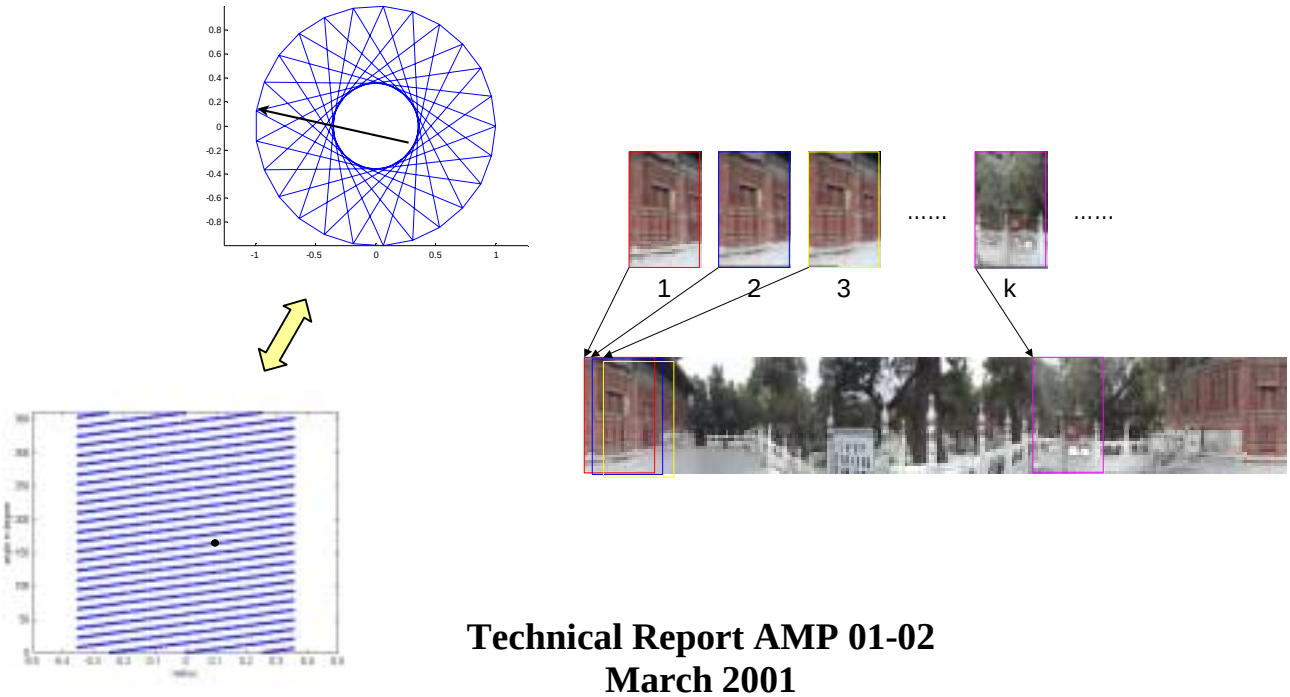
Carnegie Mellon

Electrical & Computer ENGINEERING

LINE-SPACE REPRESENTATION AND COMPRESSION FOR IMAGE-BASED RENDERING

Wing Ho Leung and Tsuhan Chen
Advanced Multimedia Processing Lab



Technical Report AMP 01-02
March 2001

Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

LINE-SPACE REPRESENTATION AND COMPRESSION FOR IMAGE-BASED RENDERING

Wing Ho Leung and Tsuhan Chen

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA
Email: {wingho,tsuhan}@andrew.cmu.edu

ABSTRACT

In image-based rendering applications, images of a scene are captured along a specific camera trajectory and are then used to render new views of the scene. In this paper, we first study several possible camera trajectories and analyze their pros and cons. Then we introduce the line space which is a representation of light rays and use it to analyze a number of implementation issues in image-based rendering. We propose a novel sprite-based compression scheme in this application. The performance of the image compression under this scheme is examined.

I. INTRODUCTION

In image-based rendering (IBR) applications, light rays from different directions are captured and then resampled in order to generate different views of a scene. In terms of sampling the light rays, Shum and He proposed concentric mosaics [1] where a 3D plenoptic function [2] is used to represent the light rays. Concentric mosaics generalize the 2D panorama [5] and can be considered as a special case of the lumigraph [3] or lightfield [4]. In this paper, although we focus our discussion on concentric mosaics, many techniques we present can easily be extended to the lumigraph or lightfield.

To examine how image-based rendering can be applied to render new views from captured images, we present our experimental setup for capturing the images and propose some evaluation criteria for the camera trajectory to capture images efficiently. We will introduce the line space which is a representation of light rays for image-based rendering applications. The line space is then used to explain the reconstruction of a new view from a virtual camera. We will also discuss how caching can lead to efficient rendering of the scene if we have random access to each of the acquired images.

In image-based rendering, often the number of acquired images is very big, therefore it is necessary to compress the images in order to store them efficiently. The acquired images form a video sequence so they can be compressed using a conventional video codec. Given such a sequence, we propose a new compression scheme based on the prediction of each image from a small number of images called sprites. A sprite is constructed by merging all images in a certain manner in order to form a good prediction for each original image. The residue between the original images and the prediction is coded using a general video codec scheme with modification to motion estimation and compensation. We will analyze the performance of our proposed algorithm and compare it with other compression schemes in rate-distortion sense.

This paper is organized as follows: In Section II, we will discuss how images are acquired for image-based rendering. The line space analysis will be introduced in Section III to better understand how the view from a virtual camera can be rendered from the acquired images. In Section IV, we will propose our sprite-based compression scheme and evaluate its performance. In Section V, we will discuss enhancement to our basic sprite-based compression scheme to improve performance. Then the conclusion and future work will be given in Section VI.

II. IMAGE-BASED RENDERING

When a user navigates in a virtual environment, the scene rendered should be updated according to the user's perspective. There are two approaches in rendering the scene: geometry-based rendering and image-based rendering. The geometry-based rendering is the traditional computer graphics approach using 3D models to represent objects in an environment. Image-based rendering, on the other hand, generates new views by a set of pre-acquired imagery. This approach is not computationally intensive therefore it is suitable for real-time implementation. Besides, the scene rendering for this approach does not depend on the scene complexity. This is an advantage over geometry-based rendering which becomes inefficient when the scene is very complex since the number of 3D models to be rendered increases.

An image is an array of light intensity and the process of acquiring an image is equivalent to the sampling of light rays passing through the physical camera. Under image-based rendering, the sampled light rays are resampled in order to reconstruct new views. Given a free space where the physical camera can be positioned, we would like to find a good camera trajectory such that based on images acquired along that trajectory, we can generate new views in certain area inside the region bounded by the trajectory. Figure 1 illustrates this new view generation process. In this example, the physical cameras are positioned in a circular trajectory for acquiring the images. After image acquisition a new view can be generated at locations different from the physical camera trajectory as if a virtual camera is placed in the position as shown in Figure 1. This new view is synthesized by interpolating corresponding light rays that are captured as the pixels in the acquired images by the physical cameras.

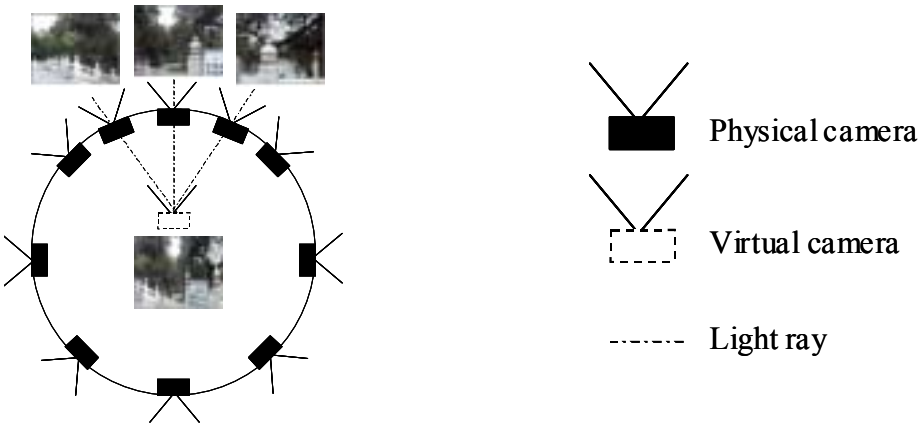


Figure 1 Rendering a new view from a virtual camera with images acquired by physical cameras along a circular trajectory

In the previous example a circular trajectory was used. An interesting question here is whether there are other types of trajectories we can use. The answer is clearly affirmative but we need some measures in order to compare different trajectories. There are several criteria that can be used to evaluate the trajectory of the physical camera positions:

1. Area of the possible virtual camera positions: we would like to maximize the area in which a virtual camera can be rendered.
2. Length of the physical camera trajectory: since the length of the camera trajectory and the number of images captured by the camera are positively correlated, it is desirable to minimize the length of the physical camera trajectory so that minimum number of images is required.
3. Presence of redundancy: redundancy occurs when the same light ray is captured by more than one physical camera. For example, Figure 2 shows a camera trajectory that will result in redundancy since a ray is captured by more than one physical camera. Redundancy should be avoided so that the light rays are captured and stored efficiently.

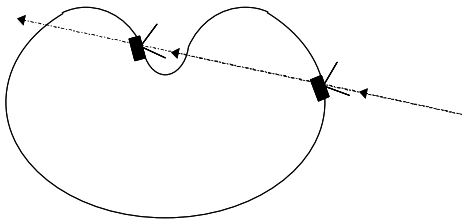


Figure 2 An example of a camera trajectory that results in redundancy

In order to avoid redundancy, special consideration should be taken for the physical camera trajectory. In particular, redundancy always occurs if the trajectory is concave. This is because in a concave trajectory there exists a line that intersects the trajectory with more than two points. Since only two light rays with opposite direction can lie on the same line but more than two cameras are used to capture the two light rays, there must be a light ray that is captured by at least two cameras.

4. Vertical field of view: we would like to capture an image with larger vertical field of view so that a larger area of the scene can be reconstructed. In other words, the camera should preferably be far enough from objects in the scene to cover the whole scene.

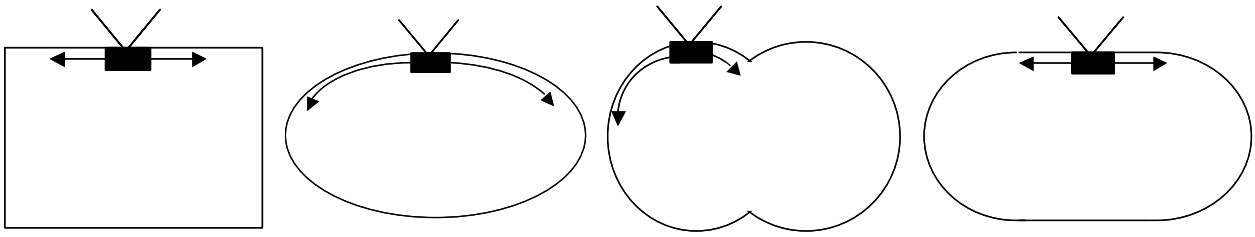
For different shapes of the region bounded by the camera trajectory there is a tradeoff between criteria 1 and 2 (a larger area usually requires a longer trajectory). The perimeter efficiency [11] can be used to evaluate a shape by looking at both the area and the length at the same time. The perimeter efficiency k is defined as

$$k = \frac{2\sqrt{\pi A}}{p} \tag{1}$$

where A is the area and p is the perimeter of the region

It should be noted that two similar shapes have the same perimeter efficiency and a circle has the maximum perimeter efficiency with k=1.

Inside a rectangular free space in which a physical camera can be positioned, several physical camera trajectories as shown in Figure 3 are examined in terms of the perimeter efficiency by varying the aspect ratio of the trajectory shape. Here it is assumed that the field of view of the camera is 180° such that it can capture all the light rays entering the enclosed region in order to render any virtual views inside the enclosed region. The issues of limited field of view will be addressed in the next section. As shown in Figure 4, among the four trajectories, it is found that the "running track" trajectory, which is a combination of circular arcs and a rectangle, provides the most efficient shape. For the two circle trajectory, when the aspect ratio is small, the two circles are not intersecting therefore the perimeter and the area remain unchanged and this explains the flat region of the perimeter efficiency.



Rectangle Ellipse Two circles "Running track"

Figure 3 Different camera trajectories

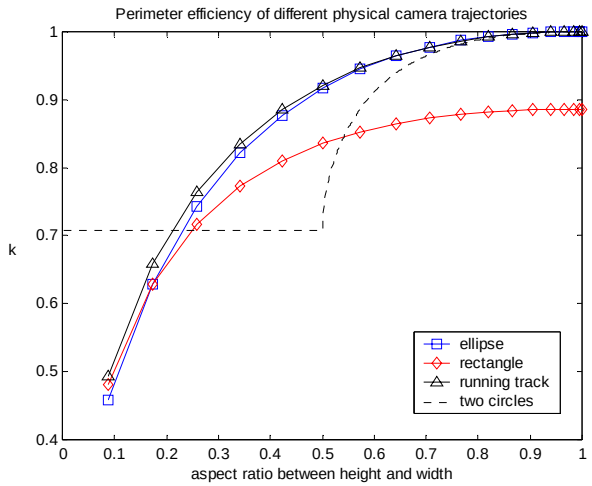
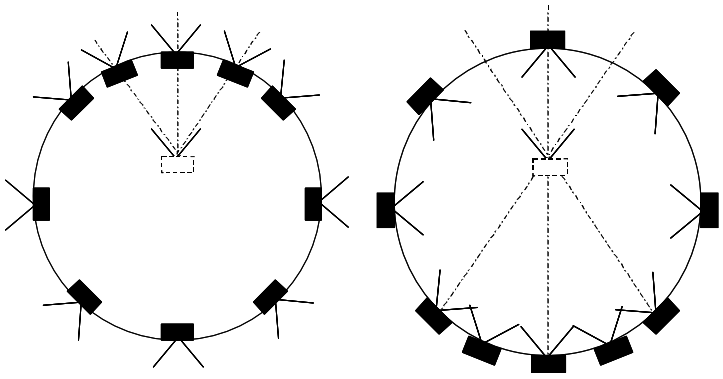


Figure 4 Perimeter efficiency of several shapes inside a rectangular free space

For our experimental setup to capture the images, we choose a circular trajectory so that the perimeter efficiency is maximized.

