Visual Comput (2007) 23: 707–719
DOI 10.1007/s00371-007-0130-y

ORIGINAL ARTICLE

**Parag Chaudhuri**
**Prem Kalra**
**Subhashis Banerjee**

# Reusing view-dependent animation

Published online: 26 June 2007

P. Chaudhuri (✉)
MIRALab, Centre Universitaire d'Informatique, University of Geneva, Switzerland
parag@miralab.unige.ch

P. Kalra · S. Banerjee
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India
{pkalra, suban}@cse.iitd.ernet.in

**Abstract** In this paper we present techniques for reusing view-dependent animation. First, we provide a framework for representing view-dependent animations. We formulate the concept of a view space, which is the space formed by the key views and their associated character poses. Tracing a path on the view space generates the corresponding view-dependent animation in real time. We then demonstrate that the framework can be used to synthesize new stylized animations by reusing view-dependent animations. We present three types of novel reuse techniques. In the first we show how to animate multiple characters from the same view space. Next, we show how to animate multiple characters from multiple view spaces. We use this technique to animate a crowd of characters. Finally, we draw inspiration from cubist paintings and create their view-dependent analogues by using different cameras to control different body parts of the same character.

**Keywords** View-dependent character animation · Stylized animation · Animation reuse

## 1 Introduction

The reuse of previously created animation to synthesize new animation is a very attractive alternative to creating animation from scratch. The reuse of stylized animation, however, is a very challenging problem because the stylizations are often generated for a particular viewpoint. View-dependent animation allows us to overcome this limitation. View-dependent animation is a technique of stylized animation, which captures the association between the camera and the character pose. Since the character's action *depends* on the view, changing the viewpoint generates a view-dependent *instance* of the character. These can be reused to synthesize new animation. We show that view-dependent variations can be reused to animate multiple instances of the same character, a group of different characters and even different body parts of the same character.

The view-dependent approach, however, demands that we define a formal representation of the camera-character pose association. We introduce the concept of a *view space*, defined by the key views and associated key character poses, which captures all the information required to produce a view-dependent animation. The framework allows the creation of a view-dependent animation in real time, whenever the animator traces out a new camera path on the view space. The animator can explore all the view-dependent variations quickly.

We present three broad classes of reuse methods. In the first we show how to animate multiple characters from the same view space. Next, we show how to animate multiple characters from multiple view spaces. We use this technique to animate a crowd of characters. Finally, we draw inspiration from cubist paintings and create their view-dependent analogues. We use different cameras to control different body parts of the same character. We combine the different body parts to form a single character in the final animation.

We begin by providing the background for our work in Sect. 2. Next, we present our framework for view-dependent animation in Sect. 3. We then present our techniques for reusing view-dependent animations in Sect. 4. Section 5 concludes with a summary of the work done.

## 2 Background

We start by exploring the work that has been done toward capturing the relationship between the pose of the character and the view.

### 2.1 Associating the camera and the character pose

The idea of the dependence of the character's geometry on the view direction was introduced by Rademacher [12] in his work on *view-dependent geometry* (VDG). In this work, the animator manually matches the view direction and the shape of a base mesh model with the sketched poses of the character and creates a view sphere. Tracing any camera path on this view sphere generates the appropriate animation with view-dependent deformations. Chaudhuri et al. [5] present a system for doing view-dependent animation from sketches using the VDG formulation. Our framework is more general and we demonstrate that it reduces to the VDG formulation as a special case. Martín et al. [10] use hierarchical extended non-linear transformations to produce *observer-dependent deformations* in illustrations, in order to capture the expressive capabilities. However, the technique does not present any method for authoring such transformations to obtain the desired animation.

Since we reuse view-dependent animation to synthesize new animations, our work bridges across the two themes of stylized animation and animation synthesis. We discuss the related work pertaining to these two areas in the next section.

### 2.2 Synthesis of stylized animation

Several artwork styles have been explored in stylized animation and rendering literature [8, 11]. Stylizations based on innovative use of the camera have also been researched [1, 13]. Coleman and Singh [6] make one of Singh's [13] exploratory cameras a boss (or primary) camera; this camera represents the default linear perspective view used in the animation. All other exploratory (or secondary) cameras, when activated, deform objects such that when viewed from the primary camera, the objects will appear non-linearly projected. Though this type of camera based stylization can produce striking effects, which can be aesthetically harnessed by an artist to create interesting animations, there is no direct one-to-one correspondence between the viewpoint and the pose of the character. Reusing view-dependent variations allow us to produce effects very similar to those produced in [6]. Glassner [7] talks about using *cubist* principles in animation, i.e., rendering simultaneously from multiple points of view in an animation using an abstract camera model. In one of our reuse techniques, we draw inspiration from cubist paintings and synthesize a new animation where different parts of the same character are controlled by separate cameras.

