Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces

Michael S. Bernstein[1], *Joel Brandt*[2], *Robert C. Miller*[1], *and David R. Karger*[1]

[1] MIT CSAIL
Cambridge, MA 02139
{msbernst, rcm, karger}@csail.mit.edu

[2] Adobe Systems, Advanced Technology Labs
San Francisco, CA 94103
joel.brandt@adobe.com

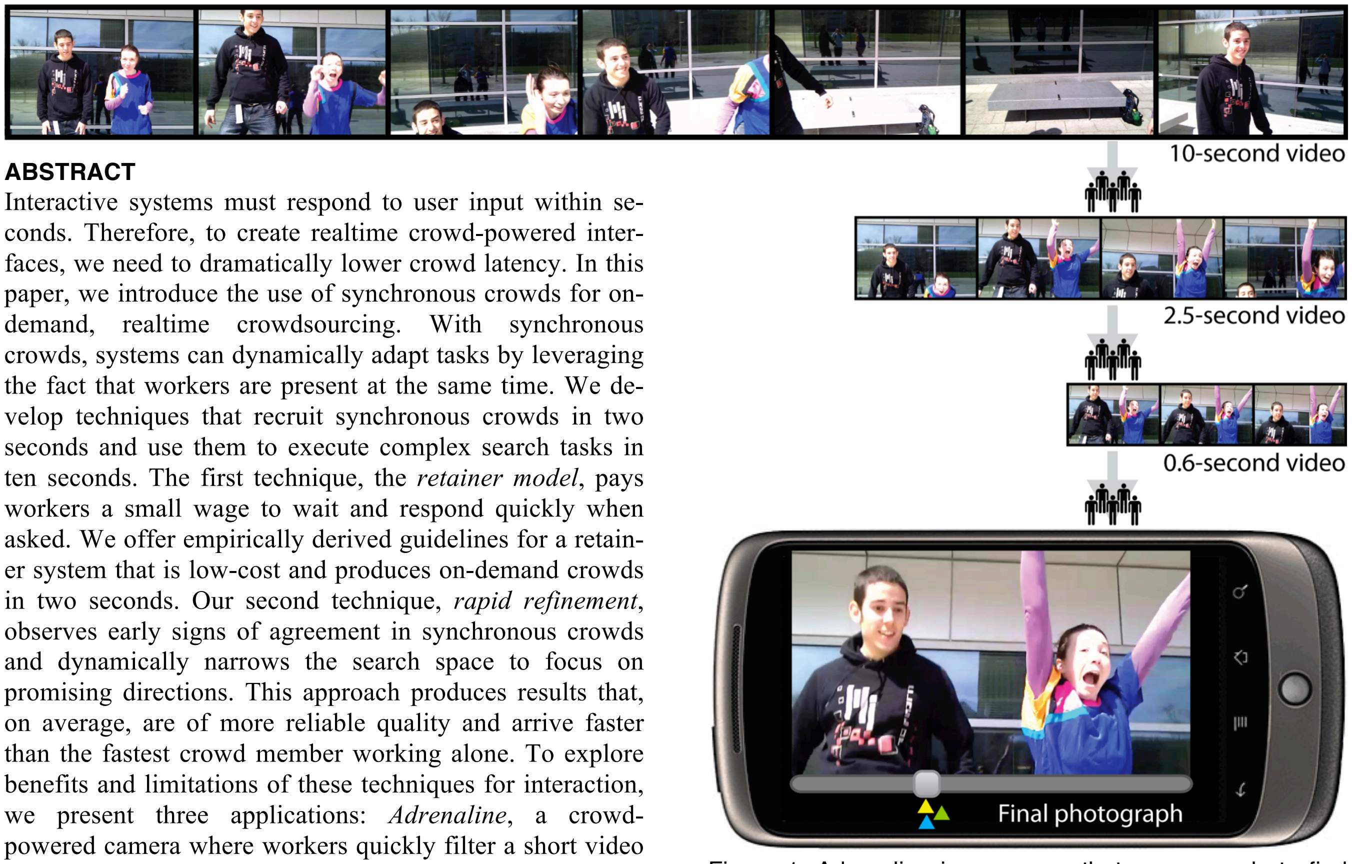

Figure 1. Adrenaline is a camera that uses crowds to find the right moment for a picture by capturing ten-second movies. It looks for early agreement to filter the timeline down quickly to a single frame. A photo is typically ready about one second after the user reviews the movie.

ABSTRACT
Interactive systems must respond to user input within seconds. Therefore, to create realtime crowd-powered interfaces, we need to dramatically lower crowd latency. In this paper, we introduce the use of synchronous crowds for on-demand, realtime crowdsourcing. With synchronous crowds, systems can dynamically adapt tasks by leveraging the fact that workers are present at the same time. We develop techniques that recruit synchronous crowds in two seconds and use them to execute complex search tasks in ten seconds. The first technique, the *retainer model*, pays workers a small wage to wait and respond quickly when asked. We offer empirically derived guidelines for a retainer system that is low-cost and produces on-demand crowds in two seconds. Our second technique, *rapid refinement*, observes early signs of agreement in synchronous crowds and dynamically narrows the search space to focus on promising directions. This approach produces results that, on average, are of more reliable quality and arrive faster than the fastest crowd member working alone. To explore benefits and limitations of these techniques for interaction, we present three applications: *Adrenaline*, a crowd-powered camera where workers quickly filter a short video down to the best single moment for a photo; and *Puppeteer* and *A|B*, which examine creative generation tasks, communication with workers, and low-latency voting.

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

General terms: Design, Human Factors

Keywords: Crowdsourcing, Human computation

INTRODUCTION
Crowd-powered interfaces have demonstrated the potential to revolutionize interactive applications [2,3], but they are limited by the problem of crowd latency. Paid crowdsourcing markets like Amazon Mechanical Turk allow us to embed human intelligence inside applications like word processors [2], question-answering services [3] and image search [28], but today crowdsourcing is only a reasonable choice if the user can wait a minute or more for a response. Existing "nearly realtime" techniques on average produce a single, unverified answer to a question in 56 seconds [3]. More complex workflows require 22 minutes or longer [2].

Users are not used to waiting, and will abandon interfaces that are slow to react. Search engine usage decreases linearly as delays grow [21], and Jakob Nielsen argues that ten seconds is the maximum delay before a user loses focus on the interaction dialogue [20]. The much longer delays with crowdsourcing make it difficult for crowds to help with tasks in the moment-to-moment workflow. For example, using crowds to create a smarter copy-paste command would be difficult: one minute waits between copying and pasting drive users away. We need new approaches if we want to realize the vision of the user pushing a button and seeing a crowd-powered result just seconds later.

