Ztitch: A Mobile Phone Application for Immersive Panorama Creation, Navigation, and Social Sharing

Andrew Au, and Jie Liang
School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada
{aau1, jiel}@sfu.ca

Abstract—This paper presents the design of Ztitch, a mobile phone application that can create immersive panoramic scenes in real time, where multiple photos are arranged in the 3D space according to their camera poses, maintaining a realistic perspective of the scene. By taking full advantage of the touchscreen, accelerometer, and gyroscope in the phone, Ztitch allows users to easily fine-tune each photo’s position, navigate the scene, and edit the scenes created by others. When there is an undesired gap or overlap between the two ends of a 360° immersive panorama due to unknown camera focal length or accumulated errors, a fast and accurate algorithm is developed to jointly adjust the entire sequence to achieve the desired panorama. This is again achieved by leveraging the touchscreen of the phone. In addition, a fast color-balancing and exposure-compensation technique is developed to blend neighboring images.

Once the panorama is created, users can smoothly navigate the scene via touchscreen or accelerometer. The latter allows the user to explore by simply tilting the phone instead of dragging the photos. Using the touchscreen, users can also fine-tune or add photos to panoramas created by others, giving the application a strong social network flavor. Users can also share the scenes with others via the Ztitch website and various social networks.

A typical problem when creating a 360° immersive panorama is that there could be a gap or too much overlap between the first and the last photos, due to unknown camera focal length or accumulated errors. Two techniques were developed in the past to address this problem. The first one can estimate the focal length for pure panning motion and cylindrical images [3], but it only works when the camera is continuously turning in the same horizontal direction. Therefore it cannot be used in many user-generated panoramas with relatively arbitrary camera poses. Another approach is to use the bundle adjustment method [4], which simultaneously aligns all the images under a least-square framework to correctly distribute the mis-registration errors. While it works for arbitrary camera motions, this method is time-consuming, especially when running on the phones.

In Ztitch, a user-aided algorithm is developed, where instead of using automatic but computationally demanding algorithm to estimate and adjust the focal length and the scene radius, we allow the user to drag one of the boundary photo on the touchscreen, and move it to the desired position with respect to the other boundary photo. The radius of the panorama is updated based on the amount of the dragging, and all other images will be adjusted automatically, as if they are glued together. The method is very fast and accurate. It is also more flexible than the method in [3].

Another common problem for image stitching is that different images could have different brightness. In Ztitch, a fast and efficient color-balancing and exposure-compensation technique is also developed to blend neighboring images.

Ztitch was partly motivated by the desktop application PhotoSynth developed by Microsoft [5], which uses various computer vision algorithms such as bundle adjustment to recreate the 3D scenes from multiple photos [6]. However, these algorithms are time-consuming, and are difficult to be ported to mobile phones. As a result, the mobile PhotoSynth was not released until April 2011, six months after Ztitch, and it is still only available on iPhone/iPad. The mobile
PhotoSynth for Windows Phone still has not been released due to hardware limitations.

Compared to Photosynth, Ztitch adopts a user-aided approach by taking full advantage of the touchscreen and other sensors, rather than relying solely on the computer vision algorithms and the processing power of the phones. This allows it to achieve similar results and arrive at the market earlier. Ztitch shows that mobile application development is different from desktop development. By exploiting the rich sensors of the phones, the constraints imposed by the limited computational power of the phone can be circumvented, and sometimes with even better user experience.

Ztitch also enjoys some advantages over PhotoSynth. For example, PhotoSynth only outputs the final panorama as a whole. The result is irreversible with no way to manually readjust the position of each individual image; it does not have the gap filling feature; and it does not have the sensor-based navigation. Moreover, images in Photosynth are captured in video mode, leading to low resolution and quality.

Ztitch has been downloaded over 80,000 times and is one of the top photo apps on Windows Phone. The source code is available at the website [1]. This paper describes the design and implementation of the application.

II. IMMERSIVE PANORAMA CREATION

In this section, we describe the detailed algorithms for immersive scene creation, editing, and photo seamline blending.

