Starburst: a Target Expansion Algorithm for Non-Uniform Target Distributions

Patrick Baudisch, Alexander Zotov, Edward Cutrell, and Ken Hinckley
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{baudisch,alexz,cutrell,kenh}@microsoft.com

ABSTRACT

Acquiring small targets on a tablet or touch screen can be challenging. To address the problem, researchers have proposed techniques that enlarge the effective size of targets by extending targets into adjacent screen space. When applied to targets organized in clusters, however, these techniques show little effect because there is no space to grow into. Unfortunately, target clusters are common in many popular applications. We present Starburst, a space partitioning algorithm that works for target clusters. Starburst identifies areas of available screen space, grows a line from each target into the available space, and then expands that line into a clickable surface. We present the basic algorithm and extensions. We then present 2 user studies in which Starburst led to a reduction in error rate by factors of 9 and 3 compared to traditional target expansion.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

Keywords: target acquisition, target expansion, labeling, Voronoi, mouse, pen, touch input.

1. INTRODUCTION

Acquiring a small target on a computer screen can be challenging, resulting in long targeting times and high error rates. One technique designed to help users acquire small targets is snap-to-target (e.g., [23]), which continuously sets the selection focus to the closest target. Snap-to-target effectively partitions screen space. Figure 1b labels pixels according to which target they snap to; the result is a so-called *Voronoi tessellation* [12]. Users benefit from this target expansion: instead of having to aim for the small target, users click anywhere inside the tile containing the target. This generally reduces targeting time and error rate.

Unfortunately, performance benefits depend on the homogeneity of the target layout. When applied to a target located inside a cluster of targets snap-to-target shows little effect. As illustrated by Figure 1b, targets located inside a cluster are surrounded by little empty screen space. As a result, the tiles generated by the expansion are small—associated targets remain hard to acquire. When used on a device with imprecise input, such as a touch-screen kiosk, the acquisition of such targets will be error prone. The same holds for pen input, as we demonstrate in the two user studies presented in this paper.

In real-world applications locally dense clusters of targets emerge for a variety of reasons. The user interface may represent a real-world geometry with a non-uniform structure, such as cities on a map (Figure 2a). In other cases, it is users who manually create clusters, e.g., when grouping icons on their desktops or when organizing links inside a web page (Figure 2b and c). Or clusters may emerge from the structure of visualized data (Figure 2d).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AVI'08, May 28–30, 2008, Napoli, Italy.
Copyright 2008 ACM 1-58113-000-0/00/0004…$5.00.

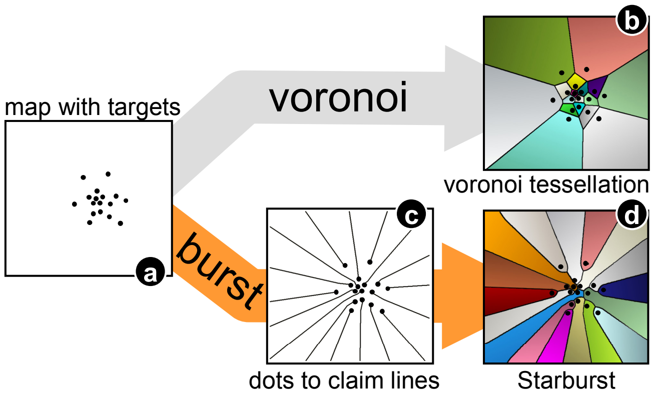

Figure 1: (a→b) Traditional snap-to-target techniques expand targets into *immediately adjacent* space. For targets located inside a cluster, however, that expansion is minimal. (a→c→d) The proposed *Starburst* algorithm connects targets to peripheral screen space to produce reasonably sized tiles for all targets, including those located inside a target cluster.

Limitations in handling target clusters are not unique to snap-to-target, but faced by all techniques based on the repartitioning of screen space, such as Bubble Cursor [14]. Some techniques even impact performance negatively if applied to target clusters. Interactions between closely adjacent *Expanding Targets* cause targets to "escape" from the user [22], resulting in a fisheye navigation problem, as discussed by Gutwin [17].

We propose addressing the problem by expanding targets in a goal-directed way.

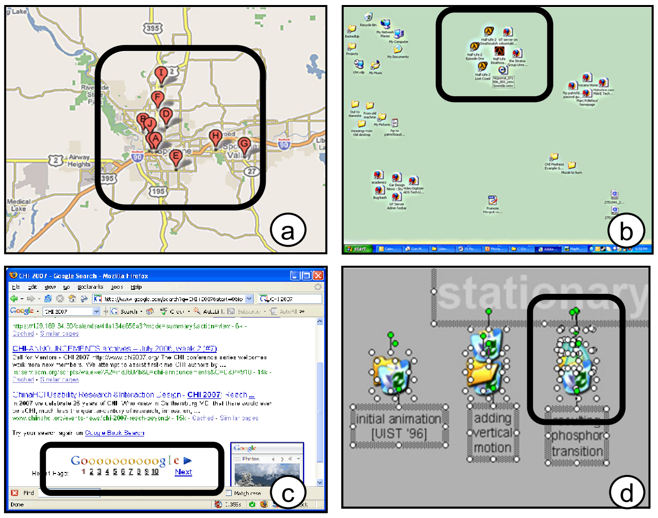

Figure 2: Non-uniform target distributions are commonplace. Examples: (a) yellow-page application showing a map of restaurants, (b) icons on a computer desktop, (c) links in a web page, and (d) handles on geometric objects in PowerPoint™.

2. THE STARBURST ALGORITHM

Figure 1 illustrates the main idea of the proposed *Starburst* technique: While the Voronoi tessellation behind a traditional snap-to-target expands targets directly into target tiles (Figure 1a→b), the proposed Starburst algorithm expands targets first into so-called *claim lines* (Figure 1a→c). Claim lines lead away from the centers of clusters and into empty screen space. Then claim lines expand into clickable surfaces (Figure 1d). The resulting layout is characterized by lines escaping from the cluster center, which gave the technique its name.

By providing targets located inside a cluster with access to empty screen space, the Starburst algorithm is able to assign screen space to targets that remain small if expanded using the traditional Voronoi approach. If used on a device with limited input accuracy, such as a pen-based tablet or a touch screen-based kiosk system, this can lead to substantial performance improvements. In our user studies, expanding targets using Starburst led to a reduction in error rate by a factor of up to 9 compared to target expansion using traditional Voronoi tessellation. The proposed algorithm thereby makes the concept of target expansion applicable to scenarios that have not been accessible to these techniques so far.

2.1. Walkthrough of the algorithm

Figure 3 shows how the Starburst algorithm converts a given target layout (Figure 3a) into a Starburst tile layout (Figure 3i).