In this paper, we introduce recruitment strategies, algorithms, and applications for *realtime crowdsourcing*. Our goal is to complete non-trivial crowdsourced computation within seconds of the user's request — fast enough to feed back into an interface before the user loses focus. Realtime crowdsourcing can open up a broad new class of applications for crowd-powered interfaces.

Our core contribution is to introduce *on-demand synchronous crowds* as a key to realtime crowdsourcing. In synchronous crowds, all crowd members arrive and work simultaneously. While prior work has focused on recruiting individual crowd members quickly (e.g., [3]), we believe that *minimizing recruitment time is not enough*. Even fast workers may not produce a first result for tens of seconds [3], and complex workflows require several minutes [23,28]. Worse, fast workers may not be available, and unpredictable wait times hurt user experience. With synchronous crowds, we can reliably execute complex search tasks within ten seconds of request. However, to do so, we need to recruit synchronous crowds on-demand and guide them to complete tasks effectively.

To recruit on-demand synchronous crowds, we present the *retainer model* and a set of empirically derived guidelines for its use. It hires crowd members in advance, then places them on hold for low cost and alerts them when a task is ready. Our most effective design results in a majority of workers returning within two seconds and 75% within three seconds. The retainer model's performance is striking in that it approaches human limits on the cognitive recognize-act cycle and motor reaction times [4]. It nearly zeroes out wait times, which in previous work ranged from twenty seconds [3] to twenty minutes [2]. Most importantly, however, it makes on-demand synchronous crowds available as a new resource for crowdsourcing.

Developers need ways to guide or program synchronous crowds for realtime results. We introduce *rapid refinement,* the first design pattern for synchronous crowds, which focuses on low-latency, reliable work. The fundamental insight behind rapid refinement is that *synchronicity enables an algorithm to recognize crowd agreement early.* The rapid refinement design pattern quickly reduces a large search space by focusing workers' attention to areas they are beginning to agree on independently (Figures 1 and 4). Repeatedly narrowing the search space to an agreement region encourages quality results because it is built around independent agreement. It also allows the interface to provide incremental, trustable feedback before a final answer is available. Critically, rapid refinement leads to fast results: faster than approaches that keep workers separate, and faster on average than even the fastest individual worker.

We use the retainer model and rapid refinement to explore new avenues for realtime crowd-powered interfaces through a system called *Adrenaline*. Adrenaline is a smart camera shutter powered by crowd intelligence: it finds the right moment to take a photo. Instead of taking a single shot, Adrenaline captures a short video — allowing the user to move around the scene, the subject to strike multiple poses, or action in the scene to unfold unpredictably — then uses rapid refinement to identify the best moment as a still photo about ten seconds later. Low latency means that users can preview and share photos they just took, like they would with any other digital camera.

To investigate the larger design space of realtime crowd-powered interfaces, we also present two other systems, *Puppeteer* and *A|B*. Puppeteer focuses on large-scale creative generation tasks, allowing a graphic artist or designer to fill a page with a large collection of crowd-posed figures (for example, a stadium of excited concert attendees or a full dance floor). The designer interacts directly with the crowd as they work, and sees the first results 2.1 seconds after request. A|B focuses on extremely low latency responses for A-or-B questions, allowing users to ask which clothes to wear, which font looks better, or which version of a sentence reads more clearly. A|B returns a histogram of five votes to the user in 5–7 seconds.

This paper carries realtime crowdsourcing from recruitment, through programming patterns, to applications. We make the following contributions:

1. The *retainer model*, which produces synchronous crowds by recruiting workers in advance and alerting them when work is available. We present empirically derived design guidelines for retainer systems.
2. The *rapid refinement* programming pattern, which looks for emerging agreement in synchronous crowds to narrow down a large search space to a single result.
3. *Adrenaline*, a realtime crowd-powered camera that crowdsources the selection of the best photographic moment in a short movie in roughly ten seconds. Puppeteer and A|B extend these ideas into creative design tasks, opinion tasks, and online feedback to workers.

We implement these ideas on Amazon's Mechanical Turk platform. However, we also suggest ways that realtime crowdsourcing techniques might impact future platforms.

The remainder of this paper proceeds as follows. We first survey related work on systems and studies that gather crowds for quick results. We then introduce the retainer model and describe several experiments exploring its effectiveness. We introduce Adrenaline, and use it to motivate and explain the rapid refinement design pattern. We evaluate Adrenaline to test the effectiveness of our approach, and finally introduce Puppeteer and A|B to demonstrate the reach of realtime crowd-powered interfaces.

RELATED WORK

The work presented in this paper builds upon research on low-latency crowdsourcing, synchronous crowds, and crowd-powered interfaces.

Latency in Crowdsourced Tasks

An important thread of work models and influences wait times on crowdsourced task markets. We use several of these techniques, then push beyond single workers to crowds and from half-minute results to 5–10 second results.

Reducing task completion time has become one of crowdsourcing's holy grails. Increased payment leads to higher work output, which translates to faster completion times [19]. Task type (e.g., audio transcription vs. product review) impacts completion time in long-running tasks, but time of day and day of week have a much weaker effect [27]. quikTurKit recruits workers in advance to solve old tasks, then replaces the old tasks with new work when requests arrive. It also re-posts tasks to keep them near the top of the "most recent" ordering, which is a popular view that workers use to find tasks [5]. We adopt versions of both these techniques in Adrenaline, then introduce new techniques to gather crowds simultaneously, reduce variance in lag times, and reduce wait times by a factor of ten — without added cost.

No work to our knowledge has investigated ways to reduce latency once workers arrive. This is the focus of the rapid refinement algorithm.

Synchronous Crowds

In addition to low-latency crowdsourcing, researchers have begun investigating synchronous, collaborative crowds. Our work introduces the notion that low-latency crowds can enable synchronous crowds on demand, and that algorithmic designs can coordinate synchronous crowds through complex workflows quickly.

The ESP Game pioneered the use of multiple individuals working simultaneously on human computation tasks [26]. Our work adopts the game's notion of simultaneous independent agreement, as well as the idea of replaying old episodes when there are no collaborators.

Synchronous online experiments have gathered some of the largest simultaneous crowds so far. One approach is to build a panel of workers prior to the study, e-mail the night before the study, and recruit three times as many workers as needed [18]. The retainer model is a more sustainable, shorter-term variant of the waiting room in this approach. Our work recruits somewhat smaller crowds, but needs orders of magnitude less time to do so.

Finally, synchronous crowds open up the possibility for crowd collaboration. For example, Mao et al. used Mason and Suri's techniques to recruit crowds to solve distributed problems like graph coloring [17]. Kittur used collaborative text authoring software to enable collaborative text translation, finding that workers would coordinate and communicate with each other while working [13]. The Shepherd system likewise recognizes that workers overlapping in time could provide synchronous peer feedback [7]. The retainer model can bring crowds of this kind together relatively quickly, further enabling this kind of work. Many of these techniques draw on synchronous group collaboration research (e.g., [8,11]).