A. Coordinate Pipeline and Spherical Coordinate

A key step in the immersive scene creation and navigation is to apply various perspective projection transforms to a photo. Homogeneous coordinate is usually used in for this purpose. Our system uses the pinhole camera model. The corresponding graphics pipeline workflow consists of a sequence of transformations. Each transformation is represented by a $4 \times 4$ matrix. Specifically, a 3D point is first augmented to its projective equivalent $s$, which then undergoes the following matrix multiplications that yields a 2D Euclidean point $p$ in the output device coordinates [7]:

$$
M_V \cdot C_P \cdot T_v \cdot M_W s \rightarrow p,
$$

where $M_W$ is the positioning transformation, $T_v$ the viewing transformation, $C_P$ the perspective projection transformation, and $M_V$ the viewport transformation [7]. Finally, the right-hand side of the equation is projected as a 2D point.

Ztitch runs on the Windows Phone operating system, and is developed using the Microsoft Silverlight framework, which provides the necessary APIs to implement various 3D projective transforms to images/videos. For example, the Matrix3D structure defines the $4 \times 4$ matrix required by various 3D perspective projections. The Matrix3DProjection class makes it easier to apply arbitrary transformations to Silverlight elements, such as translate, scale, rotate, and perspective transforms. However, significant efforts are still needed in order to create user-friendly 3D and immersive multimedia applications from these basic APIs.

In this paper, we only consider panoramic scenes, so all photos will be placed on the surface of a sphere. The spherical coordinate is thus very handy, which maps the center of each photo to a point $(x, y, z)$ on a sphere with radius $r$ using the following equations, as shown in Fig. 1,

$$
x = r \sin \theta_y \cos \theta_x,
$$

$$
y = r \cos \theta_y,
$$

$$
z = r \sin \theta_y \sin \theta_x,
$$

where $\theta_x$ is the horizontal angle of the photo in radians, and $\theta_y$ is the vertical angle of the photo. An additional tangent rotation $\theta_T$ is also needed in Ztitch, which denotes the rotation of the image in the plane tangent to the sphere at $(x, y, z)$.

The radius $r$ is directly related to the focal length of the camera. In our system $r$ is in units of pixels. Usually for a digital camera, the focal length can be extracted from the EXIF tags in the original photo file. The focal length also varies a lot between different devices. For example, the Nokia Lumia 800 phone uses a camera with focal length of 28 mm. The corresponding value of $r$ is 530. The camera focal length of the Samsung Focus phone is 4 mm, with $r = 680$. All other Windows Phone 7 devices we tested have $r$ between these two values. However, some manufacturers do not disclose the camera focal length. An unknown focal length could lead to an undesired gap or overlap when creating panoramas. In Sec. II-C, we present a fast method to resolve this problem.

B. Immersive Panorama Creation and Editing

The earlier Windows Phone 7.0 did not allow taking photo in the app. To create panorama scenes using Ztitch, users need to take all photos first, then bring them into Ztitch. Using the touchscreen, users can drag each photo to its desired location in the 3D space. To avoid running time-consuming automatic image matching algorithms, each photo is displayed in a semi-transparent mode. This allows the user to visually identify the best overlapped position between neighboring photos. Although bundle adjustment and feature-based algorithms can be used to align the images automatically, these methods are computationally expensive on the phones. Besides, sometimes these algorithms do not work well, due to, e.g., large exposure differences between images, fast moving objects, and lack
of distinctive features. The user-aided approach in Ztitch is intuitive and the result is quite satisfactory. It effectively walks around the limitation imposed by the hardware.

With the latest Windows Phone 7.5 Mango, photos can be taken directly in the app. Therefore, the latest Ztitch allows the users to rotate the camera, take new pictures at constant rotation intervals, and add them to the scene directly. All photos are still displayed in semi-transparent mode to help the user identifying the best position to take the next photo.

After the scene is created, Ztitch still allows the user to manually drag each image using the touchscreen to fine-tune its position. The coordinate of the center of the photo is updated during the dragging. That is, when the user drags the photo horizontally or vertically, we adjust $\theta_x$ or $\theta_y$ in Fig. 1. The tangent rotation $\theta_T$ is obtained by a traditional button-based user interface, since this action is not frequently used.

Let $\theta_x(t-1)$ be the photo’s horizontal angle when the photo was refreshed last time, and let $\Delta_x(t-1)$ be the $x$-axis change of the user’s finger position on the touchscreen from the last refreshing time, the photo’s new X angle will be

$$\theta_x(t) = \theta_x(t-1) + \frac{\alpha}{f} \Delta_x(t-1),$$

where $\alpha$ is the field of view of the camera, and $f$ is the focal length. The photo’s vertical angle $\theta_Y$ can be updated similarly.

