Multimedia Tools and Applications, February 2015, Volume 74, Issue 3, pp 1067-1089. The final publication is available at http://link.springer.com/article/10.1007/s11042-013-1711-6.

SpriteCam: Virtual Camera Control using Sprite

Yi Chen and Ramazan S. Aygün

Computer Science Department, University of Alabama in Huntsville

Huntsville, AL 35899 U.S.A.

Email: {yichen, raygun}@cs.uah.edu

Corresponding Author: Ramazan S. Aygun

Phone: (256) 8246455

Fax: (256) 8246239

Abstract

Editing or browsing video based on spatial content without distortion or cropping has been challenging because of providing unavailable information (view) in a single frame. Virtual camera control enables the user to view the video from her perspective by using camera functions such as pan, tilt, and zoom on a recorded video. In this paper, we propose our SpriteCam system that provides virtual camera controls by first generating the background sprite or mosaic of the video. We provide the theoretical framework and then explain how pan, tilt, and zoom operations are applied using the sprite. SpriteCam allows centralizing objects, aspect ratio conversion, and fixing the camera view without distortion or information loss for videos which sprite can be generated.

Keywords: Virtual camera control, sprite generation, video editing

1 Introduction

Since the introduction of digital videos in late 1980s, video technology has experienced a lot of improvements with respect to capturing devices such as cameras, camcorders, and smartphones. With the availability of good video compression standards, digital videos can be stored and transferred in less bandwidth with preserved quality by ordinary people. However, nowadays, users are not satisfied with only watching and saving videos. Instead, they desire to walk through, edit, and reproduce the videos. Some video editing software such as Premiere [1] and PowerDirector [2] already allows users to walk through the video and provide some editing techniques. These techniques normally include video playback, rearrangement of video clips, splitting video and deleting specific frame or frames, and aspect ratio change conversion from 4:3 to 16:9. These techniques are limited to information included in a single frame. They are not able to utilize spatially correlated information among the frames of a video. Nevertheless, users may expect to visualize extra information that cannot be covered by purely browsing the sequential frame but could be achieved by controlling the (virtual) camera of a video. Virtual camera control based on the overlapping information content among frames can help the users walk through the videos from their perspective and reproduce the videos.

1.1 Virtual Camera Control

In today's world, managing cameras and providing the best view to users are very challenging especially when the scene includes a set of moving objects that the cameraperson has to capture properly. In addition to basic camera functions like pan/tilt/zoom, the cameras are allowed to have six-degrees of freedom. However, it is still very challenging to provide the best view. Basic virtual camera control may provide camera controls such as cropping or digital zoom by just using the current frame in a video. However, it is still possible that the viewers might be interested in additional views that could be available in other frames of a video. In the past, spatial browsing or a virtual walkthrough [3] has been achieved by mapping the real-world to a virtual 3D world. The scene is continuously regenerated by using computer graphics methods during these interactive walks. However, this does not give the feeling of real video experience. Virtual camera control enables post-processing of already captured video. Virtual camera controls combine zoom, pan, and tilt operations to retrieve the interesting region by also using the scenes from other frames in a video. If a video has some camera motion, then virtual camera control can benefit from it.

1.2 Sample Applications

Let's consider two cases for virtual camera control. In the first case, the object in a video frame is not centralized and the user would like to have the object to be centralized. In the second case, the video was captured in a video with 4:3 aspect ratio, and the user would like to convert it to 16:9 aspect ratio. In both cases, the user does not want to sacrifice resolution, remove unnecessary parts from the scene, or see a distorted video.

For the first case, we would like to give an example from the 'Stefan' sequence. For example, moving the camera of a specific frame is especially important in deteriorated scene such as shown in Figure 1. Figure

1(a) presents a sample frame from Stefan video where the player is on the boundary. Figure 1(b) shows the segmentation mask of corresponding frame in Figure 1(a). The user may rather expect the player to be centralized as indicated by the red rectangle in Figure 1(c). The black rectangle in Figure 1(c) indicates the original frame boundaries. However, the shaded area is missing in the original frame. Therefore, it is not possible to present the player in the center of a frame to the user without distortion, loss of resolution, or scene removal.

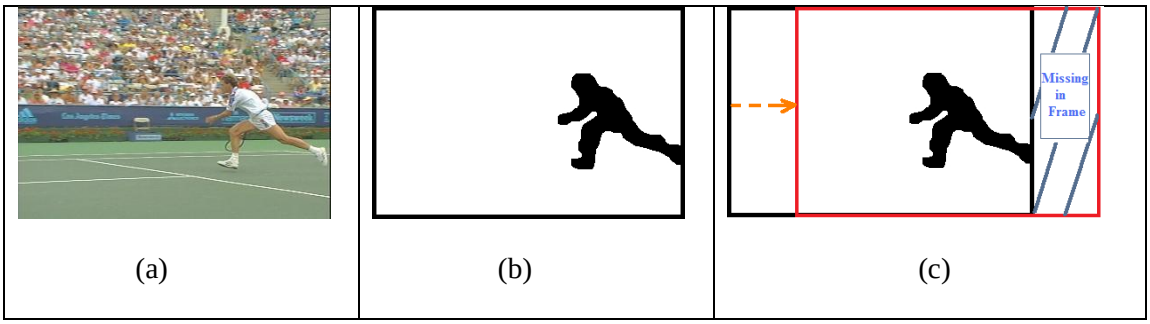

Figure 1. (a) Sample frame in Stefan video with player (Stefan) on the boundary, (b) corresponding object mask, and (c) centralizing object in the middle.

Similar case also applies to aspect ratio conversion. Without distortion or information loss, it is not possible to change the aspect ratio. Traditional methods convert aspect ratio by stretching (this introduces distortion), cropping (loss of information) or pillar box (this has missing information at the borders).

These two cases can be solved if we could extract missing regions from other frames in the video. We should also note that we plan to provide camera controls similar to actual camera controls. There could be many variations of what to present to the user based on the application.

1.3 Our Approach

In this paper, we propose our SpriteCam system that provides virtual camera controls on a video. We generate a complete view of the scene by processing all frames in the sequence. There is similar research such as photo stitching, mosaic generation, sprite generation, and panorama generation. We use the term sprite generation for generating the complete view of an object by processing frames that contain parts of the object. Sprite is a complete view of an object that is obtained by processing frames that contain parts of the object. If sprite is generated for the background, it is called as background sprite. Since background sprite is generated by correctly overlapping the frame sequence, it contains spatial relationships between frames. The term ‘mosaic’ [8] is also used for ‘background sprite’. Background sprite is generated by the process that is called as sprite generation. In this paper, sprite refers to background sprite unless stated otherwise.

The accuracy and quality of the sprite are key factors for virtual camera control. The traditional sprite generation methods introduce error due to motion models that are used, and blur the sprite. In this paper, we are going to use our sprite fusion technique [17] which is more appropriate for this type of applications where

the subjective visual quality of the sprite is important. After generating the sprite using the sprite fusion, we align original frames on the sprite. This automatically reveals the problems with sprite generation if any. We explain how virtual controls are applied based on the global motion between consecutive frames by providing the theoretical framework. We utilize intrinsic camera parameters and the fundamental matrix to explain how virtual controls are applied. We provide examples on a variety of videos.

This paper is organized as follows. The following section discusses the related work. Section 3 provides information on the sprite fusion technique and its application for virtual camera controls. Section 4 discusses virtual camera controls. Section 5 provides information on applications of SpriteCam. The experiments are explained in Section 6. The last section concludes our paper.

2 Related work

Basic virtual camera control enables walk-through by regenerating frames by cropping the regions of user interest from a high-resolution or panoramic video. This region usually named as Region of Interest (ROI) in the literature can be retrieved from a larger view of the environment or background sprite (panoramic image) of a video. In [4], Guo et al. created a system to crop ROI automatically for small display with maximum information by considering physical locations and semantic meaning of object regions. The virtual control system proposed by [5] identifies ROI in a high-resolution video and displays ROI by cropping from the high-resolution video. In both [4] and [5], the proposed systems are limited to work on the current frame without benefiting from other frames. The system proposed in [6] crops ROI from the panoramic image that is generated by stitching images from multiple stationary cameras. In [7], Aygun et al. propose a virtual camera control for video for users to crop ROI from background sprite (or mosaic) after generating the sprite. Our proposed approach also uses information from other frames in the scene using the background sprite.