Crowd-Powered Interfaces

Our work aims to break ground on a new class of realtime interactivity in crowd-powered interfaces. There are currently two approaches to managing latency in crowd-powered interfaces: streaming and batching results. Streaming is used by VizWiz [3] and CrowdSearch [28]: they return work to the user, unfiltered, as soon as it is available. Streaming returns work relatively quickly. However, because the system does not have time to aggregate answers, it cannot perform much quality control via redundancy, and it tends not to use complex workflows. Soylent [2] takes a second approach, batching: it waits longer, but has the time to execute a multi-stage workflow and combine all results into a single interface. Our work is closer to a batching approach, but it shortens batch time to a few seconds. High-frequency batching enables workflows while still providing feedback to the user every few seconds.

Adrenaline, Puppeteer and A|B use the crowd to expand on previous applications. Adrenaline shares the goals of the Moment Camera [6] by recording continuously and using crowds to make the hard semantic decision of what is a good photo. The Moment Camera and Adrenaline complement each other: computational photography can check whether subjects' eyes are open and can tune camera settings, but it is only trained on certain classes of images and it is much harder to train an algorithm to make subjective judgments. Puppeteer gives the crowd a rigid shape deformation tool [10], then relies on the crowd to generate plausible deformations based on a text description. It is inspired by projects like The Sheep Market [14], and introduces crowd assistance as part of the tight loop of the creative process. Furthermore, it allows the user to interact with the workers as they explore. A|B gathers opinions quickly from multiple independent people, as suggested by Surowieki [24], though it focuses on subjective assessment rather than guesses at factual questions. Toomim [25] used crowds to evaluate alternatives more thoroughly, though with much longer latency.

ADRENALINE: A REALTIME CROWD-POWERED CAMERA

"You must know with intuition when to click the camera. That is the moment the photographer is creative. […] The Moment! Once you miss it, it is gone forever."
— Henri Cartier-Bresson [1], 1957

Novice photographers often struggle to capture what Cartier-Bresson called *The Decisive Moment*. The instant when the shutter opens, the subject might have broken into an awkward smile, the angle might be poor, or the decisive moment might have passed. As a result, photos taken by novices can feel stilted or 'off'. Experienced photographers learn to compensate by taking many photographs — tens of photos rather than one — then sorting through them later to find the gem. They try multiple angles, capture photos over several seconds, or ask subjects to strike different poses.

Adrenaline is a realtime crowd-powered camera that aims to find the most photogenic moment by capturing many alternatives. Rather than taking a single photo, Adrenaline captures a ten-second movie and recruits a crowd to quickly find the best photographic moment in the movie (Figure 1). Within five seconds after the movie is captured, the user can see crowd members exploring the timeline (colored triangles in Figure 1). A few seconds later, the user can see that the crowd has narrowed down to a small fraction of the timeline, then again to a few adjacent frames, and finally to a final photo in a total of about eleven seconds. This means that a final photo is ready just moments after the user finishes reviewing the movie they just captured.

Adrenaline's goal is to mimic the instant-review capabilities of cameras today. Seeing a photo quickly means that users can take another photo if they want, show it instantly to friends, or post it to the web. The visceral thrill of novice photography can be lost when the photo is not available for minutes or days. Many novices never review the pile of extra photos, so we believe that is extremely important to complete the image selection process while Adrenaline still has the user's attention.

Existing crowdsourcing techniques cannot support Adrenaline's goal of 10–12 second latency. To achieve this, we introduce the use of synchronous crowds via the *retainer model* for recruitment, and the *rapid refinement algorithm* to guide the search process quickly and reliably.

RETAINER MODEL

To power realtime applications like Adrenaline, we need to gather not just one individual but an entire crowd quickly: a synchronous, 'flash' crowd. We would like the crowd to turn their attention to our task as soon as it is available, and to do so without paying for more tasks than necessary.

While it was clear from our explorations that workers would respond more quickly when paid money, it was not certain that they could respond fast enough for an interactive system. How should a retainer system be designed to 1) guarantee a fast response time, 2) be cheap enough to scale, *and* 3) maintain that response time after a long wait?

In this section, we introduce the *retainer model* for synchronous crowdsourcing and empirically derived design guidelines for its use. This approach solves all three issues by placing workers on retainer — signed up to do work when it is available — for a small fee, then allowing them to pursue other work while they wait. When the user makes a request, the retainer model alerts the workers. Our designs result in a majority of workers returning two seconds after request, all together, enabling synchronous crowds.

Retainer Design and Wait Time

Workers agree to be put on retainer by accepting the task. They are given task instructions and an example, and told that they will be alerted when a task is ready. We scale the task price up by expected wait time, usually 0.5¢–1¢ per minute on retainer. After workers agree, they are free to leave the browser tab open and pursue other work.

The worker's browser polls a work server to see if tasks are available. When a user posts a task, the work server notifies the client, and the client's browser issues a Javascript `alert()` and an audio chime to signal the worker. Optionally, the work server may also offer a small bonus to reward quick responses. Workers dismiss the alert when they arrive, then begin the task. If no new tasks are ready by the end of the retainer period, the retainer model gives workers an old task to perform. As work arrives more consistently, however, the chances of this happening become lower.

Retainer Field Experiments

Can workers react quickly enough to support a realtime application, especially when they may be distracted with other tasks? This section describes field experiments of the retainer model that investigate its effectiveness.

We created a benchmark Mechanical Turk task that instructed workers to click on all the verbs in a random paragraph from a blog or a book. Workers were told that the task would be ready within a specific retainer time limit, then the web page began an invisible countdown that sampled randomly between zero seconds and the maximum retainer time. When the countdown finished, the page alerted the worker and showed a task. We prevented workers from accepting more than one of our tasks at a time.

Study 1: Retainer Time

To test how long we could keep workers primed, we experimentally manipulated retainer time to vary between 0.5, 1, 2, 5, 10, and 30 minutes. We scaled payment roughly linearly with retainer time: 2¢, 3¢, 4¢, 7¢, 12¢, and 32¢. We hypothesized that worker response time would increase after 1–2 minutes, as workers stopped monitoring the page.

To reduce the chance that workers would see multiple price points for the same task, we posted each set at different hours. We ran the experiment over a period of six days, in six separate one-hour periods each day, and randomized the order of conditions. A total of 280 workers completed 1545 tasks. We removed and rejected 103 tasks because they disagreed significantly with our ground truth.