Besides, there are two configurations for the camera orientations, inward or outward looking, which are shown in Figure 5. Traditionally, outward-looking camera is used for scene rendering and inward-looking camera is used to render objects inside the circular trajectory. To capture the scene, people in the past used the outward-looking camera as shown in Figure 5(a). Here we propose to use the inward-looking camera to capture the scene outside as shown in Figure 5(b) which illustrates that light rays from the scene into a virtual camera can be obtained from light rays already captured by the physical camera. In addition, this configuration has the advantage of allowing a greater vertical field of view of the scene.



(a) outward-looking camera (b) inward-looking camera

Figure 5 Two different camera orientations

Figure 6 illustrates the vertical field of view between the outward-looking camera and the inward-looking camera orientations. The ellipse in Figure 6 represents the circular camera trajectory. The outward-looking camera in position 1 captures only part of the body whereas the inward-looking camera in position 2 captures the whole body. When the new view is rendered from a virtual camera in the center of the circle, it can be seen that the view synthesized from images acquired by the outward-looking camera does not contain the top part and the bottom part of the body because of the limited vertical field of view of the acquired images. On the other hand, it can be seen that the new view synthesized from the images acquired by the inward-looking camera contains the whole body.

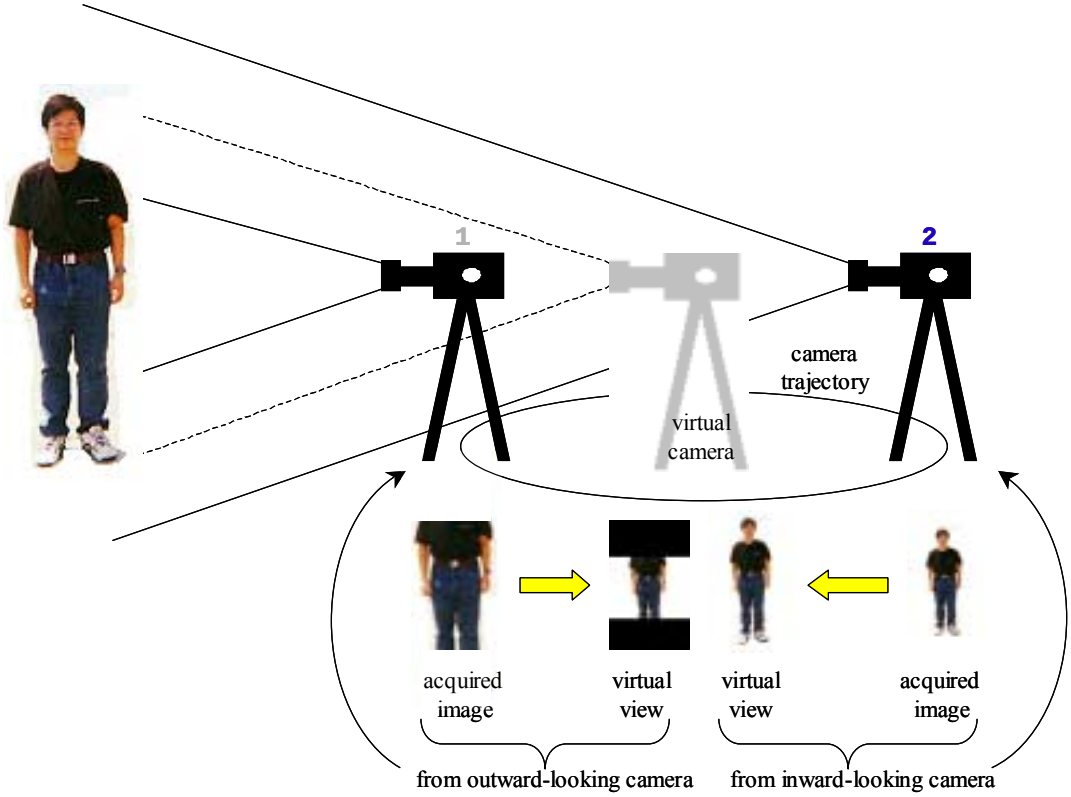


Figure 6 Vertical field of view for outward-looking camera and inward-looking camera

For the approach mentioned in this section, the circular trajectory of the physical cameras constrain them to be located in a 2D plane. Here it is assumed that the parallax effect in the vertical direction is small compared with the parallax effect in the horizontal direction. In fact, this approach, both outward-looking and inward-looking, can be extended to a more general case in which the physical cameras are located in the surface of a 3D region [13].

III. LINE-SPACE ANALYSIS

Levoy and Hanrahan use the line space to visualize light rays [4]. A sample light ray in the Cartesian space is transformed to a point in the line space as shown in Figure 7. The magnitude of r is the distance from the origin to the light ray, the sign of r represents the direction of the light ray and θ is the angle between the light ray and the x-axis. Note that when r is positive, the direction of the light ray is the same as the direction of the tangential component of a clockwise rotation. The sampling of light rays in the Cartesian space is equivalent to the sampling of points in the line space. We will use the line space to explain the reconstruction of new image views.

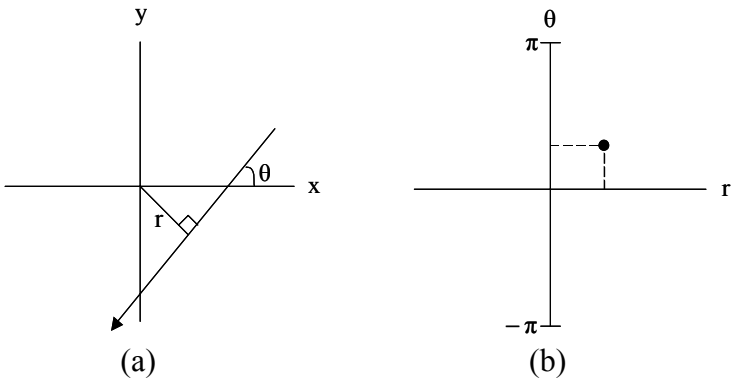


Figure 7 Mapping of a light ray from (a) the Cartesian space to (b) the line space

As mentioned earlier, we use a circular camera trajectory and position our camera inwards to capture the scene outside. Figure 8 demonstrates the coordinate systems in the Cartesian space and in the line space. The arrow in the Cartesian space shown in Figure 8(a) is transformed to a point in the line space shown in Figure 8(b). In the Cartesian space, the camera is assumed to be positioned on the outer circle with radius R and there are two lines diverging outwards which bound the range of the light rays acquired by that camera. The angle extended by these two lines is determined by the horizontal field of view (*HFOV*) of the camera. The set of acquired light rays in the Cartesian space maps to the array of tilted grid points in the line space.

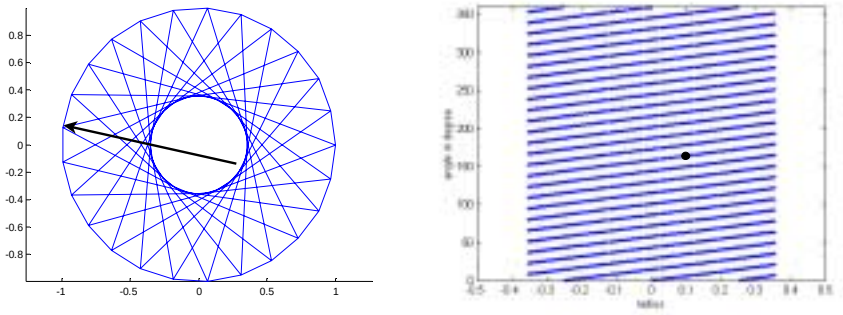


Figure 8 Light rays captured by physical camera under our experimental setup in (a) Cartesian space and in (b) line space

As illustrated in Figure 9, light rays captured at a specific camera with Cartesian coordinates of $(x, y) = (R\cos\phi, R\sin\phi)$ are mapped to an arc-sine function in the line space defined as follows:

$$\theta = \phi - \pi + \sin^{-1}\left(\frac{r}{R}\right) \qquad \text{where } -\frac{HFOV}{2} \le \sin^{-1}\left(\frac{r}{R}\right) \le \frac{HFOV}{2} \qquad (2)$$

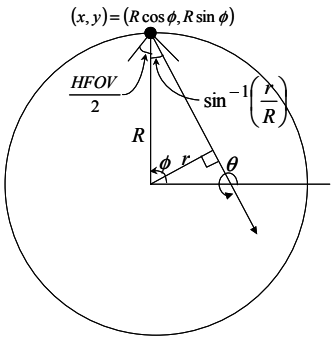


Figure 9 Mapping between Cartesian coordinates (*x*,*y*) and line-space coordinates (*r*, *θ*)

The new view generated by a virtual camera in a position not lying on the physical camera trajectory is obtained by assembling the corresponding acquired light rays together, also forming an arc-sine function in the line space. It should be noted that the light rays in the Cartesian space form an inner circle. Inside this inner circle, any light ray can be generated by interpolating neighboring rays, as detailed in the next section.

III.1 INTERPOLATION OF VIRTUAL CAMERA LIGHT RAYS

Since the sampled light rays may not contain the exact light rays required to generate a new virtual view, we need to obtain those light rays by interpolating the acquired light rays. If the depth information is not available, then we can interpolate by using parallel rays from the neighboring acquired image with the assumption that the depth is at infinity. If the depth information is available, then we can pick the light rays that correspond to a closer match from the acquired image [1]. The following example illustrates the interpolation process:

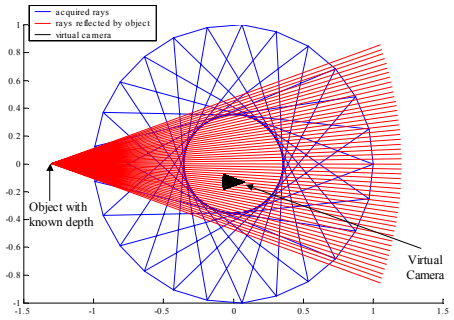
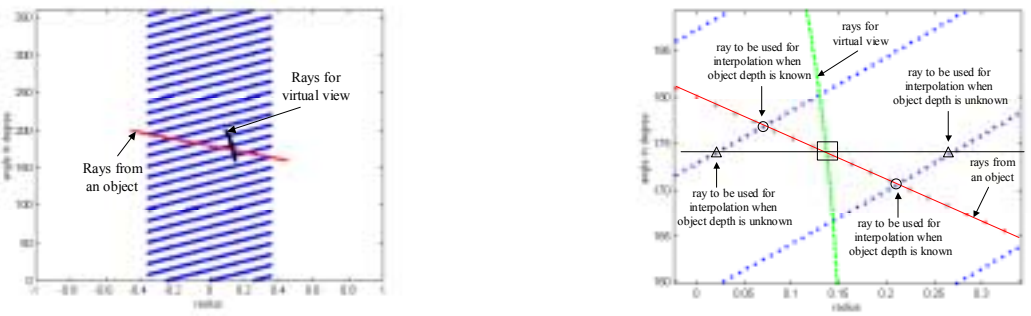


Figure 10 Rendering with a virtual camera

In Figure 10, we would like to render the view from the virtual camera inside the inner circle. The corresponding rays in the line space is shown in Figure 11(a). When the depth is unknown or the objects are assumed to be at infinite depth, then light rays that are parallel in the Cartesian space (light rays that form a horizontal line in the line space) have the same value. In order to look for the parallel ray samples to be interpolated, a horizontal line is drawn for each point in the line space that corresponds to the virtual camera light rays. For example, the intersections with the acquired light rays (the points inside the triangles as shown in Figure 11(b)) are used to interpolate for the virtual camera light ray at the center of Figure 11(b). On the other hand, when the depth of the object is known, then we can consider the set of light rays reflected by the object as shown in Figure 10. The corresponding points of these light rays in the line space is shown in Figure 11(a). We make use of the depth information in order to search for better acquired rays to be used for the interpolation process (the point inside the circles as shown in Figure 11(b)) since these points, which are intersections between the ray reflected by an object and the acquired rays, contain more accurate information about the object.



(a) Line-space representation

(b) Interpolation in the line space

Figure 11 Line-space representation

III.2 POSSIBLE REGIONS AND ORIENTATION OF VIRTUAL CAMERA

A new view from a virtual camera can be rendered from any orientation inside a circle and limited orientation outside this circle. The radius of this circle depends on *HFOV* as shown in Figure 12. In particular, if the radius of the camera trajectory is *R*, then a virtual camera can be rendered at any orientation inside the circle with radius $R\sin(\frac{HFOV}{2})$.