Most sprite generation methods are composed of three steps: global motion estimation (GME) ([9-14]), warping, and blending. Global motion estimation calculates the transformed coordinates between the sprite and original frame by minimizing the sum of MSE (mean squared error), warping the frames using calculated global motion parameters; and then blending to determine the pixel values on the sprite after merging with the frame. The quality of sprite is not important for video compression but critical if it is presented directly to the user. There is a lot of work to increase the accuracy of global motion estimation and there is some work aiming to increase the accuracy of sprite during the blending step by reducing blurring caused by inaccurate segmentation masks [15]. In [15], inaccurate segmentation mask for every frame is divided into three regions including reliable (background) regions, unreliable (object) regions and undefined regions between background and objects. Glantz et al. [23] provide automatic object segmentation by using the local background sprites to avoid segmentation preprocessing step. Bahoumi et al. [24] and Kunter et al. [25] propose that using a single sprite for a sequence may produce a distorted sprite and propose using multiple sprites for a video. In [16], Aygun developed a high-resolution mosaic (sprite) to reduce blurring in the sprite by introducing a histemporal filter for blending. In [14], spatial correlation is used to further increase the

accuracy since the most probable values for a region may not actually be spatially correlated. Reducing blurring is not an easy task since object removal adds to the complexity of sprite generation. In [17], Chen et al. developed sprite fusion method to generate sprite without segmentation for videos that track an object or a group of objects.

To perform proper aspect ratio conversion without distortion is also not easy to achieve. Traditional methods including expanding, cropping and adding black bands ([18-20]) are not able to resize the frame without distortion or information loss. In [18], Hsia et al. convert to higher aspect ratio by cropping the center part of the frame and loses the information at the borders. In [20], Guo et al. use a saliency model to perform aspect ratio conversion as they consider the human attention into account. They only use the current available frame. In [21], Avaidan et al. propose seam carving technique for resizing images based on the energy of pixels. Since it is not applied on video, their goal is to expand or compand the image by minimal distortion.

3 Video Editing using Sprite Fusion

3.1 Adapting Sprite Fusion

Since a video is very likely to include objects, the most general blending approach in sprite generation is to aggregate aligned pixel values from frames using averaging and to eliminate pixels that yield error over a threshold over a long time. These eliminated pixels are assumed to belong to moving objects, and the sprite is expected not to be blurred with correct global motion if objects are eliminated. However, these long-term aggregation methods are not able to completely eliminate the moving objects (especially if objects have repeating patterns) even with correct global motion estimation, and therefore, produce inaccurate sprites. This problem can be solved by considering object segmentation mask during blending. However, obtaining correct object segmentation is hard and computation-intensive despite several existing algorithms [23].

After considering the camera motion and object influence, Abhidnya et al. [22] classified the videos for sprite generation into six classes. The *complex* class is the class of videos that are difficult to generate sprite. The complex class can be further classified into *tracking* and *crowded* classes. In the *crowded* class, there may be at least two inconsistent object motions entering the field of view. Our sprite fusion method in [17] is targeted at tracking videos. Moving objects in tracking videos do not appear at the border of frames and the video covers a scene that does not intersect with the first frame of the video. Fusion technique is able to generate a high quality sprite by fusing two types of sprites: conservative sprite and assertive sprite. In *conservative sprite*, the pixel on the sprite will not be updated as new pixels from a frame map onto existing areas. Thus, conservative sprite will always have the first frame. In *assertive sprite* generation, a pixel on the sprite will always be replaced with the most current value. Thus, assertive sprite will always have the last frame. In [17], either fusing conservative sprite on assertive sprite or vice versa is able to generate an object-free high quality sprite. In our experiments, the clarity of sprite fusion is more acceptable than the traditional sprite since the blurring almost does not exist [17].

3.2 Playing the Original Frames on the Sprite

Traditionally, the quality of sprite is measured by comparing original frames with regenerated frames from the sprite. However, this does not reveal all the problems with sprite generation. If original frames are placed on the sprite, a user can see a) whether the frame is aligned properly with respect to the sprite and b) compare the blurring in the sprite with respect to the clarity in the frame. So overlaying the frame enables us to view whether the frames are aligned properly or not by checking the continuity at the borders of frames.

3.2.1 Locating Original Frames on the Sprite

2D motion models in general provide acceptable good results. 2D motion can be determined by the following transformation matrix:

$$T = \begin{bmatrix} m_2 & m_3 & m_0 \\ m_4 & m_5 & m_1 \\ m_6 & m_7 & 1 \end{bmatrix} \qquad \text{(Eq. 1)}$$

For mapping the points in the original frame onto the sprite, we also adapt this 8-parameter matrix. m_2 and m_5 are scale parameters. m_2, m_3, m_4, and m_5 may be related with the rotation angle. m_0 and m_1 are related to the horizontal and vertical displacement. Since the affine camera motion model usually works fine for most digital videos, m_6 and m_7 are set to 0. We use the matrix in Eq. 1 to describe the camera motion between two sequential frames. The calculated camera motion between neighboring frames can be described as: $M_{(0, 1)}$, $M_{(1, 2)}$, $M_{(2, 3)}$, …, $M_{(n, n+1)}$ and so on. Here $M_{(i,i+1)}$ indicates the transformation matrix to map pixels in frame *(i+1)* to pixels in frame *I* as provided in Eq. 2 as follows:

$$M_{(i,i+1)} \times \begin{bmatrix} X_{i+1} \\ Y_{i+1} \\ 1 \end{bmatrix} = \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \qquad \text{(Eq. 2)}$$

Then the relation between frame *n* and the first frame can be described as $M_{(0, n)} = M_{(0, 1)}M_{(1, 2)}M_{(2, 3)}\ldots M_{(n-1, n)}$ using homography:

$$M_{(0,n)} \times \begin{bmatrix} X_n \\ Y_n \\ 1 \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ 1 \end{bmatrix} \qquad \text{(Eq. 3)}$$

Since this can be easily proved, we skip the proof.

A typical camera has intrinsic parameters that include the focal length, a coefficient to map pixels to distance, distortion, skew coefficient between x-axis and y-axis, and the principal point (u_x,u_y). The principal point usually corresponds to the center of the image plane and it is critical for rotation and zoom operations.

Therefore, for each frame *i* a vector, V_i, of 14 elements is maintained. Table I lists a sample vector for frame 19 that contains motion parameters ($M_{(0,19)}$) for the Stefan sequence.

Table I. Sample vector for frame 19 of Stefan video

i	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	u_x	u_y	s_x	s_y	MSE
19	-45.78	1.2017	0.8417	0.0214	0.0073	0.8417	0	0	88	73	0	1	150.2089

The parameters in this sequence from left to right are *i* (frame number=19), m_0 (horizontal displacement since the first frame:-45.78), m_1 (vertical displacement since the first frame: 1.2017), m_2 (scale/rotate/affine parameter: 0.8417), m_3 (scale/rotate/affine parameter: 0.0214), m_4 (scale/rotate/affine parameter: 0.0073), m_5 (scale/rotate/affine parameter: 0.8417), m_6 (perspective motion parameter: 0), m_7 (perspective motion parameter:0), the center of the first frame or the principal point (u_x=88, u_y=73), s_x (shift of pixels along x-axis: 0), s_y (shift of pixels along y axis: 1), and MSE (mean squared error: 150.2089). Note that we also maintain two parameters related to shifting. As the sprite grows to left or top, the sprite (hence, the principal point) is shifted (right or down) to accommodate new pixels.

Now we show how to use these global motion parameters to align the original frame on the sprite. The relative global motion between any two frames frame *i* and frame *k* can be described as:

$$M_{(0,i)} \times M_{i,k} = M_{0,k} \qquad \text{(Eq. 4)}$$

Thus, $M_{(i,k)} = M_{0,i}^{-1} \times M_{0,k}$.

If we want to regenerate frame *n+1* based on the coordinates of the first frame on the sprite, we need to calculate it as:

$$M_{(0,n+1)}^{-1} \times \begin{bmatrix} X_0 \\ Y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} X_{n+1} \\ Y_{n+1} \\ 1 \end{bmatrix} \qquad \text{(Eq. 5)}$$

If we figure out the mapping parameters (between the first frame and the current frame) and the region of the first frame on the sprite, then we could figure out the mapping parameters between the current frame and the sprite. In Figure 2, the location of the current frame (shaded) region can be expressed with respect to the first frame as in Eq. 5. Since we use a compact sprite for maintaining optimal space, the location of the first frame changes on the sprite as more regions are added to the left or upper side of the sprite. Figure 4 shows where the first frame is (in terms of coordinates) at *T=0* and *T=last frame*.