Synthesis of animations using motion synthesis techniques has been widely researched. All previous work in the direction of animation reuse has generally focused on creating newer, meaningful motions given a database of previously recorded motion capture data [2, 9]. Bregler et al. [3] describe a method to capture the motion from a cartoon animation and retarget to newer characters. Chaudhuri et al. [4] present a technique for stylistic reuse of view-dependent animation that uses Rademacher's [12] view sphere formulation to generate an animation. Their method is a subset of the reuse method that we present in Sect. 4.1. Our method is a more general formulation.

## 3 Our framework

In this section we present our framework for view-dependent animation.

### 3.1 The view space

At a given instant of time the character may be potentially viewed from a set of different viewpoints. The character may possibly have a different pose associated to each of these viewpoints (see Fig. 1). We consider this set of viewpoints and associated character poses as one sample. We assume, for simplicity of explanation, that we are animating a single character and that the camera is looking toward the character (i.e., the character is in the field of view of the camera). We also assume that the view direction is a unit vector.

We define a representation that enables the aggregation of such samples as an ordered sequence. These sets of viewpoints and associated character poses sampled (or ordered) across time form a *view space* (see Fig. 2). Every point on the *envelope* of this view space represents a viewpoint (and a unit view direction), $v$. If we do not consider the sampling order then the view space is simply a collection of viewpoints. Since for every viewpoint there is a unique view direction, we use these terms interchangeably. We denote the pose of the character as $m_v$, associated

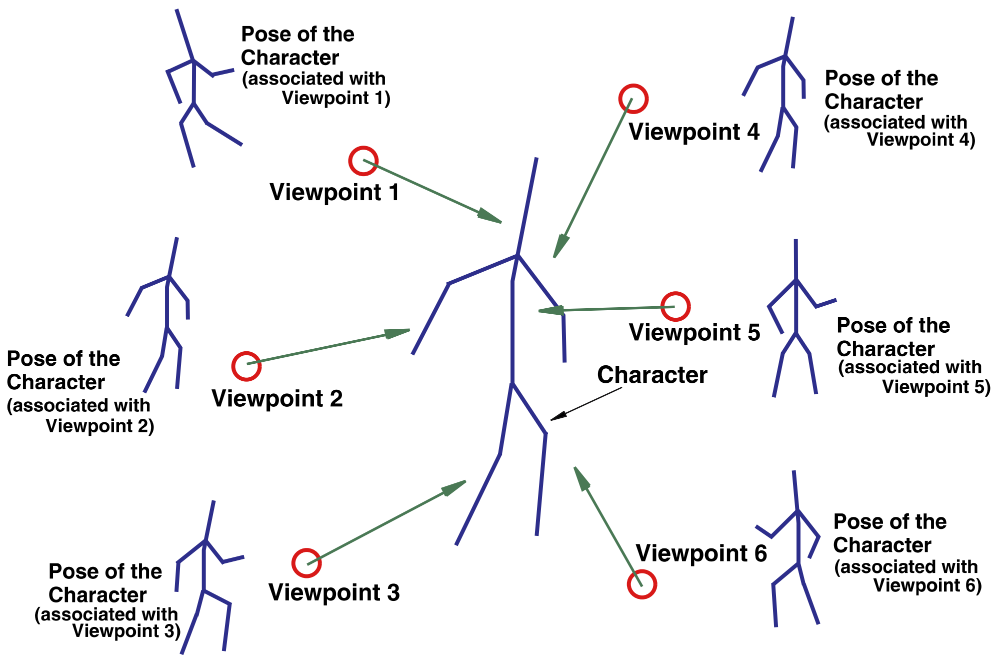
Pose of the Character (associated with Viewpoint 1)
Viewpoint 1
Viewpoint 4
Pose of the Character (associated with Viewpoint 4)
Pose of the Character (associated with Viewpoint 2)
Viewpoint 2
Viewpoint 5
Character
Pose of the Character (associated with Viewpoint 5)
Pose of the Character (associated with Viewpoint 3)
Viewpoint 3
Viewpoint 6
Pose of the Character (associated with Viewpoint 6)