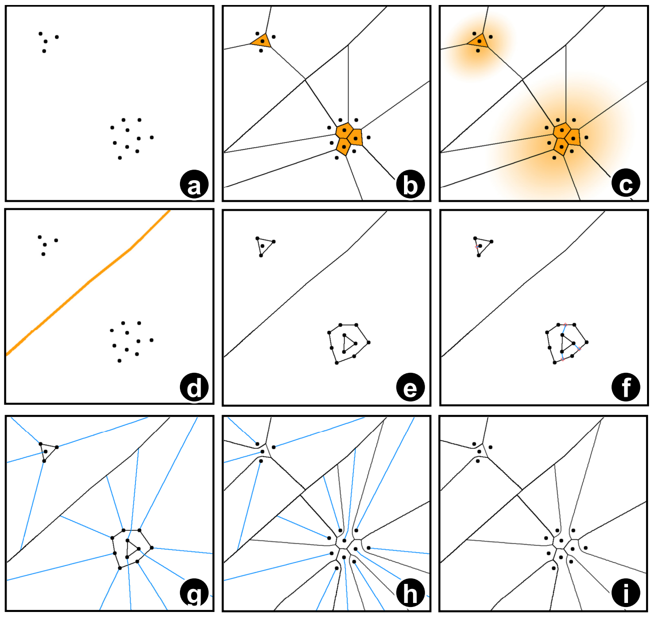

Figure 3: A walkthrough of the Starburst algorithm (a) Targets to be expanded, (b) Voronoi tessellation and identification of *recipients*, (c-d) clustering of targets into cliques, (e) nested rings, (f-g) claim line construction, (h) expansion of claim lines into tiles, and (i) final removal of claim lines.

1. Identifying targets that require additional expansion. The Starburst algorithm begins by performing a Voronoi tessellation [12] on the targets (Figure 3b). The algorithm then identifies small tiles in that Voronoi layout. Tiles which have surfaces that fall below the average tile size by a threshold (we used a factor of 5) are tagged as tiles in need of expansion. In Figure 3b these *recipients* are highlighted in orange. All other targets are tagged *donors*.

2. Organizing targets into cliques. Starburst manages the redistribution of screen space based on what we call *cliques*. A clique is a set of collocated donors and recipients. Within a clique, donors provide the screen space used to expand recipients. The Starburst algorithm computes cliques in three steps. First, it creates cliques by clustering recipients based on adjacency. In Figure 3c this results in a clique with three recipients and a clique with a single recipient. Second, the algorithm adds all donors immediately adjacent to a clique of recipients to that clique. In case a donor is adjacent to multiple cliques the donor is added to the clique with the smallest average tile size. In the case of Figure 3c this adds three donors to the single-recipient clique in the top left, all others to the three-recipient clique. Third, the Starburst algorithm adds additional donors if they are particularly large or if they are located in an area in which the clique lacks good donors. In order to be included, a candidate must be adjacent to a clique and its surface must significantly exceed the average tile size *in that clique* (we used a threshold factor of 5). In Figure 3b, all donors were already added in the previous step, so no further addition takes place. Once cliques have been formed, the Voronoi tessellation and the recipient/donor labeling is dropped (Figure 3d).

The goal of the next steps is to provide targets located on the inside of a clique with access to screen space in the periphery of the clique. In order to reach the periphery, claim lines of inner targets need to pass between outer targets and so passages between targets become potential bottlenecks. We therefore create a representation that reflects these potential bottlenecks.

3. Organizing targets into nested rings: Starburst organizes the targets of each clique into a set of nested rings (Figure 3e). The algorithm starts by computing the convex hull over all targets of a clique. All targets located on that convex hull form the *outer ring*. Then Starburst computes the second ring by computing a convex hull over the remaining targets, and so on.

4. Routing claim lines. Next the algorithm creates the claim lines. The algorithm starts with the innermost ring and connects all its targets to the immediately enclosing ring (Figure 3f). Each claim line is connected to the *nearest* edge of the outer ring that can be reached with a straight line without intersecting the inner ring. This guarantees that claim lines never intersect. If multiple claim lines are connected to the same edge, the algorithm spaces them out equidistantly; single claim lines are connected in the middle of the ring edge. This helps balance the width of the tiles at the point where they pass between the targets. Then the algorithm repeats this step, i.e., all targets on the next ring plus the newly added targets are routed to the ring another layer out. In Figure 3e, the deepest clique has two nested rings, so a single iteration is sufficient for connecting all targets to the outer ring. Now the algorithm spreads the claim lines radially into the clique's peripheral screen space (Figure 3g).

5. Growing claim lines into tiles. In the last step, Starburst creates the target tiles. The algorithm does this by assigning all pixels on screen to the target with the closest claim line as shown in Figure 3h. This completes the processing and Figure 3i shows the final result without the claim lines.

2.2. Algorithms, complexity, and performance

The overall complexity of the Starburst algorithm is O(n^2) with n being the number of targets, which allows for real-time performance with dozens of targets or several hundred targets if only a subset of them moves.

Details on the computational complexity: Step 1: We compute the initial Voronoi tessellation and its Delaunay triangulation [20] using a modified Fortune algorithm in O($n \log n$) time [12]. We compute the size of the Voronoi tiles in O(h), where h is number of edges, by reusing the quad edge data structure from the Fortune algorithm. Step 2: We compare the size of each tile with its O(h) neighbors in O(n^2) worst case time (O($n \log n$) average time).

Step 3: Constructing of the nested rings, known as *onion-peeling*, can be performed in O(n log n) time [25], but since we already computed the Delaunay triangulation we perform onion-peeling in O(n) time. Step 4: In the worst case, onion-peeling generates O(n) rings, in which case routing n claim lines through n rings requires O(n^2) time. Step 5: We perform another Fortune Voronoi tessellation, this time on the claim line segments, resulting in straight line segments and parabolic segments in O(n log n) time.