Results. For retainer times under ten minutes, 46–61% of workers dismissed the alert within two seconds and 69–84% of workers dismissed the alert within three seconds. These curves in Figure 2 asymptote to a completion rate of 83–87%: the rest of the workers never returned to complete the task. Retainer times of ten minutes or more resulted in much lower completion rates, 49–66%. The median time between dismissing the alert and completing the first incremental piece of work (clicking on a verb) was 3.35 seconds across all conditions.

These results suggest that for wait times under ten minutes, we could expect to produce a crowd in two seconds, and a larger crowd in three seconds. In the next study, we investigate how to improve response time and completion rates further through retainer designs.

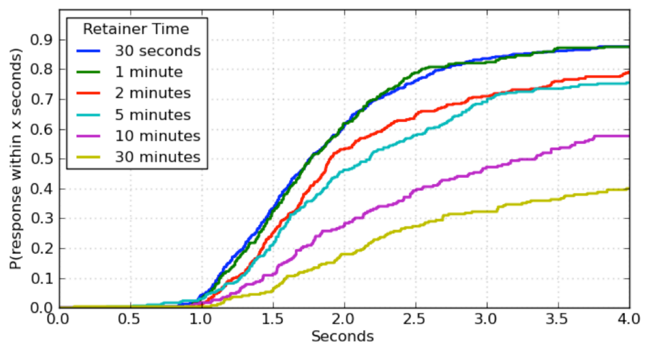

Figure 2. For retainer times under ten minutes, a majority of workers responded to the alert within two seconds and three-quarters responded within three seconds. N=1442.

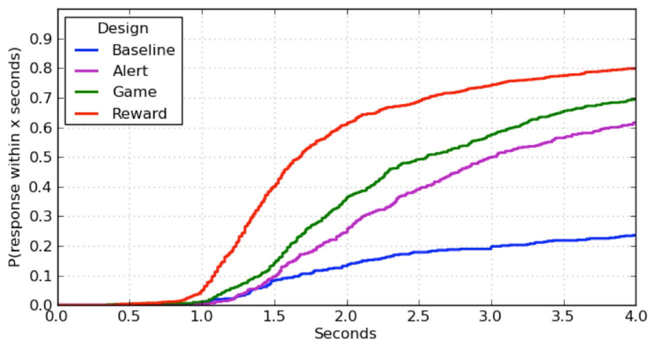

Figure 3. A small reward for fast response (red) led workers in a ten-minute retainer to respond as quickly as those on a two-minute retainer without reward (Figure 2, red). N=1913.

	30 sec	1 min	2 min	5 min	10 min	30 min
Median	1.77 s	1.77 s	1.91 s	2.18 s	3.34 s	10.32 s
3rd quartile	2.44 s	2.39 s	3.46 s	3.75 s	—	—
Completion	86.6%	87.2%	82.9%	75.1%	66.4%	49.4%

Table 1. A tabular representation of Figure 1.

	Baseline	Alert	Game	Reward
Median	36.66 s	3.01 s	2.55 s	1.68 s
3rd quartile	—	6.92 s	5.01 s	3.07 s
Completion	64.2%	76.5%	76.7%	85.5%

Table 2. A tabular representation of Figure 2.

Study 2: Alert Design
While Study 1's results are already good enough to get a crowd quite quickly, can we improve on them by changing the reason that workers would pay attention? Can we incentivize the slow workers to move more quickly?

We investigated design and financial incentives to shift the curve so that more workers came within the first 2–3 seconds. We used the 12¢ 10-minute retainer condition from Study 1, which exhibited a low completion rate and a slower arrival rate. The *alert* condition functioned as in Study 1, with a Javascript alert and audio chime. Bonuses can be powerful incentives [19], so we designed a *reward* condition that paid workers a 3¢ bonus if they dismissed the alert within two seconds. Two seconds is short enough to be challenging, but not so short as to be out of reach. To keep workers' attention on the page, we created a *game* condition that let workers optionally play Tetris during the waiting period. Finally, to isolate the effectiveness of the Javascript alert, we created a *baseline* condition that displayed a large Go button but did not use an audio or Javascript alert. We hypothesized that the bonus and game conditions might improve response time and completion rate.

For Study 2, we implemented a between-subjects design by randomly assigning each worker to a condition for the same verb-selection task. We posted tasks for four hours per day over four days. Workers completed 1913 tasks — we removed 90 for poor work quality.

Results. Paying a small reward for quick reactions had a strong positive impact on response time (Figure 3). In the reward condition, 61% of workers responded within two seconds vs. 25% in the alert condition, and 74% responded within three seconds vs. 50% in the alert condition. Roughly speaking, the ten-minute retainer with reward had similar performance to the two-minute retainer without reward. In addition, workers in the reward condition completed 2.25 times as many tasks as those in the alert condition (734 vs. 325), suggesting that the small bonus has a disproportionately large impact on work volume. Predictably, the baseline condition without the alert dialog performed poorly, with 19% returning within three seconds. The game was not very popular (5.7% of completed tasks cleared a row in Tetris), but had a small positive impact on reaction times.

Retainer Model Discussion
Our data suggest that the retainer model can summon a crowd two seconds after the request is made. In exchange for a small fee, the retainer model recalls 50% of its workers in two seconds and 75% in three seconds. Though reaction times worsen as the retainer time increases, a small reward for quick response negates the problem. Our experiment commonly produced 10–15 workers on retainer at once, suggesting that users could fairly reliably summon a crowd of ten within three seconds. Applications with an early indication that the user will want help (for example, a mouseover on the icon or an "Are You Sure?" dialog) can eliminate even this delay by alerting workers in advance.

The cost of the retainer model is attractive because it pays workers a small amount to wait, rather than spending money to repeat old tasks. The cost of the retainer model depends on the desired arrival time t, the empirical arrival distribution $P(arrive \leq t)$ as in Figure 2, and the desired workers w. The number of retainer workers r needed is:

$$r = \frac{w}{P(arrive \leq t)}$$

For example, to recruit 5 workers within 3 seconds in the ten-minute retainer with reward, the system should place 8 workers on retainer (rounded from 7.8), then expect that 7 will return to complete the task and 4 will earn rewards.

Assuming that we set the retainer length longer than the expected time period between requests T, the hourly (60-minute) cost of the retainer model depends on the retainer wage per minute c_r, and the base cost for the task c_t:

$$cost = r\left(\frac{60}{T}c_t + 60c_r\right)$$

Paying a half-cent per minute on retainer and with no wasted tasks, one worker on retainer costs 30¢ per hour.

RAPID REFINEMENT: COORDINATING SYNCHRONOUS CROWDS FOR FAST RESULTS

Once recruitment times are negligible, slow work time dominates the user experience. Even straightforward tasks like question-answering can take thirty seconds under good conditions [3], and the wait can be much longer before there are enough answers to make a clear judgment.

