Efficient Algorithms for Shared Camera Control*

Sariel Har-Peled† Vladlen Koltun‡ Dezhen Song§ Ken Goldberg§

Abstract

We consider a system that allows n networked users to share control over a robotic webcamera. Each user guides the camera pan, tilt and zoom, by drawing a rectangle in the user interface. The server adjusts the camera to best satisfy the user requests, by solving a geometric optimization problem that requires fitting one rectangle to many. We improve upon previous results with an $O(n^{3/2}\log^3 n)$ time exact algorithm for this problem. We also present a simple near-linear time ε-approximation algorithm. We have implemented the latter and report on experimental results.

1. Introduction

Background. Robotic webcameras are now commercially available. They can be placed at sites of major interest, such as a sports event, rock concert, rescue operation, or space station. Live images from the site are accessible to many users via the Internet. Since the camera is robotic, its pan, tilt and zoom parameters can be controlled by the users on-line. The challenge is to find the parameters that best satisfy many simultaneous users.

In existing robotic webcamera implementations [2, 3], each user gets exclusive control of the camera for a given time slot, after which the control moves to the next user in the queue. This has numerous disadvantages. At any given time, all users but one are inactive and prone to frustration. A potentially malicious user has complete control over the camera for a whole time slot. At events of wide interest, the majority of users never get to participate, since the event is long over before their turn arrives.

We propose a system where users share control over the camera at all times. Every user specifies the desired camera parameters via an intuitive interface. The server determines an orientation and zoom for the camera that aim to best satisfy the requests. Users are free to update their requests at any time, and the camera adjusts dynamically. The system is scalable, allowing any number of users to join or leave.

Underlying our "ShareCam" system are efficient geometric algorithms for selecting the optimal camera position. Their task is to dynamically adjust the camera in a way that best satisfies the desires of the users. Clearly, these server-side algorithms in charge of the camera have to successfully reconcile sometimes drastically different user requests. These algorithms are the focus of this paper.

Additional applications. Aside from the obvious applications to entertainment, the ShareCam system can be used for education, journalism and research. Moreover, the geometric algorithms presented in this paper are applicable in other scenarios of shared (possibly Internet-based) control of a single mechanism, for instance the collaboratively operated industrial robot arm described by Goldberg and Chen et al. [12, 13], the remotely managed waste cleanup system presented by Cannon and McDonald et al. [7, 17], and the "Tele-Actor" system of Goldberg and Song et al. [14], in which the motion of a single human agent is driven by multiple user votes. Our algorithms can also be applied to certain other optimization problems that require fitting one rectangle to a set of input rectangles, encountered, e.g., in architecture and city planning.

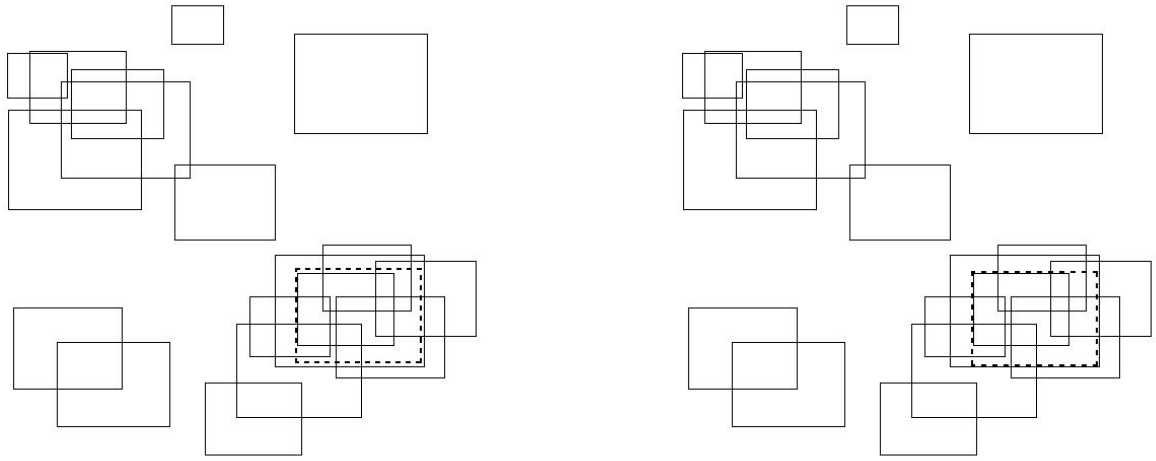
Problem formulation. The ShareCam user interface consists of two windows. The first displays the live visual stream received from the robotic camera. The second is a panoramic window that shows a fixed image of the whole accessible region of the camera (e.g., the overview of the stadium). This is the window in which the users specify their requested camera parameters (pan, tilt, and zoom). This is done by simply drawing an axis-parallel rectangle that delimits the part of the accessible region that is currently of interest to the user. The rectangle has a fixed aspect ratio corresponding to the aspect ratio of the camera. Thus such a rectangle uniquely specifies the orientation and zoom of the camera, as requested by the user. A small rectangle corresponds to a large requested zoom level, implying that the user wishes to see the delimited area in detail. The panoramic window also

†Department of Computer Science, DCL 2111, University of Illinois, 1304 West Springfield Ave., Urbana, IL 61801, USA. *sariel@uiuc.edu*

‡Computer Science Division, University of California, Berkeley, CA 94720-1776, USA. *vladlen@cs.berkeley.edu*

§IEOR Department, University of California, Berkeley, CA 94720-1777, USA. {*dzsong,goldberg*}*@ieor.berkeley.edu*

*Work on this paper by Vladlen Koltun has been partially supported by NSF Grant CCR-01-21555. Work on this paper by Sariel Har-Peled has been partially supported by NSF CAREER award CCR-0132901. Work on this paper by Ken Goldberg and Dezhen Song has been partially supported by NSF Grant IIS-0113147 and by Intel Corporation.



(a) (b)

Figure 1: The content of the panoramic window of the ShareCam user interface is shown in (a) and (b). For the sake of clarity, we do not show the preview image of the whole region accessible by the camera that is usually shown to ShareCam users in this window. The solid rectangles are requests made by 18 users of the system. The camera rectangle computed by our approximation algorithm is shown dashed in (a). For comparison, (b) shows the optimal camera rectangle, computed by a brute force algorithm.

displays the rectangles drawn by other users currently logged into the system, as well as the *camera rectangle*, showing the area currently viewed by the camera; see Figure 1.