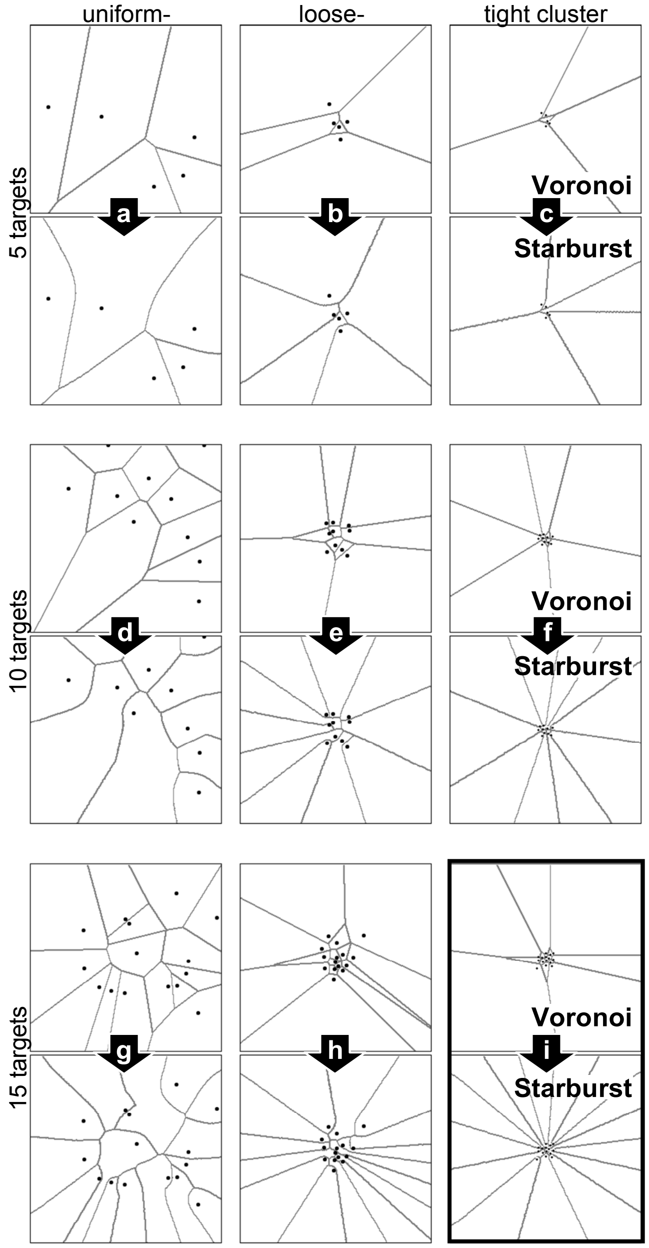

Figure 4: Examples of Voronoi tessellations (top of each pair) and the corresponding Starburst tessellation (bottom of each pair) for target layouts with 5, 10, or 15 targets and clusters of different tightness (layouts used in the user study).

2.3. Sample layouts

Figure 4 and Figure 5 show sample layouts generated using the Starburst algorithm described above and contrasts them with the corresponding Voronoi layouts.

Figure 4 shows Starburst layouts for nine single-cluster target layouts, a subset of the layouts we evaluated experimentally in our user studies. The left column of Figure 4 shows tile layouts resulting from uniform target distributions. The Voronoi-based approach was designed for uniform target distributions [14] and works as expected. Tiles in the Starburst layouts are rounder, but overall of similar quality as the Voronoi tiles. The layouts in the center column, in contrast, contain clusters. The clusters cause the Voronoi layouts to degrade visibly and inside-the-cluster targets are assigned very small tiles. The Starburst layouts, in contrast, continue to offer reasonably-sized tiles for all targets. For target layouts containing tighter clusters this effect intensifies. The vertical axis in this figure reflects the number of targets in each layout. As we move down in the diagram the target count increases. As a result, the number of inaccessible targets in the Voronoi condition increases as well. The Starburst layouts, in contrast, remain functional. Figure 5 shows a selection of multi-cluster layouts. We see similar effects as in the single cluster examples.

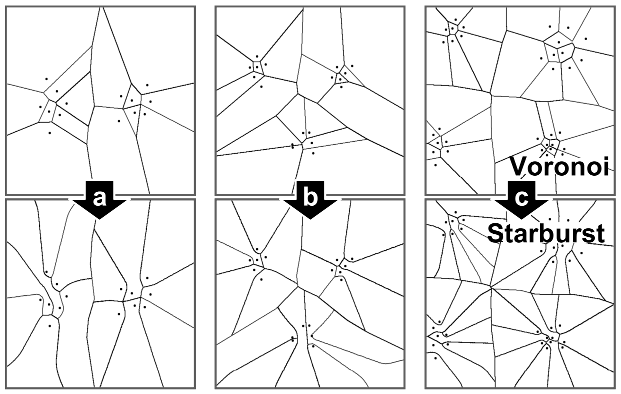

Figure 5: Examples of Voronoi (top) and Starburst (bottom) tessellations for layouts with (a) 2, (b) 3, and (c) 4 clusters.

Layouts generated by the Starburst algorithm are quite robust, i.e., insertion, removal, or relocation of targets impacts the tile layout only locally. This helps users build up spatial memory when using a Starburst layout over time.

2.4. User interface for Starburst

To outlines of Starburst tiles are irregular and therefore generally not “guessable”. A user interface deploying Starburst therefore needs to convey tile shapes to the user.

On devices supporting a hover state, such as table computers, target expansion using Starburst can be presented to the user interactively—on hover as shown in Figure 6a-c. (a) By default, only screen content is visible. (b) As the pointer moves across the screen, targets within an n-pixel radius around the pointer get increasingly “excited” and the respective tile overlays turn opaque. The tile under the pointer is highlighted. (c) Tiles away from the pointer fade to transparent, yet stay opaque long enough to allow users to tap.

Some devices, such as resistive touch screens or table top systems, do not support a tracking state. On these systems, tile boundaries are overlaid permanently onto screen content, rather than revealing them on hover. Tiles outlines can interfere with line-shaped document features as shown in Figure 6d. Such interference can often be reduced by encoding tile outlines using features not contained in the underlying document (see *multiblending* [2]).

Note that screen devices without a tracking state support *none* of the interactive expansion techniques mentioned earlier, such as Expanding Targets. Also Bubble Cursor is not applicable to such display systems; removal of the bubble visuals would reduce bubble cursor to the underlying space partitioning algorithm, i.e., a Voronoi tessellation.

2.5. Limitations

Similar to other target expansion techniques, Starburst helps overcome limitations in the clickable size of targets. A potential limitation of Starburst is that it tends to generate long and narrow targets, a type of shape that can be more difficult to acquire than rounder, wider targets [15]. In the section "Improving space allocation", we describe extensions to the algorithm that result in wider target shapes.

What remains are limitations based on the visual size of target layouts. Larger and tighter clusters result in thinner pathways at the point where claim lines pass the rings. When these pathways get so thin that they are hard to visually trace or when their thickness reaches screen resolution, some targets cannot be expanded anymore and the Starburst algorithm has reached its limit. Fortunately, as our user studies indicate, this point is reached much later than the motor space limits faced by Voronoi-based approaches.

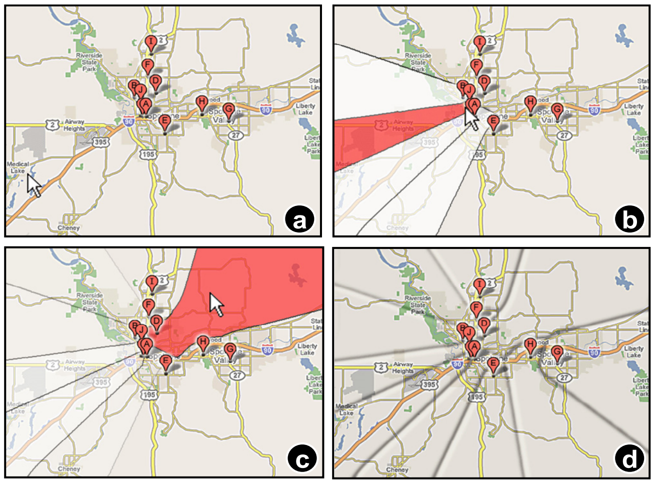

Figure 6: (a-c) on-hover exploration and (d) permanent overlay of Starburst tessellation using an emboss effect

3. RELATED WORK

Starburst is related to target acquisition and labeling.

3.1. Targeting and target expansion

In order to help users acquire small targets, researchers have proposed expanding targets in various ways.

Expansion of targets in motor space: researchers have proposed slowing down the pointer motion on and around small targets (e.g., *sticky icons* [30], also suggested by [29], *semantic pointing* [9]). Such adjustments of *control display ratio* (or *cd ratio*) increase the target's size in motor space. *Object pointing* [16] suggests removing space between targets altogether, letting users jump between targets.