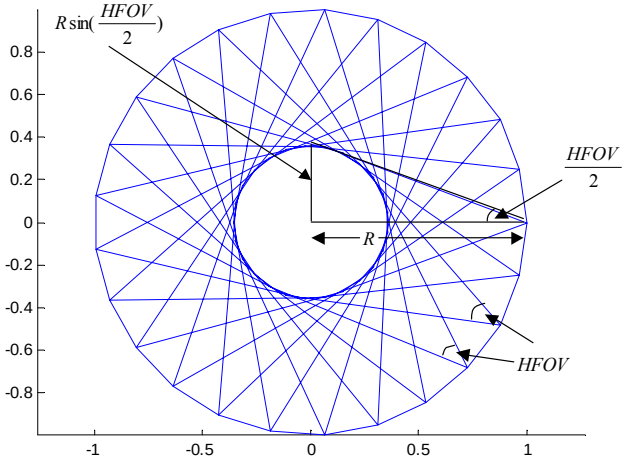


Figure 12 Relationship between the inner circle and the radius of the camera trajectory

We will look at two cases when the virtual camera is inside the inner circle and when it is outside the inner circle.

Case 1: Virtual camera inside the inner circle that can be obtained by interpolating acquired light rays

Figure 13 shows the case when the virtual camera is located inside the inner circle in the Cartesian space. The six beams of light rays represent the range of virtual light rays to be rendered under six different virtual camera orientations. It can be seen that in the line space, the points corresponding to the virtual light rays, form a curve which is an arc-sine function similar to eq. (1) that falls inside the region of the tilted grid points. Therefore the new view for each virtual camera can be generated by interpolating the tilted grid points near the virtual light points in the line space.

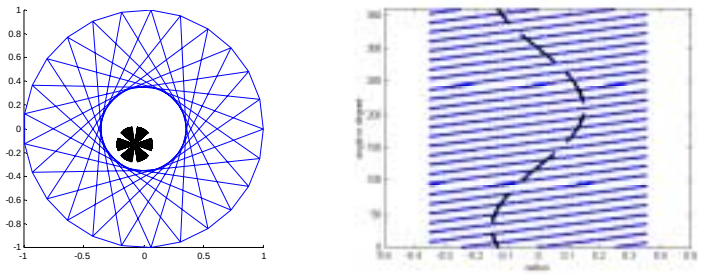


Figure 13 Virtual camera inside the inner circle that can be obtained by interpolating acquired rays

Case 2(a): Virtual camera outside the inner circle that can be obtained by interpolating acquired light rays
Figure 14 shows the case when virtual cameras are located outside the inner circle in the Cartesian space. It can be seen that in the line space, the virtual light points form a curve which is the flat part of an arc-sine curve and can be approximated by a linear function. They fall inside the region of the tilted grid points therefore the new view for each virtual camera can still be generated by interpolating the tilted grid points near the virtual light points in the line space. Comparing this case with the previous case, it can be seen from the line-space representation that when the virtual camera is inside the inner circle (case 1), the virtual view is generated by interpolation with a larger number of acquired images than when the virtual camera is outside the inner circle (case 2(a)). As a result, case 1 results in a better resolution from the interpolation.

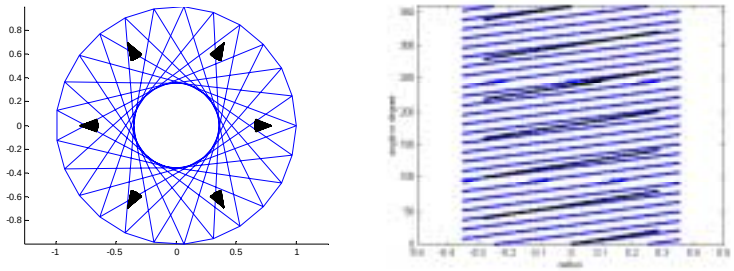


Figure 14 Virtual camera outside inner circle that can be obtained by interpolating acquired rays

Case 2(b): Virtual camera outside the inner circle that cannot be obtained by interpolating acquired light rays
Figure 15 shows another case when six virtual cameras are located outside the inner circle in the Cartesian space. The locations of these six virtual cameras are the same as those in case 2(a) but with a different orientation. It can be seen that in the line space, the virtual light points form a curve which only falls in half of the region of the tilted grid points. Therefore the new view for each virtual camera cannot be generated by the interpolation method.

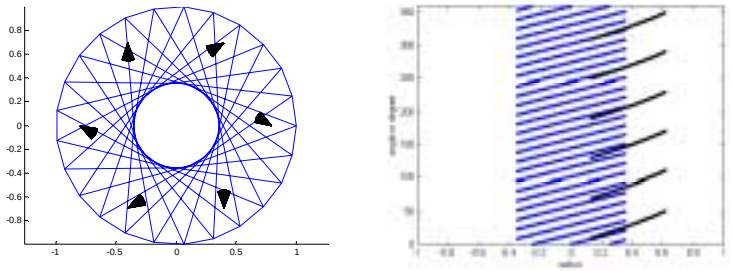
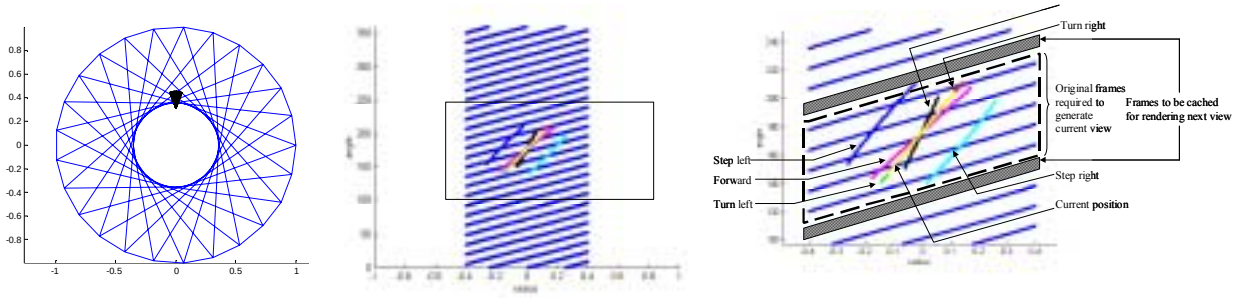


Figure 15 Virtual camera outside inner circle that cannot be obtained by interpolating acquired rays

Combining the above two cases it can be concluded that a virtual camera can be rendered with any orientation anywhere inside the inner circle in the Cartesian space. However, only certain limited orientations of a virtual camera are allowed outside the inner circle for generating new views.

III.3 CACHING AND RANDOM ACCESS

The user can navigate through the scene by generating views from a virtual camera trajectory. Successive views are highly correlated, therefore additional information required to generate the next view from the current view is relatively small. As a result, this additional information can be cached so that the next view will be ready to be rendered no matter which direction the user moves. Figure 16(a) shows the location of the virtual camera where the current view is rendered. Figure 16(b) shows the corresponding virtual light points in the line-space. In addition, the virtual light points of the next view that is generated by moving forward, moving backward, stepping left, stepping right, turning left and turning right are also shown in Figure 16(b). It can be seen that each view requires rays from a limited number of acquired images.



(a) Virtual Camera

(b) Current view and possible next views in the line space

(c) Zoom-up of the current view and possible next views in the line space

Figure 16 Current view of the virtual camera and possible next views in the line space

Figure 16(c) show the zoom-up of the line-space region that contains the virtual light points for the current view and the virtual light points for next possible views. The regions enclosed by the dashed parallelogram represent the acquired images used to generate the current view. The shaded regions represent the incremental information that is required to generate the next possible views. The number of acquired images for this incremental information is much smaller than the number of acquired images used to generate for the current view.

This caching scheme assumes frame-level random access such that an individual acquired image can be loaded independently into the cache. Each time the user moves, the new view is generated from the corresponding light rays that are already in the cache and the images that are required to render next possible views are loaded into the cache. The memory containing those images that are not longer needed will be released. This caching scheme together with frame-level random access facilitates efficient and fast rendering of the scene.

IV. SPRITE-BASED COMPRESSION

In order to have a seamless rendering of a virtual view, the number of acquired images should be large, i.e., a dense sampling of the light rays in the line-space representation. It is thus necessary to compress the images, in other words, to resample the light rays, in order to store them efficiently. One possible way is to compress the acquired images as a video sequence using a conventional video codec. Alternatively, we can exploit the high correlation of the acquired images and compress them based on the prediction from a sprite that is an image constructed from common regions of an object or a scene visible throughout multiple images. In MPEG-4, a static sprite describing a background is used to perform global motion compensation [9]. A similar concept to sprite-based compression can also be found in other video compression applications [8]. We extend these concepts and combine them with the knowledge that the images are taken from a camera with a known path. Figure 17 shows some examples of IBR images and the generation of sprite. The sprite is first constructed by merging all images and then used to predict each individual image. The residue between the original image and the prediction is coded using a codec that is similar to a general video codec but with modification to the motion estimation and compensation. In the following sections we will first explain how a general video codec works and then we will describe the modifications needed for performing sprite-based compression in more details.

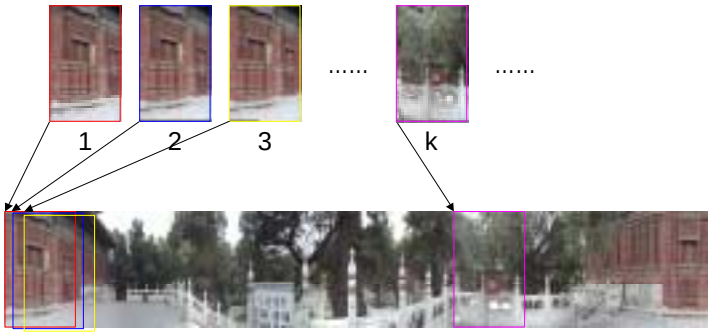


Figure 17 Examples of IBR images and the generation of sprite

IV.1 GENERAL VIDEO CODEC

Figure 18 shows the block diagram of a general video codec [7]. Each image can be coded either in the intra or inter mode. In the intra mode, each switch is connected to the top node and the feedback loop is not used. Discrete Cosine Transform (DCT) and quantization are performed on the current image directly and the result is passed to an entropy encoder. The decoder reverses the process to reconstruct the images. In the inter mode, each switch is connected to the bottom node. The prediction is made by motion estimation (ME) which is the process of finding a motion vector that gives a good match between a block on the current image and a region in a previously decoded image. Motion compensation (MC) is then performed to use the best-match region as the prediction and subtract it from the original block in the current image to produce the residue. DCT and quantization are performed on this residue and then the result is passed to the entropy encoder. On the decoder side, the process is reversed except that motion estimation is not required.

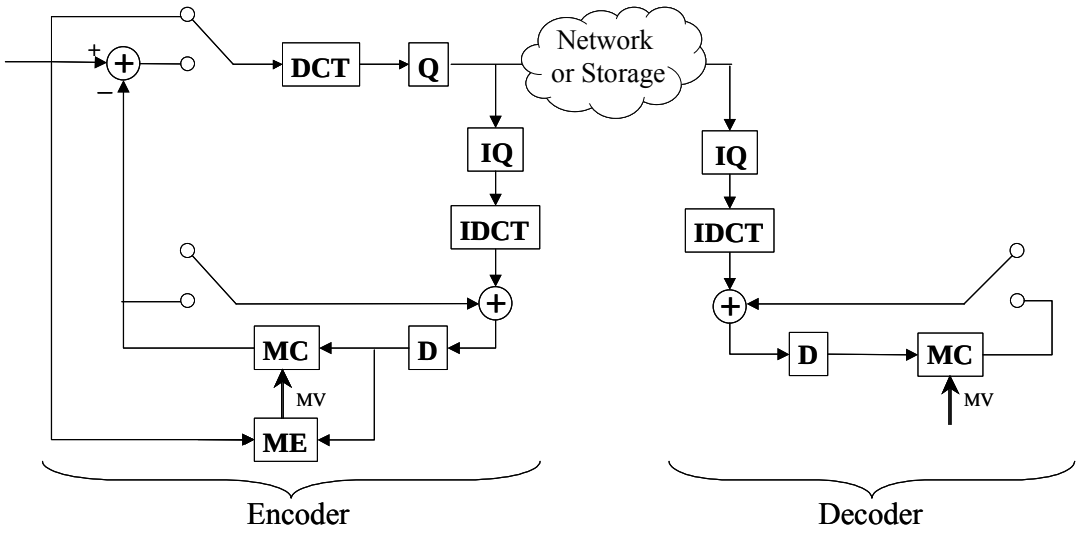


Figure 18 Block diagram of a general video codec

In our application, the acquired images have high correlation. As a result, intra coding does not yield optimal compression. On the other hand, inter coding takes advantage of exploiting the correlation between successive images by using the previously decoded image to predict the current image, thus the result yields a higher compression ratio. However, the problem with inter coding is that the current image depends on the previous decoded image. Unlike video coding, in image-based rendering applications the images to be decoded are not in a fixed order. For example, consider the two cases where the new views are generated by rotating a virtual camera clockwise in one case and counter-clockwise in another case. These two cases represent two different sequences of the same set of images. Decoding will be very inefficient if we use inter coding because of the lack of random access to each individual image.

In order to maintain the random access nature, yet at the same time to result in a good compression ratio by exploiting the correlation between images, we will use a sprite-based compression scheme that is described in the next section.