At any given time, the ShareCam server is thus faced with the following geometric optimization problem (referred to below simply as *the ShareCam problem*). Given a collection $\mathbf{R} = \{R_1, \ldots, R_n\}$ of axis-parallel rectangles with a common aspect ratio (called the *user rectangles*), where n is the current number of users, find an axis-parallel rectangle C_{opt} (called the *optimal camera rectangle*), with the same aspect ratio, that maximizes the *global satisfaction function*

$$\mathcal{S}(C) = \mathcal{S}_{\mathbf{R}}(C) = \sum_{i=1}^{n} S_i(C),$$

where $S_i(\cdot)$ is the *individual satisfaction function* of user i. This function measures the similarity of R_i and C, and can be defined in several possible ways. We introduce the following function, which we call Intersection Over Maximum (IOM):

$$S_i(C) = IOM(C, R_i) = \frac{Area(C \cap R_i)}{\max(Area(C), Area(R_i))}.$$

If C is disjoint from R_i, $S_i(C) = 0$; if C overlaps R_i exactly, $S_i(C) = 1$; if C partially overlaps R_i, $0 < S_i(C) < 1$; increasing the size of C while keeping the overlap area $C \cap R_i$ fixed causes $S_i(C)$ to diminish. The last property encourages the camera to not only have the requested area in view, but also at the appropriate zoom level. After all, if the user requests the camera to concentrate on a certain detail, having this detail on screen but at a much lower zoom level does not bring perfect satisfaction. Otherwise, the optimal camera rectangle would always tend to contain the whole accessible region, with little consideration of user requests.

Another similarity function that can be adopted as $S_i(C)$ is the Intersection Over Union (IOU) measure, also known as the Jaccard measure [5, 16, 21]:

$$IOU(C, R_i) = \frac{Area(C \cap R_i)}{Area(C \cup R_i)}.$$

Intersection Over Union is directly related to the Symmetric Difference (SD) measure:

$$\begin{aligned} SD(C, R_i) &= \frac{Area(C \cup R_i) - Area(C \cap R_i)}{Area(C \cup R_i)} \\ &= 1 - IOU(C, R_i). \end{aligned}$$

The unnormalized version of the SD measure was used by de Berg et al. [10] to assess the dissimilarity of two polygons.

Although Intersection Over Union satisfies the properties described above for Intersection Over Maximum, it does not possess the favorable properties of the latter, as analyzed in Section 2. We thus adopt Intersection Over Maximum throughout the sequel.

Previous work. The ShareCam problem bears some resemblance to other geometric optimization problems, in particular to variants of the p-center and the p-median problems [4]. However, no question substantially similar to the ShareCam problem has, to our knowledge, been previously studied.

The ShareCam problem was introduced by Song, Van der Stappen, and Goldberg [20], who have also outlined preliminary solutions. They discretize the range of zoom values that the camera can adopt into a set of m equally spaced values. A specific zoom fixes the side lengths of the camera rectangle, since its aspect ratio is fixed *a priori*. Hence, for a given zoom value, the ShareCam problem simplifies to fitting a *fixed* rectangle to the set of user rectangles. Song et al. [20] show how to solve this problem in time $O(n^2)$. They thus compute the optimal camera rectangle independently for each zoom value, yielding m candidate rectangles C_i, each of them optimal for a specific zoom. They then choose the one that provides the highest value of the satisfaction

function $\mathcal{S}_{\mathbf{R}}(C_i)$. The overall running time of the algorithm is thus $O(mn^2)$.

This procedure finds the camera rectangle that is optimal among all rectangles with one of the m allowed zoom levels. The discretization of the range of zoom levels makes sense in light of the limitations of current robotic camera technology, and allows for practical algorithms. The iteration over the set of m different zoom levels slows down the computation by only a small constant. Unfortunately, the above algorithm takes time $O(n^2)$ even when the zoom is fixed.

Our contribution. Following Song et al. [20], we search for an optimal camera rectangle independently for each zoom level of the camera, and then choose the global optimum. The focus of our work is thus obtaining more efficient algorithms for solving the ShareCam problem when the size of the camera rectangle is fixed.

For this setting, we present an exact algorithm that runs in time $O(n^{3/2}\log^3 n)$. This compares favorably with the quadratic running time achieved in [20]. This result is described in Section 4.

We also present a simple ε-approximation algorithm that runs in time $O(N \log N)$, where $N = O\left(n\frac{\log^2(1/\varepsilon)}{\varepsilon^2}\right)$. Specifically, for a given parameter $\varepsilon > 0$, our algorithm finds in this time a camera rectangle C that satisfies $\mathcal{S}(C) \geq (1-\varepsilon)\mathcal{S}(C_{opt})$, where C_{opt} is the optimal camera rectangle for the given zoom level. In other words, the algorithm computes, in near-linear time, a camera rectangle whose satisfaction value is as close to optimal as specified by ε. This result is described in Section 3.

We have implemented this approximation algorithm and compared its performance to the algorithm presented in [20]. Surprisingly, despite the strong asymptotic superiority, the current implementation of the new algorithm is slower unless the number of users is impractically large. This is discussed in Section 5.

2. Preliminaries

As described in the previous section, we can assume that the zoom level of the camera is fixed. The camera rectangle is thus determined by the coordinates of its center point p in the plane, and can be expressed as $C(p)$. We are given a set $\mathbf{R} = \{R_1, \ldots, R_n\}$ of user rectangles. For a given user rectangle R_i, the satisfaction function $S_i(\cdot)$ can be viewed as a bivariate function defined over the two-dimensional plane:

$$S_i(p) = \delta_i Area(C(p) \cap R_i),$$

where $\delta_i = 1/(\max(Area(C(p)), Area(R_i)))$ is constant, since $Area(C(p))$ is determined by the fixed zoom level. The graph of S_i has the form of a plateau, as illustrated in Figure 2. The locus of points p for which $C(p) \cap R_i \neq \emptyset$, and thus $S_i(p) > 0$, is the Minkowski sum of R_i with $2C(o)$, where $o = (0,0)$ is the origin. We denote this rectangle by $D(R_i)$. It can be partitioned into nine regions over which $S_i(p)$ is differentiable:

(i) **The central region**, over which $C(p) \subseteq R_i$ or $R_i \subseteq C(p)$. Over this region, $S_i(p) = \min(Area(C(p)), Area(R_i))/\max(Area(C(p)), Area(R_i))$, which is constant.

(ii) **Four side regions**, over which two corners of $C(p)$ are inside R_i, or two corners of R_i are inside $C(p)$. Over each of these regions, $S_i(p)$ is a linear function.

(iii) **Four corner regions**, over which one corner of $C(p)$ is inside R_i and one corner of R_i is inside $C(p)$. Over each of these regions, $S_i(p) = a + bx + cy + dxy$, where $(x, y) = p$ and a, b, c, d are appropriate constants.

A useful property of $S_i(p)$ is that it is piecewise linear over any x-parallel or y-parallel line. That is, $S_i(p)$ is a piecewise linear univariate function when p ranges over (x, c) or over (c, y), for any constant c. In particular, consider sweeping $D(R_i)$ with a y-parallel line ℓ, say from left to right. At the first phase of the sweep, ℓ intersects the two left corner regions and the left side region; during this phase, the graph of $S_i(p)$ over ℓ is a trapezoid whose height grows linearly, while the lengths of its bases remain fixed. At the second sweeping phase, ℓ intersects the central region and the top and bottom side regions; during this phase, $S_i(p)$ over ℓ is fixed. The last sweeping phase is a mirror image of the first, at which the height of the trapezoidal graph of $S_i(p)$ over ℓ diminishes linearly until it becomes zero.