Approaches based on cd ratio adjustment require users to cross the target for the cd ration enhancement to become active [3]. Researchers have therefore proposed *magnetism* [5] and *gravity* [8]. *Snap-and-go* [3] uses invisible guides that direct the user's motion while the actual propulsion still comes from the user.

On touch and pen-based systems, motor space enhancements are typically applied by using take-off selection [24]. The 1:1 mapping of these screen devices is used only to determining the initial contact position; then users iterate under a local cd ratio adjustment and commit by lifting their pen or finger off the screen (*high-precision touch screen* [26]). Benko et al. allow users to control cd-ratio manually using a second finger or the non-dominant hand [7].

Expansion of targets in visual and motor space: Some researchers have proposed manual expansion of targets using an intermitted zoom step [1, 26]. In order to apply target expansion to touch and pen-based systems with *land-on* selection [24], the motor space size of targets needs to be increased permanently. *Expanding targets,* proposed by McGuffin and Balakrishnan, refers to an expansion of the target in visual *and* motor space as the pointer approaches it [22]. For an isolated target, the motor space of the target is determined by the expanded space and McGuffin et al. found that the targeting performance is largely determined by the size of that expanded state [21].

For clusters of adjacent targets, however, target expansion in visual space causes targets to push each other away [21]. Although the visuals of each target expand fully, the proximity of the adjacent targets affects a target's ability to expand in motor space. In tightly packed clusters, *no* motor space expansion can take place.

Expansion of cursor vs. expansion of targets: To prevent these problems, researchers have looked at ways to expand targets without pushing other targets away. Bubble cursor is one such solution [14]. It shows an on-hover bubble around the pointer that varies in size, such that it contains the closest target. Bubble cursor has been applied to a variety of target acquisition techniques, such as the tractor beam [23]. There are three different ways of looking at bubble cursor. When focusing on its effect on motor space, bubble cursor divides screen space up resulting in a Voronoi diagram. The second way of looking at bubble cursor is to consider it a snap-to-target mechanism. And third, it can be considered an *area cursor* (*sticky icons* [30], also *prince technique* [18]) of adaptive size. With respect to the underlying motor space properties all three viewpoints are equivalent, although each perspective inspires a different visual user interface.

3.2. Target acquisition as a labeling problem

Another approach to associating small targets with larger motor space areas is to create a layer of *handles*—one handle for each target—that is overlaid onto the actual document content. Many programs, such as MS PowerPoint™ and Adobe Illustrator™ use little white circles to represent corners of graphical primitives that would otherwise measure only a single pixel (Figure 7a). In case a primitive is too small to fit all handles (Figure 7b, c), PowerPoint drops some of them, and finally (Figure 7d) it decouples the handles from the actual object in order to prevent handles from overlapping. Despite the decoupling, the association between handle and target is clear because of their proximity. In the case of multiple objects (Figure 7e), handles do overlap and once more it is difficult to acquire them.

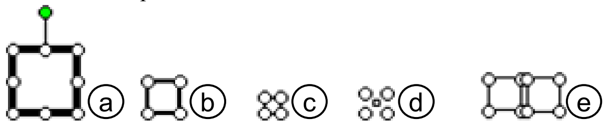

Figure 7: Resize handles in MS PowerPoint™

The idea of decoupling handles from the target can be pushed further. While we are not aware of any such research specifically designed to help users acquire small targets, a lot of research has been done on *labeling* screen objects (e.g., [19]). *Excentric labels* [11] assign labels to an entire cluster of small objects by using an explosion-drawing-like display (Figure 8). To avoid overlap between labels, they are placed at a distance from the actual targets. To associate labels and targets, this approach relies on lines and in some cases also color. While the purpose of the labels is to hold a piece of text or an icon explaining the referenced object, one could imagine using external labels for the purpose of making the associated object clickable.

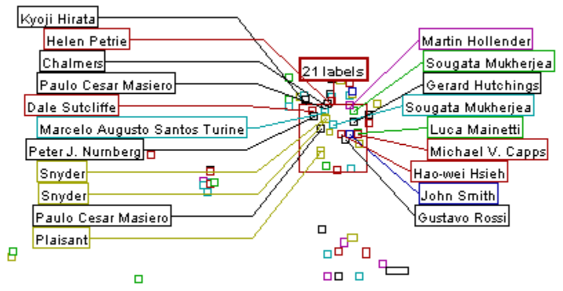

Figure 8: Excentric labels [11]

A potential limitation of this approach is that the lines produce clutter. This can make it hard for users to locate a label belonging to a particular target. Bell et al. propose an algorithm that minimizes connecting lines by placing labels onto the actual object whenever the size and shape of the target permit it [6]. The use of such *internal* labels can reduce visual search as targets and label are associated by proximity, while users need to trace a line in order to locate an *external* label.

Following this analogy, layouts produced by the Voronoi algorithm consist exclusively of internal "labels", at the expense of offering no control over their size. The Starburst algorithm, in contrast, keeps internal "labels" only if they are large enough. Otherwise it expands into an external "label". Unlike external and excentric labels, however, Starburst creates lines, tiles, and targets in the same plane, so labels never occlude targets. In that sense, Starburst shares some properties with circuit board routing [10].

4. DESIGN DISCUSSION

In this section, we give a brief overview of the design alternatives we explored and discuss their strengths and limitations. Our first two approaches were based on refining Voronoi tessellations.

4.1. Refining Voronoi by moving boundaries

Figure 9 shows a Voronoi layout and a modification obtained by moving and rotating a tile boundary. While this approach allowed for certain layout improvements, the use of straight tile boundaries turned out to be a major limitation, because many target layouts require non-straight boundaries (see, for example, the center areas of the layouts generated by Starburst in Figure 4).

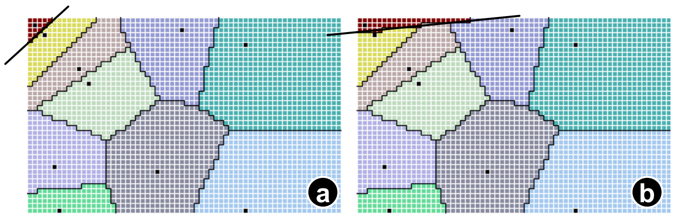

Figure 9: Boundary adjustment approach: (a) Voronoi Tessellation; (b) expansion of the tile in the top left corner by moving and rotating its boundary.

4.2. Refining Voronoi by reassigning pixels

To address this limitation, we explored algorithms that represented screen space as pixels, rather than tile boundaries. Cellular automata and pixel rewriting allow creation of a rich spectrum of shapes [13]. The high degree of flexibility, however, made it difficult to control tile growth and to direct target growth towards available space. We often obtained inefficient shapes (Figure 10c) and improving one tile often came at the expense of making another tile significantly worse (Figure 10d).

4.3. Claim lines

Based on these insights, we started looking for an algorithm that would offer flexibility *and* control. Claim lines provide tiles with a much-needed skeleton—a concept well understood in computer graphics [28]. That skeleton allowed us to direct target growth towards available space and prevent uncontrolled expansion. Yet, the resulting target tiles were not limited to straight edged or convex shapes.