Let $\begin{bmatrix} X_n \\ Y_n \\ 1 \end{bmatrix}$ be the vector for a pixel coordinate on frame *n*. Its value can be computed from the sprite (or the first frame) as:

$$\begin{bmatrix} X_n \\ Y_n \\ 1 \end{bmatrix} = M_{(0,n)}^{-1} \times \begin{bmatrix} X_{sprite} \\ Y_{sprite} \\ 1 \end{bmatrix} + \begin{bmatrix} \sum_{i=0}^{lastframe} s_x(i) \\ \sum_{i=0}^{lastframe} s_y(i) \\ 0 \end{bmatrix} \qquad \text{(Eq. 6)}$$

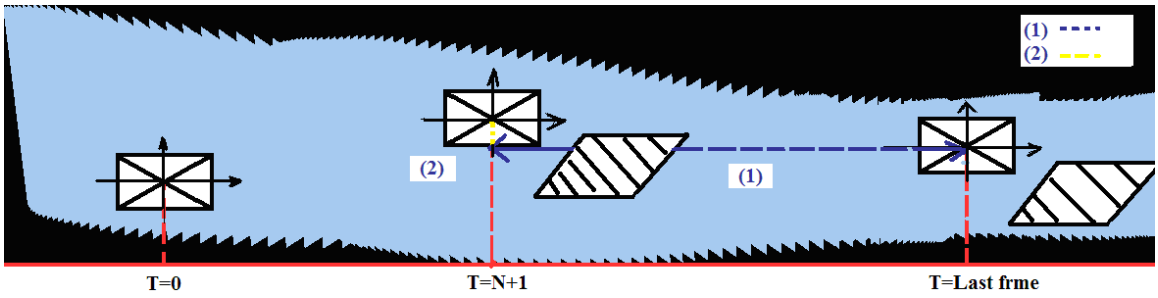

Figure 2. The spatial relation between current frame and sprite

Note that the second term related to shifts is needed to compensate the translation of the first frame's center on the sprite. (X_{sprite}, Y_{sprite}) is equivalent to (X_0,Y_0) since all frames are mapped to plane of frame *0*, but they may not appear within the boundaries of the first frame.

3.2.2 Blending Frames on the Sprite

After we apply the transformation matrix on the original frame, we retrieve the warped region. Using Eq. (6), all the points in the area of the current frame can be located in the sprite. We use four coordinates of the current frame to decide the corresponding transformed region. Let *w* and *h* represent the width and height of a frame, respectively. The center of the frame corresponds to the origin, and the four coordinates of the area are **A** *(-w/2, h/2)*, **B** *(w/2, h/2)*, **C** *(-w/2,- h/2)*, and **D** *(w/2,- h/2)*.

The inverse of matrix $M_{(0,n)}$ can be expressed as:

$$M_{(0,n)}^{-1} = \frac{1}{m_2(m_5-m_1m_7)-m_3(m_4-m_1m_6)+m_0(m_4m_7-m_5m_6)} \begin{bmatrix} m_5-m_1m_7 & m_0m_7-m_3 & m_3m_1-m_0m_5 \\ m_1m_6-m_4 & m_2-m_0m_6 & m_0m_4-m_1m_2 \\ m_4m_7-m_5m_6 & m_3m_6-m_2m_7 & m_2m_5-m_3m_4 \end{bmatrix} \quad \text{(Eq. 7)}$$

In the affine motion model, Eq. 7 can be simplified as

$$M_{(0,n)}^{-1} = \frac{1}{m_2m_5-m_3m_4} \begin{bmatrix} m_5 & -m_3 & m_3m_1-m_0m_5 \\ -m_4 & m_2 & m_0m_4-m_1m_2 \\ 0 & 0 & m_2m_5-m_3m_4 \end{bmatrix} \quad \text{(Eq. 8)}$$

Using Eq. 7 or Eq. 8, we find the minimum and maximum x and y coordinates among these four coordinates. In other words, we find the region A'B'C'D' in Figure 3. We scan all the points such as *(a, b)* in region A'B'C'D' and map to the points *(x', y')* in region ABCD using:

$$\begin{cases} x' = m_2a + m_3b + m_0 \\ y' = m_4a + m_5b + m_1 \end{cases} \quad \text{(Eq. 9)}$$

Figure 5 represents the mapping from region ABCD to A'B'C'D' on the sprite.

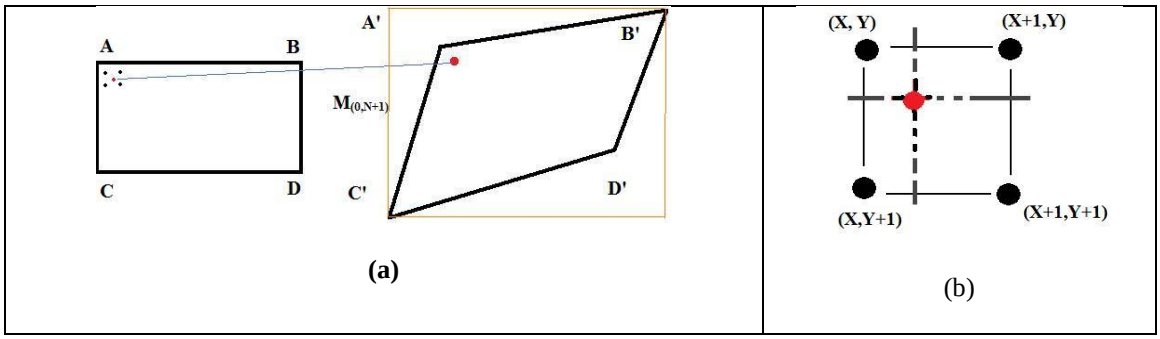

Figure 3. Mapping original frame on the sprite (a) region mapping and (b) pixel mapping

Figure 3 (a) represents the shape transformation from ABCD to quadrangle A'B'C'D'. If a pixel at location (a, b) maps to a location as the red point inside the polygon, it is likely to map a non-integer coordinate as shown in Figure 3 (b) rather than integer values. We apply bilinear interpolation method. The value of point (a, b) can be expressed as follows:

$$I_{(a,b)} = (1 - x' + \lfloor x' \rfloor)(1 - y' + \lfloor y' \rfloor) * S_{(x,y)} + (x' - \lfloor x' \rfloor)(1 - y' + \lfloor y' \rfloor) * S_{(x,y+1)} + (x' - \lfloor x' \rfloor)(y' - \lfloor y' \rfloor) * S_{(x+1,y+1)} + (1 - x' + \lfloor x' \rfloor)(y' - \lfloor y' \rfloor) * S_{(x+1,y)} \quad \text{(Eq. 10)}$$

where $S_{(x,y)}$ represents the pixel value at (x,y) on the sprite. Figure 4 shows blending the original frame on sprite using bilinear interpolation.

Figure 4. Positioning an original frame on sprite for Stefan video using bilinear interpolation

3.2.3 Preprocessing to Remove Boundaries

Video frames may have black band around boundaries which are actually not part of the video. To correctly display the original frame on the sprite, we need to crop the frame properly and automatically. Figure 1(a) shows the problem for Stefan video. For Stefan video, the frame has 2-pixel-width black band for image top and image right. Let $\mu_{(*,i)}$ and $\mu_{(i,*)}$ represent the average pixel value for the i^{th} row and the i^{th} column, respectively. In the same way, let $\sigma_{(*,i)}$ and $\mu_{(i,*)}$ represent the standard deviation for the i^{th} row and the i^{th} column, respectively. To correctly identify the boundary region, we use two measures to separate the content region and black band region. These two measures are based on two assumptions:

- If the two neighboring rows or columns belong to the boundary band, the sum of intensity difference should be less than 10 (i.e., $|\mu_{(i,*)} - \mu_{(i+1,*)}| < 10$ or $|\mu_{(*,j)} - \mu_{(*,j+1)}| < 10$ where $0 \leq i < h - 1$ and $0 \leq j < w - 1$).
- If the two neighboring rows or columns belong to the boundary band, the deviation of intensity difference should be less than 5 (i.e., $|\sigma_{(i,*)} - \sigma_{(i+1,*)}| < 5$ or $|\sigma_{(*,j)} - \sigma_{(*,j+1)}| < 5$ where $0 \leq i < h - 1$ and $0 \leq j < w - 1$).

Both measures are based on the assumption that the intensity and intensity variation of neighboring rows or columns will be very close if both belong to the black band region.

3.2.4 Handling Boundaries on the Sprite

While selecting regions from the sprite, it is possible that some areas may belong to areas outside the actual sprite. In order not to crop the parts outside the sprite, we need to consider which actions may cause this. Figure 5 shows the sample image planes after virtual camera controls leading movement of planes (quadrangles) on the sprite: (1) tilt-up, (2) tilt-down, (3) pan-left, (4) pan-right, (5) zoom-out, and (6) rotation. Since zoom-in will crop from the content of the original frame, it is not shown here.

