AN EXPLORATION OF USER INTERFACE DESIGNS FOR REAL-TIME PANORAMIC PHOTOGRAPHY

Patrick Baudisch, Desney Tan, Drew Steedly, Eric Rudolph,
Matt Uyttendaele, Chris Pal, and Richard Szeliski

Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
Email: {baudisch, desney, steedly, erudolph, mattu, szeliski}@ microsoft.com
pal@ cs.umass.edu

ABSTRACT

Image stitching allows users to combine multiple regular-sized photographs into a single wide-angle picture, often referred to as a panoramic picture. To create such a panoramic picture, users traditionally first take all the photographs, then upload them to a PC and stitch. During stitching, however, users often discover that the produced panorama contains artifacts or is incomplete. Fixing these flaws requires retaking individual images, which is often difficult by this time. In this paper, we present Panoramic Viewfinder, an interactive system for panorama construction that offers a real-time preview of the panorama while shooting. As the user swipes the camera across the scene, each photo is immediately added to the preview. By making ghosting and stitching failures apparent, the system allows users to immediately retake necessary images. The system also provides a preview of the cropped panorama. When this preview includes all desired scene elements, users know that the panorama will be complete. Unlike earlier work in the field of real-time stitching, this paper focuses on the user interface aspects of real-time stitching. We describe our prototype, individual shooting modes, and provide an overview of our implementation. Building on our experiences with Panoramic Viewfinder, we discuss a separate design that relaxes the level of synchrony between user and camera required by the current system and provide usage flexibility that we believe might further improve the user experience.
Keywords: Panorama, Panoramic Viewfinder, user interface, interactive, stitching, real-time, preview.

INTRODUCTION

To allow digital camera users to take pictures with a viewing angle wider than the one supported by the built-in lens, researchers have proposed techniques for stitching multiple photographs (Mann and Picard, 1994; Szeliski, 1996) or a stream of video (Teodosio and Bender, 1993; Irani and Anandan, 1998) into a single contiguous panoramic picture, or panorama. Recent full-view stitching methods even allow automatic stitching of photos taken in arbitrary order and spatial arrangement (Szeliski and Shum, 1997; Bergen et al., 1992). While some recent off-the-shelf cameras offer simple stitch assist modes (see related work section), stitching is still generally performed as a post-hoc step to picture taking.

Making flawless panoramas using a post-hoc stitcher can be challenging. We conducted an informal survey on an internal company mailing list for stitching users and received 26 responses out of 163 mailing list members. Each respondent had taken between 4 and 180 panoramas (median 20). All except one participant who had shot all his panoramas using a tripod had discovered at least one of the following flaws when stitching their pictures:

- Ghosting (88% respondents): Objects that move between frames end up appearing translucent when the stitcher blends frames together. Since cameras typically show only one picture at a time, ghosting can be hard to detect.

- Missing content (65% respondents): Users may miss relevant areas or relevant areas may disappear when cropping the panorama to its final rectangular shape, as illustrated in Figure 2. Even if users are aware of the danger, the difficulty of understanding perspective projections (Edwards, 1999) in combination with the particular geometry of panoramas (cylindrical or spherical projection) can make it hard for users to estimate what space needs to be covered.

- Stitching failed (38% respondents): The stitcher was unable to connect one or more frames to the rest of the panorama. Reasons include lack of overlap between pictures, lack of the texture, or flaws in the pictures, such as poor focus or motion blur.

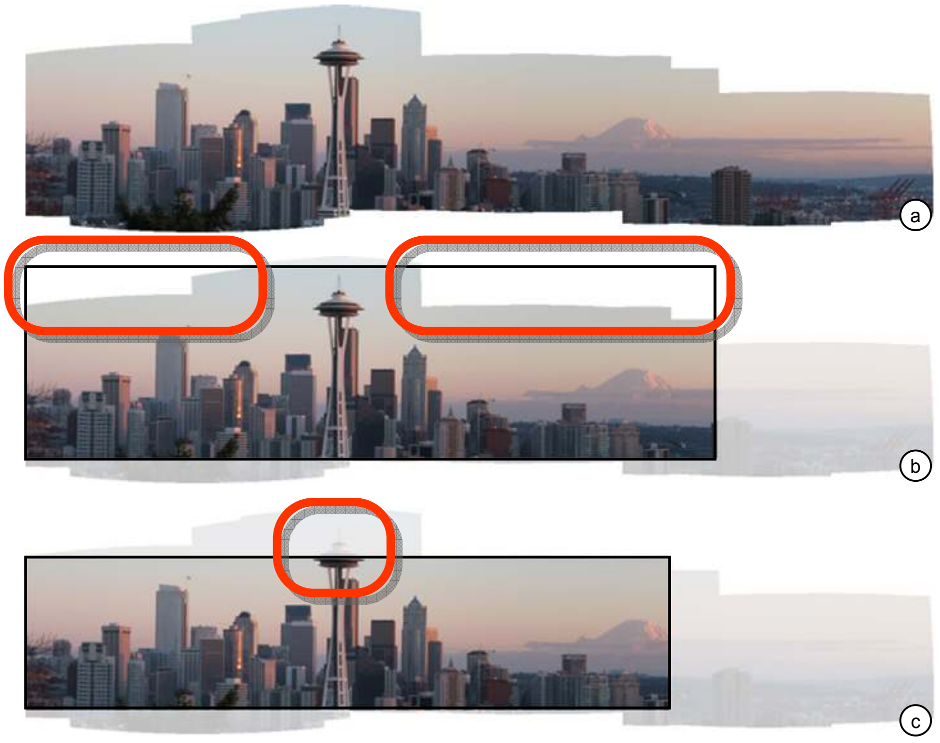


Figure 2: Common mishap when taking panoramas: (a) The user is trying to cover the Seattle Space Needle, but does not cover enough of the sky around it. During cropping, the user has the choice between (b) including white fringes and (c) cropping relevant content.