IV.2 GENERATION OF SPRITE

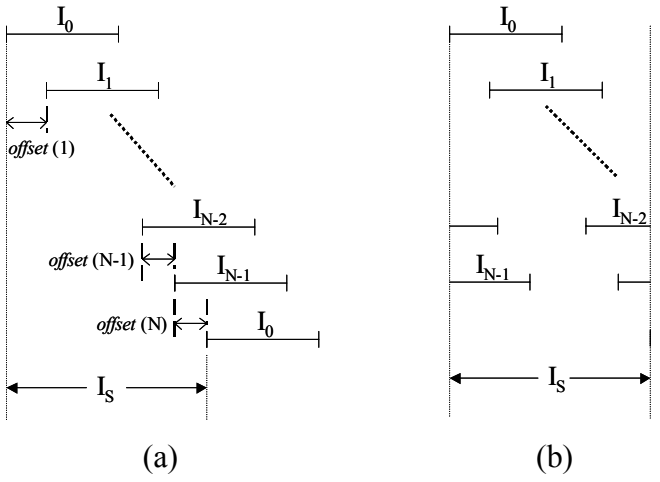


Figure 19 (a) Finding the offsets (b) Constructing the sprite by averaging

The first step in sprite-based compression is the generation of a sprite. To generate the sprite, the images are assumed to align vertically and successive images are shifted horizontally until the mean absolute difference (MAD) [10] between every two successive images, including between the last image and the first image, in the overlapping area is minimum. The amount of overlap between successive images is constrained to be larger than a prescribed threshold to avoid the trivial solutions. We refer to these horizontal shifts as *offsets* and the process of finding the offsets is shown in Figure 19(a). The (n+1)-th acquired image is denoted by I_n, n = 0,1,…,N-1 and the sprite is denoted by I_S. The sprite is composed by averaging the overlapped images using the offsets as shown in Figure 19(b). It can be observed that the generated sprite is a circularly wrapped around image whose size is data dependent. Assume that each acquired image is of size $H\times W$ (H rows and W columns) and the sprite is of size $H\times M$. For the (n+1)-th acquired image I_n, the mapping between each column x of I_n and the corresponding column $f(x)$ of the sprite I_S is defined by the following:

$$f(x) = \mathrm{mod}\left(x + cumulative\_offset(n), M\right) \tag{3}$$

$$\text{where} \qquad cumulative\_offset(n) = \sum_{i=0}^{n} offset(i) \tag{4}$$

$$\text{and} \quad offset(i) = \begin{cases} 0 & i = 0 \\ \text{horizontal shift between } I_{i-1} \text{ and } I_i & i = 1,2,...,N-1 \\ \text{horizontal shift between } I_{N-1} \text{ and } I_0 & i = N \end{cases} \tag{5}$$

After finding the mapping between the column of I_n and the column of I_S, we update I_S from I_n using a weighting function $w(x)$ which depends on the column as shown in the following:

$$I_S\left(f(x), y\right) \leftarrow I_S\left(f(x), y\right) + w(x) I_n(x, y) \qquad x = 0,1,...,W-1 \quad y = 0,1,...,H-1 \tag{6}$$

The total weight tw is also updated as follows:

$$tw\left(f(x)\right) \leftarrow tw\left(f(x)\right) + w(x) \qquad x = 0,1,...,W-1 \tag{7}$$

After all the acquired images have been processed, the sprite is normalized by the total weight as shown in the following:

$$I_S\left(x', y\right) \leftarrow \frac{I_S\left(x', y\right)}{tw(x')} \qquad x' = 0,1,...,M-1 \quad y = 0,1,...,H-1 \tag{8}$$

Figure 17 shows an example of a sprite generated in this manner. The constructed sprite is similar to a 2D panorama except that the images are captured by positioning the camera along a circle instead of fixing the camera at the center.

The sprite is formed by the weighted sum of the overlapping images and there are various ways of assigning weights to $w(x)$. Three weighting functions will be examined: constant weighting function, triangular weighting function and delta weighting function. The constant weighting function is defined by:

$$w(x) = \frac{1}{W} \qquad x = 0,1,...,W-1 \tag{9}$$

The triangular weighting function is defined by:

$$w(x) = \begin{cases} \dfrac{x}{W(W/2-1)} & x \le \dfrac{W}{2}-1 \\ \dfrac{W-1-x}{W(W/2-1)} & x > \dfrac{W}{2}-1 \end{cases} \tag{10}$$

The delta weighting function is defined by:

$$w(x) = \begin{cases} \dfrac{1}{offset} & \dfrac{W}{2}-1-\dfrac{offset}{2} < x \le \dfrac{W}{2}-1+\dfrac{offset}{2} \\ 0 & otherwise \end{cases} \tag{11}$$

These three weighting functions are shown in Figure 20.

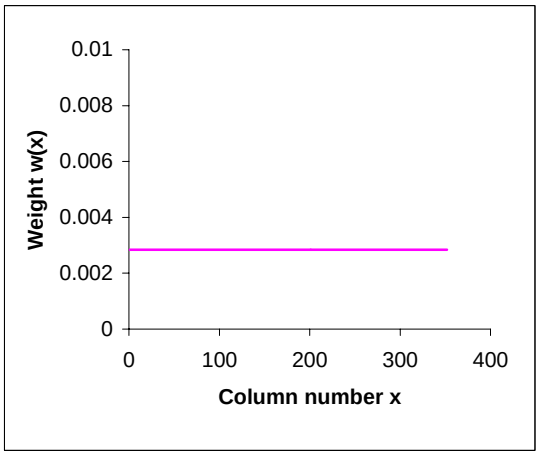
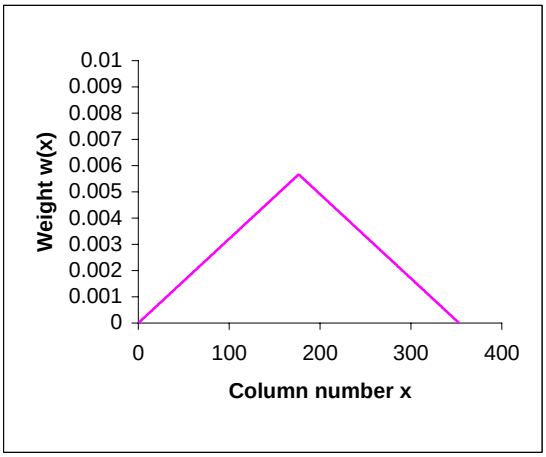


Figure 20 (a) Contant weighting function



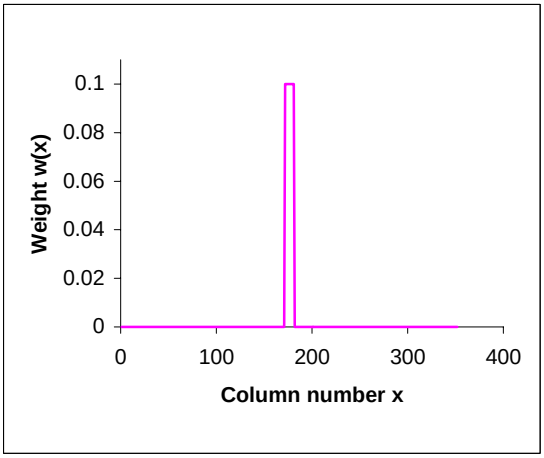


Figure 20 (b) Triangular weighting function (c) Delta weighting function

Figure 21 shows the sprites generated from these different weighting functions. It can be seen that the sprite generated by the constant weighting function is more blurred while the sprite generated by the delta weighting function is the sharpest.

Figure 21 (a) Sprite generated by the constant weighting function

Figure 21 (b) Sprite generated by the triangular weighting function

Figure 21 (c) Sprite generated by the delta weighting function

IV.3 SPRITE-BASED COMPRESSION WITHOUT MOTION COMPENSATION

The codec for sprite based compression can be obtained by modifying the general video codec described in Section IV.2. The general video codec is modified such that a sprite is generated first and then the corresponding part of the sprite is taken as the prediction image. The decoder receives the sprite at the beginning therefore the sprite is present on both the encoder and decoder, and there is no dependency on the previous decoded image. The corresponding part of the sprite is selected as the prediction image according to the camera orientation. Figure 22 shows the block diagram of the modified scheme.

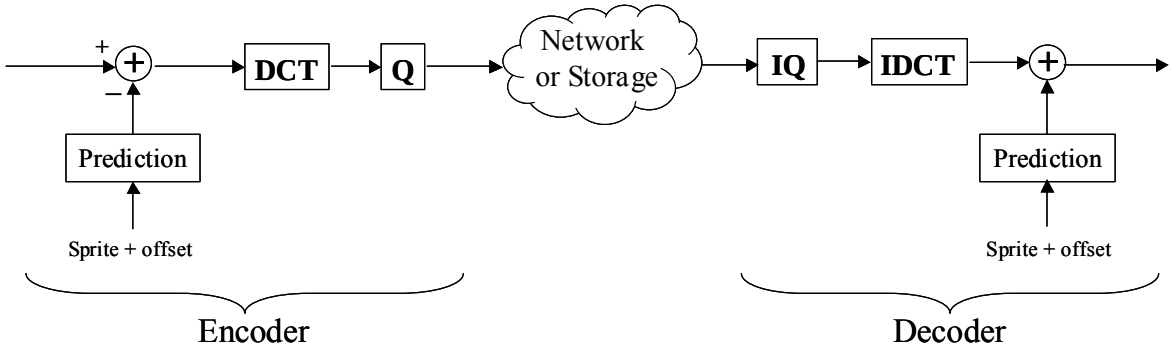


Figure 22 Block diagram of the modified compression scheme using prediction from sprite without motion compensation

IV.4 SPRITE-BASED COMPRESSION WITH MOTION COMPENSATION

Since the sprite is obtained from averaging the original images, prediction of a block in the current image using the exact spatial corresponding block in the prediction image may not give the best match. In order to have a better match, motion compensation can be performed on the prediction from the sprite by estimating the motion vector for each block. Figure 23 shows the block diagram of this scheme.

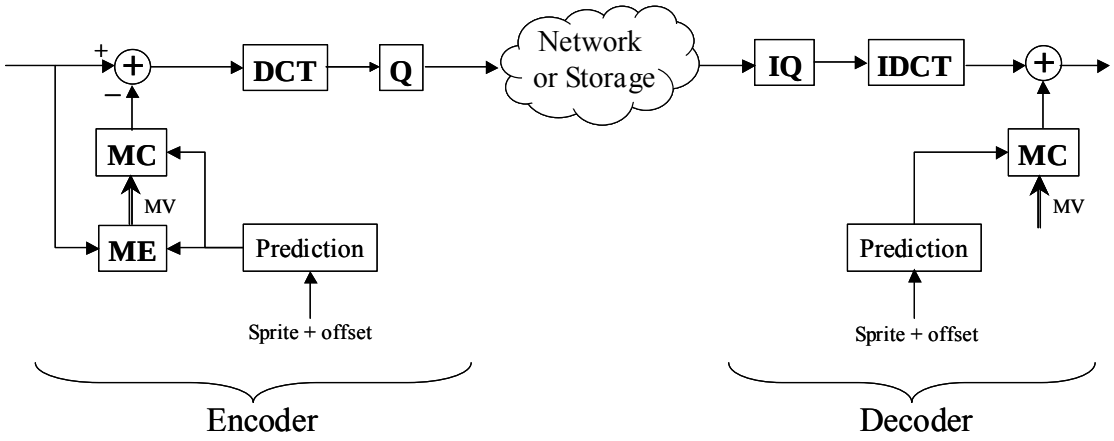


Figure 23 Block diagram of the modified compression scheme using prediction from sprite with motion compensation

IV.5 EXPERIMENT

Four test sequences were used to test our algorithms. Two of the sequences are synthetic scenes as shown in Figure 24 and the other two are real scenes as shown in Figure 25.

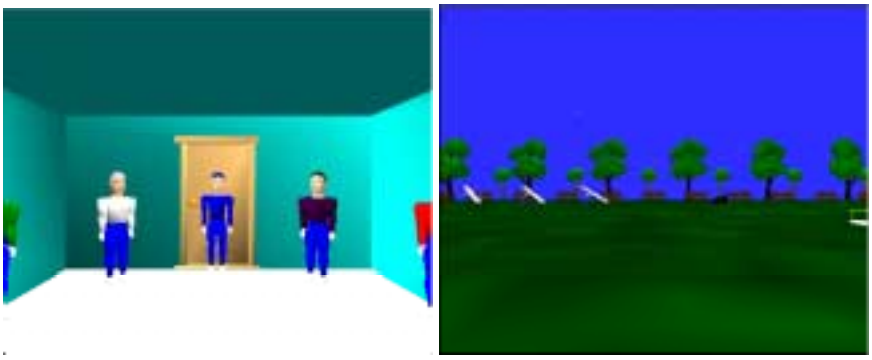


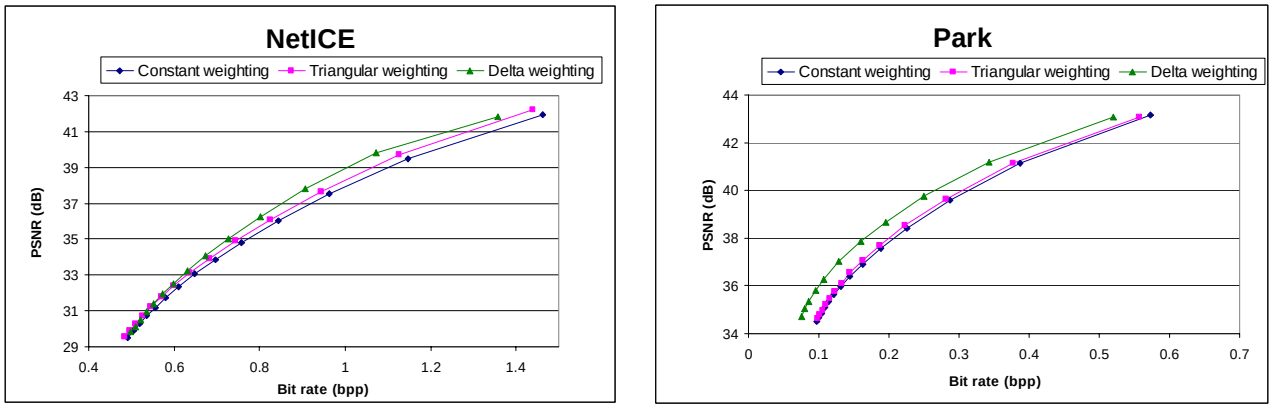
Figure 24 Synthetic test sequences (a) NetICE (b) Park

Figure 25 Real test sequences (a) Kids (b) Kongmiao

Each test sequence consists of images obtained by positioning the camera along a circle. The synthetic sequences are obtained by an inward-looking camera whereas the real sequences are obtained by outward-looking camera. The images are stored in YUV 4:2:0 Common Interface Format (CIF, 352×288 pixels per frame).