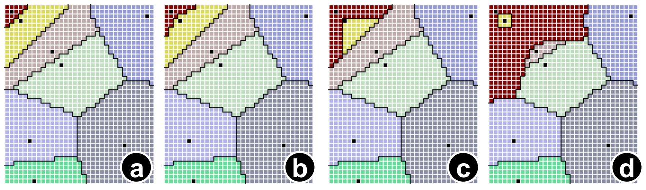

Figure 10: Pixel rewriting approach: (a) Voronoi tessellation; (b) expansion of the top left target using pixel rewriting; (c, d) further expansion leading to undesirable target shapes.

We went through several design iterations to determine a claim line skeleton that would offer enough flexibility to avoid bottlenecks yet be simple enough to allow for good control.

Our first attempt used single-segment claim lines, which it created by drawing a straight line from a common "center point" located inside the cluster through the individual targets. This approach turned out to be too limited and long strips of targets resulted in inefficient space usage.

To address these shortcomings, we switched to multi-segment lines. We tried to avoid bottlenecks by making claim lines repel each other, yet that made it difficult to direct claim lines towards available screen space.

Our final version, the nested ring approach, finally, reduced the number of line segments to what was absolutely necessary and offered a good handle on bottlenecks. This resulted in cleaner layouts, faster computation, and the desired degree of control.

5. IMPLEMENTATION

Figure 11 shows our Starburst test environment. It allows placing targets and generating tile layouts using a variety of algorithms. It was implemented using the .NET WinForms framework and runs on Microsoft Windows XP Tablet PC Edition.

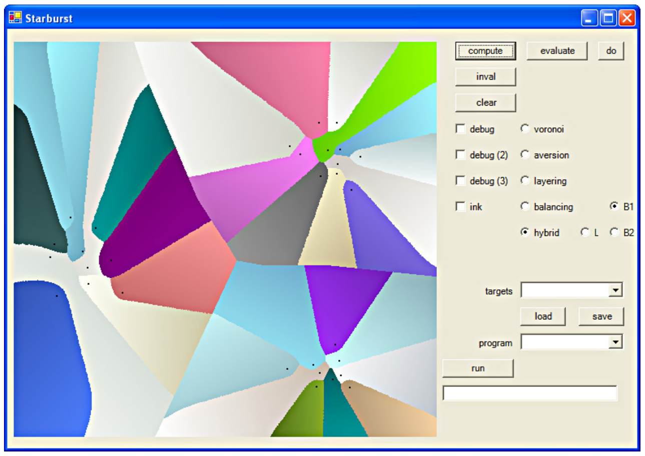

Figure 11: The Starburst test environment for Tablet PC

6. USER STUDIES

To objectively evaluate the performance of the Starburst algorithm, we conducted two controlled experiments comparing Starburst with traditional Voronoi target expansion.

The goal of the first experiment was to verify that our technique indeed reduces the motor skills required to select clustered targets. Voronoi and Starburst both make use of the entire screen space—the average size of generated tiles is therefore the same. Starburst does not *increase* tile sizes compared to Voronoi, but *balances* tile sizes; its median target size is higher that Voronoi's, not its mean. On the flipside, as discussed earlier, targets generated by Starburst tend to be longer and thinner. We were wondering how the two effects would play out against each other. The first study investigated this by highlighting the entire target tile (Figure 12a).

After finding a very strong effect in the first study (a reduction of error by a factor of nine) we conducted a second study. This time we looked at a more realistic scenario simulating a user encountering a target layout for the first time or who works with a layout undergoing perpetual change. How effectively would users acquire targets now? We implemented this scenario by highlighting the target only, not the tile, so that users had to visually examine layouts for every trial to determine where to tap (Figure 12b).

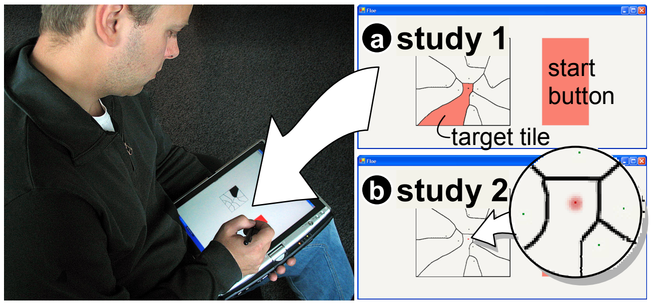

Figure 12: Participants tapped the start button and then the tile associated with the target. (a) In study 1, the entire target tile was highlighted, (b) in study 2 only the target itself.

7. USER STUDY 1

The participants' task in the first study was to acquire targets with a pen on a tablet computer (Figure 12). Target acquisition was supported by expanding all targets in the target layout into a space-filling layout of tiles. Participants could acquire a target by acquiring any part of the associated tile. As mentioned above, the entire tile associated with the target was shaded red (Figure 12a).

Our main hypothesis was that participants would acquire with less errors if layouts were generated using Starburst.

7.1. Interfaces

There were two interface conditions. In the *Starburst* condition, target tile layouts were generated using the algorithm described at the beginning of this paper. In the *Voronoi* condition, target tile layouts were generated using the traditional Voronoi approach.

Both interfaces provided permanently visible tile boundaries, i.e., a set of black lines as shown in Figure 12. We chose this interface style, because it is available on all devices—unlike interface styles relying on hover.

7.2. Target layouts

Target layouts measured 256 x 256 pixels and 2" x 2" (5 x 5 cm) on screen. To keep the number of trials manageable and since multi-cluster layouts are structurally similar (Figure 5) we used uniform and single-cluster layouts only. Figure 4 show examples for each of the nine types of target layouts used in the study: each target layout contained 5, 10, or 15 targets; targets were organized either in a uniform distribution (*uniform*), in a normal distribution with standard deviation of 32 pixels (*loose*), or in a normal distribution with standard deviation of 8 pixels (*tight*). For each of the nine layout types we randomly generated 5 target layouts. Each participant completed each layout using each of two interfaces. This resulted in 3 target counts x 3 densities x 5 layouts x 2 interfaces = 90 layouts.

7.3. Task

The participants' task was to acquire targets using the pen. Each trial proceeded as follows. (1) The current target was highlighted in gray and the start button turned red as shown in Figure 12. (2) Participants tapped the start button (100 x 256 pixels, 0.8" x 2"/2cm x 5cm) located right of the target layout. This was acknowledged with a "click" sound and started the timer for that trial. (3) Participants acquired the highlighted target by tapping anywhere within its tile using the pen. This stopped the timer. Success/failure was confirmed using auditory feedback.

While participants acquired one target per trial, performance was measured on a per-layout basis. A per-target comparison did not make sense, because target sizes and shapes of the tiles in a layout were not independent from each other; adding space to one target to make it easier to acquire came at the expense of making another one smaller and thus harder to acquire.