3. Approximation Algorithm

We start with an informal overview. The algorithm proceeds as follows. First, for every i, we construct a collection of weighted rectangles that ε-approximates $S_i(p)$ in the following sense:

Definition 3.1 (a) A function $g(\cdot)$ is said to *ε-approximate* a function $f(\cdot)$, if:

- For all $x \in \mathbb{R}^2$, $g(x) \leq f(x)$.
- For all $x \in \mathbb{R}^2$, such that $f(x) \geq (\varepsilon/50)c$, we have $(1-\varepsilon)f(x) \leq g(x)$, where $c = \max_{p\in\mathbb{R}^2} f(p)$.

(b) For a set $\mathcal{R}$ of weighted rectangles in the plane, let $W_{\mathcal{R}}(\cdot)$ be the corresponding *weight function*, returning for each point in the plane the cumulative weight of the rectangles that contain it. A set $\mathcal{R}$ of weighted rectangles ε-approximates a function $f(\cdot)$ if $W_{\mathcal{R}}(\cdot)$ ε-approximates $f(\cdot)$.

The ε-approximation of $S_i(p)$ is obtained by subdividing $D(R_i)$ into a collection $\mathcal{R}_i$ of rectangles, and setting the weight of a rectangle $r \in \mathcal{R}_i$ to be the minimal value attained by $S_i(p)$ over r. This is similar to the standard procedure of discretizing a function in order to efficiently compute its integral. The precise way in which we generate the subdivision of $D(R_i)$ is described in Lemma 3.2. Informally, we 'slice' the graph of $S_i(p)$ at numerous heights, project the slices onto the xy-plane, obtaining a number of level sets of $S_i(p)$, and, for each pair of (almost) consecutive level sets, approximate the area bounded by these two curves by axis-parallel rectangles, whose weight we define as the height of the lower among the two level sets that bound them.

Consider the overall collection $\mathcal{R} = \cup_i \mathcal{R}_i$ of weighted rectangles obtained by independently approximating each function $S_i(p)$ by $\mathcal{R}_i$, for $1 \leq i \leq n$, as above. Consider the *maximally covered* point p_0 that maximizes the value of $W_{\mathcal{R}}(\cdot)$. Lemma 3.3 shows that $C(p_0)$ is a 3ε-approximation of the optimal camera rectangle. Our problem thus reduces to finding the maximally covered point given a collection of axis-parallel rectangles. Theorem 3.4 shows how this can be done efficiently using a sweep-based algorithm.

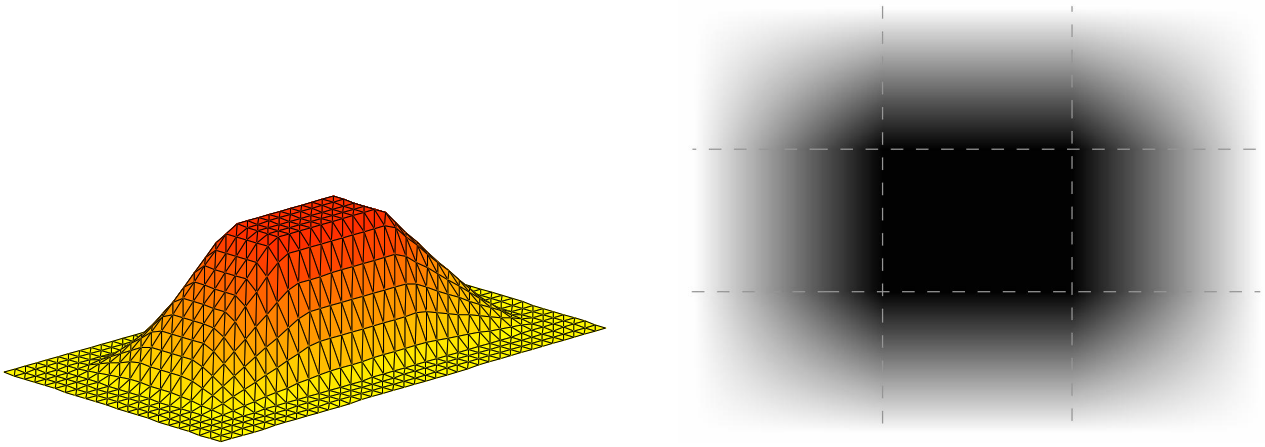


Figure 2: The graph of the satisfaction function $S_i(\cdot)$ is shown on the left. The height map of another such function is shown on the right, partitioned by dashed lines into nine regions over which it is differentiable. Darker colors indicate higher values; black marks the maximal value reached by the function; white marks value zero. Note that perceptual distortions introduced by the human visual system might make the gradation of color from black to white seem non-linear in the side regions, despite its actual linearity.

3.1 Approximating the Individual Satisfaction Functions

Let us now fill the gaps left by the above high-level description. We start with the ε-approximation of $S_i(\cdot)$ by a collection of weighted rectangles.

Lemma 3.2 *For a user i, one can compute a set $\mathcal{R}_i$ of $O\big((\log(1/\varepsilon)/\varepsilon)^2\big)$ rectangles, such that $W_{\mathcal{R}_i}(\cdot)$ is an ε-approximation of $S_i(\cdot)$.*

Proof. See Figure 3 for an illustration of our construction. By an appropriate affine transformation we can assume that both the camera rectangle and the user rectangle R_i are squares, and that the side length of the camera rectangle is 1. Such affine transformation exists because the camera rectangle and R_i have the same aspect ratio. We assume without loss of generality that the bottom left corner of R_i lies at $(1/2, 1/2)$, and its side length is $\alpha > 1$. The latter assumption can be made due to the fact that the definition of $S_i(C) = IOM(C, R_i)$ is symmetric with respect to C and R_i.

We can trivially ε-approximate $S_i(\cdot)$ in the side regions where it is a linear function, using $O(1/\varepsilon)$ weighted rectangles that form a 'staircase'. The central region of $S_i(\cdot)$ can be represented precisely by just one rectangle. We are left to treat the four corner regions. Since the function $S_i(\cdot)$ is symmetric, it suffices to show how to construct an ε-approximation for only one of these regions. We thus show how to approximate $S_i(\cdot)$ on the square $[0,1] \times [0,1]$, which is the bottom left corner region of $S_i(\cdot)$.

We note that the construction described below is insensitive to scaling, and for convenience multiply $S_i(\cdot)$ by α^2. This ensures that $\max_{p \in \mathbb{R}^2} S_i(p) = 1$.

For $j = 1, \ldots, M$, let $\gamma_j = \Big\{ p \in [0,1]^2 \;\Big|\; S_i(p) = \beta_j \Big\}$, where $\beta_j = \min((1+\varepsilon)^j \varepsilon/50, 1)$ and $M = \big\lceil \log_{1+\varepsilon}(50/\varepsilon) \big\rceil = O\Big(\frac{\log(1/\varepsilon)}{\varepsilon}\Big)$. Thus the curve γ_j is the β_j-level set of $S_i(\cdot)$, i.e., the planar projection of the horizontal cross-section ('slice') of the graph of $S_i(\cdot)$ at height β_j. M will be referred to as the *number of slices*.

Notice that our carefully made assumptions imply that $S_i(p) = xy$ for $p = (x, y) \in [0,1]^2$. Thus, $S_i(p) = \beta_j$ implies $y = \beta_j / x$. The curve γ_j is thus simply the image of the univariate function $f_j(x) = \beta_j / x$ inside the unit square.