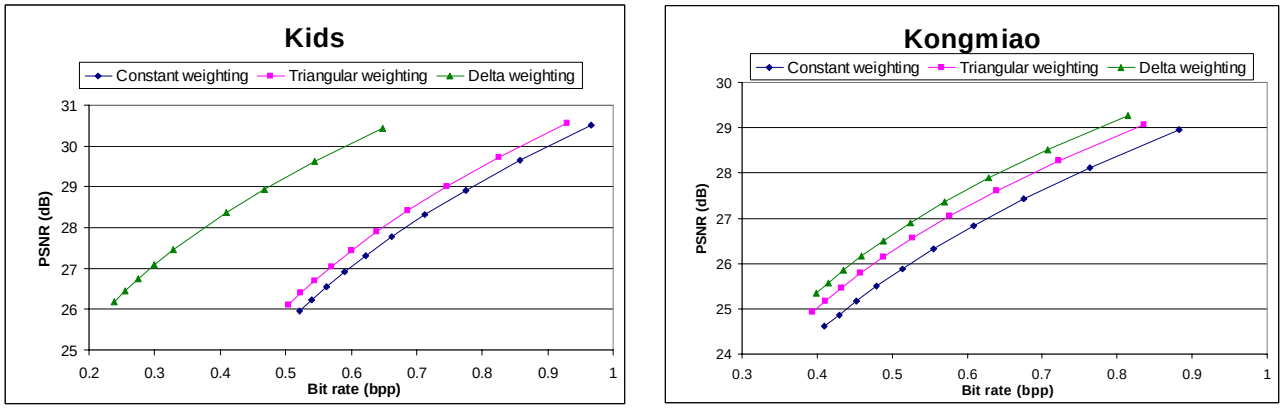
To evaluate performance, rate-distortion curves are generated by varying the DCT quantization levels. Bit-streams are generated under each scheme and the Peak Signal-to-Noise Ratio (PSNR) of the luminance is computed. The rate is measured by the average number of bits per pixel (bpp). Under the sprite-based compression schemes with and without motion compensation, the bit count includes the bit count of the reconstructed sequence, the bit count of the compressed sprite and the bit count of the offsets.

Firstly we examine the rate-distortion curves of the test sequences using the sprites generated by the constant, triangular and delta weighing functions. Figure 26 shows the results for the synthetic sequences and Figure 27 shows the results for the real sequences. For all the sequences, the rate-distortion curves that correspond to the delta weighting function are above those that correspond to the other weighting functions, thus, for the same quality (in terms of PSNR), the delta weighting function offers lower bit rate than the other two weighting functions. As a result, the delta weighting function yields a higher performance in rate-distortion sense. Since the delta weighting function provides the best performance, we will use it to construct the sprite in the subsequent experiments.



(a) (b)

Figure 26 Rate-distortion curves for the synthetic sequences (a) NetICE and (b) Park with different weighting functions



(a) (b)

Figure 27 Rate-distortion curves for the real sequences (a) Kids and (b) Kongmiao with different weighting functions

We will compare the performance of four methods. The first method is intra coding and the second method is inter coding, both using a conventional video codec compliant to H.263 [6]. The third and fourth methods use sprite-based compression. The difference between the third and the fourth methods is that the third method does not include motion compensation but the fourth method does.

The resulting rate-distortion curves are shown in Figure 28 and Figure 29. For both the synthetic sequences as shown in Figure 28 and the real sequences as shown in Figure 29, it can be seen that for the same PSNR value, inter coding results in the lowest bpp. Sprite-based compression with motion compensation requires a smaller bit rate than sprite-based compression without motion compensation. This shows that by using sprite-based compression with motion compensation, it is possible to find a region in the sprite prediction image closely matched to a block in the original image. On the other hand, sprite-based compression without motion compensation performs worse than intra coding. Since the results show consistency among all the sequences, this shows that our proposed compression algorithm is robust to different types of scenes. In particular, it can be observed that for real sequences as shown in Figure 29, the performance of the sprite-based compression with motion compensation is very close to inter coding. This shows that the performance of our proposed algorithm is not affected by scene complexity, since the real sequences contain a lot of detail (i.e. textures) yet the performance of our proposed algorithm is still comparable with inter coding.

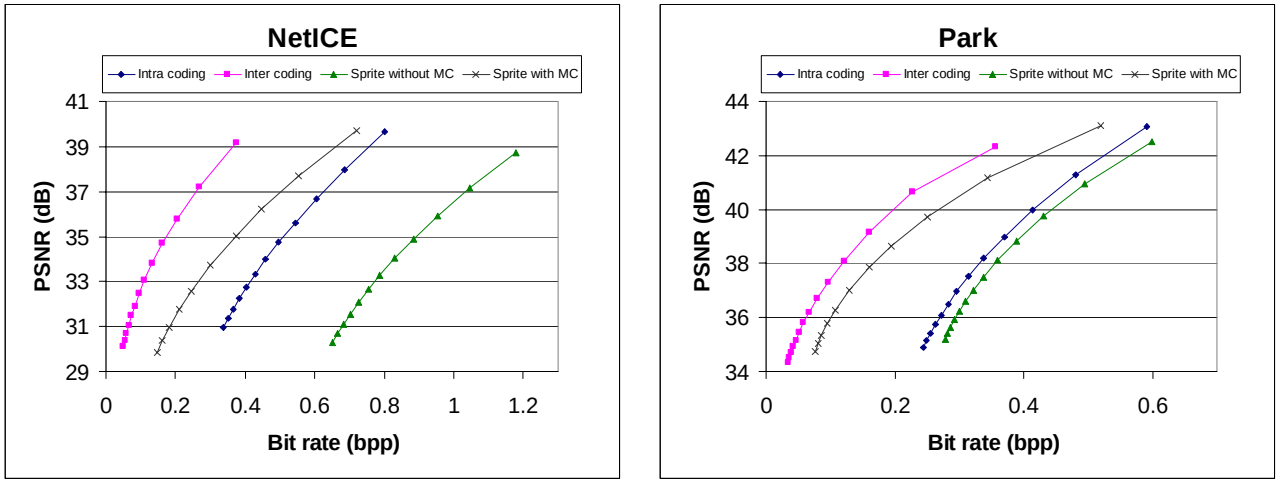


Figure 28 Rate-distortion curves for the synthetic sequences (a) NetICE and (b) Park

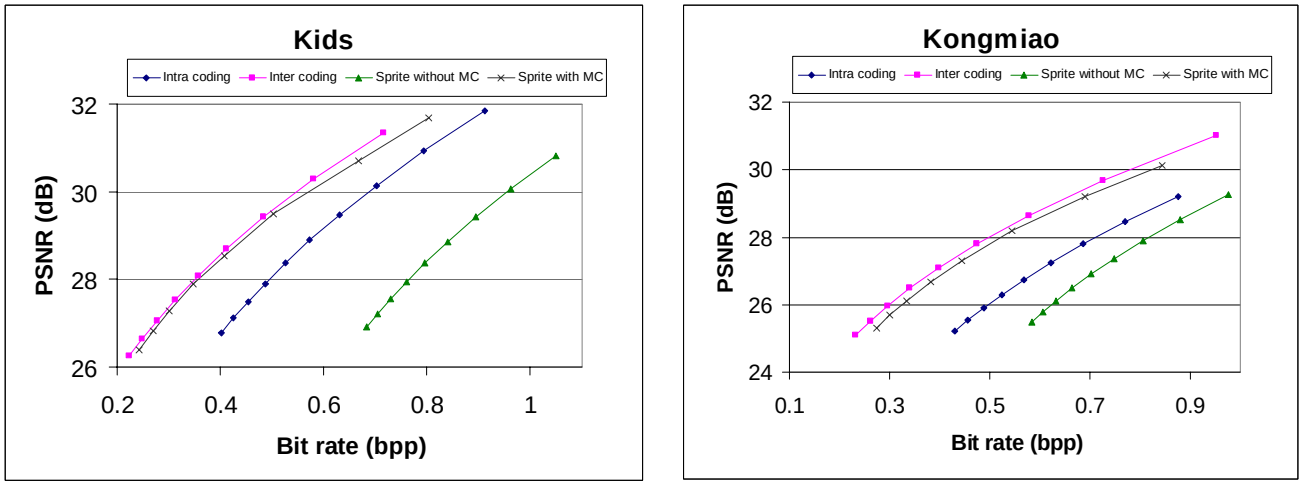


Figure 29 Rate-distortion curves for the real sequences (a) Kids and (b) Kongmiao

V. SPRITE-BASED COMPRESSION ENHANCEMENT

In the previous section we describe the implementation of the basic sprite-based compression scheme. In this section we will propose some enhancement in several components to reduce the amount of computation and to provide better compression performance.

V.1 WINDOW SIZE FOR SEARCHING OFFSET

As mentioned in the previous section the offset is found by minimizing the MAD of an overlapping area between successive frames. To compute the MAD, a window is moved around the next frame and it is compared with a fixed window in the current frame, as indicated in Figure 30. The rectangles with solid boundaries represent the windows in the current frame to be matched and the rectangles with dashed boundaries represent the windows in the next frame used to match the windows in the current frame. The arrows indicate that the windows in the next frame are moved horizontally in order to search for the best match. If the window size is too big then the amount of computation will be large. On the other hand, if the window size is too small, then the offset will be more sensitive to noise and may not reflect the real horizontal shift.

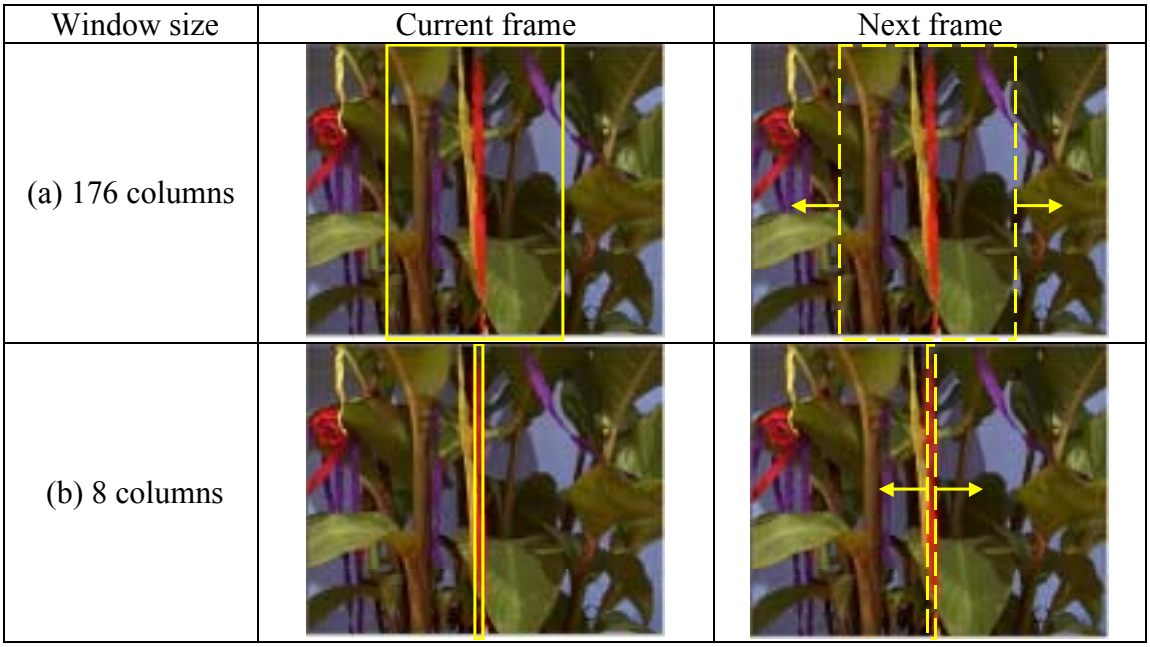


Figure 30 Finding offset using a large window and a small window

We perform an experiment to compare the computation time for the construction of sprite and the compression process using windows with different sizes (number of columns) while maintaining the same bit rate and quality (in terms of PSNR). The computation time is the longest when the window size is the biggest (176 columns). As the window size decreases, the computation time also decreases and it does not vary too much as the window size gets smaller. For the real sequences, the computation time for the window size of 176 columns is 25% less than the computation time for the window size of 8 columns. For synthetic sequences, the computation time for the window size of 176 columns is 63% less than the computation time for the window size of 8 columns. In the subsequent experiments, the window size of 8 columns will be used to search for the offset.