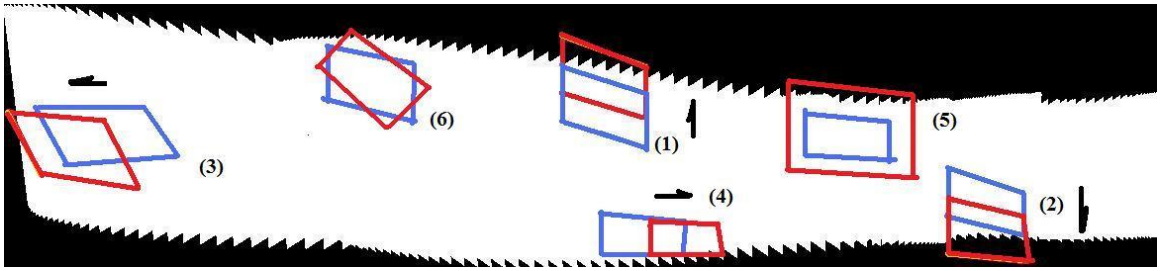

Figure 5. Intersections caused by movement on the sprite

We may determine whether a plane (quadrangle) intersects outside region by using the fact that there must be at least one intersection on the boundary line of quadrangle that belongs to the outside region. In other words, there must be at least one point that should be marked as 0 on one of the boundaries for the quadrangle if the outside region is coded with 1 and the sprite region is coded with 0.

4 Virtual Camera Control in SpriteCam

In this section, we provide information about how we apply camera motion using the sprite and motion parameters.

4.1 Intrinsic Camera Parameters

A camera model has intrinsic and extrinsic parameter sets. Intrinsic parameters are related to the manufacturing of a camera and is not expected to change. The intrinsic parameter set is represented with the following matrix:

$$K = \begin{bmatrix} \alpha_x & \gamma & u_x \\ 0 & \alpha_y & u_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 11)}$$

where α_x and α_y are dependent on the focal point and the size of pixels; u_x and u_y correspond to the principal point coordinates; and γ corresponds to the distortion (skew coefficient) between x and y axis of the camera. The parameters α_x and α_y depend on the focal point f and can be represented as $\alpha_x = f.m_x$ and $\alpha_y = f.m_y$ where m_x and m_y correspond to the ratio of mapping distance to pixels. Typical cameras allow users to change the focal depth of a camera by zoom-in and zoom-out operations. The coordinate (u_x,u_y) corresponds to the center of the image plane. If the camera is manufactured with precision, u_x=*width/2* and u_y=*height/2*. The γ factor is normally expected to be 0.

4.2 Extrinsic Camera Parameters

Extrinsic camera parameters correspond to the movement of a camera in a 3D coordinate system. In most cases, the camera is fixed on a platform and there is no change along the z-axis (or depth). This is a typical case for pan-tilt-zoom (PTZ) cameras. With six degrees of freedom, a camera may move (or be translated) in each axis and rotate around each axis (yaw, pitch, roll). A typical camera system allows all these three rotations, but may restrict the translations. A PTZ camera has two rotations, and the zoom factor is modified by changing the focal point. The extrinsic parameter set can be represented as a multiplication of the following matrices:

$$K_{ext} = R \times T \qquad \text{(Eq. 12)}$$

where R is a sequence of rotations around each axis: $R = R_\theta \times R_\varphi \times R_\omega$. To avoid the confusion between the order of rotation operations, R rotation matrix is often represented as:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \text{(Eq. 13)}$$

4.3 Anatomy of Virtual Camera Control

Before explaining how we implement virtual camera control, we would like to provide information about how the sprite is generated based on camera operations. Figure 6 displays how sprite is generated with respect to a variety of operations. There are three image planes in the figure. The image plane in the middle corresponds to the image plane where the sprite is mapped. The image plane on the lower-right shows the plane corresponding to the translation-left and zoom-out. The arrow corresponds to the mapping of new scenes after this operation. The image plane at the upper-right shows the plane after rotation and zoom-in. Again, the arrow shows how the new scenes are mapped on the sprite. As a result an elongated central image plane is generated, and this corresponds to the sprite.

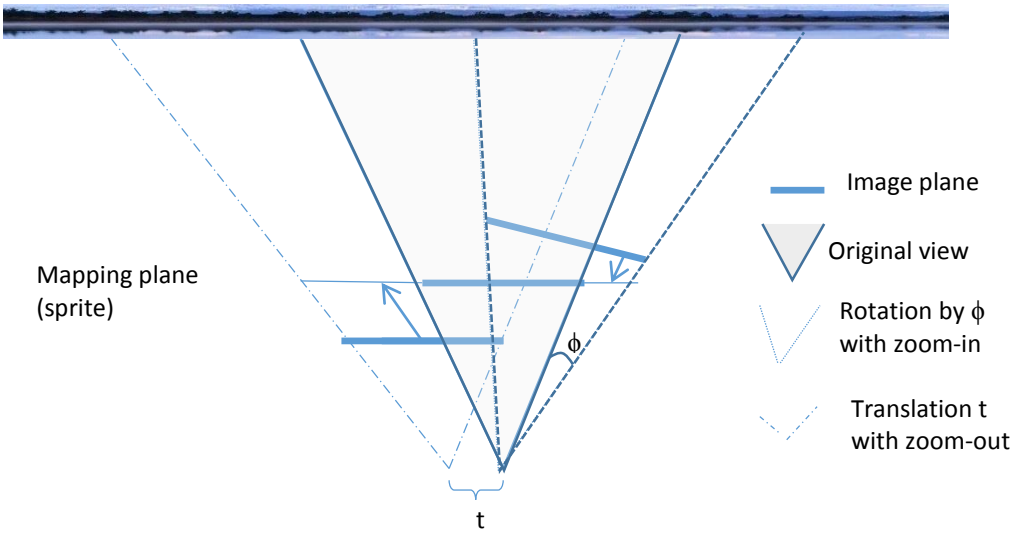

Figure 6. Representation of camera operations.

Virtual camera control actually corresponds to finding the proper image plane of the camera operation. After finding the image plane, the pixels on the sprite are mapped onto this image plane. Figure 7 shows how the image plane is determined for the translation. Figure 8 displays how zoom-in and zoom-out operations are handled with respect to image planes. The change of image planes is clear for zoom operations. The next section provides mathematical overview of how image pixels are mapped to these image planes.

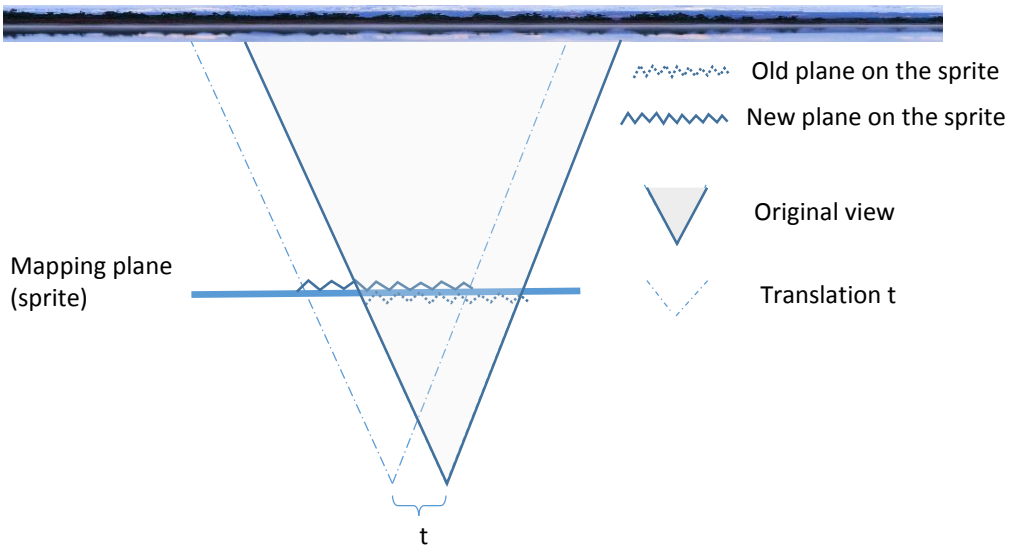

Figure 7. Representation of translational operation.