Let P_j be the minimum link polygonal path that lies between the curves γ_j and γ_{j+1}. We consider the set of all P_j, for $1 \le j \le M-1$, and construct its vertical decomposition [11]. This results in a collection of rectangles, all lying between γ_j and γ_{j+2}, for various $1 \le j \le M-2$. We define the weight of each such rectangle as the height β_j of the lower curve γ_j. These are the weighted rectangles that approximate $S_i(\cdot)$ in the bottom left corner region. Their number is bounded by the complexity of the vertical decomposition, which is $O(\sum_{j=0}^{M} |P_j|)$.

We now show how to compute the chains P_j and analyze the quantity $\sum_{j=0}^{M} |P_j|$. Fix a number j and assume for simplicity that $\beta_{j+1} < 1$. Define the chain as $P_j = p_1 p_1' p_2 p_2' \ldots p_{m_j - 1}' p_{m_j}$. The points p_k and p_k' are computed as follows.

Let $x_{j,1} = \beta_{j+1} = (1+\varepsilon)^{j+1}\varepsilon/50$ and

$$p_1 = (x_{j,1}, f_{j+1}(x_{j,1})) = \Big((1+\varepsilon)^{j+1} \frac{\varepsilon}{50}, 1 \Big) \in \gamma_{j+1}.$$

p_1 is the first vertex of P_j. The odd vertices of P_j lie on γ_{j+1} and the even ones lie on γ_j. We compute an odd vertex p_k of P_j by shooting a horizontal ray to the right from p_{k-1}' until it hits γ_{j+1}. We compute an even vertex p_k' by shooting a vertical ray down from p_k until it hits γ_j.

More formally, let $p_k = (x_{j,k}, f_{j+1}(x_{j,k})) \in \gamma_{j+1}$, and consider the vertical segment emanating from p_k downward until it hits γ_j. Let $p_k' = (x_{j,k}, f_j(x_{j,k})) \in \gamma_j$ denote the endpoint of this segment on γ_j. Next, let $x_{j,k+1}$ be the x coordinate of the intersection point of γ_{j+1} with the horizontal ray emanating from p_k' to the right. By construction, $x_{j,k+1}$ is the solution to the equation $\frac{\beta_j}{x_{j,k}} = \frac{\beta_{j+1}}{x_{j,k+1}}$. Thus, either $x_{j,k+1} = 1$ (i.e., we have reached the right 'opening'

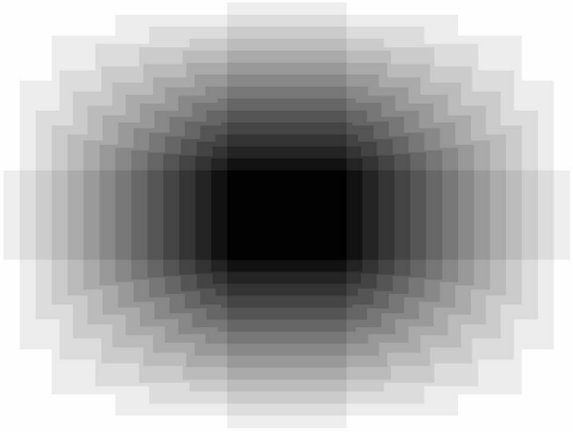


Figure 3: The height map of the weight function $W_{\mathcal{R}_i}(\cdot)$ of the rectangles $\mathcal{R}_i$ that approximate $S_i(\cdot)$. The number of slices M is 15.

of the area between γ_j and γ_{j+1}), or $x_{j,k+1} = (1+\varepsilon)x_{j,k} = (1+\varepsilon)^k x_{j,1} = (1+\varepsilon)^k \beta_{j+1}$. This yields

$$\begin{aligned} p_k &= (x_{j,k}, f_{j+1}(x_{j,k})) = \left((1+\varepsilon)^k \beta_{j+1}, \frac{\beta_{j+1}}{x_{j,k}}\right) \\ &= \left((1+\varepsilon)^{k+j+1}\frac{\varepsilon}{50}, \frac{1}{(1+\varepsilon)^{k-1}}\right), \end{aligned}$$

and

$$p_k^{'} = \left((1+\varepsilon)^{k+j+1}\frac{\varepsilon}{50}, \frac{1}{(1+\varepsilon)^{k}}\right).$$

This completes the specification of the chain P_j. We have

$$|P_j| = O\left(\log_{1+\varepsilon}\frac{1}{\beta_{j+1}}\right) = O(M-j),$$

and $\sum_{j=0}^{M}|P_j| = O(M^2) = O\big((\log(1/\varepsilon)/\varepsilon)^2\big)$. This completes the proof of the lemma. ■

3.2 Approximating the Global Satisfaction Function

We have shown in the previous subsection that an individual satisfaction function $S_i(\cdot)$ can be efficiently ε-approximated by a set $\mathcal{R}_i$ of weighted rectangles. We next prove that the union $\mathcal{R} = \cup_i \mathcal{R}_i$, for $1 \le i \le n$, is a good approximation for the global satisfaction function $\mathcal{S}_{\mathbf{R}}(\cdot)$. Specifically, the following lemma implies that the weight function $W_{\mathcal{R}}(\cdot)$ 3ε-approximates $\mathcal{S}_{\mathbf{R}}(\cdot)$.

Lemma 3.3 *Let $g_1, \ldots, g_n$ be n functions that respectively ε-approximate the functions $S_1, \ldots, S_n$. Let $\mathcal{U} = \max_{p\in\mathbb{R}^2}\mathcal{S}_{\mathbf{R}}(p)$ and let $U = \max_{p\in\mathbb{R}^2}\sum_{i=1}^{n} g_i(p)$. Then $\mathcal{U} \ge U \ge (1-3\varepsilon)\mathcal{U}$.*

Proof. Let $p_{opt} = p_{opt}(\mathbf{R})$ be the point realizing $\mathcal{U}$. Clearly, if for $i = 1, \ldots, n$, we have $g_i(p_{opt}) \ge (1-\varepsilon)S_i(p_{opt})$ then the claim trivially holds, as $U \ge \sum_{i=1}^{n} g_i(p_{opt}) \ge (1-\varepsilon)\mathcal{U}$.

This fails if for some i, $g_i(p_{opt}) < (1-\varepsilon)S_i(p_{opt})$, which by Definition 3.1 holds only if $0 < S_i(p_{opt}) < (\varepsilon/50)c_i$, where $c_i = \max_{p\in\mathbb{R}^2} S_i(p)$. Let

$$I = \left\{ i \;\middle|\; S_i(p_{opt}) < (\varepsilon/50)c_i, i = 1, \ldots, n \right\}$$

and define

$$\mathcal{U}^- = \sum_{i\in I} S_i(p_{opt}) \quad \text{and} \quad \mathcal{U}^+ = \sum_{i\notin I, i=1,\ldots,n} S_i(p_{opt}).$$

Note that $\mathcal{U} = \mathcal{U}^- + \mathcal{U}^+$. If $\mathcal{U}^- < \varepsilon\mathcal{U}$, we are done, as $U \ge (1-\varepsilon)\mathcal{U}^+ \ge (1-\varepsilon)^2\mathcal{U} \ge (1-3\varepsilon)\mathcal{U}$. Thus assume that $\mathcal{U}^- > \varepsilon\mathcal{U}$. Consider a function $S_i(\cdot)$ that contributes (positively) to $\mathcal{U}^-$ and observe that $C(p_{opt}) \cap R_i \neq \emptyset$. Thus, $S_i(p_{opt}) < \varepsilon \max_{p\in\mathbb{R}^2} S_i(p)$. Also note that either at least one of the corners of $C(p_{opt})$ is inside R_i, or, alternatively, two of the corners of R_i are inside $C(p_{opt})$.