V.2 STRIPE MOTION COMPENSATION

So far in our sprite-based compression approach, the reference frame is generated from a single offset for each frame, therefore the reference frame generation from the sprite can be considered as global motion compensation. Here we are assuming that the horizontal translations for all the columns do not vary too much within a frame such that a single offset is good enough to model for all the horizontal translations within a frame. Since the horizontal translation, i.e., global motion vector, is related to the depth of the objects and there may exist objects with various depths in a frame, we may want to use more than a single value to model these global motion vectors. Although motion estimation is performed in the codec, the search range as specified in the codec may not be large enough to provide good prediction. Therefore, when composing a reference frame from the sprite, instead of just compensating the global motion for the entire frame, we would first divide each frame into stripes which are groups of columns and then compensate the motion for each stripe as shown in Figure 31.

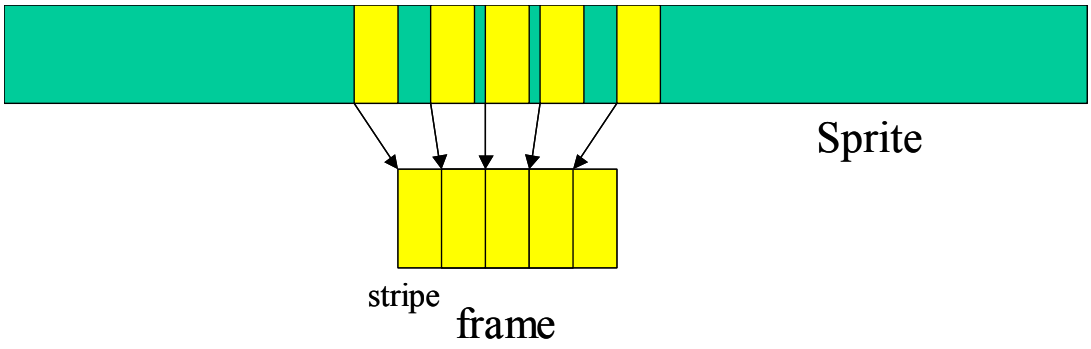
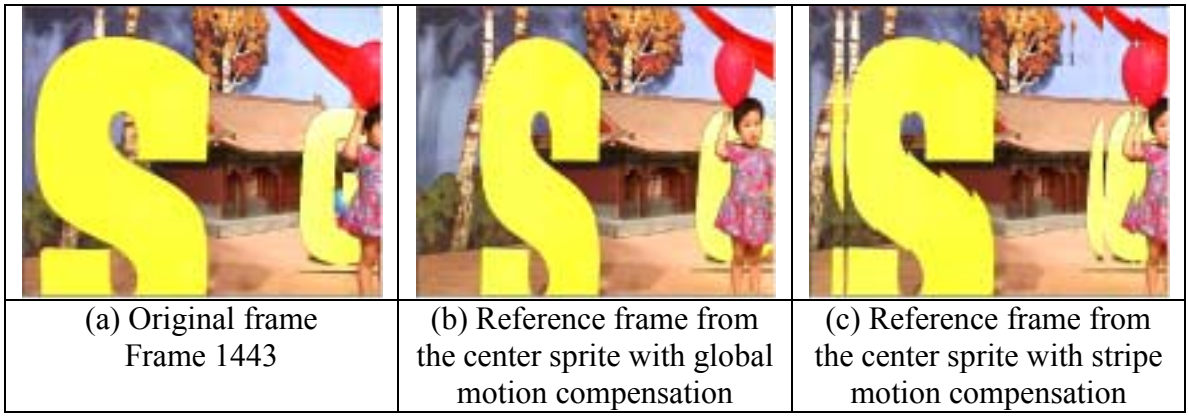


Figure 31 Stripe motion compensation



(a) Original frame Frame 1443	(b) Reference frame from the center sprite with global motion compensation	(c) Reference frame from the center sprite with stripe motion compensation

Figure 32 Comparison between global motion compensation and stripe motion compensation

Figure 32 shows an example of a reference frame generated from global motion compensation using a single offset and a reference frame generated from stripe motion compensation using multiple offsets for each frame. It can be seen that objects like the letter 'S' and the kid demonstrate a relatively larger motion and distortion between the original frame (Figure 32(a)) and the reference frame with global motion compensation (Figure 32(b)). If we compensate for the motion for each stripe when constructing the reference frame (Figure 32(c)), then it can be seen that the resulting reference frame matches more closely to the original frame. The tradeoff with using motion compensation for each stripe is that there is

an overhead for storing additional motion vectors. However, since only a single motion vector is stored for each stripe, the overhead is only a small percentage compared with the bit rate required to represent the content of the frames.

There are two parameters in searching for the stripe motion vectors: the stripe size and the motion vector search range. If the stripe size is too big, then the motion vector may not be representative enough for all the motions within the same stripe. If the stripe size is too small, then the search result is more sensitive to noise. On the other hand, the motion vector search range should be large enough so that the stripes with large motions can be compensated. However, the search range should also be limited otherwise the overhead required to encode the motion vectors will become significant.

We perform an experiment to analyze the optimal combination between the stripe size and the motion vector search range. Firstly the sprite is compressed and then decompressed. Each frame from the original sequence is divided into M stripes. Starting with the center stripe, we search along the decompressed sprite around the region specified by the offset. Then for each neighboring stripe, we limit the search range to be $[mv_{pres}-dmv, mv_{pres}+dmv-1]$ where mv_{pres} is the motion vector of the neighboring previously searched stripe and dmv specifies the relative search range. The motion vector that minimizes the MAD is chosen. Note that the same sprite is used to generate the reference frame and the same quantization step size is used under different stripe sizes and motion vector search range, thus keeping the quality constant. For the four sequences, the minimum bit rate is achieved with different combinations of stripe sizes and relative motion vector search range due to the variation of the motion parallax effects. For example, the minimum bit rate is achieved for the sequence Kids with $M = 16$ and $dmv = 16$ whereas for the sequence Kongmiao the minimum bit rate occurs when $M = 8$ and $dmv = 2$. By performing this motion compensation for the stripe, the bit rate reduction is between 3% and 9% for the four test sequences compared with the case when this approach is not used.

Using the combination of the stripe size and relative motion vector search range that results in the best bit rate, we perform the experiment using this stripe motion compensation for the test sequences and compare it with the case without stripe motion compensation. The stripe motion compensation gives better performance for all the test sequences. When stripe motion compensation is performed, the improvement for the four test sequences is between 0.2dB and 0.6dB in PSNR.

V.3 MOTION COMPENSATION USING A LARGE REFERENCE FRAME

In the previous section we perform motion compensation to compensate for large motion for each stripe. The same idea can be extended to motion compensation for large motion for each macroblock (MB). Essentially we would like to combine stripe motion compensation and the local motion compensation performed by the codec for each macroblock in order to search for a better match. This method is shown in Figure 33. Using a better match will reduce the number of bits required to code the residue while the tradeoff is that more overhead is also required to code the motion vectors for each macroblock.

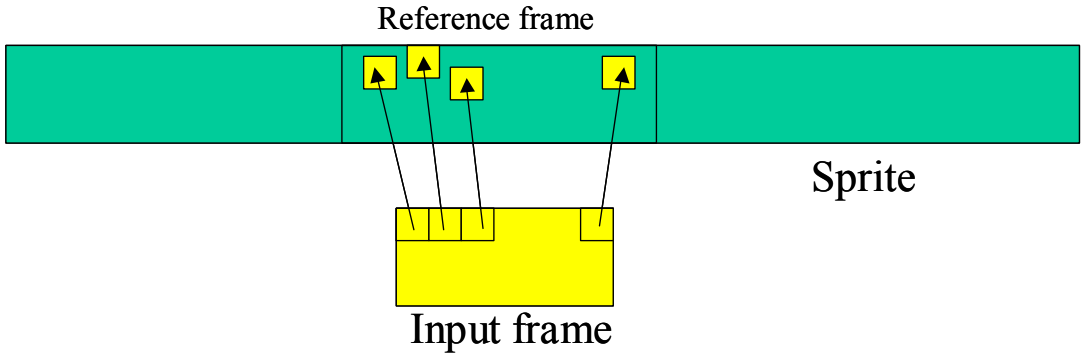


Figure 33 Motion compensation with large reference frame

When we compare the performance for the motion compensation with a large reference frame and the performance for the stripe motion compensation method mentioned in the previous section, it is found out that the two methods yield similar performance for the sequences Kongmiao and NetICE but the motion compensation with a large reference frame performs better than the stripe motion compensation for the sequences Kids and Park. This is because the former two sequences have more uniform horizontal motion for each macroblock in the same stripe whereas for the latter two sequences there are variations in the motion for each macroblock in the same stripe. In particular, the performance gain for Kids is 1.4dB and the gain for Park is 0.5dB in PSNR.

V.4 MULTIPLE REFERENCE FRAMES FROM MULITPLE SPRITES

The sprite is generated by taking the center columns of each input frame as specified by the delta weighting function. As a result, we would expect that the prediction from this sprite near the center columns is good and the prediction towards the end columns is not as good. In order to verify this argument, we perform an experiment to compare the bit rate required to encode the macroblocks from different stripes for all the input frames. As shown in Figure 34, each input frame is divided into stripes where each stripe consists of a vertical list of macroblocks, i.e., 16 columns. We sum up the bit rate required to encode the residue for each macroblock on the same stripe for all the frames and plot the result in Figure 35.

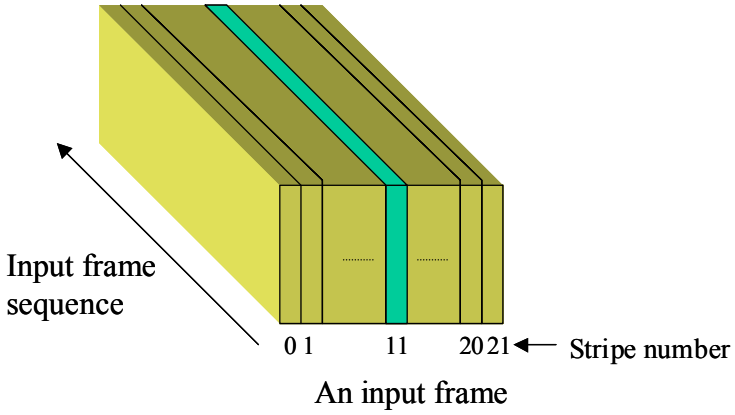


Figure 34 Division of the input frame sequence into stripes

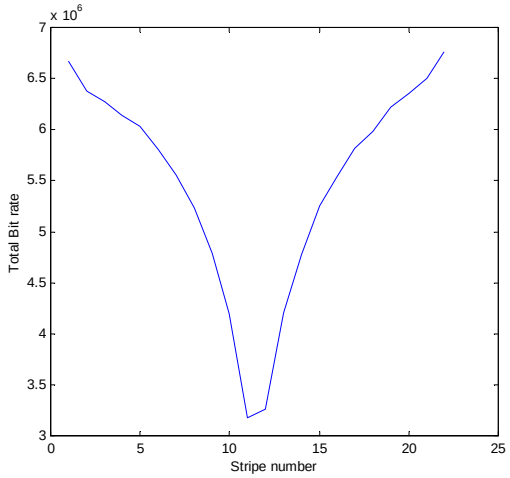


Figure 35 Analysis of the prediction from the sprite constructed from center columns

In Figure 35, the horizontal axis is the stripe number and the vertical axis is the total number of bits required to code the residue of the stripe using the prediction from the sprite constructed from the center columns of the input frames. Figure 35 shows that the bit rate of the central stripe is the minimum, and it increases monotonically towards the end stripes in either direction. In fact, the bit rate required to encode the residues at the end stripes is twice as much as the bit rate required to encode the residues at the center stripe. As a result, we need to find a better prediction for the stripes that are far away from the columns used to construct the sprite. It is then intuitive to generate the reference frame using multiple sprites constructed from columns at various locations. Although there will be more overhead to encode the additional sprites, the reduction in bit rate of residue due to better prediction can still lead to better performance.

There are various ways of picking the columns to construct multiple sprites. For example, when we use three sprites, Figure 36 shows three example configurations which specify which columns are used to construct the three sprites. These column numbers correspond to the peak of the delta weighting function whose width is equal to the offset as mentioned in Section IV.2. Three sprites are generated by three different delta weighting functions and they are used to predict the input frame. For each macroblock of the input frame, the sprite used for the prediction is chosen adaptively. The sprite containing the macroblock that minimizes the MAD will be chosen.