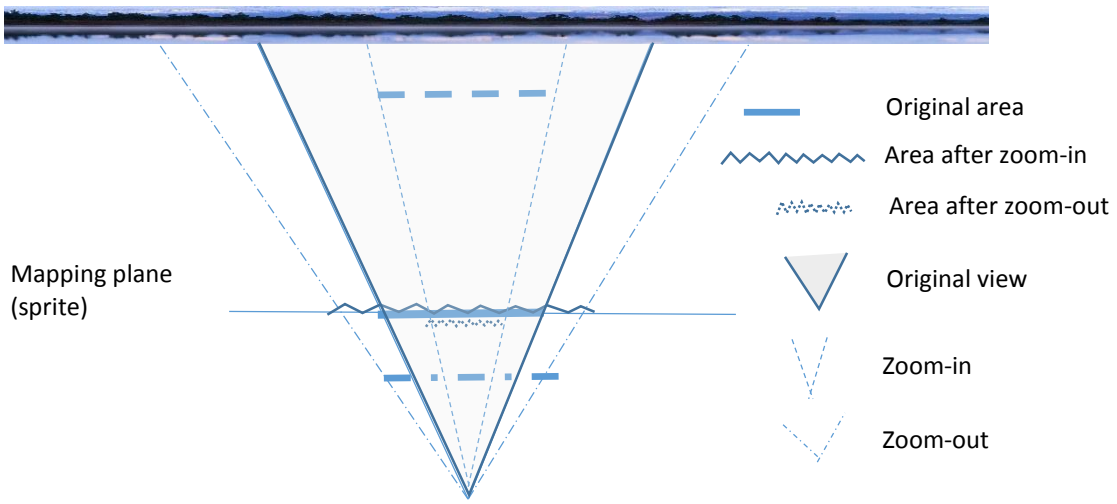

Figure 8. Representation of zoom operation.

4.4 Our Virtual Camera Control Model

Our virtual camera control model is defined based on the supported camera operations. Roughly, our virtual camera model supports pan, tilt, rotation, and zoom. There are two reasons why we focused on these virtual camera controls. Firstly, the rotation around x-axis (pan) and y-axis (tilt) has significant overlap with translation along x and y axes. We can see the correlation as follows. Let *β, d, L,* and *t* represent the angle of view, the distance of the camera from the scene, the length of the visible area, and the length of the translation. Initially, $L = \tan\left(\frac{\beta}{2}\right) * 2d$. Initial coverage area can be represented as *[-L/2,L/2]*. After translation *t*, the coverage area becomes *[t-L/2,t+L/2]*. If the camera is rotated such that the rotation angle generates translation t around the center, then the rotation angle is equivalent to *arctan(t/d)*. With respect to this rotation, the coverage becomes

$$\left[-\tan\left(\frac{\beta}{2} - \arctan\left(\frac{t}{d}\right)\right) * d, \tan\left(\frac{\beta}{2} + \arctan\left(\frac{t}{d}\right)\right) * d\right] \quad \text{(Eq. 14)}$$

Now assume that the angle of view is 30 degrees and the camera is 100 feet away from the scene. We want a translation about 5 feet around the center of the scene. The initial coverage area is computed as [-26.79, 26.79]. After translation by 5 feet, the new coverage becomes [-21.79,31,79]. If we rather apply rotation rather than translation, the corresponding angle of rotation is arctan(5/100)=2.86. The new coverage becomes [-21.51,32.23]. For small angle changes and small translations, this difference can be ignored. The application of realistic pan and tilt operations cause distortion for far objects due to perspective motion model. With the proposed translational operations that type of distortion is minimized. We should also mention that it is easier to control the camera by moving the camera along axes since the visible area changes consistently. For the purposes of video editing, we feel that translational operation is more fitting for the user.

The pan and tilt operations actually correspond to rotation around y-axis and x-axis, respectively. Typically, the goal of these operations is to cover new areas distant from the camera when the camera is mounted on a fixed pole. In our camera model, we map pan and tilt operations to translational displacement along x-axis and y-axis, respectively. Since the sprite is generated as a planar sprite, we have chosen to provide pan-tilt in terms of translational motion. We believe that translational control better fits the needs of a user. The only rotation is along z-axis (depth). So when the rotation is mentioned, it is implied that it is around z-axis unless stated otherwise for the rest of the paper. In a similar way, pan/tilt is used to represent translational motion along x-axis and y-axis unless stated otherwise. Our camera also supports zoom-in and zoom-out operations.

Due to homography, a camera view *a* can be mapped to camera view *b* in the following way:

$$P_b = K_b H_{ba} K_a^{-1} P_a \qquad \text{(Eq. 15)}$$

where P_a and P_b are the positions of point *P* in camera view *a* and *b*, respectively; K_a and K_b are the intrinsic parameters of the camera at view *a* and *b*, respectively; and H_{ba} is the homograhy matrix for the mapping. The matrix $M_{ab} = K_b H_{ba} K_a^{-1}$ is also called as the fundamental matrix. The actual goal is to find this fundamental matrix.

We approach virtual camera control in two ways. In the first method, we consider in terms of the essential matrix. In the second one, we try to express it through intrinsic camera parameters.

4.4.1 Virtual Camera Model through the Essential Matrix

In the previous section, we have already explained how to map the pixels from the sprite to a specific frame *i*: $M_{(0,i)}^{-1} P_s = P_i$. One of the aims of this project is to enable virtual camera control at any frame. For consecutive frames, the essential matrix has already been computed. Our virtual camera provides or defines new essential matrices for the available functions. Our virtual camera control, provides five types of camera control (pan, tilt, rotation clock-wise, rotation counter-clock-wise, zoom-in, and zoom-out) and these yield the following essential matrices:

$$C_p = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, C_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, C_{zi} = \begin{bmatrix} z_i & 0 & 0 \\ 0 & z_i & 0 \\ 0 & 0 & 1 \end{bmatrix}, C_{zo} = \begin{bmatrix} z_o & 0 & 0 \\ 0 & z_o & 0 \\ 0 & 0 & 1 \end{bmatrix} C_{rc} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$C_{rcw} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 16)}$$

where C_p, C_t, C_{zi}, C_{zo}, C_{rc}, and C_{rcw}, are essential matrices for pan, tilt, zoom-in, zoom-out, and rotation clock-wise and counter-clock-wise, respectively. The zoom parameters z_i and z_o can be correlated such as $z_i = 1/z_o$. Zoom operation can be represented with a single matrix C_z for simplicity. Each virtual control calls the corresponding essential matrix. We may just use C_r for rotation matrix for simplicity although rotation in each direction is possible. As a result, we get a series of multiplications of these essential matrices:

$$C = \prod_{k=1}^{m} C_x^k \qquad \text{(Eq. 17)}$$

where C_x^k represents the k^{th} essential matrix for the camera operation and $x \in \{p, t, z_i, z_o, r_c, r_{cw}\}$. The final matrix for virtual camera control at frame *i* for a sequence of *m* virtual controls looks like:

$$M_{(0,i)} \prod_{k=1}^{m} C_x^k \qquad \text{(Eq. 18)}$$

For mapping, we use

$$(M_{(0,i)} \prod_{k=1}^{m} C_x^k)^{-1} P_s = P_v \qquad \text{(Eq. 19)}$$

where P_v is the desired pixel location and P_s is the location on the sprite.

4.4.2 Virtual Camera Model through the Intrinsic Camera Parameters

The virtual camera model using the homography matrix may work; however, it may not correspond to actual transformation. There are three characteristic operations: zoom, translation, and rotation. Zoom operation is managed by changing the focal depth of a camera. Hence, zoom operation updates the intrinsic camera matrix not the *H* matrix.

The translation can be considered in two ways: translation of the camera or the translation of the principal points. The second operation affects the intrinsic parameters whereas the first one maintains the same intrinsic parameters. We cover both scenarios below. Whenever the principal point is updated, we assume the intrinsic parameters are updated for translation. If we assume the second case, the *H* matrix includes only rotation. Otherwise, it may have rotation or translation.

Zoom operation at view *a* updates intrinsic parameters for view *b* as follows:

$$K_b = C_z K_a = \begin{bmatrix} z.\alpha_x & \gamma & u_x \\ 0 & z.\alpha_y & u_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 20)}$$

In a similar way, translation t_x and t_y updates the principal points from view a to view b as follows (note that we merged pan-tilt into a single translation matrix for simplicity):

$$K_b = C_t K_a = \begin{bmatrix} \alpha_x & \gamma & t_x + u_x \\ 0 & \alpha_y & t_y + u_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 21)}$$

The *H* matrix may have a rotation only. For rotation θ, it is represented as

$$H_\theta = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 22).}$$

Note that in a virtual control normally these operations are applied one at a time. If we have only zoom, the fundamental matrix is:

$$C_z K_a H_0 K_a^{-1} = C_z K_a I K_a^{-1} = C_z K_a K_a^{-1} = C_z K_a K_a^{-1} = C_z \qquad \text{(Eq. 23)}$$

where *I* is the identity matrix. We get the same essential matrix provided in the previous section.