Consider the former case first. Let q_1, q_2, q_3, q_4 be the four corners of $C(p_{opt})$, and let q_1 be a corner that lies inside R_i. Clearly, $S_i(q_1) \ge c_i/4$; this is true whenever the center of the camera is placed inside the rectangle R_i.

In the latter case there are two corners of R_i inside $C(p_{opt})$, and R_i is smaller than $C(p_{opt})$. It can be easily verified that this implies (again) that

$$\max\left(S_i(q_1), S_i(q_2), S_i(q_3), S_i(q_4)\right) \ge \frac{c_i}{4}$$

Let $\varphi(p) = \sum_{i\in I} S_i(p)$. We have

$$\begin{aligned} \varphi(q_1)+\varphi(q_2)+\varphi(q_3)+\varphi(q_4) &\ge \sum_{i\in I}\frac{c_i}{4} \\ \ge \sum_{i\in I}\frac{50}{4\varepsilon}S_i(p_{opt}) > \frac{12}{\varepsilon}\mathcal{U}^- &> \frac{12}{\varepsilon}\varepsilon\mathcal{U} = 12\mathcal{U}, \end{aligned}$$

which implies that

$$\max_c \varphi(q_c) > 3\mathcal{U} > \max_c \mathcal{S}(q_c) \ge \max_c \varphi(q_c),$$

which is a contradiction. ■

3.3 Computing the Approximate Camera Rectangle

The previous subsections show that we can reduce the problem of finding an ε-approximate camera rectangle to the problem of finding, given a collection of weighted axis-parallel rectangles, the *maximally covered* point in the plane that maximizes the cumulative weight of the rectangles that contain it. Indeed, put $\mathcal{E} = (\log(1/\varepsilon)/\varepsilon)^2$. Lemma 3.2 shows that we can $(\varepsilon/3)$-approximate $S_i(\cdot)$ with a collection $\mathcal{R}_i$ of $O(\mathcal{E})$ weighted rectangles. This collection can trivially be computed in time $O(\mathcal{E}\log\mathcal{E})$. (With a little more insight, it can actually be computed in time $O(\mathcal{E})$, but the crude bound $O(\mathcal{E}\log\mathcal{E})$ will suffice.) Consider the collection $\mathcal{R} = \cup_{i=1,\ldots,n}\mathcal{R}_i$. Lemma 3.3 implies that $W_{\mathcal{R}}(\cdot)$ ε-approximates $\mathcal{S}(\cdot)$. To compute an ε-approximate camera rectangle it thus suffices to find the point $p_0 = \max_{p\in\mathbb{R}^2} W_{\mathcal{R}}(p)$, which is the maximally covered point in the above sense. In the proof of the following theorem we show how this can be done in time $O(N\log N)$, where $N = |\mathcal{R}| = O(n\mathcal{E})$.

Theorem 3.4 *Given a set $\mathbf{R} = \{R_1, \ldots, R_n\}$ of user rectangles and a parameter $\varepsilon > 0$, we can compute a point $p_0 \in \mathbb{R}^2$, such that $\mathcal{S}_{\mathbf{R}}(p_0) \ge (1-\varepsilon)\max_{p\in\mathbb{R}^2}\mathcal{S}_{\mathbf{R}}(p)$, in time $O(N\log N)$, where $N = O\left(n\frac{\log^2(1/\varepsilon)}{\varepsilon^2}\right)$.*

Proof. In light of the above, we concentrate on computing the point $p_0 = \max_{p\in\mathbb{R}^2} W_{\mathcal{R}}(p)$. This is done with a sweep-based algorithm. Our approach is very similar to the one used by de Berg et al. [9] to find, given two collections of

respectively red and blue squares, a point in the plane that maximizes the difference between the number of red and the number of blue squares that contain the point.

We sweep the collection $\mathcal{R}$ with a vertical line ℓ, from left to right. At any given time, the intersection $\mathcal{R} \cap \ell$ is a collection of weighted intervals on ℓ. This collection changes only when ℓ reaches the left (resp., the right) side of some rectangle $r \in \mathcal{R}$. At this point, a weighted interval on ℓ appears (resp., disappears). Throughout the sweep we maintain the maximally covered point on ℓ. That is, given the collection of weighted intervals on ℓ, we maintain the point on ℓ that maximizes the cumulative weight of the intervals that contain it. The heaviest point among those maintained in this fashion during the sweep is the point p_0 we are looking for.

It remains to give the details of the data structure we maintain on ℓ. Recall that at any given time we have a set of weighted intervals $\mathcal{I}$ and we want to keep track of the maximally covered point. We use a modified version of the segment tree [11]. It is a balanced binary tree on the elementary intervals into which ℓ is partitioned by $\mathcal{I}$. Every leaf u corresponds to an elementary interval $i(u)$; every inner node v corresponds to the interval $i(v)$ that is the union of the intervals corresponding to its children: $i(v) = i(\text{left}(v)) \cup i(\text{right}(v))$. In addition, every node stores pointers to a set $S(v) \subseteq \mathcal{I}$ of weighted intervals from $\mathcal{I}$. The intervals $S(v)$ are those intervals of $\mathcal{I}$ that contain $i(v)$ but do not contain $i(\text{parent}(v))$.

In addition to the above standard components of the segment tree structure [11], we associate with every node v the weight $w(v)$, defined to be the cumulative weight of the intervals $S(v)$, and the ordered pair $(p(v), w(p(v)))$ that specifies the maximally covered point $p(v)$ among all points in $i(v)$, and the weight $w(p(v))$ of this point. These parameters, as the rest of the segment tree, are initialized bottom-up. For a leaf u, the point $p(u)$ is set to be an arbitrary point in $i(u)$, and its weight $w(p(u))$ is set to $w(u)$. For an interior node v, if $w(p(\text{left}(v))) > w(p(\text{right}(v)))$, we set $p(v) = p(\text{left}(v))$ and $w(p(v)) = w(p(\text{left}(v))) + w(v)$; otherwise, $p(v) = p(\text{right}(v))$ and $w(p(v)) = w(p(\text{right}(v))) + w(v)$.

It is clear that the maximally covered point on ℓ is the point $p(r)$ associated with the root r of the tree. Standard analysis implies that the tree can be constructed in $O(m \log m)$ time, where $m = |\mathcal{I}|$ [11]. Recall, however, that we need to maintain the data structure dynamically through the sweep. We do this as follows. Project all the horizontal edges of all the rectangles of $\mathcal{R}$ horizontally onto the y-axis. This results in a subdivision of this axis into $O(N)$ intervals. We construct a balanced binary tree on the elementary intervals of this subdivision. This tree serves as a backbone for our segment tree, and ensures that the segment tree remains roughly balanced through the sweep. Inserting or removing a weighted interval now amounts to updating the information associated with certain nodes of this fixed tree. The properties of the segment tree imply that the number of nodes affected by any single insertion or deletion is $O(\log N)$, and that they can be traversed in $O(\log N)$ time [11].