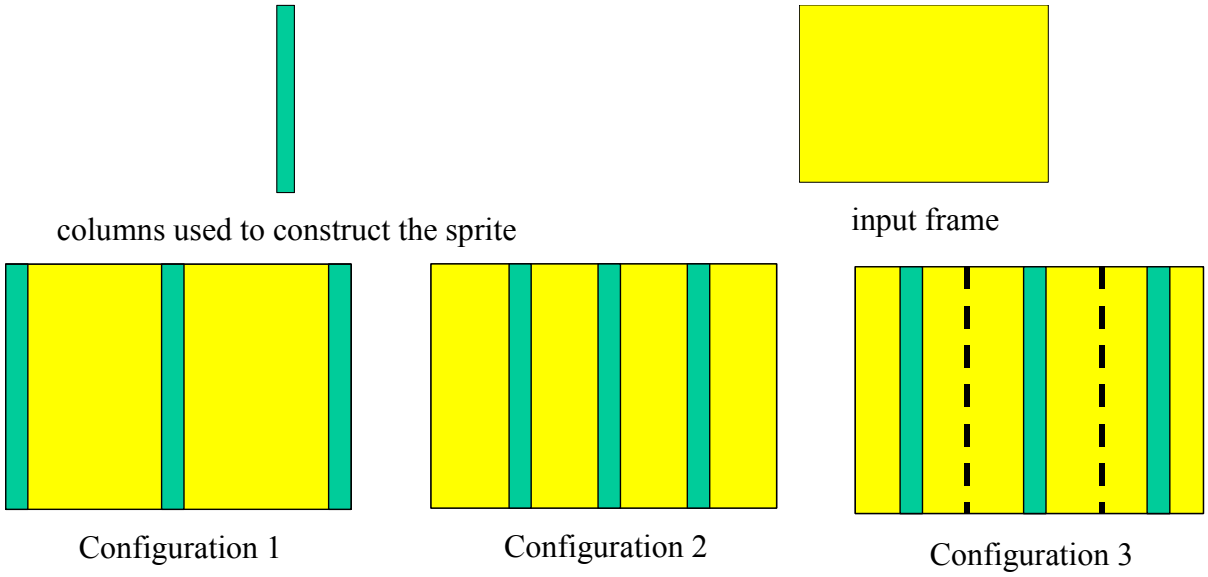


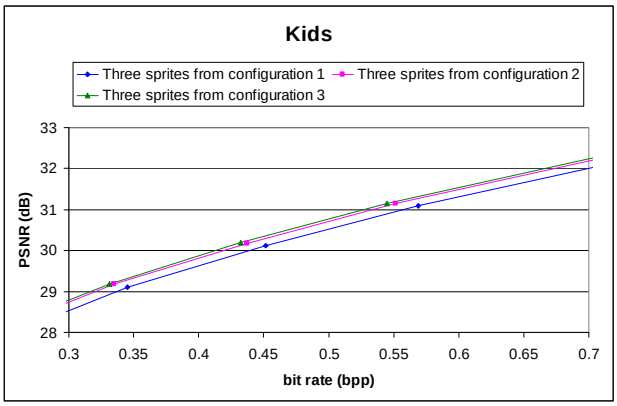
Figure 36 Three example configurations for generating three sprites

All the three configurations contain the sprite which is generated from the center columns. In configuration 1, the two other sprites are generated from the end columns. These sprites generated under configuration 1 capture the maximum amount of motion parallax information since the end columns correspond to the light rays with maximum angular deviation for each frame. In configuration 2, the two other sprites are generated by dividing the frame into four equal areas and picking the columns near the three dividing boundaries. In configuration 3, the two other sprites under configuration 3 are generated by dividing the frame into three equal areas and picking the columns near the centers of each region. These sprites under configuration 3 yield the minimum average distance between the columns to be predicted and the nearest column used in generating the prediction. Figure 37 shows a reference frame generated by each of these three configurations. In this example, it can be seen that the kid's face on the reference frames from configuration 1 and configuration 2 is not very clear whereas it looks good on the reference frame from configuration 3.

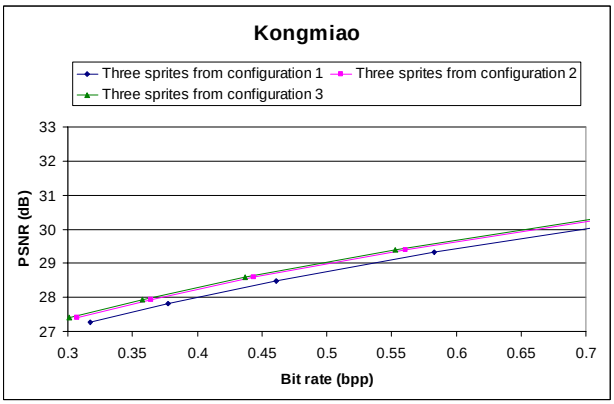
(a) Reference frame from config. 1 using three sprites	(b) Reference frame from config. 2 using three sprites	(c) Reference frame from config. 3 using three sprites

Figure 37 Reference frame generated by the three configurations using three sprites

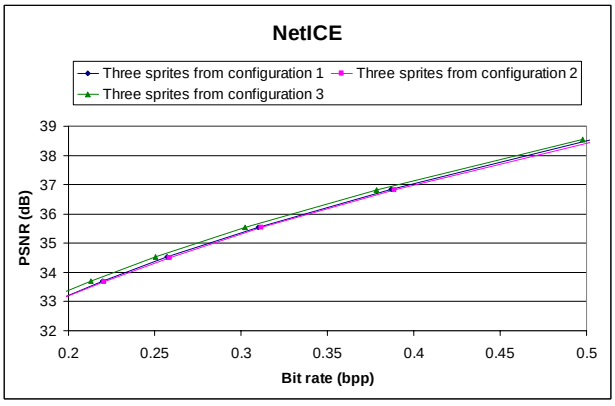
The performance under these three configurations is show in Figure 38. It can be observed that configuration 3 provides the best performance for all the test sequences. The difference between the three configurations is not very noticeable for the sequence NetICE and it shows that the parallax effect is not so strong in this sequence therefore the sprites generated by different delta weighting functions in these three configurations do not deviate too much. The performance gain for the other sequences is between 0.2dB and 0.3dB in PSNR.



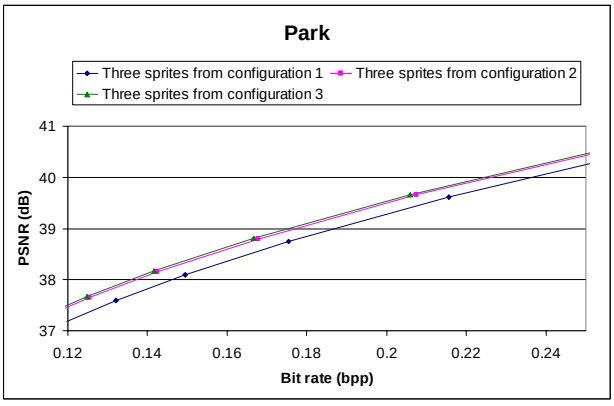
(a)



(b)



(c)



(d)

Figure 38 Comparison of the performance for the test sequences from the three configurations using three sprites

As a result of the previous experiment, we know that if we want to generate *N* sprites, we first divide the frame into *N* equal regions and pick the columns near the center of each region. Using more sprite will result in the reduction of the residue bit rate but at the same time will increase the overhead required to encode the sprites. We perform an experiment to compare the performance using multiple sprites and the result as shown in Figure 39:

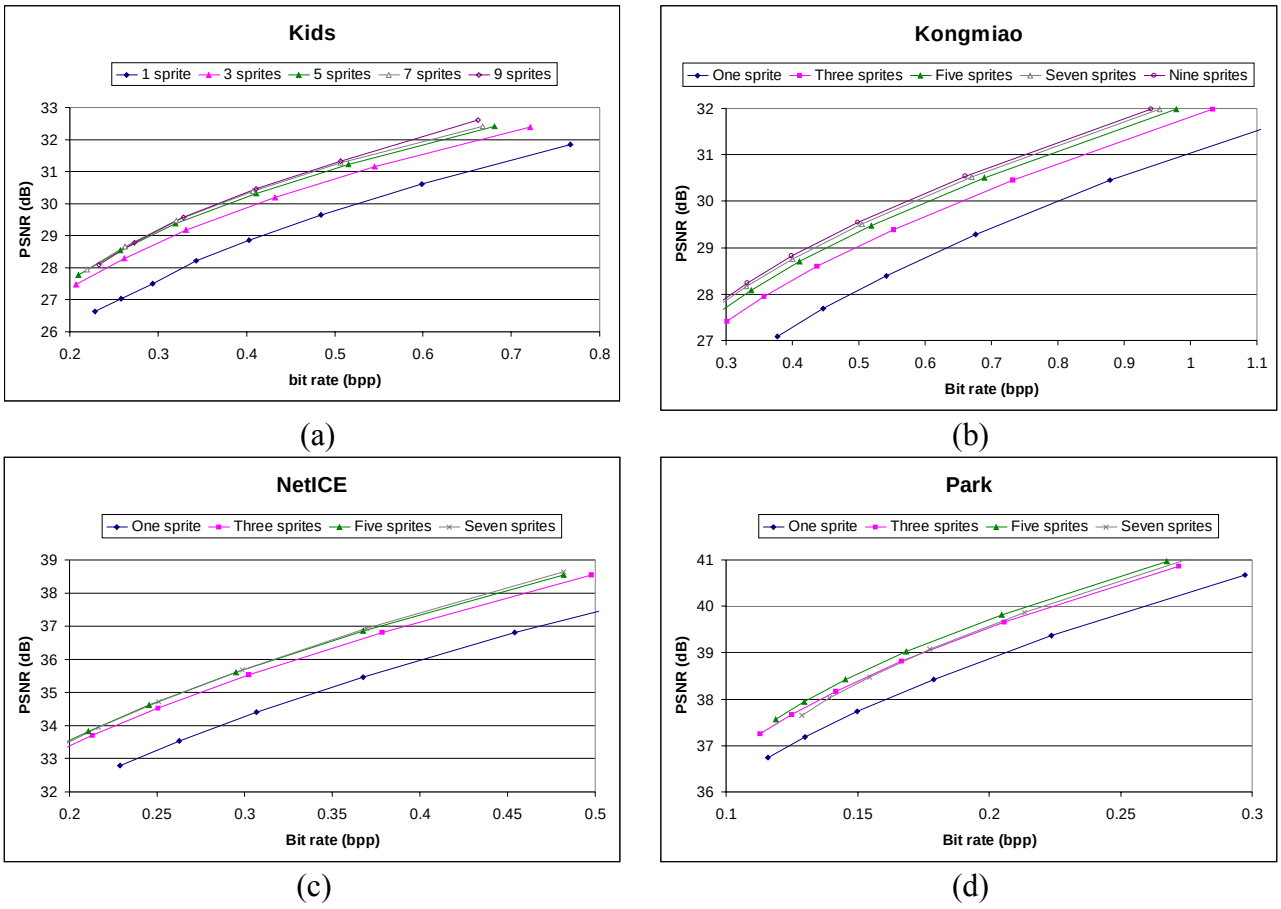
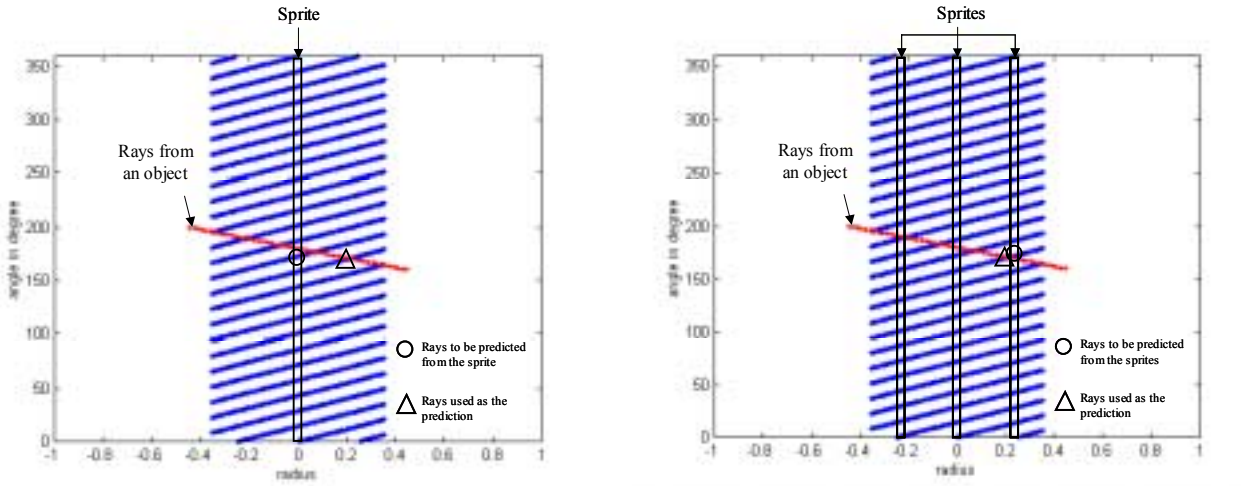


Figure 39 Comparison of the performance for the test sequences with multiple sprites

From Figure 39, it can be seen that initially the performance increases with an increase in the number of sprites. However, the gain resulting from each additional sprite is diminishing. For the sequences Kids and Kongmiao, the gains become saturated when nine sprites are used. The overall performance using nine sprites demonstrates a gain of 1.5dB compared with the performance using only one sprite. For the sequence NetICE, the gain becomes saturated when seven sprites are used. The overall performance using seven sprites demonstrates a gain of 1.5dB compared with the performance using only one sprite. For the sequence Park, the gain actually decreases when seven sprites are used instead of five sprites. The overall performance using five sprites demonstrates a gain of 0.8dB compared with the performance using only one sprite.