We then obtain the rotations $R_x(t)$ and $R_y(t)$ from Eq. (4) and (5), and their combination $R_x(t)R_y(t)$ is applied to matrix $M_V$ in Eq. (1). The new location of the photo can be calculated accordingly using the spherical coordinate.

$$R_x(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x(t) & \sin\theta_x(t) & 0 \\ 0 & -\sin\theta_x(t) & \cos\theta_x(t) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(t) = \begin{bmatrix} \cos\theta_y(t) & 0 & \sin\theta_y(t) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y(t) & 0 & \cos\theta_y(t) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$
this technique works only for pure panning motion and when the images are perfectly orthogonal to the ground (θy = 90°), which is difficult to achieve in practice.

In Ztitch, we develop an efficient user-aided algorithm to solve this problem. Our solution is more flexible than [3], because our method works even if the first image and the last image have different vertical positions, and when the images do not have constant inter-frame rotations.

Consider a panoramic sequence of N images, where neighboring images are well aligned, but there is a gap or overlap between the first and the last image that has both horizontal and vertical components, as shown in Fig. 2 (a). Let θix, θiy, and θi denote the rotations that define the location of the i-th image on the sphere with radius radius r, as in Fig. 1. We use two angles θgi and θgy to describe the undesired gap or overlap, i.e., if Point A at (r, θxA(A), θyA(A)) is in the first image, and its matching point B in the last image is at (r, θxB(B), θyB(B)), then the gap or overlap is defined as

θgi = θxB(B) − θxA(A),
θgy = θyB(B) − θyA(A). 

Suppose the values of the two angles above have been measured by a method, which will be described in the end of this section. In order to eliminate the gap or overlap, we adjust the rotations of all images uniformly, i.e., the updated orientation of each image will be:

θi′x = θix + θgi/N,
θi′y = θiy + θgy/N, 

Note that since the vertical position of each image is changed, to maintain the alignment between neighboring images, the last line in Eq. (7) is used to slightly adjust the tangent angle of each image. This empirical formula works well if the 360° scene contains sufficient images (N ≥ 10), which is usually the case.

We also need to update the radius, for which the following empirical formula is used:

r′ = r + Cθgi, 

where C is a constant dependent on several factors. We found that by setting C = 126, we can get good results even when r needs to be updated by 90 pixels, which is the worst case (the default radius is 620. The Nokia Lumia 800 has the smallest radius of 530, and the Samsung Focus has the largest radius of 680).

With this method, the gap or overlap can be eliminated satisfactorily while all the images in the sequence would remain relatively aligned with each other.

As described above, the key to the method is to measure θgi and θgy, the values of the gap or the overlap. This could be achieved by running featured-based alignment algorithms to calculate the amount of misregistration between the first and the last images. However, this increases the complexity of the algorithm, which means the user has to wait during the computation, leading to unpleasant user experience. In addition, these algorithms do not always work.

In Ztitch, a user-aided approach is used, where the user can manually drag one of the boundary images towards the other on the touchscreen, and visually identifies the best alignment position. This operation is intuitive, fast, and accurate because both images are displayed semi-transparently. The dragging can include both horizontal and vertical components; hence our method is more flexible. The required gap/overlap values can thus be easily measured, and all other images and the radius will be jointly updated according to (7) and (8).

Fig. 3 shows the result after manually closing the gap of the scene in Fig. 2. As can be seen, the gap can be satisfactorily closed, and all other images remain perfectly aligned.

D. Photo Seamline Blending

In photo panoramas, after stitching photos, a seamline is usually visible at the intersection of two neighboring photos, due to their different exposures and/or white-balance levels, as shown in the example of Fig. 4 (a).

For applications on iPhone/iPad such as Photosynth, developers have an ample amount of controls over the camera’s functionalities through the provided APIs, such as locking the exposure and white-balance when the user takes different pictures in the scene. This allows the panorama to have a more homogeneous appearance. However, it also creates another problem, as some images could be too bright or too dark, especially for 360° scene taken in sunny days.

Windows Phone 7 does not provide such APIs to the developers, and the exposure and white-balance are handled automatically by the camera. To produce a smoother looking final result, a color correction method has to be used. The challenge is that it should have low complexity in order to run on mobile phones.