We start the sweep from $-\infty$ with the weight $w(v)$ of every node v in the tree being 0; the number of weighted intervals intersecting ℓ in the beginning is 0. We then sweep the collection $\mathcal{R}$, inserting and removing intervals into the data structure as we go along, until we get to $+\infty$, where all the weights are 0 again. We make $O(N)$ insertions and deletions, and standard analysis shows that each such operation can be carried out in time $O(\log N)$ [11]. The overall running time of the sweeping algorithm is thus $O(N \log N)$, which subsumes the time $O(N \log \mathcal{E})$ needed to construct the approximation rectangles $\mathcal{R}$. This completes the proof of the theorem. ■

Remark 3.5 After the submission of this extended abstract, Chan [8] found a way of representing each individual satisfaction function by only $O(1/\varepsilon^2)$ weighted rectangles, such that the algorithm of Theorem 3.4 still produces an ε-approximate solution using these rectangles. This improves the running time of the algorithm to $O(N \log N)$ for $N = O(n/\varepsilon^2)$. The main idea behind the alternative construction of rectangles is that the cumulative weight function they define does not have to ε-approximate $\mathcal{S}(\cdot)$. It is sufficient for the purpose of the algorithm that the value of this weight function be ε-close to the value of $\mathcal{S}(\cdot)$ when $\mathcal{S}(\cdot)$ achieves its maximum.

4. Exact Algorithm

We will extend the sweeping algorithm of Theorem 3.4 to compute the exact maximum of $\mathcal{S}(\cdot)$. Unfortunately this extension is not trivial, since we are now not sweeping a collection of weighted rectangles, but rather a collection of n individual satisfaction functions $S_i(\cdot)$. The cross-section of each such function with the vertical sweep-line ℓ changes continuously for part of the sweeping process, and this cross-section is no longer a weighted interval on ℓ, but rather a trapezoid-like function. We need to maintain the maximum of the sum of n such continuously changing functions throughout the sweep. Fortunately, the favorable properties of the functions $S_i(\cdot)$, described in Section 2, will come to our rescue.

For a specific $1 \leq i \leq n$, let L_i be the set of eight lines (four horizontal and four vertical) that partition the plane into rectangular cells, such that inside each cell, the function $S_i(\cdot)$ is differentiable. See Figure 2 and note that $S_i(\cdot) = 0$ inside the 16 unbounded cells of the arrangement of L_i. Let the collection of the 9 bounded cells (rectangles) of this arrangement be E_i, and define $E = E_{\mathbf{R}} = \cup_{i=1,\ldots,n} E_i$. Also let $\mathcal{G} = \mathcal{G}_{\mathbf{R}}$ denote the (grid) arrangement of L, where $L = L_{\mathbf{R}} = \cup_{i=1,\ldots,n} L_i$. The following lemma is equivalent to [20, Theorem 1] and is given here for completeness.

Lemma 4.1 *The maximum of $\mathcal{S}(\cdot)$ is achieved at a vertex of $\mathcal{G}$.*

Proof. Assume, for the sake of contradiction, that the maximum of $\mathcal{S}(\cdot)$ is not achieved at a vertex of the grid $\mathcal{G}$, but rather at a point p that lies in the interior of a cell or an edge of $\mathcal{G}$. Assume, without loss of generality, that the edge in the latter case is horizontal. Let ℓ be the horizontal line spanned by p. Let $f(t) = \mathcal{S}(\ell(t))$ be the restriction of $\mathcal{S}$ to ℓ. The properties outlined in Section 2 imply that the univariate function $f(t)$ is piecewise linear, and is linear inside each elementary interval on ℓ (these are the intervals bounded by consecutive intersections of ℓ with the vertical lines of L). Since, by assumption, p lies inside such an elementary interval, the value of $f(\cdot)$ on one of the endpoints of this interval is at least as large as $f(p)$. If p lies on an edge of $\mathcal{G}$, this brings an immediate contradiction. Otherwise, if

p lies in the interior of a cell of $\mathcal{G}$, consider the endpoint p' of the elementary interval on ℓ that contains p, for which $f(p') \geq f(p)$. (Since p is assumed to be the maximum, we actually have that $f(p') = f(p)$.) p' lies on some vertical line $\ell' \in L$. We can repeat the same reasoning as above with respect to p' and ℓ', showing that $\mathcal{S}(p'') \geq \mathcal{S}(p')$, for one of the endpoints p'' of the elementary interval on ℓ' that contains p'. This is a contradiction that proves the lemma. ■

This lemma shows that it is sufficient to search for the maximum of $\mathcal{S}(\cdot)$ on the horizontal lines $\ell_1, \ldots, \ell_m$ of L. Let $y_1, \ldots, y_m$ be the y-coordinates for which $\ell_i \equiv (y = y_i)$, respectively. Define also $f_i(t) = \mathcal{S}(\ell_i(t))$ to be the restriction of $\mathcal{S}$ to ℓ_i. As follows from Section 2, $f_i(t)$ is piecewise linear, and is linear in between consecutive vertical lines of L. We seek to locate $\max_t \max_{i=1,\ldots,n} f_i(t)$. This will be achieved with the help of the data structure described in the following lemma.

Lemma 4.2 *Let $P(t) = \{(a_1, g_1(t)), \ldots, (a_u, g_u(t))\}$ be a set of u points that are moving along vertical trajectories in the plane, at constant speeds. That is, $a_1, \ldots, a_u$ are the fixed x-coordinates of the points and $g_1, \ldots, g_u$ are linear functions. Then*

(a) The upper part $\mathcal{CH}(P(t))$ of the convex hull of $P(t)$ undergoes $O(u)$ combinatorial changes throughout time.

(b) $\mathcal{CH}(P(t))$ can be maintained in a kinetic data structure of size $O(u)$ that allows estimating its extreme point in any direction in time $O(\log u)$. The data structure can be initialized in time $O(u \log u)$ and undergoes $O(u \log u)$ structural changes (events) throughout time. Each change can be processed in time $O(\log^2(u))$. The overall time needed to maintain the data structure is thus $O(u \log^3 u)$.

Proof. Part (a) is a standard exercise: Since all points move at fixed speeds, no point can leave $\mathcal{CH}(P(t))$ and then appear on it again. Any point thus appears and disappears at most once. Since any combinatorial change of $\mathcal{CH}(P(t))$ is caused by some point appearing or disappearing, this proves part (a).