If we have only rotation θ, the intrinsic parameters at view *a* and view *b* are the same. However, the rotation is not performed with respect to original frame but it is performed with respect to the last camera world. The typical fundamental matrix is: $K_b H K_a^{-1}$. We did not specify *H* explicitly. First, we should take the world of view *a*, apply rotation, and take it back to the original world. So, *H* matrix is represented as

$$H = K_a^{-1} H_\theta K_a \qquad \text{(Eq. 24)}$$

The fundamental matrix becomes

$$K_b K_a^{-1} H_\theta K_a \; K_a^{-1} = K_b K_a^{-1} H_\theta = K_a K_a^{-1} H_\theta = H_\theta \qquad \text{(Eq. 25)}$$

Note that K_b is equal to K_a since there is no change in the intrinsic parameter set.

We may apply the same logic for translation (without changing the intrinsic parameters) in the camera world with respect to view *a*. The fundamental matrix becomes

$$K_b K_a^{-1} C_t K_a \; K_a^{-1} = K_b K_a^{-1} C_t = K_a K_a^{-1} C_t = C_t \qquad \text{(Eq. 26)}$$

Note that this translation is in the world with respect to view *a*. We may analyze this matrix with respect to the principal points and obtain the same result. So, translations can be handled at the sprite plane. Translation at the sprite plane corresponds to the change of the principal point. We do not have actual translation but we have displacement of principal points leading to change in the intrinsic parameter view *b*. The fundamental matrix is $K_b I \; K_a^{-1}$. The principal points for K_b is shifted with respect to the world of view *a* to indicate the same ratio of change at each world. So K_b is represented as

$$K_b = K_a (K_a^{-1} C_t K_a) = C_t K_a \qquad \text{(Eq. 27)}$$

The matrix multiplication $(K_a^{-1} C_t K_a)$ returns 3x3 matrix where diagonal values are 1 and the rest of the values except translational values are 0. The translational values correspond to changes with respect to view *a*. The fundamental matrix becomes

$$C_t K_a I \; K_a^{-1} = C_t \qquad \text{(Eq. 28)}$$

4.4.3 Summary

The application of virtual camera controls is not complex. Let $M_{0,i}$ be the essential matrix when a user freezes (or applied controls with respect to) frame *i* for virtual controls. Let V_k^i be the k^{th} virtual operation applied with respect to frame i.

The following operations are applied to get the essential matrix for the first virtual control:

Zoom: $V_1^i = M_{0,i} C_z$

Translation: $V_1^i = M_{0,i} C_t \qquad \text{(Eq. 29)}$

Rotation: $V_1^k = M_{0,i}C_\theta$

For the affine transformation, V_k^i has 6 parameters as follows:

$$V_k^i = \begin{bmatrix} m_2^k & m_3^k & m_0^k \\ m_4^k & m_5^k & m_1^k \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 30)}$$

Table II. Updates of motion parameters after virtual camera controls.

Operation	**Direction**	**Computing V_{k+1}^i based on V_k^i parameters**
Pan	Left	$m_0^{k+1} = m_2^k * (-t) + m_0^k$ $m_1^{k+1} = m_4^k * (-t) + m_1^k$
	Right	$m_0^{k+1} = m_2^k * (t) + m_0^k$ $m_1^{k+1} = m_4^k * (t) + m_1^k$
Tilt	Down	$m_0^{k+1} = m_3^k * (-t) + m_0^k$ $m_1^{k+1} = m_5^k * (-t) + m_1^k$
	Up	$m_0^{k+1} = m_3^k * (t) + m_0^k$ $m_1^{k+1} = m_5^k * (t) + m_1^k$
Zoom	In *(z>1)*	$m_2^{k+1} = m_2^k * z$ $m_3^{k+1} = m_3^k * z$ $m_4^{k+1} = m_4^k * z$ $m_5^{k+1} = m_5^k * z$
	Out *(z>1)*	$m_2^{k+1} = m_2^k / z$ $m_3^{k+1} = m_3^k / z$ $m_4^{k+1} = m_4^k / z$ $m_5^{k+1} = m_5^k / z$
Rotation	Counter clock-wise	$m_2^{k+1} = m_2^k cos\theta + m_3^k sin\theta$ $m_3^{k+1} = m_3^k cos\theta - m_2^k sin\theta$ $m_4^{k+1} = m_4^k cos\theta + m_5^k sin\theta$ $m_5^{k+1} = m_5^k cos\theta + m_4^k sin\theta$
	Clock-wise	$m_2^{k+1} = m_2^k cos\theta - m_3^k sin\theta$ $m_3^{k+1} = m_3^k cos\theta + m_2^k sin\theta$ $m_4^{k+1} = m_4^k cos\theta - m_5^k sin\theta$ $m_5^{k+1} = m_5^k cos\theta + m_4^k sin\theta$

Assume that the translation (pan,tilt) parameters correspond to *t*; the rotation angle corresponds to θ; and zoom operation can be zoom-in or zoom-out with *z* and *1/z* parameters. Then the parameters of V_{k+1}^i is

computed after each operation as in Table II. Although these can be easily extracted from the matrix multiplications, we provide them for avoiding ambiguity.

The essential matrix after *k* operations for frame *i* becomes:

$$M = M_{0,i} \prod_{m=1}^{k} V_m^i \qquad \text{(Eq. 31)}$$

The pixel values after k operations are obtained as:

$$\begin{bmatrix} X_v \\ Y_v \\ 1 \end{bmatrix} = M^{-1} \begin{bmatrix} X_0 \\ Y_0 \\ 1 \end{bmatrix} \qquad \text{(Eq. 32)}$$

where (X_v, Y_v) corresponds to pixel coordinates after applying virtual controls. This expression simply shows that the new matrix can be computed as a sequence of matrix multiplications.

5 Applications

In this section, we provide three applications: video editing, object centralization, and aspect ratio conversion. If virtual camera controls are enabled as stated in the previous section, the following applications can be managed easily.

5.1 Automated vs. Manual Video Editing

The idea of virtual controls is to regenerate the video with new viewpoints. In the manual mode, the user is expected to play with the virtual camera control for each frame for regeneration. However, applying virtual control at every frame can be tedious. In addition to manual mode for every frame, we provide automated video editing. In the automated mode, the user can begin with any frame and perform a sequence of camera motions. Then the user can keep the camera static to capture the content automatically of that position on the sprite for the following frames.

5.2 Object Centralization

If the object does not appear in the center of a frame, we would like to centralize the object. To centralize the object, we need to locate the object first. When we play the original frame on the sprite, we place the transformed region of original frame on the sprite with object and calculate the intensity difference of this region against the corresponding region on the original sprite. And we assume that difference above a threshold is caused by the object occlusion. To quickly decide the location of the object, we adapted our padding method. Our padding method assumes that the object has misalignment only in the horizontal direction, and we adjust the object in the horizontal direction towards center. When we take the difference, we choose the column that has the largest intensity difference from the sprite as the column where the object center is located. We first regenerate frame *i* using $M_{(0,i)}^{-1}$ without the object and take the difference.

Assume that the current frame for object centralization is frame *i*. Based on the difference of the columns, let (x_o) be the column that contains the center of the object. Then the displacement (x_d) is (x_o- *w/2*). The virtual control translation matrix becomes

$$C = \begin{bmatrix} 1 & 0 & x_d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_o - \frac{w}{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 33)}$$

The new image plane would map a quadrangle region as shown with red borders in Figure 9.

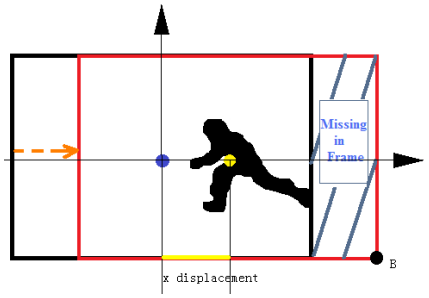

Figure 9. Horizontal shift for object centralization

5.3 Aspect Ratio Conversion

To regenerate a frame using different aspect ratio without distortion and information loss, we perform aspect ratio conversion based on the sprite rather than just using a single frame. Let the original aspect ratio, r, be w/h. Let $r_n = w_n/h_n$ be the new aspect ratio to map to new frame size with width w_n and height h_n. Based on the comparison of aspect ratios, we may need to elongate the image plane along either the x-axis or y-axis. The new frame size is computed as shown in Table III.