Fixing these flaws requires retaking images, but by the time users discover the errors during stitching, it is often too late.

While other researchers have looked at real-time previews generated using real-time stitching from a technical perspective (e.g. Peleg and Herman, 1997), we address the above issues by providing users with a novel user interface that allows them to utilize these previews in order to verify which areas of the scene have been successfully covered. The user interface of our system, called Panoramic Viewfinder, is designed to match the interaction model of existing digital cameras. In fact, we have constrained our designs to depend only a mode selector and a shutter button so that it may be ported to existing camera interaction mechanisms.

We begin with a walkthrough of the system. We then go over related work and describe the implementation of our prototype. We describe our experiences with this prototype and discuss the broader space of solutions. We also propose a second design, which alleviates some of the shortcomings of our current design. We conclude with a summary of our findings and a description of future work.

PANORAMIC VIEWFINDER

Panoramic Viewfinder is an interactive system for panorama construction that provides users with a real-time stitched preview of the panorama while shooting (Baudisch et al., 2005).

Figure 3 shows the user interface of our prototype system, here running on a Sony U50 device with an attached webcam. Panoramic Viewfinder offers three main user interface elements designed to help complete the desired panorama, i.e., (1) the preview shows the panorama in its current state of completion, (2) the viewfinder shows what the camera sees at this moment, integrated at the correct location into the panorama, and (3) the real-time cropping frame shows the extent the panorama will have after cropping if the user stops shooting at this instant.

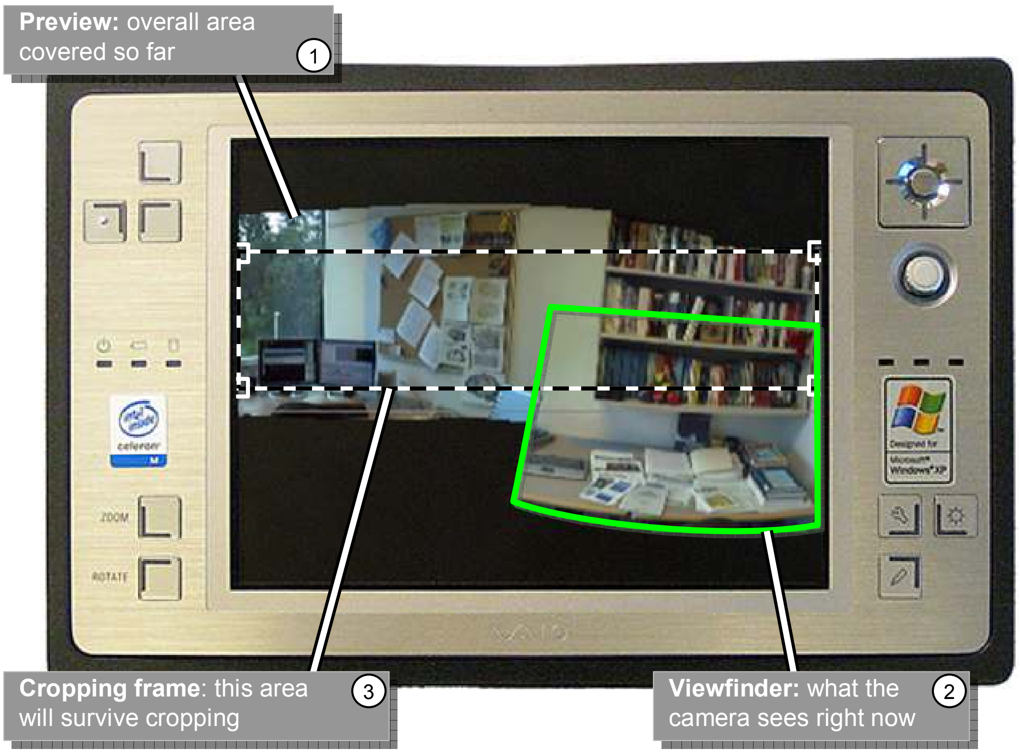

Preview: overall area covered so far
1
Cropping frame: this area will survive cropping
3
Viewfinder: what the camera sees right now
2

Figure 3: The Panoramic Viewfinder user interface on a Sony U50 ultra portable PC

Walkthrough

Figure 4 illustrates the user's interaction with the Panoramic Viewfinder using an example scenario. The user's task is to take a panoramic picture of the front of a building complex[16]. This example features our system's manual mode, in which the user manually triggers each photo. We explain the automated modes later in the paper.

1. The camera is ready, but the user has not triggered the shutter button. The LCD shows what the camera sees, just as in any other camera.

2. The user has pressed the shutter button and the camera has taken the first picture. From now on, the viewfinder is displayed in its relative location with respect to the panorama. This is made possible by continuously matching the viewfinder contents with the panorama (real-time stitching, see Implementation section).

3. The user pans the camera and takes additional pictures. Since these are automatically added to the panorama, the panorama grows. Whenever necessary, the panorama preview is automatically re-scaled to fit the screen. The cropping frame is also updated continuously, and it always represents the largest rectangle that can be inscribed into the current

[16] For a demo video of our system see http://www.patrickbaudisch.com/projects/panoramicviewfinder

panorama. Before stopping, the user checks the cropping frame and sees that stopping now would cause part of the building to be removed during cropping!

4. The user takes an additional shot of the top left corner. This completes the bounding box and the cropping frame now contains the entire building complex.