To construct the data structure of part (b) we use a divide and conquer approach, and modify the data structure of Overmars and van Leeuwen [18] to our needs. Here is a sketch of the basic idea: Split the points into a left half and a right half, and process the halves recursively. This yields a binary tree. For every node in the tree, the recursive processing yields the two separate hulls associated with the children of this node. We connect them by a bridge that we associate with the node. Part (a) implies that this bridge undergoes $O(u_i)$ changes throughout time, where u_i is the number of points handled by the subtree rooted at this node. The data structure thus has $O(u \log u)$ structural changes at all of its levels. Its size is linear in u since every node maintains only a constant amount of information that specifies the bridge between the hulls of its two children. The above recursive initialization process takes $O(u \log u)$ time. We omit the technical details, which can be found in Overmars and van Leeuwen [18]. In particular, each structural change can be handled in time $O(\log^2 u)$.

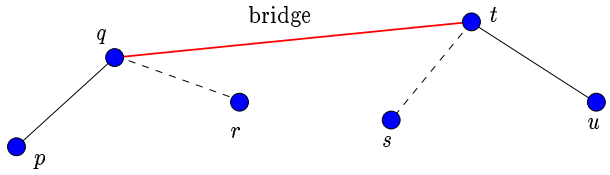


Figure 4: For a bridge qt stored at a node v, we have to maintain a certificate that testifies that qt is a valid bridge. Here p, q, r (resp., s, t, u) are the three consecutive vertices of the convex hull of the points stored in the left (resp., the right) subtree of v. These certificates testify that the triplets (u, s, t), (s, t, q), (q, r, t), (p, q, r) of moving points are not collinear. For each such certificate, we compute the time when it is violated; namely, when the three relevant points become collinear. These are the times when the convex hull of the moving points changes and we need to update the data structure.

To correctly handle the motion of the points we need to transform the above into a kinetic data structure (see Basch et al. [6] and Guibas [15]). Any node in our tree contains a bridge (between the convex hulls of its left and right children), which can be certified, in the sense of [6], using a constant number of certificates, see Figure 4. Namely, by inspecting the four adjacent edges to the bridge in the children convex chains, and the (moving) points that define them, we can in constant time compute the first time where this bridge is no longer legal. Putting these time events into a queue, we can detect the next time when the convex hull undergoes a combinatorial change, and then handle these internal events by performing the appropriate insertions and deletions in the data structure. See [6] for further details.

As for estimating the extreme points of $\mathcal{CH}(P(t))$, observe that at any time t, our data structure is (essentially) the data structure of Overmars and van Leeuwen. As such, it can answer the queries as claimed. ■

This lemma suggests the following approach to our problem of maintaining the maximum of $\mathcal{S}(\cdot)$ over the sweep-line ℓ: Maintain the convex hull of the moving points $(y_i, f_i(t))$ as described in the lemma, and keep track of its highest point through time. The problem is that the functions $f_i(t)$ are not linear but piecewise-linear, so we might need to appropriately update the data structure whenever the sweep-line ℓ reaches a vertical line of L. At each of the $O(n)$ such vertical lines, a linear number of points $(y_i, f_i(t))$ might change their velocity (this happens when the function $f_i(t)$ changes its slope), so overall the number of updates we have to perform might be as large as quadratic.

We overcome this difficulty by batching successive lines ℓ_i into batches of cardinality $k = \lceil \sqrt{n} \rceil$. Namely, we define $\mathcal{L}_i = \{\ell_{k\cdot(i-1)+1}, \ldots, \ell_{k\cdot i}\}$, for $i = 1, \ldots, r$, where $r = \lceil n/k \rceil$. We will also denote by $\mathcal{P}_i$ the batch of points

$$\{(y_{k\cdot(i-1)+1}, f_{k\cdot(i-1)+1}(t)), \ldots, (y_{k\cdot i}, f_{k\cdot i}(t))\}$$

that corresponds to $\mathcal{L}_i$. Let us now look closer at the behavior of these points.

The velocity of a point belonging to $\mathcal{P}_i$ changes whenever the sweep-line ℓ reaches a vertical edge of one of the rectangles in E, such that this edge contains the mentioned point. However, if this rectangle edge contains all the points in the

Figure 5: The robotic camera used by our system.

batch $\mathcal{P}_i$, we can handle the velocity changes of all these points together. Specifically, we can associate a function $\mathcal{B}_i(y,t)$ with $\mathcal{L}_i$, such that for a point $(y_j, f_j(t))$, $f_j(t) = \mathcal{B}_i(y_j,t) + g_j(t)$, where i is the number of the batch that contains this point. For all i, the points

$$\left\{(y_{k\cdot(i-1)+1}, g_{k\cdot(i-1)+1}(t)), \ldots, (y_{k\cdot i}, g_{k\cdot i}(t))\right\}$$

are stored in the data structure of Lemma 4.2. Thus, handling a rectangle edge that contains all of $\mathcal{P}_i$ amounts to updating $\mathcal{B}_i(y,t)$ (in constant time), and the invariant $f_j(t) = \mathcal{B}_i(y_j,t) + g_j(t)$, for all points $(y_j, f_j(t))$ in $\mathcal{P}_i$ still holds. We call an event as above *global* with respect to the batch i. Otherwise, if the encountered rectangle edge contains only some of the points in $\mathcal{P}_i$, the event is said to be *local* with respect to this batch. Every event is local with respect to at most two batches, and global with respect to at most $O(\sqrt{n})$ (which is their total number). Whenever a batch i encounters a local event, we explicitly update the velocities of all the points in the batch, and simply rebuild the data structure associated with it.

Recall that our goal is to locate the maximum of $\mathcal{S}(\cdot)$. Lemma 4.1 implies that it is sufficient to compute, for all $i = 1, \ldots, r$, the highest point on the upper hull $\mathcal{CH}(\mathcal{P}_i)$ of $\mathcal{P}_i$, and to do so only whenever this batch is being updated because of a global or local event.

Let us explain how we handle this computation when $\mathcal{P}_i$ encounters a global event. Since the convex hull $\mathcal{CH}(\mathcal{P}_i)$ is maintained in the data structure described in Lemma 4.2 we can locate its highest point in time $O(\log n)$. The points stored in the data structure do not take into account the global function $\mathcal{B}_i(y,t)$. Since $\mathcal{B}_i(y,t)$ is linear in y for a fixed t, adding this function to all the points, as described above, simply 'skews' the convex hull. Finding the highest point on the 'skewed' hull $\mathcal{B}_i(y,t)+\mathcal{CH}(\mathcal{P}_i(t))$ is equivalent to finding the extreme point of $\mathcal{CH}(\mathcal{P}_i(t))$ in a specific direction, which can be inferred from $\mathcal{B}_i(y,t)$ in constant time. As follows from Lemma 4.2, this can be done in time $O(\log n)$.

Overall, a global event can be easily be handled in time $O(\log n)$ per affected batch. We first update in constant time the associated function $\mathcal{B}_i(y,t)$, and then find in time $O(\log n)$ the highest point on the skewed convex hull. This point is a candidate for being the highest point of $\mathcal{S}(\cdot)$, and as such, it is compared with the highest point encountered so far. Since there are $O(\sqrt{n})$ batches and $O(n)$ events (vertical edges of rectangles from E) throughout the sweep, the processing of all global events takes time $O(n^{3/2}\log n)$.