**Fig. 1.** Different character poses associated with each viewpoint

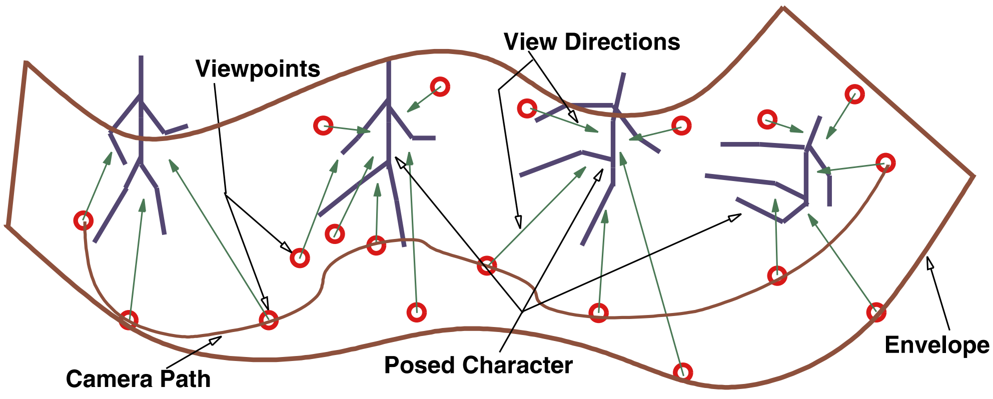

**Fig. 2.** Tracing a camera path on the envelope of the view space generates an animation. One character pose is shown for each set of viewpoints

with a view direction $v$. A *character pose*, in this paper, is the resulting mesh model of the character having undergone any change that may be rigid or non-rigid, i.e., it includes mesh deformations as well as changes in the mesh due to articulation. We couple the character pose to the view direction. Hence, changing the view direction changes the pose of the character.

An animation is generated by tracing a path, $P$, on the envelope (see Fig. 2). A point $p$ on this path consists of the view direction associated with the point on the envelope, $\underline{v}$, and is indexed by time (runtime of the animation) along that camera path, $\underline{t}$, measured from the start of the camera path. Note that the runtime of the animation should not be confused with the sampling time. We refer to points on a camera path $P$ as $p \equiv (\underline{v}, \underline{t})$. The animation generated is the sequence of the poses $m_{\underline{v}}$ associated to $\underline{v}$ on the path $P$, viewed along the direction $\underline{v}$. Every distinct camera path generates a distinct animation. This is the basic idea behind our framework.

In order to create the view space, the animator provides a set of *key viewpoints* or *key view directions* and the associated *key poses*. Let $v^k$ represent a key viewpoint, and $m_{v^k}$ represent the associated key character pose. The animator can provide these in the form of a set of sketches, a video, or a mix of the two. As an example consider the *Hugo's High Jump* animation (Hugo is the character used in the animation) where the animator provides the sketches for the keyframes. These key views and key poses form the

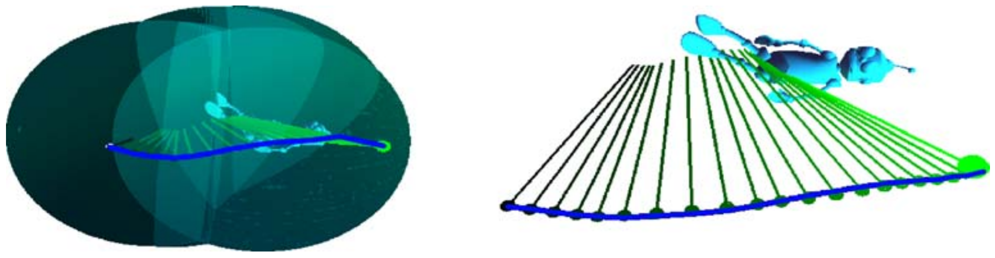

**Fig. 3.** The *left* image shows the path traced on the envelope of the view space. The *right* image shows a close-up view of the path

*view space* on which the animation is generated. Figure 5 shows some of the sketches used for the purpose.

Now, for each key view the sphere centred at the look-at point (in this case the end of the unit length view direction vector) is the set of all possible view directions from which one can look toward that point. Hence, such a sphere may be thought of as a view space generated by just one view. The complete view space is, therefore, the union of the view spaces generated by all the views, i.e., a union of all the spheres (see Fig. 3).

In order to generate an animation along a camera path $P(\underline{v}, \underline{t})$ on the envelope of the view space, we need to generate the associated character pose, $m_{\underline{v}}$, for every point $p$ on $P$. To do this, for any view direction $\underline{v}$ we determine the $r$-closest key viewpoints (closest in the metric defined on the envelope), $v_j^k$.