Various algorithms have been developed in the past to blend images together and create a more seamless result. One of the most successful algorithms is the Laplacian pyramid blending (or multi-band blending) [8]. First, it decomposes each source image into its own Laplacian pyramid, consisting of high, medium, and low frequency parts. It then tries to smoothly blend the low-frequency variations (i.e. color), but quickly blends the high-frequency variations (i.e. texture). While it has good result, the method is very slow. A more recent method is developed in [9], but it requires knowing the exposure value and uses high dynamic range images.

In this paper, an efficient algorithm is developed. Assume that the sequence of N images is arranged from left to right, and the first image is numbered i = 0, then our algorithm works as follows:

Step 1: Find the region of overlap between an image pair. For a pair of images projected on a cylinder that are set apart horizontally by θix, we can approximate the amount of overlap O (in pixels) using O ≈ w − θixr, where w is the width of the image and r is the radius of the cylinder. In the ith image, extract the region with width O from the right end. Denote it
Fig. 4. (a) Exposure and white balance differences among the three images create an unpleasant result. (b) Using our method to balance out the color of each pixel, and achieve smooth transition between images.

Step 2: Since RGB is readily available from the image, we can compare $R_{right}$ and $R_{left}$ in terms of their RGB components and spread the difference across the image pair. To do this, we first calculate the sum of the pixel values in each region for the red, green, and blue channels. Then, the difference between these sums is calculated.

$$D^i_k = \left( \sum_{x=0}^{X} \sum_{y=0}^{Y} R_{right,k}(x,y) - \sum_{x=0}^{X} \sum_{y=0}^{Y} R_{left,k}(x,y) \right) / XY$$

(9)

where $k = \{R, G, B\}$ represents the color channel, $(x, y)$ is the pixel location over width $X$ and height $Y$. Then for all pixels in the $i^{th}$ image, add $\sigma_{i,k} D_k^i$ to the corresponding channel $k$. Similarly for all pixels in the next image, add $(\sigma_{i,k} - 1) D_k^i$. The weight $\sigma_{i,k}$ is initialized to 0.5, but is adjusted in Step 5. This weighting function dictates how much of the difference should be spread to one image, and how much should be spread to the other image.

Step 3: What we have done in the previous step is augment the pixels by addition. Visually, this can lead to a contrast difference between the image pair. We compensate for this by applying a contrast factor, $f^i$. First, we normalize $D^i_k$ between 0 and 1, then calculate:

$$f^i = \sqrt{1 + (D^i_R + D^i_G + D^i_B)/3}.$$  

(10)

For all pixels in the $i^{th}$ image, we then adjust their contrast by $\sigma_{i,k} f^i$ in channel $k$. Similarly, the contrast in the next image is adjusted by $(\sigma_{i,k} - 1) f^i$. Adjusting the contrast of a pixel is achieved by transforming its range to $[-128, 127]$, multiplying by the contrast factor, then transforming the range back to $[0, 255]$, with necessary clip.

Step 4: Increment $i$ and repeat steps 1-3 for the next image pair.

Step 5: Repeat steps 1-4 multiple times for all image pairs in the sequence, including the pair of the first and last image. We find that doing it $N/2$ times is sufficient. In the first loop, all the weights $\sigma$ were initialized to 0.5. When we run a new loop, we calculate new weights using $\sigma_{i,k} \propto D_k^{i+2}/D_k^i$ for each color channel.

Step 6: Images in Silverlight are in the ARGB format, where A is the alpha or transparency channel, and the rest are regular RGB channels. Therefore, we can adjust transparency of each pixel, and achieve smooth transition between images by applying a linear gradient to each image. This gradient becomes transparent as it nears the edge of the photo. To do this, we take the region $R_{right}$ and decrease its alpha channel over its width $X$. Similarly, this is applied to $R_{left}^{i+1}$ in the opposite direction. This linear gradient is also used in the vertical direction.

The final result of our algorithm is shown in Fig. 4 (b), which has significant improvement over Fig. 4 (a). For 6 images, it took about 5.5 seconds to run our algorithm on a single-core Samsung Focus S phone. Compared to this, it takes about 9.0 seconds to run the multi-band blending technique in the AutoStitch application [10] on an iPad 2, which has a dual-core processor. This time is measured by comparing how long it takes to render the final panorama with multi-band blending enabled versus disabled.