When a batch encounters a local event, we explicitly rebuild the associated data structure. We thus need to deposit at this point $O(\sqrt{n}\log^3 n)$ units of time, to cover for the time we will need to maintain this data structure, as explained in Lemma 4.2. Also, whenever we rebuild the convex hull as above, we compute its highest point, which is a candidate for being the highest point of $\mathcal{S}(\cdot)$ as above. Overall, processing a local event takes time $O(\sqrt{n}\log^3 n)$ per batch. Since there are $O(n)$ events, and each of them is local for only two batches, this amounts to $O(n^{3/2}\log^3 n)$ time spent on local events overall. Combined with the above analysis of global events, this yields the following main result of this section.

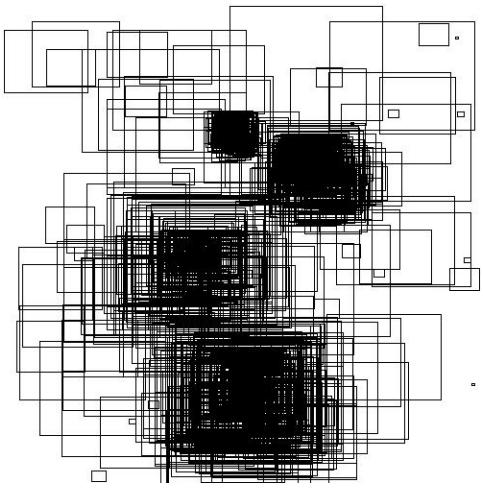


Figure 6: 400 randomly generated rectangles.

Theorem 4.3 *Given a set $\mathbf{R} = \{R_1, \ldots, R_n\}$ of user rectangles, we can compute a point $p_0 \in \mathbb{R}^2$, such that $\mathcal{S}_{\mathbf{R}}(p_0) = \max_{p\in\mathbb{R}^2} \mathcal{S}_{\mathbf{R}}(p)$, in time $O(n^{3/2}\log^3 n)$.*

Remark 4.4 After the submission of this extended abstract, Chan [8] found an alternative exact algorithm for computing the maximum of $\mathcal{S}(\cdot)$. It is based on a technique described by Overmars and Yap [19] and achieves running time $O(n^{3/2}\log n)$.

5. Implementation and Experiments

We have implemented the approximation algorithm of Section 3 and integrated it with the rest of the ShareCam infrastructure, which includes a functional networking module and is currently on-line [1]. We have carried out experiments on a Dell PowerEdge Server 2650 with a Dual Intel Xeon 2.4Ghz CPU and 2GB of RAM. (Only one CPU was utilized.) We use Visual C++ and Windows. Our pan/tilt/zoom robotic camera is the Canon VC-C4, shown in Figure 5. Figures 1 and 3 were produced using our implementation. Figure 1(a) shows 18 user rectangles and the camera rectangle computed by our algorithm. Figure 1(b) shows the optimal camera rectangle for this input that was computed by the algorithm of [20].

We have compared the performance of our algorithm to the algorithm of Song et al. [20]. The comparison used randomly generated collections of user rectangles. We have not generated the rectangles uniformly at random since the rectangles then tend to cover the plane in a homogeneous fashion, such that the global satisfaction function values are

number of users	Song et al. [20]	This paper
200	15	125 (0.88)
400	32	204 (0.86)
600	125	562 (0.87)
800	157	781 (0.88)
1000	281	1578 (0.86)

Table 1: The table shows the running times (in milliseconds) of the algorithms of [20] and of Section 3. The ratio of the global satisfaction value generated by our algorithm to the optimal value found by the algorithm of [20] is shown in parenthesis.

almost the same everywhere. This is not a realistic model of user input. As an alternative, we first select four seed points uniformly at random. To generate a rectangle, we select one of the seed points at random or, with small probability, select none. If one of the seed points is selected we compute the rectangle using a normal distribution centered at the selected seed. Otherwise the rectangle is chosen from the uniform distribution over the plane. Figure 6 demonstrates 400 rectangles generated in this fashion.

A sample from our experimental results is presented in Table 1. In this table, the number of slices made by our algorithm to approximate the individual satisfaction functions is $M = 3$ (see Section 3). As the accuracy values in Table 1 indicate, we have observed that in practice the dependence of the approximation quality of our algorithm on M is considerably better than the conservative bound guaranteed by Theorem 3.4.

On the other hand, the algorithm was consistently slower than its counterpart, despite having superior asymptotical running time. This asymptotical superiority does ensure that our algorithm outperforms [20] beyond a certain number of users. This number is, however, in the hundreds of thousands, when both algorithms are no longer practical.

The bottleneck for our algorithm seems to be the large number of segment-tree insertions and deletions performed during the sweep. For $M = 3$ slices, each individual satisfaction function is represented by about 50 rectangles. For $1,000$ users this translates to about $100,000$ insertions and deletions. We are currently looking into various ways of achieving speed-up in practice, which include optimizing our segment tree implementation.

Acknowledgements

The authors would like to acknowledge the helpful suggestions of Timothy Chan, Shankar Krishnan, Mark Pauly, Frank van der Stappen, and Suresh Venkatasubramanian.

6. REFERENCES

[1] http://tele-actor.net/sharecam

[2] http://www.x-zone.canon.co.jp/WebView-E/all/list.htm

[3] http://www.globalcam.net/demos.html

[4] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30:412–458, 1998.

[5] A.Z.Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, 1998.

[6] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[7] D. J. Cannon. *Point-And-Direct Telerobotics: Object Level Strategic Supervisory Control in Unstructured Interactive Human-Machine System Environments*. PhD thesis, Stanford Mechanical Engineering, June 1992.

[8] T. Chan. Personal communication.

[9] M. de Berg, P. Bose, O. Cheong, and P. Morin. On simplifying dot maps. In *Proc. 18th European Workshop on Computational Geometry*, pages 96–100. 2002.

[10] M. de Berg, O. Cheong, O. Devillers, M. van Kreveld, and M. Teillaud. Computing the maximum overlap of two convex polygons under translations. *Theory of Computing Systems*, 31:613–628, 1998.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[12] K. Goldberg and B. Chen. Collaborative control of robot motion: Robustness to error. In *International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[13] K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, and G. Smith. Collaborative teleoperation via the internet. In *IEEE International Conference on Robotics and Automation (ICRA)*, April 2000.

[14] K. Goldberg, D. Song, Y. Khor, D. Pescovitz, A. Levandowski, J. Himmelstein, J. Shih, A. Ho, E. Paulos, and J. Donath. Collaborative online teleoperation with spatial dynamic voting and a human "tele-actor". In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2002.

[15] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Workshop on Algorithmic Foundations of Robotics*, pages 191–209. 1998.

[16] T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. In *Proceedings of WWW*, 2002.

[17] M. McDonald, D. Small, C. Graves, and D. Cannon. Virtual collaborative control to improve intelligent robotic system efficiency and quality. In *IEEE International Conference on Robotics and Automation*, April 1997.

[18] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[19] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20:1034–1045, 1991.

[20] D. Song, A. F. van der Stappen, and K. Goldberg. Exact and distributed algorithms for collaborative camera control. In *Workshop on Algorithmic Foundations of Robotics*, 2002.

[21] R. C. Veltkamp and M. Hagedoorn. Shape similarity measures, properties, and constructions. In *Proceedings VISUAL 2000*, volume 1929, pages 467–476. Springer, 2000.