Creating VR Scenes Using Fully Automatic Derivation of Motion Vectors

Kensuke Habuka and Yoshihisa Shinagawa
University of Illinois at Urbana-Champaign

1 Introduction

We propose a new method to create smooth VR scenes using a limited number of images and the motion vectors among them. We will discuss two specific components to simulate a majority of VR scenes: MV VR Object and MV VR Panorama. They provide similar functions to QuickTime VR Object and QuickTime VR Panorama [1]. However, our method can interpolate between the existing images, and therefore, smooth movement of viewpoints is achieved. When we look at a primitive from arbitrary viewpoints, the images of an object associated with the primitive is transformed according to the motion vectors and to the location of the viewpoint.

2 Motion Vectors and Generating Intermediate Images

First, let us consider two images. The first one is called the source image and the second the destination image. Our matching technique solve the correspondence problem between pixels in the source image \((x, y)\) and pixels in the destination image \((x_{\text{dst}}, y_{\text{dst}})\). We call \(v_f(x, y) = (x_{\text{dst}}, y_{\text{dst}}) - (x, y)\) the forward motion vectors that are the motion vectors from the source image to the destination image. Our method also solve for the backward motion vectors \(v_b(x, y)\) from the destination image to the source image. The solution for \(v_b\) is not trivial because of discretization of the domain and occlusions in the two images.

In this paper, we use the correspondences computed by critical-point filters [2] as the motion vectors. The multiresolutional critical-point filters are nonlinear filters and they do not destroy the essential structures of the critical points in the images. There are four types of the critical-point filters: \(\alpha_x \alpha_y p_{(x, y)}^{(m)}\), \(\alpha_x \beta_y p_{(x, y)}^{(m)}\), \(\beta_x \alpha_y p_{(x, y)}^{(m)}\), and \(\beta_x \beta_y p_{(x, y)}^{(m)}\) that preserve stronger critical points such as the peaks, saddle points, and pits. The size of the resulting images is 1/4 of the original. The abbreviations above mean:

\[
\begin{align*}
\alpha_x f_{(x, y)}^{(m-1/2)} &= \max_{0 \leq w \leq 2} I_{(x, y)}^{(m)}(2x + w, y), \\
\alpha_y f_{(x, y)}^{(m-1/2)} &= \max_{0 \leq w \leq 2} I_{(x, y)}^{(m)}(x, 2y + w), \\
\beta_x f_{(x, y)}^{(m-1/2)} &= \min_{0 \leq w \leq 2} I_{(x, y)}^{(m)}(2x + w, y), \\
\beta_y f_{(x, y)}^{(m-1/2)} &= \min_{0 \leq w \leq 2} I_{(x, y)}^{(m)}(x, 2y + w)
\end{align*}
\]

where \(I_{(x, y)}^{(m)}\) is the intensity of the pixel at \((x, y)\) at the \(m\)-th resolution. Then, for each pixel at \((x, y)\) of the source image at each resolution level, the corresponding pixel at \((x_{\text{dst}}, y_{\text{dst}})\) of the destination image is determined so that the associated energy is minimum. The candidate is searched for inside the quadrilateral determined at the upper resolution level.

Once we obtain the pair of motion vectors, we can generate the intermediate image between the source and destination images by interpolation. First, we transform both images using the motion vectors. Transforming an image \(I\) with motion vectors \(v\) means that each square \((x, y), (x + 1, y), (x + 1, y + 1), (x, y + 1)\) is transformed to \((x, y) + v(x, y), (x + 1, y) + v(x + 1, y), (x + 1, y + 1) + v(x + 1, y + 1), (x, y + 1) + v(x, y + 1)\). The color of each pixel in the transformed image is bilinearly interpolated. We denote it by \(\text{transform}(I, v)\).

After both images are transformed, they are blended together using the following expression:

\[
\begin{align*}
&f_{\text{blend}2}(1 - \tau) \text{transform}(I_{\text{src}}, \tau v_f) + \\
&f_{\text{blend}2}(\tau) \text{transform}(I_{\text{dst}}, (1 - \tau) v_b),
\end{align*}
\]

where \(f_{\text{blend}2}\) is a blending function and \(0 \leq \tau \leq 1\) is a blending ratio. \(\tau\) is the relative position between the images. \(f_{\text{blend}2}(0) + f_{\text{blend}2}(1 - a) = 1\), where \(0 \leq a \leq 1\). We used \(f_{\text{blend}2}(\tau) = \frac{1}{2} \sin(\pi \tau - \frac{\pi}{2}) + \frac{1}{2}\).

We can generate the intermediate image at an arbitrary point inside the square formed by four viewpoints and four pairs of motion vectors as well. We call it 2D interpolation.

3 Creating VR Scenes

3.1 MV VR Object
This section describes how to display arbitrary views of an object using a limited number of images. Let us index the image of an object by the camera’s angles $\theta$, $\phi$ and distance $D$ as $I_{\theta, \phi, D}$ in the following.

Suppose there are $N$ images looking at the object from viewpoints on the plane containing the object’s center: $I_{0,0,D_0}$, $I_{\Delta \theta,0,D_0}$, $I_{2\Delta \theta,0,D_0}$, ..., $I_{(N-1)\Delta \theta,0,D_0}$, where $D_0$ is a constant distance, and $\Delta \theta = \frac{360}{N}$ is a constant angle. $I_{N\Delta \theta,0,D_0}$ is equal to $I_{0,0,D_0}$. To generate $I_{0,0,D_0}$ from the images, we use the method that we described in Section 2.