The problem is not just one of minimum response time, but also one of variance. Time variance means that wait time will not be reliable: it will depend on whether the system happened to recruit a fast worker. That worker may also produce low quality results. Worse, nontrivial human computation algorithms (e.g., [16]) often wait for several results before proceeding: it is even less likely that a system would recruit multiple fast workers. We could require workers to finish within a short time limit, but our experience is that this is stressful to workers and leads to poor results.

To complete nontrivial crowdsourcing tasks in realtime, we must develop new algorithms and programming patterns to return quality results quickly and reliably. Like traditional randomized algorithms, we may be willing to sacrifice some amount of comprehensiveness for faster runtime.

The insight behind our solution is that *low-latency crowds are synchronous crowds*: all workers are working simultaneously. Synchronous crowds can interact with, influence, and communicate with each other and the user. In most on-demand crowdsourcing approaches, workers do not overlap in time enough for synchronous designs to make sense. However, the retainer model makes it practical to assume, for the first time, that the crowd is all present simultaneously a few moments after request. So, rather than waiting for all results in order to continue, a realtime algorithm has the opportunity to influence the process *as it takes place*.

We take advantage of synchronous crowds by recognizing potential agreement early on, while workers are still exploring, and then focusing workers' attention on the area of agreement. The insight is that the majority of Turkers who are efficient satisficers [22] can guide the process: these workers decide on the gist of the solution quickly, but can take time to commit to a final answer. Rapid refinement recognizes when several workers are *likely to* agree and commits for them by focusing the task on that area. Fast workers will still find an answer quickly and contribute to the vote, whereas slow workers benefit from the focus on a smaller search space. In the next section, we will describe the rapid refinement algorithm that implements this idea.

Algorithm Design

The rapid refinement algorithm repeatedly narrows down a search region to a fraction of its existing size when it senses that workers independently agree on the subregion (Figure 4). It is appropriate for a variety of search-oriented human computation tasks.

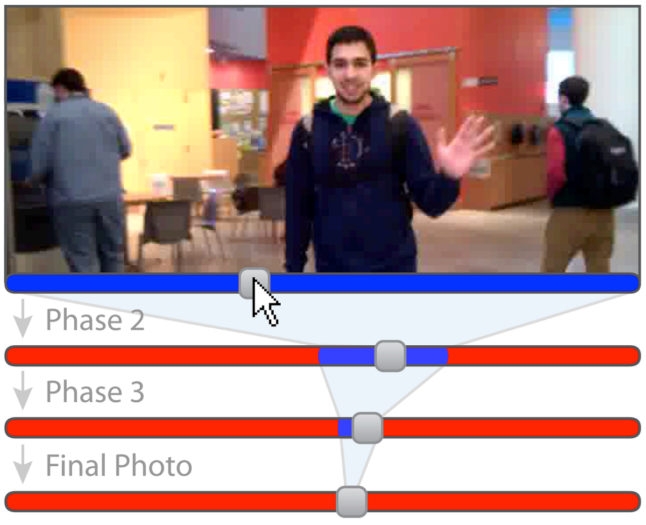

Figure 4. Rapid refinement repeatedly shrinks the working area for all workers when it detects that several independent workers are exploring the same area.

Rapid refinement begins with the entire search space available and workers initialized to a random position. The algorithm takes place in *phases* where each phase narrows to a smaller search region. Workers trigger a new phase by independently focusing on the same area for a period of time. The algorithm depends on three values: the agreement range r, agreement time t, and agreement amount a. If a fraction a of the workers stayed within a range less than a fraction r of the existing search area for at least t seconds, rapid refinement declares agreement. It then shrinks the search space to match the agreement range and begins a new phase. Workers are no longer able to explore the out-of-bounds area, and must try to agree within the new region. This process repeats until convergence. To approve or reject work, rapid refinement looks at whether the worker agreed with at least one phase change.

In Adrenaline, the algorithm begins with the entire video timeline. Workers vote on a good photo using their timeline slider. They cannot see others' sliders, which encourages independent agreement [24]. When at least 33% of the workers have been in the same 25% of the timeline for at least 2 seconds, Adrenaline declares agreement. These values can be adjusted to trade off delay for false positives. With these values, Adrenaline converges in 3–4 phases per video. The first phase is the slowest, and agreement accelerates as workers' attention is focused on one area.

The rapid refinement algorithm has several benefits aside from speed. First, it produces preliminary results that can be returned to the user early. Early results reduce interface latency and allow the user to provide feedback on the process. Second, it combines work and verification into one stage, which saves cost and time for a separate verification step. Third, workers tailor their votes toward what someone else might think, which minimizes individual bias [9].

Rapid refinement makes tradeoffs. First, the algorithm may focus too early on a single part of the search space, especially if the low quality workers are the first to respond. With four or more workers, it is possible to fork the crowd into two simultaneous groups to help avoid this problem. Forking the crowd has the additional benefit that the algorithm can explore multiple promising paths at the same time. Second, it may stifle individual expression: talented workers might be forced to agree with the majority. A future system could recognize such workers and give them more weight in the votes.

A slight modification to the algorithm can guarantee convergence. In particular, systems can increase the agreement fraction r as time passes in each phase. Then, given at least three workers, the algorithm will always converge.

EVALUATION

We have argued that the retainer model and rapid refinement combine in Adrenaline to produce a realtime crowd-powered interface by controlling variance in wait time and quality. In this section, we report on an evaluation of Adrenaline that evaluated this claim, stress-tested the retainer model and rapid refinement, and investigated whether end users understand the system.

Method

We recruited 24 participants through e-mail lists in exchange for a $20 gift certificate. Fourteen were male and ten were female, and the median age was 25. About half had taken a photograph on a cell phone camera or consumer camera in the past month, and five had taken pictures on a DSLR camera in the past month. The typical participant was a young, technically competent student who had a moderate interest in photography as a hobby (median Likert response: 4/7). Participants arrived to the study in pairs.

We gave each participant a smartphone with Adrenaline installed and introduced them to the application. We did not allow participants to immediately see the still photos that Adrenaline chose, so that we could compare rapid refinement to other approaches. Participants began by taking a video portrait of their partner. Then, the pair spent fifteen minutes in a public area capturing videos of people, actions, or landscapes. Finally, participants chose two videos in addition to the portrait to submit for the evaluation.

On a weekend afternoon, we generated candidate photographs using the following computational, expert, and crowdsourced approaches:

- *Rapid Refinement:* we required five workers to be on retainer before labeling each video.
- *Generate-and-Vote:* a standard crowdsourcing approach [15] in two stages. First, five retainer workers independently selected a frame in the video. Then, keeping the fastest three results returned, we use eight retainer workers to vote on the best photo of the three.
- *Generate-One*: using the same dataset as Generate-and-Vote, this condition simulated stopping the process as soon as the first worker submitted a photo.
- *Photographer:* a professional photographer labeled the best still frame in each video.