Table III. Aspect-ratio width and height

Aspect ratio	New width (w_n)	New height (h_n)
$r_n > r$	$w_n = h * r_n$	$h_n = h$
$r_n < r$	w_n w	$h_n = w * r_n$
$r_n = r$	$w_n = w$	$h_n = h$

Then, we use the new width and new height to map pixels on the sprite to the new image plane (quadrangle) using the same transformation matrix.

6 Experiments

We have implemented our SpriteCam using Microsoft Visual C# under Windows. We have experimented our technique on a variety of sequences including Stefan, GM_LM, Exploring_Turkey, and coastguard video. We have used our sprite fusion method [17] to generate the background sprite. In [17] we explain limitations of other sprite generation techniques. In addition, we have provided a comparison of our method with two other methods [14] and [26]. Firstly, our algorithm does not require any object segmentation step or segmentation masks. Secondly, even without using segmentation masks, our method surpasses

average PSNR value mentioned in [26]. More information about our research can be found at http://sprite.cs.uah.edu/mosaics.

6.1 Manual Mode

In the manual mode, any frame can be edited with a single camera motion. Figure 10 presents a frame in Stefan video after applying different camera motions.

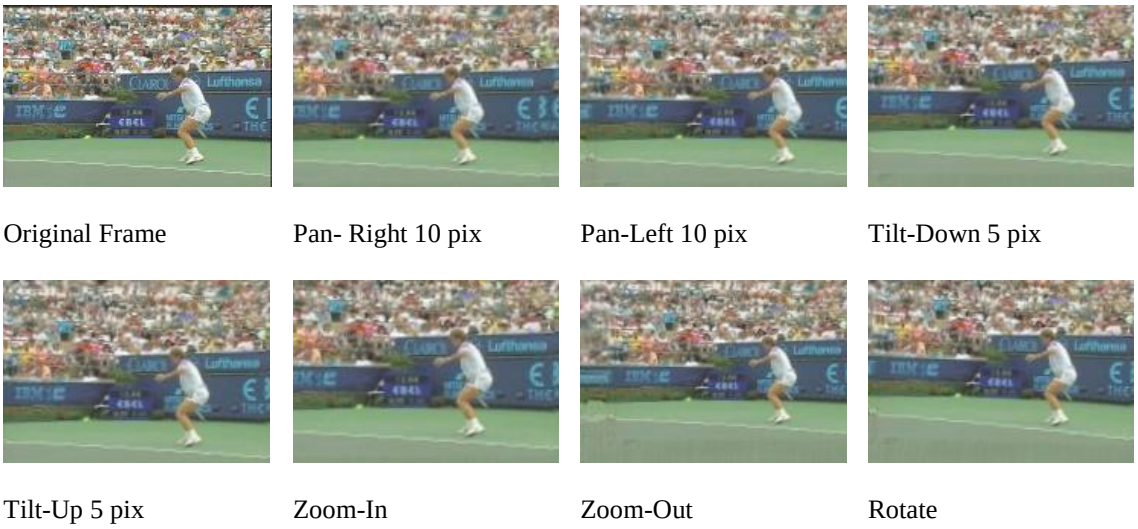

Figure 10. Regenerated frames in Stefan video after applying different camera motions

It is also possible to apply a series of camera motion to a frame. Figure 11 presents a selected frame from sequence Exploring_Turkey3 and then application of pan-right 45-pixel, zoom in (zoom rate is 1.1), and rotation by 2 degrees.

Figure 11. Manual mode for Exploring_Turkey after applying a sequence of camera motions

6.2 Automated Mode

In the automated mode, we can start with a frame and perform a series of camera motion. We could then fix the camera, and capture the content of that specific location. Figure 12 presents original sample frames beginning at frame 18 for GM_LM sequence. Figure 13 presents the sprite for GM_LM videos. Figure 14 shows regenerated sample frames when we pan left the camera at frame 18 then fix the camera. Note that this kind of sequence is not possible to generate by only cropping frames.

Figure 12. Original sample frames beginning at frame 18 for GM_LM sequence.

Figure 13. Fusion Sprite for GM_LM sequence

Figure 14. Sample regenerated frames using automated mode

Coastguard video is another good example to show how this automated mode can be used to relate any two frames. Figure 15 shows the covered scene when we stop the camera location at frame 137.

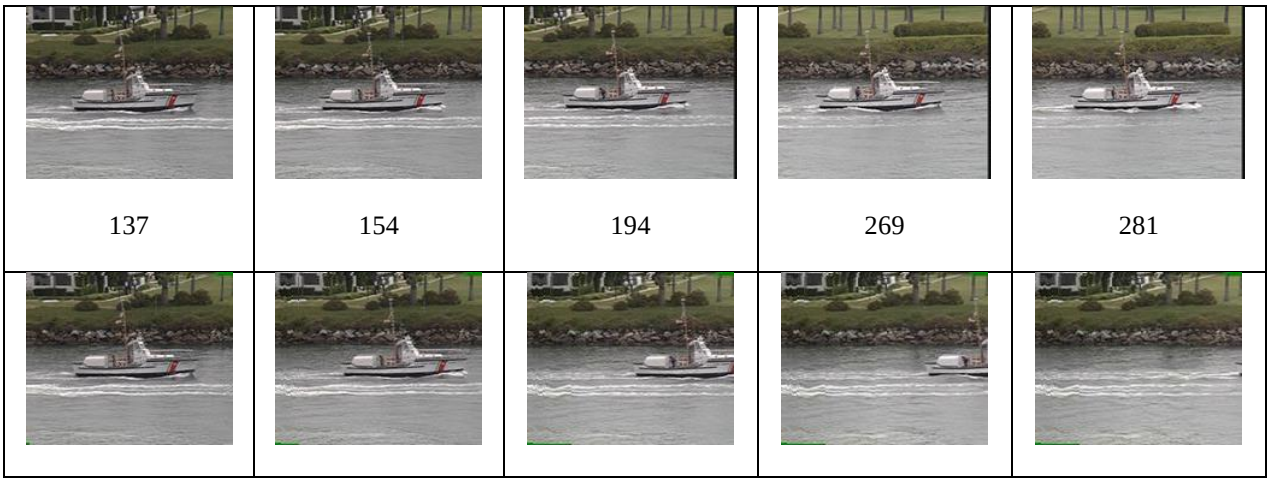

Figure 15. Sample regenerated frame using automated mode to stop at frame 137 (the first row represents the original frames, the middle row provides the frame numbers, and the last row displays the regenerated frames).

6.3 Object Centralization

For Stefan video, the cameraperson fails to locate the object (tennis player) in the middle of a frame. This problem can be observed in Figure 2(b). Hence, Chen et al. [17] defined Stefan sequence as a partially-tracking video rather than a tracking video. For partially-tracking videos, objects may appear at the boundary. Figure 16 presents sample Stefan frames where objects are not centralized in the first row. After centralizing the object, the corresponding frames are shown in the last row. For frames 56, 106 and 166 in Stefan video, the results are acceptable.

However, the regenerated frame 264 has parts outside the region. In this case, we can adjust manual mode to edit the regenerate frame. Figure 17 represents the result of making shift and zoom in result.

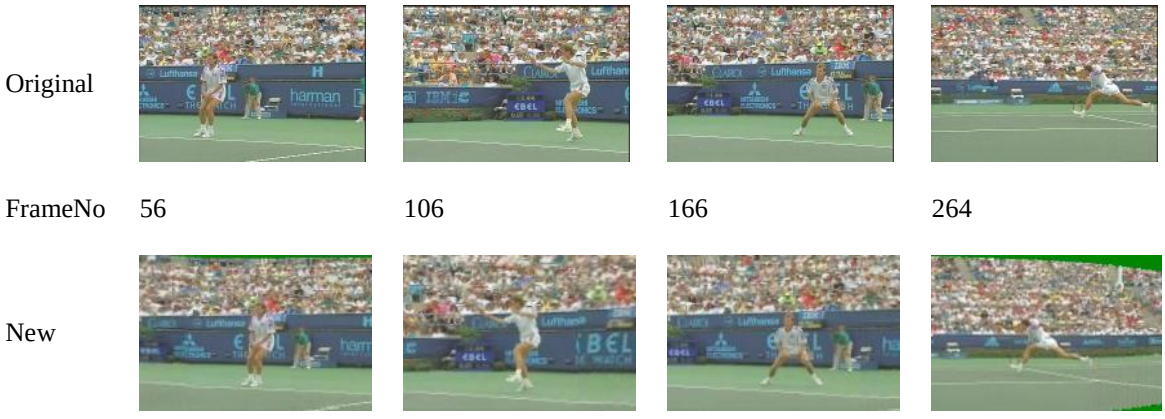

Figure 16. Original frame and object-centric frame

Figure 17. Regenerated frame 264

6.4 Aspect Ratio Conversion

We also performed aspect-ratio conversion using the sprite. Before providing our approach, we first provide the traditional methods for aspect ratio conversion. Aspect ratio conversion is usually handled using information from a single frame by stretching (this introduces distortion), cropping (loss of information) or pillar box (this has missing information at the borders) (Figure 18).