### III. Interactive Navigation of Immersive Panoramas

Zitch allows easy navigation of the immersive scene from the phone, using the touchscreen or the sensors. This includes moving to different viewing directions, zooming in and zooming out. When users navigate the scene via the touchscreen, we detect the finger motion, and calculate the corresponding changes of the rotation angles, and update the viewing transform $T_v$. This matrix then transforms the world coordinates system into the coordinate system of the user’s view (the pinhole). After applying the viewing transform, the coordinates of the photos is relative to the user’s view. By multiplying this viewing matrix by a rotation matrix and then plugging back to the pipeline Eq. (1), all the photos in the scene would appear to be rotated.

Our system can also use the accelerometer in the phone to allow the users to navigate a scene by tilting the phone in different directions, giving the application a gaming flavour. When the phone is tilted in a particular direction, the view...
of the scene will move in that direction, and the amount of tilt angle controls the speed of the movement. This creates a video effect from the static images.

The integration of the accelerometer is achieved by multiplying the viewing transform $T_v$ in Eq. (1) with rotation matrices $R_x(t)$ and $R_y(t)$, which represent the rotation matrices for the horizontal axis and vertical axis, respectively. $R_x(t)$ and $R_y(t)$ depend on the roll and pitch values given by the accelerometer, respectively. $T_v$ must be updated at every display refreshing time, and reference values for the roll and pitch must be set when the user activates the accelerometer.

We also improve the rendering performance of our application by not rendering photos that are out of view from the user (outside the viewport). Therefore, only a subset of the collection is shown at any moment, reducing the amount of rendering that the processor needs to do. This is done by running a loop to calculate the dot product of the vector normal to $T_v$ (the 3rd row of $T_v$) and that of each photo in the scene, then taking the arc-cosine of this dot product to get an angle. If this angle is greater than the field of view, we will skip the rendering of that photo.

When we redraw a frame, we also need to consider the pipeline in Eq. (1). We must go through the pipeline again whenever at least one of its matrices is updated. Three scenarios will cause this to happen: when the user drags a photo, navigates the scene, or changes the field of view (a function similar to zooming). In the last case, the perspective projection matrix, $C$, gets updated.

IV. ONLINE SHARING OF THE PANORAMAS

In our system, a scene is defined by two components: (1) all photos contained in the scene, and (2) a small configuration file, whose size is about a few hundred bytes. To allow other people to access to the scenes created by our users, we develop an accompanying Ztitch website, which is also based on Silverlight. Therefore this website can be accessed by any computers and browsers that support Silverlight, including Internet Explorer, Firefox, and even Safari on Mac. Ztitch allows users to share the link of a scene on our website to other people via email and Facebook.

When the users upload their immersive scenes to our website, each individual photo is actually uploaded separately to their Flickr accounts. All photos can be found by searching “Ztitch” on Flickr, and clicking the link in the description of each photo will direct the user to our web application showing the complete immersive scene.

In our website, only a small configuration file about each scene is stored, which contains the URL of each uploaded photo, as well as the $4 \times 4$ matrices that define the final locations of all photos in the scene. It also stores metadata information such as the geo-coordinates of the scene, and the Facebook or Flickr ID of the user. The images will only be retrieved from Flickr when someone wants to view them as a single stitched scene from our website.

Through the Ztitch website or the Ztitch app on the phone, users can browse the scenes created by other users. On the phone, users can also download and edit a scene. For example, adding a new photo to a scene to improve its coverage. This enables community-built immersive panoramas, and gives Ztitch a strong social network flavor.

V. CONCLUSION AND FUTURE WORK

This paper presented a mobile application that can easily create, navigate, and share immersive panoramic sceneries. It achieves good performance without using time-consuming algorithms, by taking full advantage of the hardware features of the phones. The application can be further improved. For example, we plan to generalize it to support non-panoramic scenes. The blending algorithm can be improved for better performance and quality. The manual stitching can be refined when the users upload their photos to our server, where we can run the state-of-the-art automatic stitching algorithms.

ACKNOWLEDGEMENT

This work was supported in part by Nokia and by NSERC Strategic Project Grant 380875.

REFERENCES

[1] www.ztitch.com