5. When uploading the photo to the PC, the panorama is restitched using our high-quality offline stitcher (Brown et al., 2005) to obtain maximum image quality. When cropping the panorama to the rectangular shape, all the desired content is preserved. The user already knows that this will be successful, because the cropping frame has indicated that all the required material has been taken. While we expect that most users will choose to keep the default cropping frame, as displayed in the preview, we do permit users to modify the final cropping if they so desire. In this way, they can perform non-standard (e.g. non-rectangular) crops. Whatever their final intent, they know from the preview that they have captured the desired material for their composition.

In summary, Panoramic Viewfinder allows users to take pictures by first shooting the desired scene elements, then filling the bounding box around these elements. One key element in this process is the real-time cropping frame, which serves as a visual reference. The real-time cropping frame inverts the traditional cropping model: traditional post-hoc cropping requires users to narrow the image down to what is desired; using the real-time cropping frame, users keep adding content until the cropping frame has grown to cover the desired area.

This approach has two main benefits. First, it avoids the need for providing the cropping frame with explicit controls, such as resize handles. This supports our goal of creating an interaction model that transfers seamlessly into a standard consumer camera, since operating a set of resize handles using the set of built-in buttons available on such cameras can be inefficient and frustrating. Second, growing the cropping frame rather than narrowing it down increases users' efficiency by helping them avoid taking undesired content in the first place.

While the default action during upload is to crop each picture automatically as previewed using the real-time cropping frame, users can always override that choice by accessing the uncropped panorama and cropping it manually with resize handles. This works well in the offline mode because users have access to a mouse rather than having to use the camera as their input device.

In addition to helping users take care of missing content, Panoramic Viewfinder can also help users detect ghosting, which typically becomes visible as an erroneous alignment between overlapping frames. Finally, Panoramic Viewfinder provides support for the third problem, i.e., failure to stitch, by immediately notifying users whenever such a failure occurs. We describe this in the following section.

Dealing with stitching failure

In order to place the viewfinder into the panorama preview, the system matches every new image received from the camera with the preview. In some situations, for example if the user moved to camera so quickly that the new frame cannot be matched, the system is unable to properly place the received image. If this happens, the viewfinder continues to update, but remains stationary and informs the user of the situation by changing the viewfinder frame from green to red and by playing a continuous error sound.

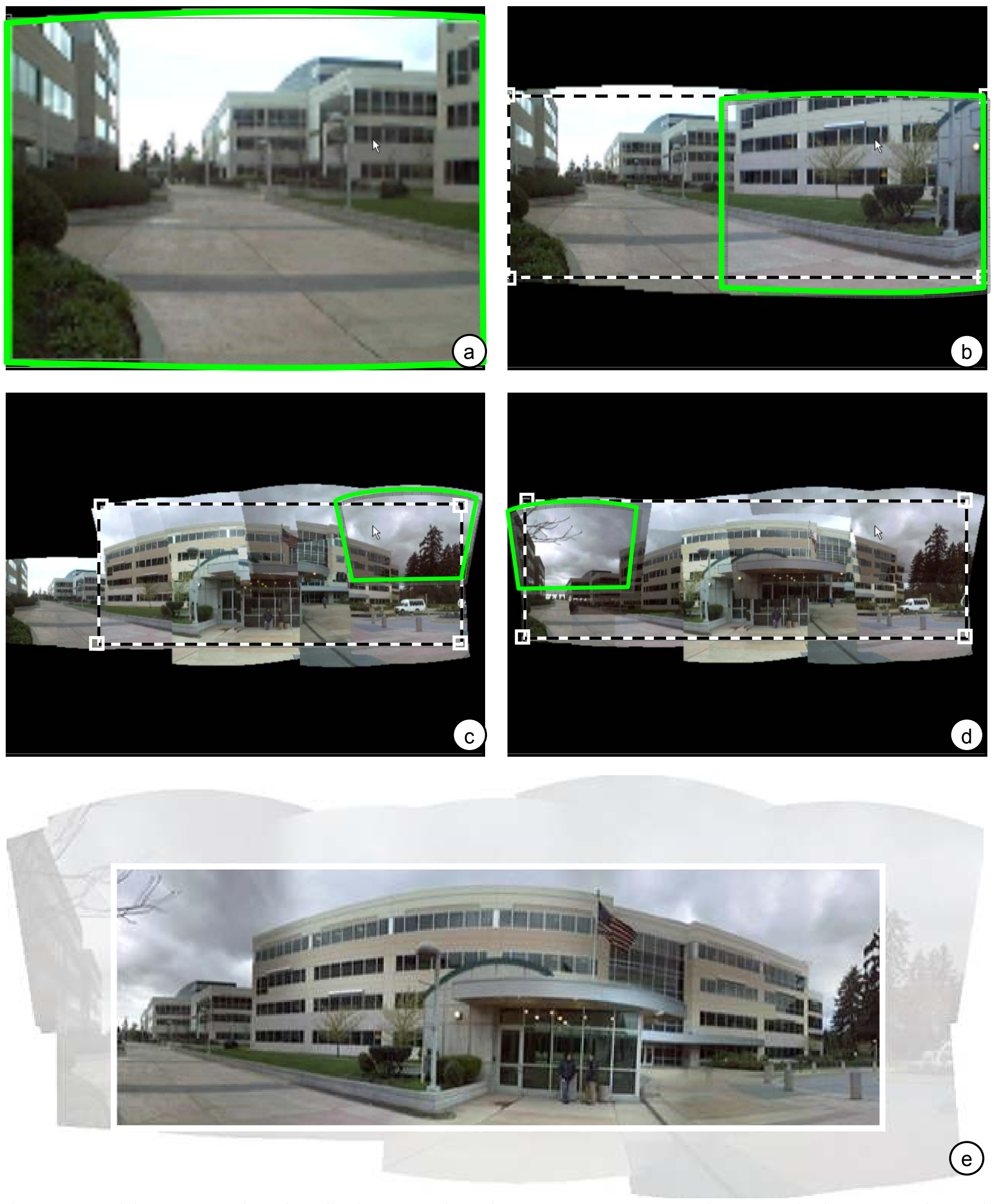


Figure 4: With Panoramic Viewfinder, panning the camera corresponds to a painting operation with a textured rectangular (actual screen captures with viewfinder and cropping frame emphasized to improve readability on b/w printouts)