This meant that participants needed to perform 10 times more target acquisitions for the same number of data points than in a normal target acquisition study. In order to keep the number of repetitions manageable, distance and angle of the target were *not* varied in this experiment. Instead we used the aforementioned start button located at a fixed position. While the start button placement could impact targeting times of individual targets, its effect balanced out across entire layouts.

7.4. Procedure

Each participant acquired every target of the 90 tile layouts once, i.e., there were 45 target layouts, each one tessellated differently for each of the two interface conditions. Each participant therefore performed a total of 3 levels of target counts (5, 10, or 15 targets) * 3 densities (uniform, loose, tight) * 5 layouts * 2 interfaces = 900 trials. To minimize learning and ordering effects, the order of all 900 trials was randomized, so that in the general case the entire target layout changed from trial to trial. Overall, the user study took about 20 minutes per participant.

7.5. Apparatus

Participants performed all tasks using a Toshiba Portégé M200 Tablet PC, with a 12.1" inch LCD monitor running the Microsoft Windows XP Tablet PC Edition operating system. The screen measured 7½" x 9¾" (19cm x 25cm), offered 1400 x 1050 pixel resolution (140dpi), and was used in portrait orientation. Participants performed all interaction using a pen. The tablet keyboard was hidden ("slate mode"). The tablet was placed on a table, but participants were allowed to hold the tablet in the lap instead, if they preferred (Figure 12). The experimental application was implemented using the .NET WinForms framework.

7.6. Participants

12 volunteers (10 male) between the ages of 20 and 40 were recruited from our institution. Each one received a lunch coupon for our cafeteria as a gratuity for their time. All had experience with graphical user interfaces, TabletPC, and pen input. Nine participants were right handed. All had normal or corrected to normal vision and normal color vision.

7.7. Hypotheses

We had the following three hypotheses:

(H1) Participants would acquire target layouts faster and with fewer errors for the clustered target layouts (*loose* and *tight* conditions) when using the Starburst interface.

(H2) The performance benefit of the Starburst condition would increase with the number of targets in a layout. The reason is that a higher target count would cause more targets to be enclosed inside clusters in the Voronoi condition.

(H3) The performance benefit of the Starburst condition would be greater in the *tight* condition. In the Voronoi condition, the tighter packing would make tiles of targets located inside a cluster even smaller.

We did not expect any performance benefits for the Starburst interface in the uniform layout conditions because neither of the techniques should produce any small targets.

7.8. Results

Performance was measured in error rates and targeting times for each condition.

7.8.1. Error rates

We aggregated selection errors across all 5 layouts per condition to compute an error metric for each condition. We then performed a 3 (*TargetCount*) × 3 (*Density*) × 2 (*Technique*) within subjects analysis of variance. We found significant main effects for all three variables. For *TargetCount* (F(2,22)=92.5, p<<0.001), accuracy decreased as the number of targets increased. Similarly for *Density* (F(2,22)=158.4, p<<0.001), as the density increased, so did the error rate. Finally, for *Technique* (F(1,11)=272.1, p<<0.001), Voronoi was associated with significantly higher error rates than Starburst (14% vs. 2% error).

In addition, all interactions tested were significant: *TargetCount x Density*, F(4,44)=24.1, p<<0.001; *TargetCount x Technique*, F(2,22)=51.2, p<<0.001; *Density x Technique*, F(2,22)=204.8, p<<0.001; and *TargetCount x Density x Technique*, F(4,44)=12.9, p<<0.001. Figure 13 illustrates all the error rates for each technique and all display conditions. Post hoc paired t-tests were performed comparing each technique at each condition and significant differences are denoted by "*" (Bonferroni adjustment for multiple tests, p<0.005).

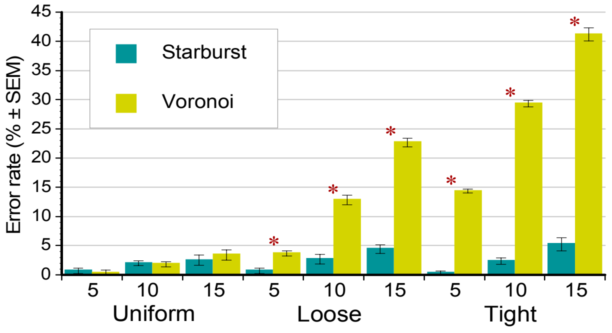

Figure 13: Error rates over layout types (+/- std error of mean)

7.8.2. Target acquisition times

Before analyzing target acquisition times, outliers were removed from the analysis based on a heuristic of any acquisition longer than 2 seconds (this is well over 3 standard deviations from the mean for a given condition). A total of 55 out of 10745 trials were removed from the data (45 from the Voronoi conditions).

As with error rates, for time analyses we collapsed target acquisition times across all 5 layouts per condition, computing the median target acquisition time for each condition. We performed a 3 (*TargetCount*) × 3 (*Density*) × 2 (*Technique*) within subjects analysis of variance for acquisition time. We found significant main effects for all three variables. For *TargetCount* (F(2,22)=244.4, p<<0.001), acquisition time increased as the number of targets increased. Similarly for *Density* (F(2,22)=76.3, p<<0.001), as the density increased, so did target acquisition time. Finally, for *Technique* (F(1,11)=65.9, p<<0.001), Starburst was significantly faster than Voronoi.

In addition, all interactions tested were significant: *TargetCount x Density*, F(4,44)=20.7, p<<0.001; *TargetCount x Technique*, F(2,22)=11.0, p<0.01; *Density x Technique*, F(2,22)=47.5, p<<0.001; and *TargetCount x Density x Technique*, F(4,44)=7.8, p<0.01. Figure 14 illustrates targeting times for each technique and all display conditions. Post hoc paired t-tests were performed comparing each technique at each condition and significant differences are denoted by "*" (Bonferroni adjustment for multiple tests, p<0.005).

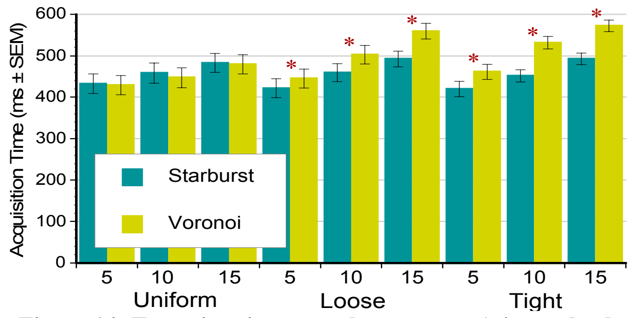

Figure 14: Targeting times over layout types (+/- standard error of the mean).

7.9. Discussion

In summary, the study results support all three hypotheses. Participants acquired tiles layouts generated using Starburst faster and with a substantially lower error rate than tiles generated by the Voronoi conditions. This supports our hypothesis that the improved balancing of target sizes outweighs the drawback resulting from the degeneration of tile shapes. Tighter clusters and more targets increased the gap in performance.

8. USER STUDY 2

As mentioned earlier, the purpose of the second study was to investigate the more realistic scenario where users encounter a target layout for the first time. The second study was identical to the first, except:

Interfaces: only the target itself was highlighted, but not the corresponding tile, so that users had to visually examine the layout to determine where to click. Since targets were very small, they were also provided with a pale red glow to make them easier to locate, as shown in Figure 12b. As before, targets were revealed upon completion of the previous trial. All participants tapped start in immediate succession to completing a trial and did not inspect layouts before tapping start.