	Delay		**Quality**
	Median	Mean	9pt Likert
Computer Vision	–	–	4.9 σ=2.2
Generate and Vote	41.9	45.3 σ=14.0	6.6 σ=2.1
Generate One	13.6	16.3 σ=9.8	5.9 σ=2.6
Professional Photographer	–	–	6.4 σ=2.3
Rapid Refinement	11.6	12.6 σ=4.3	5.8 σ=2.2

Table 3. Rapid Refinement was the fastest algorithm and had the lowest timing variance, though it sacrificed slightly on quality compared to Generate-and-Vote.

- *Computer Vision:* the still frame selection algorithm on YouTube, which uses recent computer vision research algorithms [12].

We used the quikTurKit technique to repeatedly post tasks and keep work near the top of the Mechanical Turk task list, then implemented a five-minute retainer and a 2¢ bonus for quick response. We paid 4.5¢ on average for a rapid refinement or generate task (quikTurKit posts tasks at multiple price points simultaneously), and 3.5¢ on average for a vote task. Including bonuses and removing one worker on average who never responded to the retainer, these costs added up to 4(4.5¢) base + 2(2¢) bonus = 22¢ per video for the rapid refinement and generate tasks. Voting added eight workers: we estimated 7 would appear and 3 would earn the bonus, for an additional 31¢. So, rapid refinement and generate-one cost 22¢ per video, and generate-and-vote cost 53¢ per video. In a live system, it would be feasible to use fewer workers and pay closer to 10¢ rather than 22¢. On Mechanical Turk, we posted half of the videos first in the rapid refinement and generate-and-vote conditions to compensate for order effects.

We then contacted all of our participants and asked them to rate each still photo on a 9-point Likert scale. We instructed participants to ignore aspects of the picture like contrast and color that could be computationally improved.

Results

We used the retainer model to post each video as soon as there were five workers on retainer, stress-testing the volume of the retainer model. Adrenaline had enough workers to label a video every 45 seconds (median). Worker arrivals were bursty, but the median time between retainer arrivals was 6.3 seconds. The median time between unique worker arrivals was 48.8 seconds. These numbers are dependent on the current workforce state of Mechanical Turk, and will change as the market grows or more tasks use retainers.

Timing. Table 3 lists our results. Rapid refinement returned the fastest results, with a median total time of 11.6 seconds (μ=12.6, σ=4.3). Generate-One, which used the first available photo was a few seconds slower, with a median time of 13.6 seconds (μ=16.3, σ=9.8). Generate-One's timing standard deviation was nearly twice that of rapid refinement. These variances are significantly different from each other, F(71, 71)=5.17, p<0.001.

The timing data is non-normal, so we square-root transformed it to satisfy normality assumptions. An ANOVA comparing the delay for the three human computation algorithms is significant F(2, 213) = 278.11, p < .001, and post-

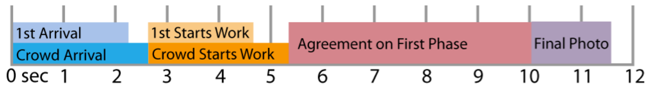

Figure 5. Timeline of the median Adrenaline execution.

hoc pairwise Tukey comparisons confirmed that all conditions were significantly different than each other (all $p <$ .05), confirming that rapid refinement is fastest.

Figure 5 outlines the median timing distributions of a rapid refinement process. One worker typically arrived 2.2 seconds after the video was uploaded, and the second and third came within 2.6 seconds. At least two workers were moving their sliders by 5.3 seconds after request. After the crowd began exploring, agreement took 4.7 seconds in the first phase. The median first phase completed a total of 10.05 seconds after request (μ=10.65, σ=3.0). The following phases lasted 0.75 seconds each, typically. These phases moved more quickly because workers were often already agreeing on a small area when the first phase completed.

Quality. Quality ratings tell an unexpected story. Participants' ratings had high variance, making it difficult to draw statistical conclusions (Table 3). However, Generate-and-Vote appears to match or beat the Photographer condition. While we believe that a photographer would take better pictures given the opportunity to operate the camera, it appears that an unskilled crowd may be equally talented at selecting good moments. Generate-One also performed better than expected, roughly matching Rapid Refinement. As with timing, however, Generate-One was less reliable and had a higher variance. The Computer Vision approach was the least successful, which suggests that using crowds in a subjective photo quality task is a good match.

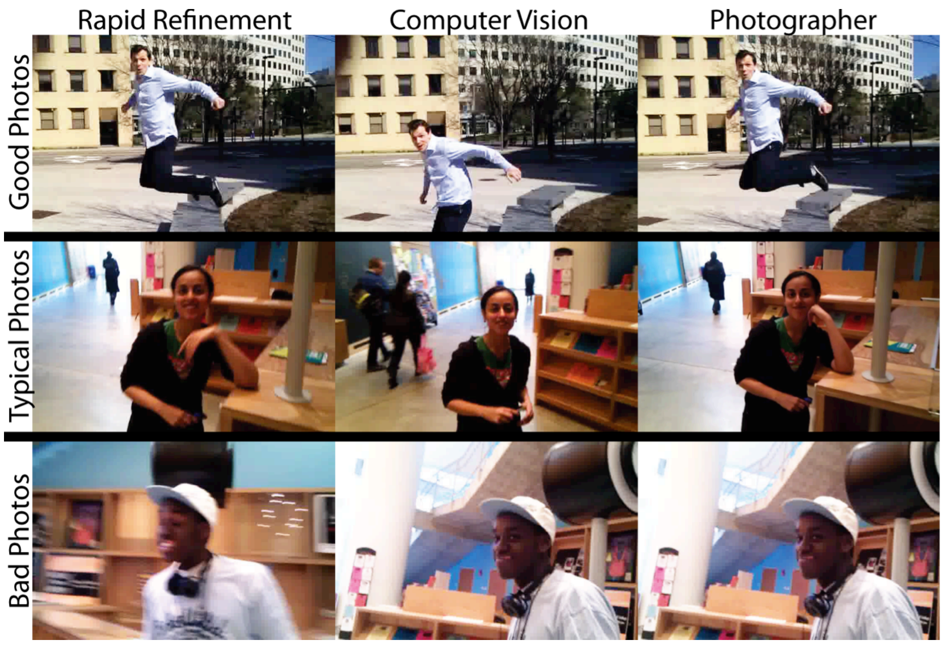

Figure 6. Photos from the Adrenaline study. The examples are good, typical, and bad photos that rapid refinement recommended. The computer vision and photographer columns demonstrate how other approaches performed on the same movie.