To fix the problem, users may pan the camera and take an additional shot that offers greater overlap with the existing panoramic image. Alternatively, users may choose to zoom out to cover more of the image, and then zoom back in to fill in with higher resolution. As soon as the next match is found, the alert ends and the viewfinder continues to track within the preview.

In addition to alerting users, Panoramic Viewfinder also provides a mechanism for reducing stitching errors in the first place. We first experimented with a video mode, i.e., adding each and every frame to the panorama. The implementation of such a video mode was simple, as every new frame is stitched temporarily anyway in order to properly place the viewfinder. While video mode eliminated most stitching errors, the large number of stored frames affected the frame rate and increased registration error (Sawney et al., 1998). Additionally, in order to optimize the stitching, we only attempted to match the current frame with the previous one (and not all others in the preview) in this mode. This meant that if a user lost track of the stitched preview, they would have to return the viewfinder to the location of the last frame taken, rather than being able to move it back to an arbitrary part of the existing image. We therefore created an automatic snapshot mode. In this mode, the system does not add frames to the panoramic mosaic until the stitcher loses track. When it does, the system automatically adds the last frame that it was able to successfully register. It then retries stitching the current frame which will typically now work, due to the extra frame in between. Compared to video mode, automatic mode creates less data, is therefore faster, and leads to less accumulated registration error. It also allowed us to compare the current frame to the set of frames that constitute the entire panoramic picture.

RELATED WORK

Panoramic Viewfinder touches three areas of related work: stitching, user interfaces designed to support stitching, and user interface and interaction research on small screen devices.

We have already mentioned some approaches to stitching in the introduction. Panoramic Viewfinder uses a feature-based approach (Brown et al., 2003). Unlike image-based techniques (Szeliski 1996; Peleg and Herman, 1997; Szeliski and Shum 1997) that attempt to match bitmaps, feature-based techniques reduce the complexity of the matching problem by using only a few salient image features, such as corners. These algorithms also tend to be more robust to scene motion and parallax induced by camera translations.

The concept of real-time stitching was first explored by the VideoBrush system (Peleg and Herman 1997; Sawhney et al., 1998; Sawhney, Hsu, and Kumar, 1998). While VideoBrush offered most of the technical features of the real-time stitcher underlying Panoramic Viewfinder, it did not offer any particular user interface beyond displaying translated images on a computer screen. We believe that computing power embedded in mobile devices has now reached the point where we can begin to explore the user experience when such techniques are used in consumer electronics. In this paper, we propose a user interface concept to be added on top of real-time stitchers, such as VideoBrush, within the constraints of the interaction affordances in digital cameras. Novel features include the scaling preview, the real-time cropping frame, warning when losing track, and the automatic snapshot mode.

Some current off-the-shelf digital consumer cameras, such as the Canon Powershot S230 (www.powershot.com), offer a Stitch Assist mode. To help users line up frames and to obtain the right amount of overlap between frames, the LCD shows the current viewfinder next to part of the last frame taken (Figure 5a). In order to test the validity of the Stitch Assist concept for large panoramas, we implemented a 2D version of Stitch Assist (Figure 5b). By showing cropped previous frames in all four directions, our 2D Stitch Assist allows users to add pictures in any of the four main directions. While we found this to work well for smaller panoramas, informal observations revealed that for larger panoramas, users quickly lost track of the shots they had taken and spots within the desired panorama that remained to be taken. Additionally, since this scheme provides no feedback of the quality of shots, users sometimes ended up with shots that were not ideal for stitching (e.g. shots that were out of focus or contained motion blur).

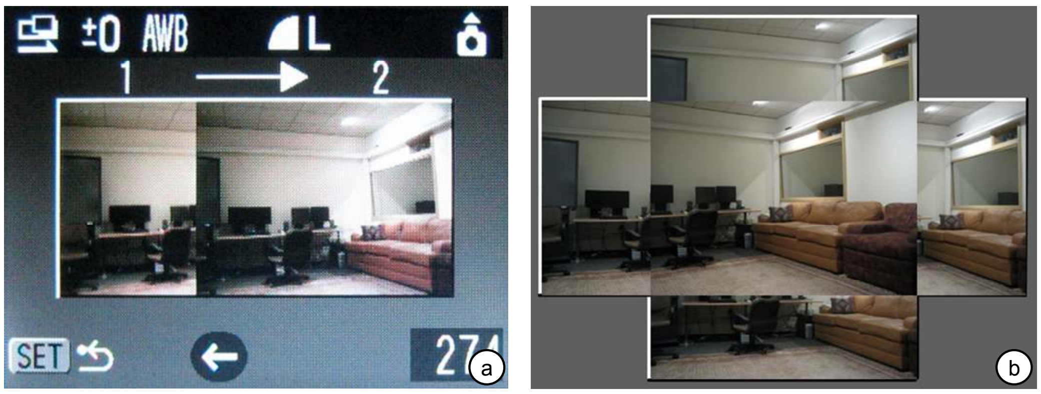


Figure 5: (a) The Stitch Assist mode, here from a Canon Powershot camera tells users where to take the next picture. The user pans the camera until the next picture on the right lines up with the piece of the previous picture on the left. (b) Our 2D adaptation of Stitch Assist allows users to continue a panoramic picture in any of the four main directions.