Additional density condition: We only tested the 5 and the 10 target conditions, but not the 15 target conditions (Figure 5). Participants therefore now performed 2 target counts x 3 densities x 5 layouts x 2 interfaces = 60 layouts.

Participants: 6 participants (5 male); all with GUI experience; 2 Tablet PC users and pen input experience; 5 right handed and one left handed. All had normal or corrected to normal vision and normal color vision.

Hypotheses: As in the first study, we expected to see a benefit in error rate. Since the visual analysis of the Starburst layout would take time, we did not expect to see a benefit in task time though.

8.1. Results

Performance was measured in error rates and targeting times for each condition.

8.1.1. Error rates

Analyses for Study 2 were nearly identical to Study 1. While the accuracy rates tended to be slightly lower (reflecting the increased task difficulty), the pattern was the same. We performed a 2 (*TargetCount*) × 3 (*Density*) × 2 (*Technique*) within subjects analysis of variance. We found significant main effects for all three variables. For *TargetCount* (F(1,5)=42.9, p<0.001), accuracy decreased as the number of targets increased. Similarly for *Density* (F(2,10)=97.9, p<<0.001), as the density increased, so did the error rate. Finally, for *Technique* (F(1,5)=37.6, p<0.002), Voronoi was associated with significantly higher error rates than Starburst (10% vs. 4% error).

Unlike study 1, only 2 interactions were significant: *TargetCount x Density*, F(2,10)=17.8, p<0.001; and *Density x Technique*, F(2,10)=39.6, p<<0.001. Figure 15 illustrates all the hit rates for each technique and all display conditions. Post hoc paired t-tests were performed comparing each technique at each condition and significant differences are denoted by "*" (Bonferroni adjustment for multiple tests, p<0.008).

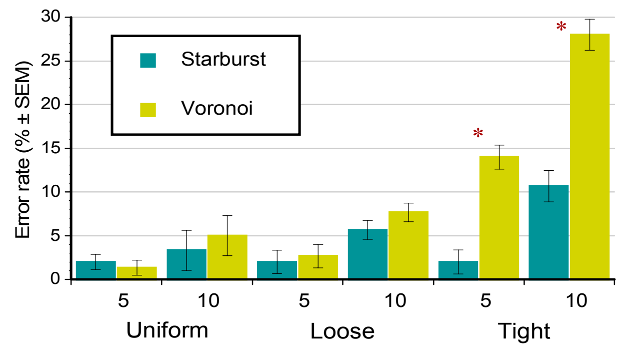

Figure 15: Error rates over layout types (+/- std error of mean)

8.1.2. Target acquisition times

As expected, the time for target acquisition was generally longer than in study 1, reflecting the greater difficulty of the task. We performed a 2 (*TargetCount*) × 3 (*Density*) × 2 (*Technique*) within subjects analysis of variance for acquisition time. We found significant main effects for all three variables. For *Target-Count* (F(1,5)=18.3, p<0.001), acquisition time increased as the number of targets increased. Similarly for *Density* (F(2,10)=6.49, p<0.02), as the density increased, so did target acquisition time. Finally, for *Technique* (F(1,5)=10.7, p<0.02), Starburst was significantly faster than Voronoi.

No interactions were significant. Figure 16 illustrates targeting times for each technique and all display conditions. As above, post hoc paired t-tests were performed comparing each technique at each condition and significant differences are denoted by "*" (Bonferroni adjustment for multiple tests, p<0.008).

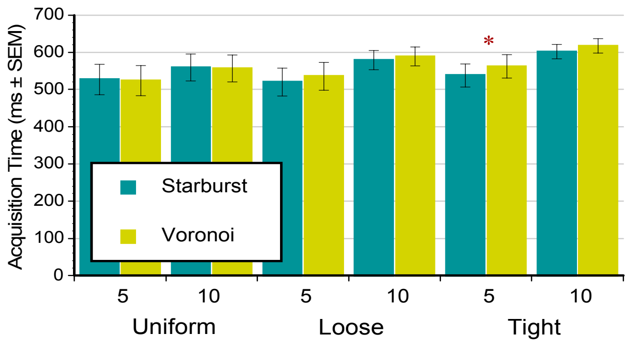

Figure 16: Targeting times over layout types (+/- standard error of the mean)

8.2. Discussion

Also the second study results support our hypotheses. While the visual analysis of the Starburst layout resulted in longer task times and higher error rates in both interface conditions compared to the first study, the Starburst layout still outperformed the Voronoi layout on both measures.

9. IMPROVING SPACE ALLOCATION

The Starburst algorithm, as described throughout this paper, improves target tile layouts by reallocating screen space from donors to recipients. While the algorithm delivers good results for the average case, it can lead to suboptimal results if the supply of screen space is distributed unequally around a cluster. In the example shown in Figure 17a, for example, the five claim lines in the bottom left access only limited amounts of screen space. In the following, we present an extension of our algorithm that causes it to take the availability of screen space into account. The extension replaces step 4 of the original algorithm as follows.

4a. Locate available screen space. To probe space availability this algorithm casts rays from the outer ring into the periphery, intersects them with the clique boundaries (dashed and dotted lines in Figure 17b), and measures the length of the ray. Sectors that are too "shallow" are excluded from the following space allocation steps (finely dotted lines in Figure 17b).

4b. Place claim line endpoints. The algorithm places claim line endpoints into the sectors marked as available. For a reasonably small number of targets per clique, such as 20, the algorithm partitions screen space radially as shown in Figure 17c.

4c. Route claim lines between targets and endpoints. The algorithm descends claim lines from the endpoints to the closest segment of the outer ring. Then it flips pairs of connections until claim lines do not intersect each other anymore. It repeats this step for all remaining ring layers.

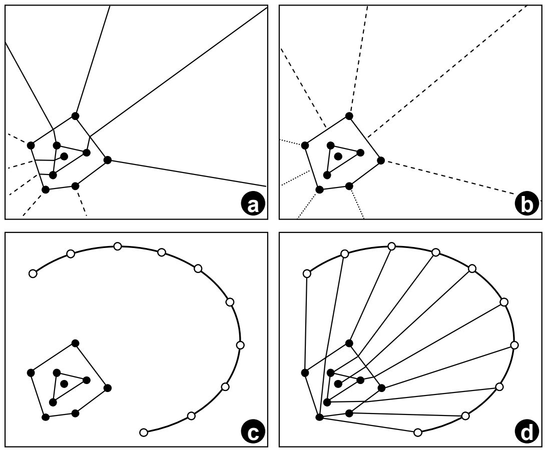

Figure 17: (a) The 5 dashed claim lines have limited access to screen space. The extension (b) locates available screen space, (c) places claim line endpoints into the available screen space, and then (d) routes claim lines from endpoints to targets.

Figure 18 juxtaposes a tile layout generated using the basic Starburst algorithm with the corresponding layout produced by the extended version.