Study Discussion

This study suggests that rapid refinement guides crowds of two to five people to select a good photo faster than even the fastest member of a similar-size group. We might expect that workers would conflict, stalemate and disagree with each other. However, bottlenecks were uncommon, especially with more than two crowd workers. This result suggests that, rather than interfering with each other, synchronous crowds may hold significant promise for exploring new forms of cooperation and collaboration (e.g., [13]).

Quality may be the most salient issue with rapid refinement: its photos were of reasonable quality, but it did not match Generate-and-Vote. One common source of error was too-fast agreement in the later phases. Sometimes the algorithm decided that workers agreed in the second phase before they had adequate time to explore the timeline and make a decision. We have prototyped designs to compensate for this: for example, requiring that workers explore a minimum fraction of the range before their votes are counted, or requiring a minimum time lapse between phases. These approaches empower the designer to trade off increased lag in exchange for better quality.

A second issue is that rapid refinement uses constants that may depend on task, crowd size and the number of items being explored. For example, constants that work well for small crowds of 3–5 may act differently when 10–15 crowd members arrive. A more principled approach would treat workers' locations as populating a probability density function over frames. Then, measures of distribution peakedness, like kurtosis, would likely ease this problem.

The retainer model was more aggressive than necessary. Often workers joined midway through the process, resulting in more workers than needed. Given a sufficiently busy system, we might re-route latecomers from a retainer into a different task. Alternatively, there may be design patterns that place latecomers into a complementary role in the computational process, like vetting.

OTHER REALTIME APPLICATIONS

In this section, we expand the design space of realtime crowd-powered interfaces beyond Adrenaline. We introduce two applications, Puppeteer and A|B, to explore questions of creative content generation and even lower latency.

Puppeteer: Creativity and Creation in Seconds

Realtime crowds can thread creation tasks into user interfaces as well. Puppeteer (Figure 7) works in conjunction with Adobe Photoshop to support large-scale content generation and synchronous feedback

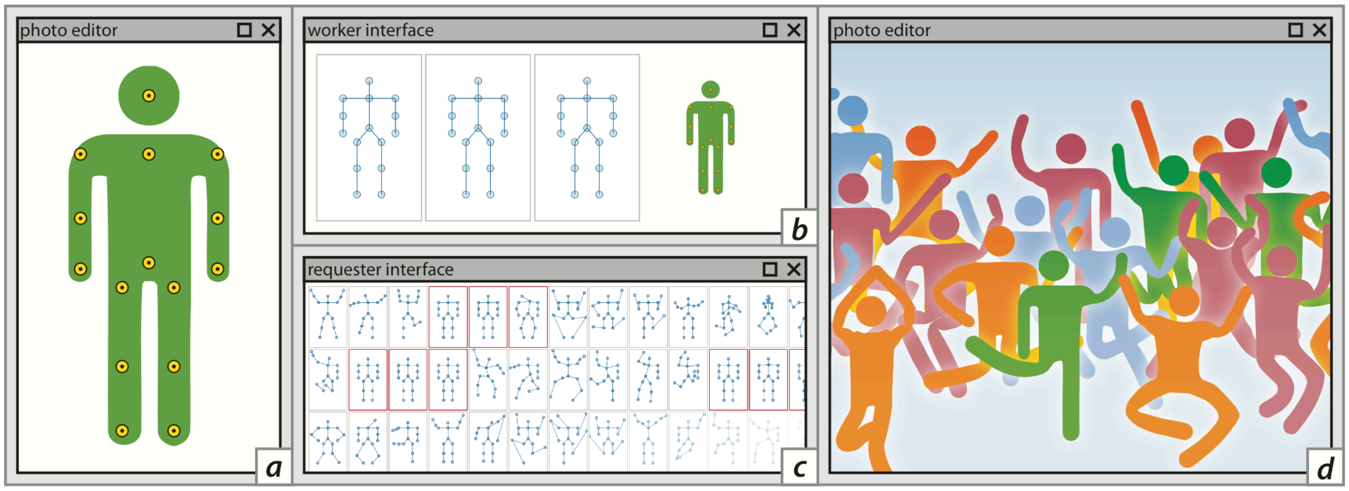

Figure 7. Puppeteer allows an artist or designer to generate a group of articulated figures quickly, and to interact with workers as they work. a) The user demonstrates control points. b) Workers move control points to match a request, like "make the person look excited!" c) The user watches incoming results and interacts with workers. d) The final result in Photoshop.

to workers. Artists often want to create a large number of believably varying objects, like an excited audience at a concert or flowers in a field. Algorithmic techniques to generate poses may not be realistic, semantically meaningful, or generalizable to new objects.

Puppeteer users specify control points on an image using Photoshop's Puppet Warp tool [10] (Figure 7a). Users give a textual description of their goal (*e.g.*, "Make the person look like he is dancing"), then workers each pose three figures to match the goal. As workers progress, the user observes them through a small multiples interface (Figure 7c). Then, because of the realtime nature of the application, the user can communicate with workers (*e.g.*, "Can you make another one more like your first?") These poses can be imported back into Photoshop to arrange the final result.

As an illustrative example, we simulated a user request to a large number of Puppeteer workers that they pose a human stick figure so that it looked excited. After they finished two puppets, we programmatically prompted workers with a request to make the third figure look like it is jumping. Anecdotally, the message was quite effective — the majority of workers' third puppets appeared to be jumping whereas the first and second puppets were rarely doing so.

To understand the latency and total throughput of Puppeteer, we repeated the "excited" task, but removed the prompt. We began the task when there were 8 workers on retainer, received the first control point movement 2.1 seconds later, and received the first completed figure in 25.0 seconds. The first worker completed the entire task (3 puppets) in 46.1 seconds. Workers completed 300 puppets in 16 minutes and 35 seconds, or one puppet every 3.3 seconds. Work output rate was relatively constant throughout.

A/B: Instant Crowd Feedback

It can be hard to escape from our own biases when we try to predict what others will think. Crowds are certainly good at having opinions, but high latency makes them less useful for snap decisions. A|B is our lowest-latency crowd feedback platform. The user asks a question and takes two pictures, then a histogram of crowd feedback appears moments later. A user might try on two different sweaters, take pictures of each, and ask which one looks better; an artist might sketch two different versions of a character and ask which one looks more engaging; a designer might want fast aesthetic feedback on a sketch.

In our tests with eight workers on retainer, A|B returned five opinions in as little as five seconds.

DISCUSSION

In this section we explore the implications and limitations of realtime crowdsourcing.

Synchronous crowds and rapid refinement open the door to many applications and techniques. Crowdsourcing has largely been confined to divide-and-conquer tasks, but synchronous crowds enable coordination and collaboration on a new set of problems. Such crowds can edit documents simultaneously (e.g., [13]), work on team tasks, and distribute large tasks in new ways. Rapid refinement also has applications in other search tasks. For example, rapid refinement might power a predictive mobile web browser that uses crowds to search the page for the user's next action and offer swipe-to-complete for that action, or redesign Soylent's Find task [2] as a synchronous, parallel process.