To address this latter problem, the HP Photosmart R707 (www.hp.com) allows users to switch to playback mode and review panoramas on the camera's LCD screen. Although this allows users to see flaws while still on-site, fixing flaws still requires retaking the entire panorama. In contrast to both these schemes, Panoramic Viewfinder offers users more freedom by allowing them to take content in random order by "sweeping" camera over the desired scene. It also provides immediate feedback about the quality of the panorama so that the user can interactively fix flaws.

The main interaction involved in using Panoramic Viewfinder, sweeping the camera over the desired scene to capture the relevant images, was inspired by various small-screen device interaction research projects. For example, early work by Rekimoto (1996) exploring how users could interact with the digital world by directly manipulating the device itself led to interaction techniques in which users could control menus and scroll bars, as well as browse maps and 3D objects, by tilting the device. Peep-hole displays (Yee, 2003) and the Chameleon system (Fitzmaurice, 1993) treat the screen as a small viewport into a much larger world and allow users to explore digital and physical spaces by moving the display around. However, because of the domain in which these systems were designed, neither addressed the issues surrounding stitching and presenting an arbitrary space or image formed by the display sweep. In a parallel line of work, Baudisch (1998) proposed paintable interfaces, which treat the cursor much like a paintbrush and allow users to manipulate large numbers of objects on screen simple by brushing over them. Panoramic Viewfinder uses the same painting or sweeping mechanism explored in all this work. Rather than using mechanical sensing means, as the other techniques did, we use vision-based techniques to formulate a contiguous view of the world that the camera has seen (the eventual panoramic image).

There are two views that must be shown to the Panoramic Viewfinder user: what the camera currently sees, as would be traditionally seen in the viewfinder; and the picture the camera has of the world it has seen, or the panorama at that point in time. Displaying these two views on the same small screen could be accomplished in a variety of ways (e.g. overview plus detail, modally switching between the two, etc). We chose to use the focus-in-context approach, which displays the focal data, in this case the viewfinder image, in-place within the overview data, the panorama. This class of techniques has been shown to be cognitively less demanding than approaches that require users to switch attention between multiple views (Baudisch et al., 2002).

IMPLEMENTATION

Our Panoramic Viewfinder prototype is written in C++ and runs under Windows XP. We have two implementations of the front end, one using the GDI+ graphics library and one based on DirectX (http://microsoft.com/windows/directx).

While we focus this paper on describing the interface innovation in this system, the technical details of our stitching engine can be found in (Steedly et al., 2005). The algorithm is loosely based on the traditional offline stitcher described in (Brown et al., 2005) and shares some libraries with it. Similar to the offline stitcher, the real-time stitcher in Panoramic Viewfinder performs stitching in four steps: 1) extracting Multi-scale Oriented Patches (MOPs) from each new image, 2) matching them with previous images, 3) estimating the orientation of the camera when the new image was taken, and 4) warping the new image onto the composite.

MOPs are 8×8 pixel patches in the image, extracted at Harris corner locations (Figure 6), which occur at the centre of patches in the image that are easily localizable in both image dimensions. This is in contrast to lines, which can only be localized in one dimension, and uniform colour patches, which are hard to localize in any dimension. The corner locations are extracted with sub-pixel accuracy at multiple resolutions or scales. Anywhere from fifty to several hundred features are extracted from a typical image.

After extracting MOPs in a new image, the features are compared to features from previously stitched images. The comparison is made efficient by using a three-dimensional hash table (10×10×10) to store the features. In order to ensure that features are well distributed in the hash table, we perform a Harr wavelet transform (Jain, 1989) on the 8×8 patches and the Harr coefficients are converted into a vector of length 64. The three hash table indices are the first three Harr transform coefficients of the image patch. The features from each new image are matched to the features from previous images using the hash table.

Only two feature matches are needed to estimate the position from which the new image was taken. In practice, many more than two matches are typically found, over-constraining the position. Hence, we use a robust least-squares optimization to estimate the new camera position.

Once we estimate the position of the new image, we display it on top of the composite of the previously captured images. If the user (or automatic snapshot algorithm) elects to keep the image, the MOP features are permanently added to the hash table and the new image is permanently added to the composite. If not, the features and image are discarded. The entire process is then repeated for the next incoming frame.

Panoramic Viewfinder is different from the offline stitcher in that it allows adding an image without reprocessing the images already stitched. To make this possible, Panoramic Viewfinder interleaves the four steps listed above. It finds the best location for each new image, but keeps all existing relationships fixed. While this leads to more error accumulation than the offline approach, which re-minimizes errors between all pairs of images, it allows new frames to be added in roughly constant rather than quadratic time.

Using the optimizations described above, our prototype reaches frame rates of up to 4 frames/sec on a 2GHz tablet PC with 1GB of RAM. We obtain slightly slower performance on an ultra portable 1 GHz Sony U50 (Figure 3).

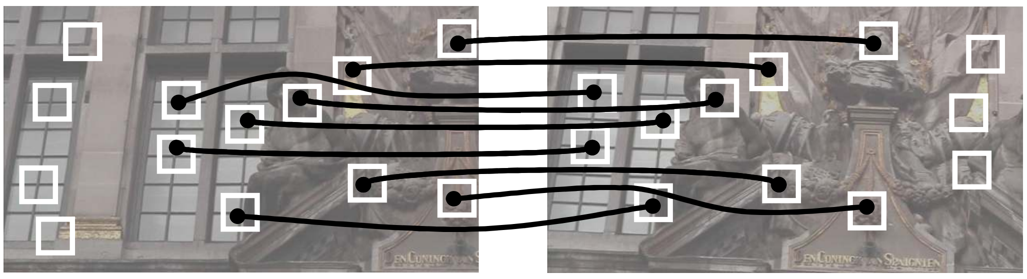


Figure 6: Feature Matching Between Two Images: The white boxes show the locations of MOPs in each image and the black lines show the correspondences that the feature matcher detects. The corner patches and correspondences were not extracted by our code and are only for illustration.