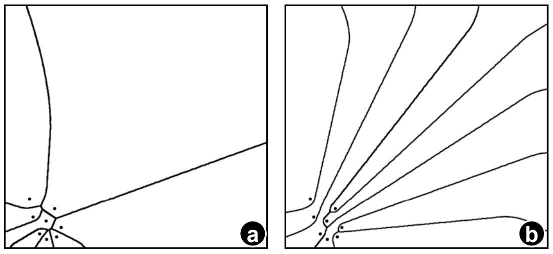

Figure 18: (a) A tile layout generated using the basic Starburst method and (b) using the extended version

For clusters with more than 20 targets, spreading claim line endpoints along a single arc produces very thin tiles that can be hard to acquire [15]. To avoid this, our algorithm handles large numbers of endpoints by laying them out in two or more layers as shown in Figure 19a (this example uses the 8 targets layout from Figure 18 to allow juxtaposing the resulting layouts). When growing claim lines into tiles in step 5, endpoints are given additional "attraction". This causes tiles to inflate around their endpoints, which provides tiles with a "handle", making them easier to acquire. Figure 19b shows the resulting tile layout.

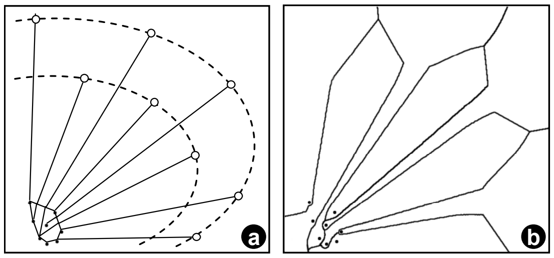

Figure 19: (a) Organizing claim line endpoints in multiple layers (b) helps thicken targets in this tile layout.

10. CONCLUSIONS

In this paper, we presented Starburst, an algorithm that extends the concept of target expansion to target layouts that contain clusters. Our user studies support our claims that the presence of target clusters limits the applicability of Voronoi-based target expansion techniques and demonstrated substantial performance benefits for the proposed Starburst technique.

As future work we plan to extend the algorithm to allow it to expand starting with arbitrary target shapes, such as buttons in graphical user interfaces. We also plan to experimentally evaluate Starburst's on-hover user interface, e.g., by comparing it against bubble cursor.

ACKNOWLEDGMENTS

We thank Heather Thorne, Raman Sarin, Tovi Grossman, and Merrie Morris for their comments on earlier drafts of this paper. Thanks to Michael McGuffin and Tovi Grossman for allowing us to use demo sequences of expanding targets and bubble cursor in our demo video.

REFERENCES

1. Albinsson, P.-A. and Zhai, S. High precision touch screen interaction. In *Proc CHI'03*, pp. 105-112.
2. Baudisch, P. and Gutwin, C. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proc. CHI'04*, pp. 367-374.
3. Baudisch, P., Cutrell, E., Hinckley, K., and Eversole, A. Snap-and-go: Helping Users Align Objects Without the Modality of Traditional Snapping. In *Proc. CHI'05,* pp. 301-310.
4. Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P. Bederson, B., and Zierlinger, A. Drag-and-Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-operated Systems. In *Proc. Interact'03*, pp. 57-64.
5. Beaudouin-Lafon, M. & Mackay, W. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proc. AVI'00,* pp.102–109.
6. Bell, B., Feiner, S., and Höllerer, T. View Management for Virtual and Augmented Reality. In *Proc. UIST '01*, 101-110.
7. Benko, H., Wilson, A., and Baudisch, P. Precise Selection Techniques for Multi-Touch Screens. In *Proc. CHI'06*, pp. 1263-1272.
8. Bier, E. and Stone, M. Snap dragging. In *Proc. SIGGRAPH'86,* pp. 233–240.
9. Blanch, R. Guiard, Y., Beaudouin-Lafon, M. Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation. In *Proc. CHI'04*, pp. 519–526.
10. Dion, J. (1987). *Fast printed circuit board routing.* ACM Press New York, NY, USA.
11. Fekete, J.-D., and Plaisant, C. Excentric labeling: dynamic neighborhood labeling for data visualization. In *Proc. CHI'99*, pp. 512–519.
12. Fortune, S. A sweepline algorithm for Voronoi diagrams. In *Algorithmica* 2(1):153-174, March 1987.
13. Furnas, G.W. and Qu, Y. Shape manipulation using pixel rewrites. In *Proc. Visual Computing 2002* (VC'02), published in *Proc. DMS2002*, pp. 630-639.
14. Grossman, T. and Balakrishnan, R. Bubble cursor: Enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proc. CHI 2005,* p. 281-290.
15. Grossman, T., and Balakrishnan, R. A probabilistic approach to modeling two-dimensional pointing, *TOCHI Volume 12,* Issue 3 (September 2005), p. 435-459.
16. Guiard, Y., Blanch, R., and Beaudouin-Lafon, M. Object pointing: A complement to bitmap pointing in GUIs. In *Proc. GI'04,* pp. 9-16.
17. Gutwin, C. Improving Focus Targeting in Interactive Fisheye Views. In *Proc. CHI'02,* pp. 267–274.
18. Kabbash, P. and Buxton, W. The prince technique: Fitts' law & selection using area cursors. In *Proc. CHI'95,* pp. 273–279.
19. Kakoulis, K. and Tollis, I. *Intl. Journal of Computational Geometry and Applications* 13(1):23–59. (2003).
20. Lischinski D. Incremental Delaunay triangulation. In *Graphics Gems IV.* Academic Press, pp. 47–59 (1994).
21. McGuffin, M, and Balakrishnan, R. Fitts' Law and Expanding Targets: Experimental Studies and Designs for User Interfaces. *TOCHI (*12)4:388-422, Dec. 2005.
22. McGuffin, M., and Balakrishnan, R. Acquisition of Expanding Targets. In *Proc. CHI'02*, pp. 57-64.
23. Parker, J., Mandryk, R., Nunes, M., and Inkpen, K. Improving target acquisition for pointing input on tabletop displays. In *Proc. INTERACT 2005*, pp 80-93.
24. Potter, R. L., Weldon, L. J., and Shneiderman, B. (1988). Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proc. CHI'*88, pp. 27-32.
25. Preparata, F., Shamos, M. 1985. Computational Geometry: An Introduction. Texts and Monographs in Computer Science. Springer-Verlag, New York
26. Ramos, G**.**, Cockburn, A., Beaudouin-Lafon, M. and Balakrishnan, R. Pointing Lenses: Facilitating Stylus Input through Visual- and Motor-Space Magnification. In *Proc. CHI'07*. pp. 757 – 766.
27. Sears, A. and Shneiderman, B. (1991). High precision touchscreens: design strategies and comparisons with a mouse. *Int. J. Man-Mach. Stud.* 34(4):593-613.
28. Sederberg, T. and Parry, S. Free-form deformation of solid geometric models. In *Proc. SIGGRAPH 86*, pp. 151-160.
29. Swaminathan, K. and Sato, S. (1997) Interaction design for large displays. In *Interactions* 4(1):15–24.
30. Worden, A., Walker, N., Bharat, K and Hudson, S. Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. In *Proc. CHI '97*, pp. 266–271.