Larger, non-visual search spaces are another avenue for future work. If it is harder for workers to skim the items or the search space is very large, agreement will be sparser. We are interested in separating tasks into overlapping segments to address this problem. The core interface insight is to stop workers from worrying about deciding on one right answer and instead quickly call out promising areas.

We envision a future where crowdsourcing markets are designed for quick requests. A centralized system could use queueing theory to manage the worker pool and decide when to hire. Queueing theory makes assumptions about the distribution of worker arrivals, request arrivals and work times that would imply how many workers to put on retainer in advance. Workers could *follow* tasks they liked, saving them time searching the task list, and be on call for several tasks simultaneously. The system could then route workers to tasks where they are needed most.

Until we have such a large-scale realtime crowdsourcing platform, scale remains an issue. Our experiments in April 2011 found it relatively straightforward to recruit 8–15 workers on retainer, but if thousands of users began using Adrenaline, it might exhaust the worker pool. Given sufficient demand, however, more workers would likely enter the market in exchange for higher wages. Successful realtime services might eventually recruit their own full-time crowds like ChaCha [www.chacha.com].

CONCLUSION

This paper introduces techniques for realtime crowdsourcing and its applications in user interfaces. Where the fastest crowd-powered interfaces used to be limited by a median response time of nearly a minute [3], we show that it is possible to recruit a crowd within two seconds, get that crowd to answer a simple query within five seconds, and complete a complex search in roughly ten seconds. We presented Adrenaline, a realtime crowd-powered mobile phone camera that captures several seconds of video, then uses the crowd to quickly find the best still photo in the set.

Our solution is to introduce on-demand synchronous crowds, where workers arrive and work simultaneously. To create on-demand synchronous crowds, we present the retainer model of recruitment, which pays workers a small amount to come quickly when asked, and a set of empirically derived design guidelines for its use. With synchronous crowds, algorithms can identify regions of likely agreement before workers would normally select a final answer. This intuition led to rapid refinement, a design pattern that focuses the search space on areas of emerging agreement to quickly narrow complex search tasks. The combination of these two ideas enable reliably fast turnarounds for Adrenaline — ten seconds to a preview and the final photo a second or two later. This speed is on average faster than even the fastest individual worker. Finally, we extended the design space of realtime crowd-powered interfaces with A|B and Puppeteer, which achieved an even lower latency of five seconds to five responses and embedded crowd contributions directly in an authoring interface.

ACKNOWLEDGMENTS

Special thanks to Katrina Panovich, Jeff Bigham, our colleagues at MIT and Adobe, and the workers on Amazon Mechanical Turk. This work was funded in part by Quanta Computer via the T-Party Project. Additional funding provided by Adobe and a NSF Graduate Research fellowship.

REFERENCES

1. Bernstein, A. The Acknowledged Master of the Moment. *Washington Post*, 2004.
2. Bernstein, M.S., Little, G., Miller, R.C., et al. 2010. Soylent: A Word Processor with a Crowd Inside. *UIST '10*.
3. Bigham, J.P., Jayant, C., Ji, H., et al. 2010. VizWiz: Nearly Real-time Answers to Visual Questions. *UIST '10*.
4. Card, S.K., Moran, T.P., and Newell, A. 1983. *The psychology of human-computer interaction*. Lawrence Erlbaum.
5. Chilton, L.B., Horton, J.J., Miller, R.C., and Azenkot, S. 2010. Task search in a human computation market. *HCOMP '10*.
6. Cohen, M.F. and Szeliski, R. 2006. The moment camera. *Computer 39*, 8, 40–45.
7. Dow, S.P., Bunge, B., Nguyen, T., et al. 2011. Shepherding the Crowd: Managing and Providing Feedback to Crowd Workers. *Ext. Abs. CHI '11*.
8. Greenberg, S. and Bohnet, R. 1991. GroupSketch: A multi-user sketchpad for geographically-distributed small groups. *Graphics Interface'91*.
9. Hacker, S. and Von Ahn, L. 2009. Matchin: eliciting user preferences with an online game. *CHI '09*.
10. Igarashi, T., Moscovich, T., and Hughes, J.F. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (TOG)*, 1134–1141.
11. Ishii, H. and Kobayashi, M. 1992. ClearBoard: a seamless medium for shared drawing and conversation with eye contact. *CHI '92*.
12. Ižo, T. and Yagnik, J. 2009. Smart Thumbnails on YouTube. *Google Research Blog*.
13. Kittur, A. 2010. Crowdsourcing, collaboration and creativity. *XRDS 17*, 2, 22–26.
14. Koblin, A.M. 2009. The sheep market. *C&C '09*.
15. Little, G., Chilton, L., Goldman, M., and Miller, R.C. 2010. Exploring iterative and parallel human computation processes. *HCOMP '09*.
16. Little, G., Chilton, L., Goldman, M., and Miller, R.C. 2010. TurKit: Human Computation Algorithms on Mechanical Turk. *UIST '10*.
17. Mao, A., Parkes, D.C., Procaccia, A.D., and Zhang, H. 2011. Human Computation and Multiagent Systems: An Algorithmic Perspective. *AAAI '11*.
18. Mason, W. and Suri, S. 2010. A Guide to Conducting Behavioral Research on Amazon's Mechanical Turk.
19. Mason, W. and Watts, D.J. 2009. Financial Incentives and the "Performance of Crowds." *HCOMP '09*.
20. Nielsen, J. 1993. *Usability engineering*. Morgan Kaufmann.
21. Schurman, E. and Brutlag, J. 2009. Performance related changes and their user impact. *Velocity 2009*.
22. Simon, H.A. 1956. Rational choice and the structure of the environment. *Psychological review 63*, 2, 129.
23. Sorokin, A., Berenson, D., Srinivasa, S.S., and Hebert, M. 2010. People helping robots helping people: Crowdsourcing for grasping novel objects. *IROS '10*.
24. Surowiecki, J. 2005. *The Wisdom of Crowds*. Random House, New York.
25. Toomim, M., Kriplean, T., Pörtner, C., and Landay, J.A. 2011. Utility of Human-Computer Interactions: Toward a Science of Preference Measurement. *CHI '11*.
26. Von Ahn, L. and Dabbish, L. 2004. Labeling images with a computer game. *CHI '04*.
27. Wang, J., Faridani, S., and Ipeirotis, P.G. 2011. Estimating the Completion Time of Crowdsourced Tasks Using Survival Analysis Models. *CSDM '11*.
28. Yan, T., Kumar, V., and Ganesan, D. 2010. CrowdSearch. *MobiSys '10*.