Computing the cropping frame in real-time

The user interface concept underlying Panoramic Viewfinder relies heavily on its ability to update the cropping frame in real-time. Here we describe our algorithm that makes this possible.
The real-time stitcher outputs the panorama as an alpha-matted composite bitmap image. In order to place the cropping frame, the system computes the largest interior rectangle that contains only fully opaque pixels. The problem of computing the largest inscribed rectangle in an arbitrary shaped polygon has been studied in the computational geometry community. However, existing algorithms (e.g. Alt et al., 1995) require the geometry to be described as a polygon, which does not cover cases involving interior transparent regions. Also, converting the rounded outlines of a panorama to a polygon can be computationally expensive. We therefore created an algorithm based on a pixel-based representation of the panorama.

At the highest level, our algorithm tests all possible combinations of top-left and bottom-right corners thereby testing all possible rectangles that can be inscribed within a given region. The algorithm obtains real-time performance by eliminating top-left corners of boxes with expected areas lower than a moving upper bound and by limiting the search for the bottom right corners to a linear number of possible candidates. By doing this, we obtain a complexity of roughly $O(n^2)$, with n being the width of the picture, which is suitable for real-time use. The detailed steps in this process are:

Step 1—down-sampling: For faster performance, our algorithm starts by creating a version of the panorama bitmap that is down-sampled by a factor of four. All further computation takes place on this down-sampled version.

Step 2—pre-compute span lengths: When computing rectangle surfaces, our algorithm uses information about the length of spans. Since the same span information is used repeatedly, it is cached as follows. The algorithm creates two arrays called width image and height image. For each pixel, width image stores the length of the horizontal span right of this pixel, height image stores the length of vertical span below this pixel. Both are created simultaneously by traversing the panorama from the last row to the first row and, for each of these rows, from the last column to the first one. Since each span value is generated based on its right or bottom neighbor, this step requires $O(n^2)$ operations.

Step 3—compute rectangle surfaces: Now the algorithm searches for the largest interior rectangle. It initializes the largest found rectangle to area 0. Then, it traverses the image top-down and left to right. At each pixel location (a) the algorithm computes an upper bound for the size of the largest possible rectangle whose top-left corner is at that pixel location. The upper bound is computed as the product of the horizontal span times the vertical span—both are looked up in the width and height images. (b) If the upper bound is smaller than the current best, this pixel cannot be the top-left corner of the sought rectangle and the algorithm moves on to the next pixel. (c) If the upper bound is larger than the current best, the algorithm computes the true largest rectangle with this particular top-left corner. In order to do so, it traverses down the current column of the width image, looking for the bottom left corner that results in the largest rectangle surface. During this traversal, it keeps track of the largest rectangle created by any bottom left corner so far. The algorithm also keeps track of the length of the shortest horizontal span seen so far in this column. The width of a rectangle with a given bottom left corner is the shortest span seen above. For each pixel in that column, the algorithm computes the surface of the rectangle that has the respective pixel as its bottom right corner as the product or that width times its distance from the top left corner. It keeps updating the best value until all pixels are traversed.

In the worst case, step 3 of our algorithm takes $O(n^3)$ operations. However, due its ability to terminate early, we find that in practice it is closer to $O(n^2)$ and is suitable for our real-time requirements. Interestingly, performance analysis shows that the algorithm spends 80% of its time in Step 2. As a future optimization we therefore plan to improve performance further by updating the width and height of images incrementally as new input images are added.

USAGE EXPERIENCE, LIMITATIONS, AND IDEAS FOR A REVISED DESIGN

We designed the interaction described above to minimize stitching flaws by providing feedback as early as possible. We had initially assumed that instant notification would allow users to fix problems immediately, thus minimizing the overall impact of errors.

After building and using Panoramic Viewfinder (Baudisch et al., 2005), we have realized that the conclusions we had initially drawn are not as straightforward as we had thought. We had expected a seamless shooting experience when shooting in auto mode, but while using our system to take panoramic pictures, we experienced more “lost track warnings” than we had initially expected. Most of these errors were caused by sweeping the camera too fast. Because the underlying stitcher requires a certain minimum overlap between consecutive frames, the overall speed at which users can sweep the camera without losing track is determined by the frame rate, which is only about 4 frames/second in our current version. The problem is not only that this “speed limit” limits how fast users can shoot panoramas, but also that staying below that speed limit requires additional attention. In fact, receiving this warning requires users to focus on the display, which often breaks the “flow” of the interaction.

While future code optimizations and increases in processing power might improve this situation by raising the speed limit, we contend that it is extremely difficult to completely eliminate all stitching failures. Examples of problems likely to persist include lack of texture or motion blur in low light situations, which makes stitching hard.

This brings up the question: does losing track need to break the flow? Or, should we avoid breaking the flow by allowing users to temporarily be in a “broken” state?