V.5 ANAYSIS OF THE PREDICTION WITH MULTIPLE SPRITES IN THE LINE SPACE

The prediction using multiple sprites can be visualized in the line space. Figure 40 shows the comparison between the prediction using one sprite and the prediction using three sprites. Recall from Section III that the array of tilted grid points in the line space represent the light rays of the acquired input images. In Figure 40(a), the tilted grid points lying inside the rectangle form the sprite and similarly in Figure 40(b), the collection of the tilted grid points inside each rectangle represents one of the three sprites. These are the light rays that are required to be stored and they are used to predict the light rays of the input images. In Figure 40(a) and (b), there is a line that corresponds to the rays emitted from an object. The intersections between this line and the array of tilted grid points form similar light rays. The triangles in Figure 40(a) and (b) represent the light rays of the input image that need to be predicted from the sprite(s). The light rays of the sprite(s) are searched and the ones that correspond to the best match will be used as the prediction. These prediction light rays are represented by the circle in Figure 40(a) and (b). It can be seen from Figure 40(b) that when three sprites are used, there are more choices for the prediction and therefore a closer match can be found compared with the case with only one sprite. By similar arguments, there will be more choices of light rays when more sprites are used and it will result in better prediction. However, the tradeoff is that there are more light rays needed to be stored for the sprites.



(a) Prediction using one sprite in the line space (b) Prediction using three sprites in the line space

Figure 40 Comparison of the prediction using one sprite and three sprites in the line space

V.6 COMPARISON WITH REFERENCE BLOCK CODEC

In [12], Zhang and Li proposed to use Reference Block Codec (RBC) to compress the acquired images. Under their RBC scheme, the images are divided into anchor frames which are intra-coded and predicted frames which are coded using the prediction generated from the neighboring anchor frames. In order to yield a high performance in rate-distortion sense, the quantization step size of the anchor frames is set to be smaller than the quantization step size of the predicted frames. The problem with this approach is that there is a large variation in quality across neighboring frames. Figure 41 and Figure 42 illustrate the variation in quality for the first 150 frames of the sequence Kids compressed by RBC and by sprite-based compression with nine sprites respectively. In this example, the test sequence is compressed with roughly the same bit rate by these two methods. As shown in Figure 41, the frames that yield high PSNR are the anchor frames. The predicted frames closer to the anchor frame have a higher quality and the predicted frame farther from the anchor frame have a lower quality. On the other hand, with sprite-based

compression, the quality across neighboring frames does not deviate too much as shown by the smooth curve in Figure 42.

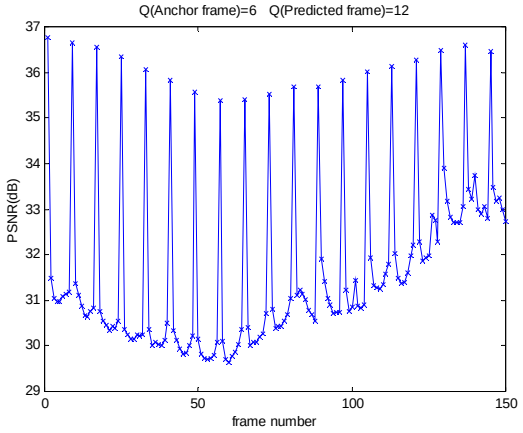


Figure 41 Quality of the sequence Kids compressed by RBC for the first 150 frames

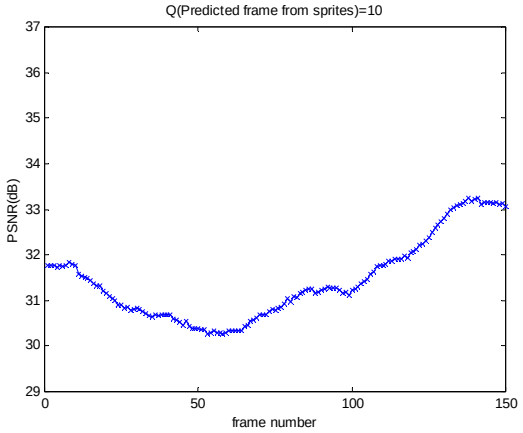


Figure 42 Quality of the sequence Kids by sprite-based compression for the first 150 frames

Since the scene complexity is very similar between neighboring frames, therefore it is reasonable to compress the frames such that neighboring frames have similar PSNR. To compare our algorithm with RBC under this assumption, we decrease the quantization step size of the predicted frames such that the quality of the predicted frames is the same as the quality of the anchor frames when performing RBC. After searching for the quantization step size that will yield a smooth quality of the compressed sequence (we call this scheme smoothed RBC), we compare the performance in rate-distortion sense between our proposed sprite-based compression and smoothed RBC. The comparison between these two methods for the test sequences is shown in Figure 43. It can be seen that sprite-based compression with multiple sprites has a better performance than smoothed RBC especially at low bit rate for the sequences Kids, Kongmiao and Park. However, the performance of the sprite-based compression with multiple sprites is worse than smoothed RBC for the sequence NetICE. This may be due to the fact that the synthetic sequence NetICE does not have too much motion parallax effects and the background does not have any texture. As a result, a very good prediction frame can be generated by the anchor frames from smoothed RBC and the coding gain from the sprite-based compression approach does not offset the bits required to encode the sprites for this simple scene.

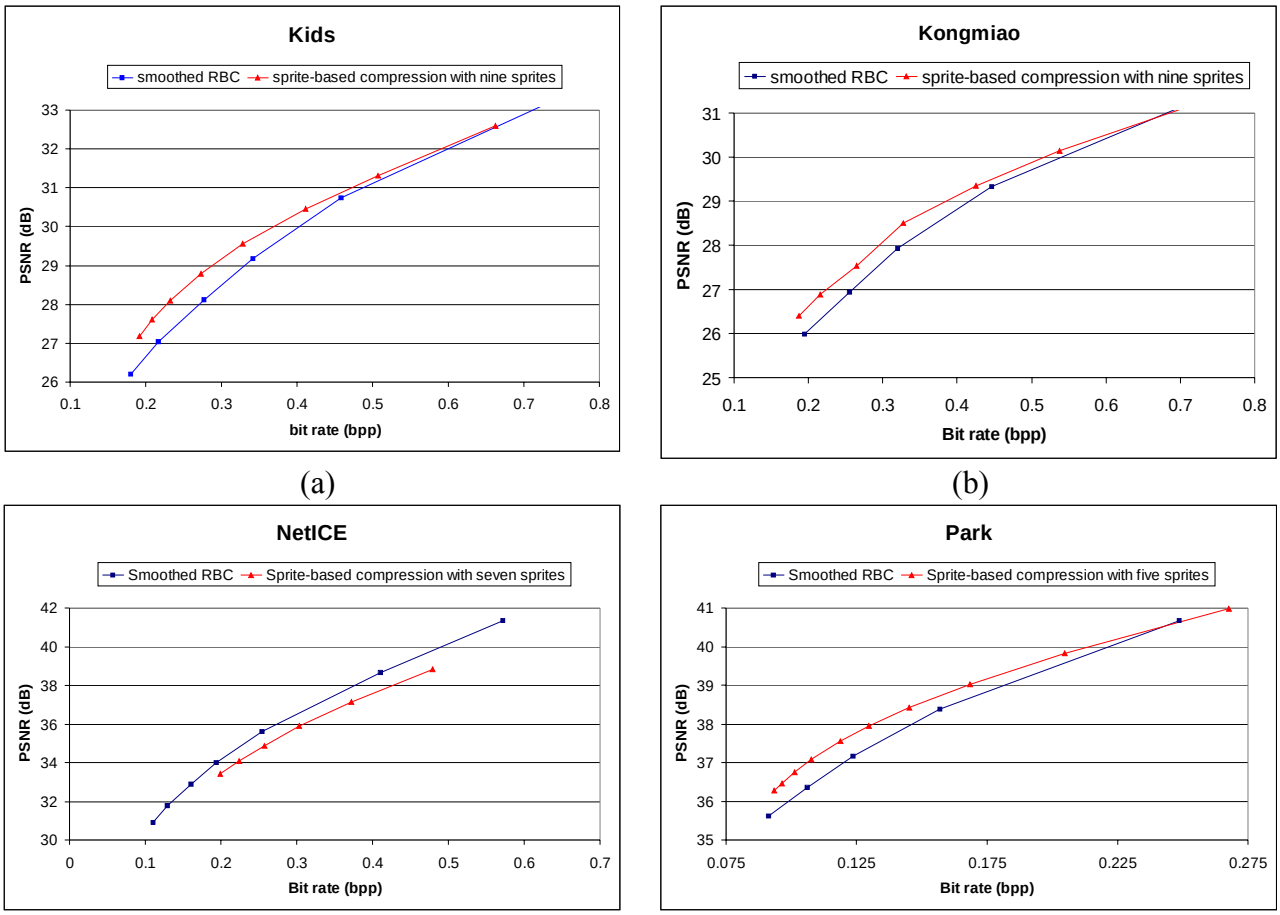


Figure 43 Comparison of the performance for the test sequences between the sprite-based compression with multiple sprites and smoothed RBC

VI. CONCLUSION AND FUTURE WORK

In this paper, we first studied several camera trajectories for acquiring images with some evaluation criteria. Then we analyze the rendering of new views using the line-space representation. We discussed how caching together with frame-level random access can provide an efficient and fast rendering of the scene. We also presented a new sprite-based compression scheme under which the sprite is constructed first and its corresponding part is served to predict the original image with or without motion compensation. It has been shown that the delta function is the best weighting function for the generation of sprite. It is concluded that the sprite-based prediction with motion compensation provides a better compression ratio than intra coding while it has the advantage of frame-level random access over the inter coding. Sprite-based compression can be enhanced by motion compensation with a larger reference frame. In addition, compression with multiple sprites further improves the performance in rate-distortion sense. We compared the sprite-based compression with smoothed RBC and demonstrated a better performance for three out of four test sequences given that the quality of the compressed sequence is smooth using both methods.

We will continue our research along the issues about the compression of images in image-based rendering applications. For example, instead of performing the caching based on frame-level random access, we may consider performing the caching based on stripe-level random access to examine the tradeoff between the overhead needed to provide an extra level of random access and the savings in memory to store the required light rays. In terms of compression, we will study how the global motion compensation, stripe motion compensation and motion compensation using a larger reference frame at the macroblock level can be combined in order to optimize the bit rate required to store the overhead motion vectors while maintaining a low bit rate to code the residue from the sprite prediction.

VII. ACKNOWLEGDEMENT

We would like to thank Dr. Harry Shum from Microsoft China for providing the test sequences Kids and Kongmiao. We would also like to thank Dr. Jin Li and Cha Zhang for fruitful discussions. In addition, we would like to give our special thanks to Ta-Chien Lin from the Advanced Multimedia Processing Lab at Carnegie Mellon University for his help in modifying the motion estimation and motion compensation modules of the H.263 Codec to enhance the sprite prediction.

VIII. REFERENCES

[1] Shum, H-Y. and He L-W. "Rendering with Concentric Mosaic"., *Computer Graphics Proceedings, Annual Conference Series, Proc. SIGGRAPH'99,* pages 299-306, Los Angeles, CA, August 1999.

[2] Adelson E. H. and Bergen J. "The plenoptic function and the early vision". *Computational Models of Visual Processing,* pages 3-20, MIT Press, Cambridge, MA, 1991.

[3] Gortler S. J., Grzeszczuk R., Szeliski R., and Cohen M. F.. "The lumigraph". *Computer Graphics Proceedings, Annual Conference Sereies*, *Proc. SIGGRAPH'96*, pages 43-54, New Orleans, August 1996.

[4] Levoy M. and Hanrahan P. "Light field rendering". *Computer Graphics Proceedings, Annual conference Series*, *Proc. SIGGRAPH'96*, pages 31-42, New Orleans, August 1996.

[5] Chen S.E. "QuickTime VR – an image-based approach to virtual environment navigation". *Computer Graphics (SIGGRAPH'95)*, pages 29-38, August 1995.

[6] Advanced Multimedia Processing Lab, Carnegie Mellon University, CMU H.263+ Decoder, http://amp.ece.cmu.edu/

[7] Video Coding for Low Bit rate Communication, ITU-T Recommendation H.263 Version 2, Jan. 1998

[8] Irani M., Anandan, P., Bergen, J., Kumar, R., and Hsu, S. "Efficient Representations of Video Sequences and Their Applications", *Signal Processing: Image Communication*, Vol. 8, No. 4, pages 327-351, May 1996.

[9] MPEG-4, Overview of the MPEG-4 Standard, ISO/IEC JTC 1/SC 29/WG 11, Melbourne, Australia, October 1999.

[10] Turaga D. and Chen T., "Estimation and Mode Decision for Spatially Correlated Motion Sequences", submitted to IEEE Trans. CSVT.

[11] The Math Forum, an Online Math Education Community Center, *http://www.forum.swarthmore.edu/geometry/blackwell/index.html*

[12] C. Zhang and J. Li, "Compression and rendering of concentric mosaic scenery with reference block codec (RBC)", SPIE Visual Communication and Image Processing (VCIP 2000), Perth, Australia, Jun. 2000.

[13] J. Li, K. Zhou, Y. Wang and H-Y Shum, "A Novel Image-Based Renedering System with a Longitudinally Aligned Camera Array", EuroGraphics'2000, Interlaken, Switzerland, Aug. 2000.