Original frame

Stretching (distorts the frame)

Pillar box (black borders at the sides)

Cropping (missing content at the top and bottom)

Figure 18. Traditional aspect ratio conversion methods.

Figure 19 presents views of different coastguard frames with different aspect ratios. The original size of the coastguard frame is 176:144. The frame size for16:9 is 256x144 and the frame size for 4:3 is 192*144. We can observe that the new generated frames do not distort the shape of the boat and picture quality. Instead, they cover more background than the original ones.

Original (176*144)	16:9 (256*144)	4:3 (192*144)
		
		
		

Figure 19. Sample frames of aspect-ratio conversion on coastguard video

7 Conclusion

In this paper, we provide our SpriteCam system, a video camera controller to perform video editing and browsing using the sprite. We have placed and blended frames on the sprite. We have provided the theoretical framework of virtual camera controls using the sprite of a video. We have shown its applications such as object centralization and aspect ratio conversion. We introduced the automated and manual modes for video editing. Our video controller adapted sprite fusion technique for sprite generation, which provides better quality than other ones. The advantage of our video camera controller is to use spatially correlated information from other frames in the sequence for video editing. We have provided our results on a variety of videos.

The major contribution of our work is the enhanced video editing using available correlated information from other frames in the sequence. We present this type of video editing through a theoretical framework based on virtual camera controls. We should note that SpriteCam can be used if the sprite can be generated. Even if the sprite is generated, there could be problems related to alignment and blurring.

Our video editing also requires that there should not be moving objects at the borders of a frame. Otherwise, if the scene is moved towards the object at the border, missing object parts are not shown since they do not exist in the original video. We should also note that since the sprite is static, good results are obtained if there are not major changes in the background. SpriteCam depends on the quality of sprite. There are two issues

with generating quality sprite: accuracy of motion estimation and blurring caused during the blending phase. As future work, we plan to increase the quality of sprite. In addition, we plan to generate 3D sprites and provide additional virtual camera controls based on the depth information.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0812307.

References

[1] http://www.adobe.com/products/premiere.html, accessed June 2013

[2] http://www.cyberlink.com/products/powerdirector/overview_en_US.html, accessed June 2013

[3] Wilson ,A., Lin, M. C., Yeo, B., Yeung, M.M., and Manocha, D.: 'A video-based rendering acceleration algorithm for interactive walkthroughs'. Proc. ACM Multimedia, 2000, pp.75-83

[4] Guo, Y.D., Gu, X.D., Chen, Z.B., Chen, Q.Q., Wang, C.: 'Adaptive Video Presentation for Small Display while Maximize Visual Information', LNCS, Springer. Visual 2007, Shanghai.

[5] Mavlankar, A., Varodayan, D., and Girod, B.: 'Region-ofinterest prediction for interactively streaming regions of high resolution video'. Proc. of 16th Intl. Packet Video Workshop,Lausanne, Switzerland, , Nov. 2007, pp. 68–77

[6] X. Sun, X., Foote, J., and Kimber, D., *et al*: 'Region of interest extraction and virtual camera control based on panoramic video capturing'. *IEEETrans. Multimedia*., Oct. 2005, 7 (5), pp 981–990

[7] Aygun, R.S., and Zhang, A.: 'Integrating Virtual Camera Controls into Video', *2004 IEEE Int. Conf. on Multimedia and Expo*, Tokyo, Japan, August, 2004

[8] Sikora, T.: ' The mpeg-4 video standard verification model'. IEEE Trans. Circuits Syst. Video Technology 7.,1997, pp. 9–31

[9] Dufaux, F., and Konrad, J.:'Effcient, Robust and Fast Global Motion Estimation for Video Coding'. *IEEE Trans. on Image Processing.,2000*, 9, (3).

[10] Smolic, A., Sikora, T., and Ohm, J.R.: ' Long-term global motion estimation and its application for sprite coding, content description and segmentation', IEEE Transactions on Circuits and Systems for Video Technology.,1999, 9, (8) ,pp. 1227–1242

[11] Smolic, A. and Ohm, J.R.:' Robust global motion estimation using a simplified m-estimator approach'. Proc. ICIP2000, IEEE Int. Conf. on Image Processing, 2000

[12] Lu, Y., Gao, W., and Wu, F.:' Fast and Robust Sprite Generation for MPEG-4 Video Coding'. *Proc. of the Second* I*EEE Pacific Rim Conf. on Multimedia: Advances in Multimedia information Processing, October 2001*

[13] Lee, M.C., Chen, W.G., Lin, C.B., GU, C., Markoc, T.,Zabinsky, S.I., and Sze liski, R.:'A layered video object coding system using sprite and affine motion model'. *Circuits and Systems for Video Technology,IEEE Transactions on* , 1997, 7, (1), pp.130-145

[14] Lu, Y., Gao, W., and Wu, F.:' Efficient background video coding with static sprite generation and arbitrary-shape spatial prediction techniques'. IEEE Trans. Circuits and Systems for Video Technology., 2003, 13, (5) , pp.394–405

[15] Lu, Y., Gao, W., and Wu, F.:' Fast and Robust Sprite Generation for MPEG-4 Video Coding'. Proc. of the Second IEEE Pacific Rim Conf. on Multimedia: Advances in Multimedia information Processing, 2001.

[16] Aygün, R. S. and Zhang, A.: 'Reducing blurring-effect in high resolution mosaic generation'. In Multimedia and Expo, 2002. ICME '02. Proc. 2002 IEEE Int. Conf., 2, IEEE Computer Society, Washington, DC, pp. 537-540.

[17] Chen, Y., Deshpande. A.A., and RS Aygun.: Sprite generation using sprite fusion. *ACM Trans. Multimedia Comput. Commun. Appl.* 8, 2, Article 22 (May 2012), 24 pages.

[18] Hsia, S.C., and Liu, B.D.:'An NTSC to HDTV video conversion system by using the block processing concept'. Journal, IEEE Transactions on consumer electronics. 1994, August, pp. 216-224.

[19] Deng, Z-L., Guo, Y-D., Gu, X-D., Chen, Z-B., Chen, Q-Q ., and Wang, C.: 'A Comparative Review of Aspect Ratio Conversion Methods'. Proc. of the 2008 Int. Conf. on Multimedia and Ubiquitous Engineering (MUE '08). IEEE Computer Society, Washington, DC, USA, 2008, pp.114-117

[20] Guo, Y-D; Deng, Z-L; Gu, X-D; Chen, Z-B; Chen, Q-Q; Wang, C. 'Aspect Ratio Conversion Based on Saliency Model'. Image and Signal Processing, 2008. CISP '08. Congress 4, 2008,pp. 92-96

[21] Avidan, S., and Shamir, A.: 'Seam carving for content-aware image resizing'. In ACM SIGGRAPH 2007 papers (SIGGRAPH '07). ACM, New York, NY, USA, 2007, Article 10 . DOI=10.1145/1275808.1276390 http://doi.acm.org/10.1145/1275808.1276390

[22] Deshpande, A., and Aygun, R.S.: ' Motion-based video classification for sprite generation'. Database and Expert Systems Applications, 20th Int. Workshop on Database and Expert Systems Application, 2009, pp. 231-235

[23] Glantz, A., Krutz, A., Sikora, T., Nunes, P., and Pereira, F.: 'Automatic MPEG-4 sprite coding—Comparison of integrated object segmentation algorithms' Multimedia Tools and Applications Vol. 49(3), pp. 483-512, 2010

[24] Barhoumi, W., Bakkay, M. C., and Zagrouba, E.: An online approach for multi-sprite generation based on camera parameters estimation Multimedia Tools and Applications, 2011

[25] Kunter, M., Krutz, A., Mandal, M., and Sikora, T.: 'Optimal multiple sprite generation based on physical camera parameter estimation'. Visual Communications and Image Processing, VCIP, IS&T/SPIE's Electronic Imaging 2007

[26] Cheung, H.-K. and Siu W.-C. 2007. Robust global motion estimation and novel updating strategy for sprite generation. In Image Processing, IET (March 2007), vol.1, no.1, pp.13-20.