Similar problems and solutions have been discussed in other application domains, such as source code editing. Many initial editors were batch-oriented and used a "compile, view error list, edit, repeat"-cycle, which could be considered similar to working with an offline stitcher. To remedy this somewhat disjoint process flow, some second generation source code editors, such as the Microsoft Visual Basic editor began using a tight control loop. They continuously checked code syntax and prevented users from moving focus to a different line of source code until the current one was error free. However, enforcing this specific editing order on users made it hard for them to stay in the flow, which led to a series of usability issues. In retrospect, this evolution of solutions is analogous to that of the path we have already taken with Panoramic Viewfinder. To complete the analogy, more recent editors, such as Microsoft IntelliSense or Word have overcome some of these problems by using an "asynchronous" design. As the user is typing, squiggly underlines inform the user of issues that need to be addressed eventually. However, they do leave the control about order with the user and allow them to continue their train of thought.

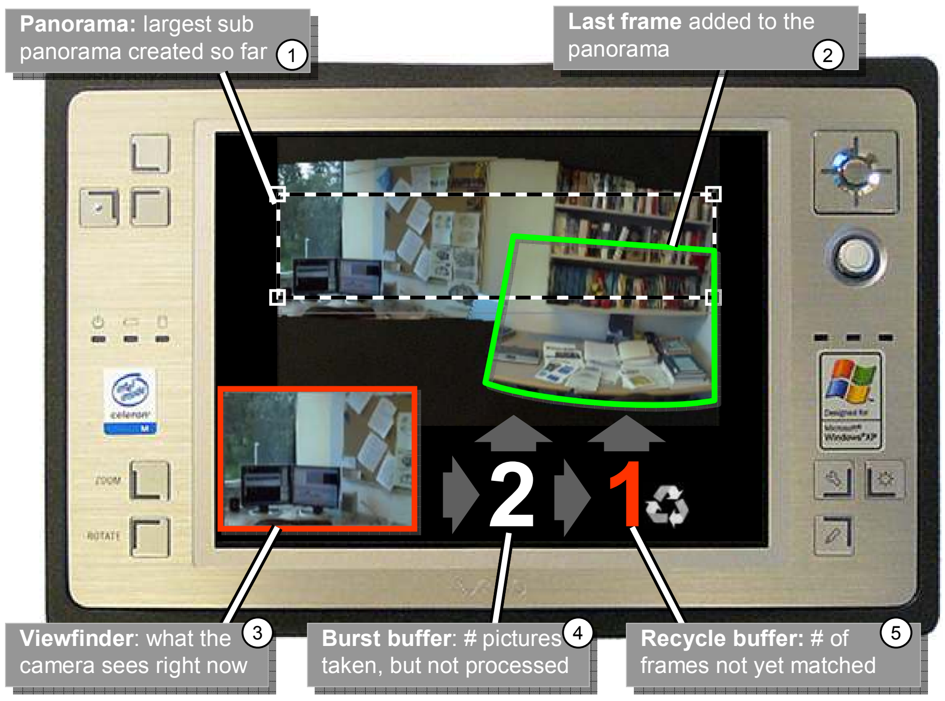
Based on these observations, we propose a modified design to Panoramic Viewfinder, one that relaxes the synchrony between stitcher and user. There are many possible designs to embody such an asynchronous user interface concept. Figure 7 shows a mock-up of one possible design. As an extension to the current instantiation of Panoramic Viewfinder, a separate viewfinder shows a live image coming from the camera (region 3). Whenever the shutter is manually triggered, a shot is taken and added to the burst buffer, as indicated by a little gray arrow (region 4). In this instantiation of our design, users can either choose to start in automatic snapshot mode while shots are able to be stitched into the panorama and fall back to manual mode once tracking is lost, or they can choose to just do everything manually. The burst buffer is similar to the burst buffer offered by many of today's digital cameras. It stores images without applying any processing, which allows shots to be taken at a very high rate. This process runs at the highest priority, which means that the other processes will be halted while the user is shooting to assure maximum shooting frame rate.

A fast stitcher, as it is implemented in panoramic viewfinder, takes images from the burst buffer and tries to stitch them to the panorama. This thread runs at medium priority. If fast stitching succeeds, the respective frame is added to the main panorama and the panorama preview (region 1) is updated accordingly. Note that the green frame inside the preview denotes the last successfully stitched frame, but unlike the previous version, this is now distinct from the view through the viewfinder. If fast stitching fails, the frame is moved to the recycle buffer (region 3).

As a third and optional thread running at lowest priority, a "quality stitcher" similar to the offline stitcher re-stitches all frames in preview, burst-, and recycle buffer. This stitcher is slower, but since it compares all sets of frames to each other, it may find additional matches. When it succeeds, the respective frames are added to the preview and removed from the respective buffers. The preview always shows the largest sub panorama. If running the second stitcher creates a sub-panorama bigger than the one currently displayed in the preview, the new sub-panorama will instead be displayed in the preview. A typical case where this will occur is when the very first shot of a panorama was blurred or otherwise unsuccessful. When the preview is switched, the previously shown sub-panorama goes back into the recycle buffer; its stitching information may be cached to speed up computation in case both sub-panoramas can be connected later.

The asynchronous Panoramic Viewfinder offers audio feedback different from the original version. As before, taking pictures results in a shutter speed sound. Instead of signaling stitching failure using a continuous error sound, however, the system now updates users on a per-event basis. Successful stitches (transition from burst buffer to preview) result in a friendly 'ping' sound, unsuccessful stitches (transition from burst buffer to recycle buffer) in a 'pop' sound, and successful recoveries (transition from recycle buffer to preview) in a "charm" sound that is repeated if multiple frames were recovered.

At the expense of offering a slightly more complex user experience, the asynchronous design allows users to shoot first and at any rate while fixing stitching failures on demand. The burst buffer within this design allows shooting at all times and at the highest possible rate. A typical usage would be for users to start a new panorama by shooting many shots very quickly. Stitching failure, in particular during this early stage, does not affect the user's ability to shoot, so stitching failures do not break the user's flow anymore.