We select an integer $i$ that satisfies $i\Delta \theta \leq \theta < (i+1)\Delta \theta$ and $0 \leq i < N$ in order to select two images from our existing images. Then, we generate the intermediate image $I_{0,0,D_0}$ using $I_{i\Delta \theta,0,D_0}$ as the source image and $I_{(i+1)\Delta \theta,0,D_0}$ as the destination image with two sets of motion vectors between them at the relative position $\tau = \frac{\theta - i\Delta \theta}{\Delta \theta}$.

We synthesize $I_{0,0,D}$ which is the image at an arbitrary distance $D$ as follows. To obtain a ray passing through the viewpoint $P$ of $I_{0,0,D}$ and each image point $Q$ of $I_{0,0,D}$, we need to identify an interpolated image $I_{0+i\Delta \theta,0,D_0}$ containing the ray $PQ$ by calculating the angle $\theta_P$. For real-time rendering using meshes, we first need to obtain the offset $\langle x_f, y_f \rangle$ from the point $(x, y)$ on the grid of motion vectors. We cannot obtain the offset from the image $I_{0+i\Delta \theta,0,D_0}$ because $\theta_P$ is unknown. Therefore, we use the offset $\langle x_f', y_f' \rangle = \tau v_f(x, y) + (x, y) - \langle x_{center}, y_{center} \rangle$ in the image $I_{0,0,D_0}$ as an approximation, where $(x_{center}, y_{center})$ is the center of the image. Then, we approximate $\theta_P'(x, y)$ by $\theta_P' = \arctan \left( \frac{(D-D_0)x_f'}{D_0D} \right)$.

The image $I_{0,0,D}'$ can be obtained using $v_f' = \tau v_f(x, y)$. However, $I_{0,0,D}'$ is distorted with the distortion ratio $\alpha_{\tau} = \frac{D\cos(\theta_P)}{D\cos(\theta_P) - \tau D\sin(\theta_P)}$. The distortion ratio $\beta_{\tau}$ for the $y$ axis is 1 because we have no images or motion vectors along the depression angle $\phi$. Finally, we can compute an approximate forward transform vector: $v_f(x, y) = \left( \begin{array}{cc} \alpha_{\tau} & 0 \\ 0 & \beta_{\tau} \end{array} \right) \tau v_f(x, y)$. The expressions to solve for an approximate backward transform vector $v_{\theta\phi}$ are almost the same, except for changing $v_f$ to $v_b$ and $\tau$ to $1 - \tau$. After $v_f'(x, y)$ and $v_{\theta\phi}(x, y)$ are obtained, we can generate an approximate image for $I_{0,0,D}$ using the following expression:

$$f_{\theta\phi}\text{transf}(1 - \tau)\text{transform}(I_{\Delta \theta,0,D_0}, v_f) + f_{\theta\phi}\text{transf}(\tau)\text{transform}(I_{(i+1)\Delta \theta,0,D_0}, v_{\theta\phi})$$

After the image $I_{0,0,D}$ is obtained, the image is zoomed in or out according to its position. Figure 1 shows the distortion effect caused by changing distance $D$. For ease of understanding, it is shown without zooming.

![Figure 1: The original image from the distance $D_0$ (a) and synthesized images from the distances $10D_0$ (b) and $15D_0$ (c) without zooming.](image)

### 3.2 MV VR Panorama

In this section, we describe the method of seamless panorama scene interpolation. Using this method, users can view the scene in an arbitrary direction from an arbitrary position.

First, we need to take panoramic images at several locations. We used approximately twelve pictures to make a single panoramic image for each viewpoint. There are many stitching methods to make panoramic images. We denote the panoramic image at the position $(p_x, p_y)$ by $P_{(p_x, p_y)}$.

To generate a dense set of panoramic images from the sparse set, 2D interpolation is used. We assume that we have $N \times M$ panoramic images $P_{(0,0)}, P_{(\Delta p_x,0)}, \ldots, P_{(0,\Delta p_y)}, \ldots, P_{(N\Delta p_x,0)}, \ldots, P_{(0,\Delta p_y)}, \ldots, P_{(N\Delta p_x,\Delta p_y)}$, where $\Delta p_x$ and $\Delta p_y$ are constant distances and $N$ and $M$ are integers. To generate an intermediate panoramic image $P_{(i,j)}$, we select integers $i$ and $j$ that satisfy $i\Delta p_x \leq p_x \leq (i+1)\Delta p_x$ and $j\Delta p_y \leq p_y \leq (j+1)\Delta p_y$ in order to determine the four panoramic images to be interpolated. The integers $i$ and $j$ must satisfy $0 \leq i < N-1$ and $0 \leq j < M-1$. Then we interpolate the image $P_{(i,j)}$ using the ratios $\tau_x = \frac{p_x - i\Delta p_x}{\Delta p_x}$ and $\tau_y = \frac{p_y - j\Delta p_y}{\Delta p_y}$.

After panoramic images are synthesized, we need to correct the distortion caused by the camera when making the panoramic images.

### References