Panorama: largest sub panorama created so far 1
Last frame added to the panorama 2
Viewfinder: what the camera sees right now 3
Burst buffer: # pictures taken, but not processed 4
Recycle buffer: # of frames not yet matched 5
2
1

Figure 7: Mock-up of an asynchronous panoramic viewfinder user interface

CONCLUSIONS

In this paper, we have presented Panoramic Viewfinder, an interactive system for panorama construction that offers a real-time preview of the panorama while shooting. Panoramic Viewfinder makes ghosting, missing content, and stitching failures visible, thereby allowing users to fix them by immediately retaking necessary pictures.

Additionally, showing users the result of their work right away offers "instant gratification", which we believe is one reason why many users today prefer digital photography to traditional photography. By automating the remaining steps in the process, i.e., high-quality restitching and auto cropping, Panoramic Viewfinder brings panoramic pictures one step closer to the user experience that users expect from regular digital photography, but which is not yet available with panoramic photography.

We have also discussed our usage experiences with Panoramic Viewfinder as well as usability issues and possible solutions to them. We propose a revised design that relaxes the forced synchrony between camera and user, and that we believe can further improve the user experience.

As future work, we are planning to optimize our algorithms to increase the frame rate and to reduce accumulated error. We also plan on running a user study evaluating user's performance and experience with Panoramic Viewfinder.

ACKNOWLEDGEMENTS

We would like to thank the members of the Interactive Visual Media Group and the Visualization and Interaction Group at Microsoft Research for their assistance and feedback on the work.

REFERENCES

Alt, H., Hsu, D., and Snoeyink, J. (1995). Computing the Largest Inscribed Isothetic Rectangle. Proceedings of 1995 Canadian Conference on Computational Geometry (CCCG), 67–72.

Baudisch, P. (1998). Don't Click, Paint! Using Toggle Maps to Manipulate Sets of Toggle Switches. Proceedings of 1998 ACM Symposium on User Interface and Software Technology (UIST), 65-66.

Baudisch, P., Good, N., Bellotti, V., and Schraedley, P. (2002). Keeping Things in Context: A Comparative Evaluation of Focus Plus Context Screens, Overviews, and Zooming. Proceedings of CHI 2002 Conference on Human Factors in Computing Systems, 259-266.

Baudisch, P., Tan, D., Steedly, D., Rudolph, E., Uyttendaele, M., Pal, C., and Szeliski, R. (2005). Panoramic Viewfinder: Providing a Real-time Preview to Help Users Avoid Flaws in Panoramic Pictures. In Proceedings of OZCHI 2005, ACM International Conference Proceedings Series, Canberra, Australia, November 2005

Bergen, J. R., Anandan, P., Hanna, K. J., and Hingorani, R. (1992). Hierarchical Model-based Motion Estimation. Second European Conference on Computer Vision (ECCV), 237–252.

Brown, M. and Lowe, D.G. (2003). Recognising Panoramas. Proceedings of the 9th International Conference on Computer Vision (ICCV), 1218-1225.

Brown, M., Szeliski, R., and Winder, S. (2005). Multi-image Matching Using Multi-scale Oriented Patches. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Volume I, 510-517.

Edwards, B. (1999). The New Drawing on the Right Side of the Brain. Jeremy Tarcher Publishing, Expanded Edition.

Fitzmaurice, G. (1993). Situated Information Spaces and Spatially Aware Palmtop Computers. Communications of the ACM, 36(7), 38–49.

Irani, M. and Anandan, P. (1998). Video Indexing Based on Mosaic Representations. IEEE Transactions on Pattern Analysis and machine Intelligence, 86(5), 905–921.

Jain, A.K. (1989). Fundamentals of Digital Image Processing. Englewood Cliffs, NJ: Prentice Hall.

Mann, S. and Picard, R. W. (1994). Virtual Bellows: Constructing High-quality Images from Video. Proceedings of First IEEE International Conference on Image Processing (ICIP-94), 363–367.

Peleg, S., and Herman, J. Panoramic Mosaics by manifold projection. (1997). Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR), 338.

Rekimoto, J. Tilting Operations for Small Screen Interfaces. Proceedings of 1996 ACM Symposium on User Interface and Software Technology (UIST), 167–168.

Sawhney, H.S., Kumar, R., Gendel, G., Bergen, J., Dixon, D., Paragano, V. (1998). VideoBrush™: Experiences with Consumer Video Mosaicing. Proceedings of IEEE Workshop on Applications of Computer Vision (WACV), 56–62.

Sawhney, H.S., Hsu, S., and Kumar, R. Robust Video Mosaicing through Topology Inference and Local to Global Alignment. (1998). In Proc. European Conf. On Computer Vision, 103-119.

Steedly, D., Pal, C., and Szeliski, R. Efficiently Registering Video into Panoramic Mosaics. Proc. Tenth International Conference on Computer Vision (ICCV 2005), Volume II, 1300-1307.

Szeliski, R. (1996) Video Mosaics for Virtual Environments. IEEE Computer Graphics and Applications, 16(2), 22–30.

Szeliski, R., and Shum, H. (1997). Creating Full View Panoramic Image Mosaics and Environment Maps. Proceedings of SIGGRAPH 1997 31st International Conference on Computer Graphics and interactive Techniques, 251–258.

Teodosio, L. and Bender, W. (1993). Salient Video Stills: Content and Context Preserved. Proceedings of ACM Multimedia 1993, 39–46.

Yee, K.P. (2003). Peephole Displays: Pen Interaction on Spatially Aware Handheld Computers. Proceedings of CHI 2003 Conference on Human Factors in Computing Systems, 1-8.