For clarity, henceforth we represent $v^k$ as $v$ and $v_j^k$ as $\bar{v}$. The character pose $m_{\underline{v}}$ is then given by

$$m_{\underline{v}} = \sum_{\bar{v}} w_{\bar{v}} m_{\bar{v}} . \tag{1}$$

Thus, $m_{\underline{v}}$ is a weighted blend of the corresponding $m_{\bar{v}}$'s (i.e., the $r$-closest key view poses). The $w_{\bar{v}}$'s are the corresponding blending weights. The $w_{\bar{v}}$'s vary inversely to the proximity of $\bar{v}$ to $\underline{v}$. An example of such a path, $P(\underline{v}, \underline{t})$, is shown in Fig. 3. Figure 4 shows the selection of the $r$-closest key viewpoint for a given position of the rendering camera on the path.

The path shown in Fig. 3 is obtained by smoothly joining the key viewpoints. Some frames from the animation obtained from this path are shown in Fig. 5. Here we see that the generated animation matches the planned storyboard frames very closely and the path generates the animation originally intended by the animator. In this animation, we have generated the actual jump as a view-dependent animation. Hugo's run-up before the jump, however, is generated using simple keyframed animation and it blends in seamlessly with the view-dependent portion. The character is a 3D mesh model with an embedded articulation skeleton. We use inverse kinematics to pose this skeleton. The character is also enclosed in a control lattice made up of tetrahedral cells, with each cell associated to one skeleton bone. This allows us to deform

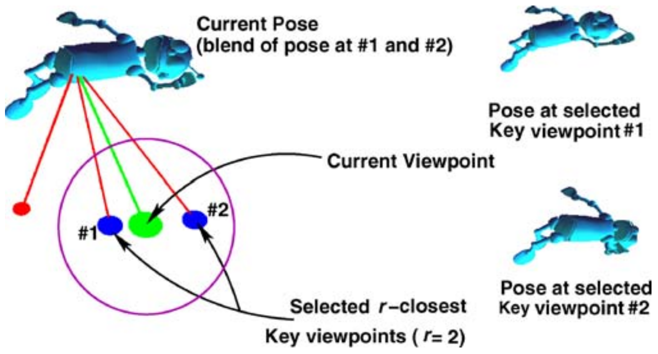

**Fig. 4.** Blending between the $r$-closest key views

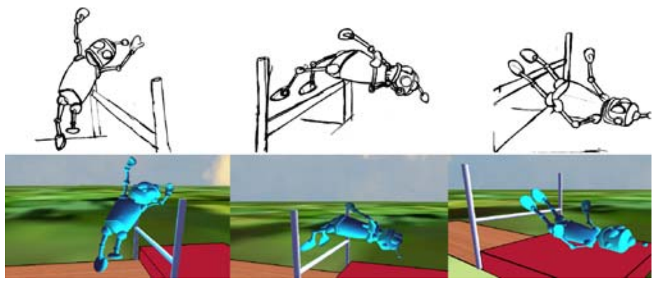

**Fig. 5.** Some frames from the planned storyboard and the final rendered animation

the character's mesh using direct free-form deformation. We use a combination of these methods and robust computer vision techniques, as explained in [5], in order to recover the key view directions and the key poses from the sketches given by the animator.

This complete process is very intuitive for the animator as she does not have to worry about the camera and the character separately, once the view space has been created. We assume coherence over a local neighbourhood around any viewpoint both in terms of the view direction and the character pose, i.e., the pose specified by the animator for any viewpoint is similar to the pose specified for any other viewpoint in its small neighbourhood. This guarantees spatio–temporal continuity in the generated animation, i.e., the animation will not have any sudden *unwanted* changes in the view or pose between successive frames.

The view space for this example (shown in Fig. 3) is an instance of the general view space formulation. The view space can have other forms depending on the spatial location and sampling order of the sets of viewpoints used to construct it. The conditions under which they are generated are enumerated below:

1. If all the view directions, corresponding to a set of viewpoints sampled at a given instant of time, intersect at a common point (i.e., they share a common look-at point), then the instantaneous view space is a single sphere (also called a *view sphere*) centered at the point of intersection. This is trivially true if there is only one view direction for some time instant. If this condition holds for all sampling time instants, then the view space is an aggregation of view spheres. The spatial location and sampling order of these sets of viewpoints (i.e., view spheres) gives rise to the following view space configurations:
   (a) If there is only one set of viewpoints (i.e., there is only one sample), then the view space is a single view sphere.
   (b) If there are multiple sets of viewpoints and each set is located at a different point in space and sampled at a different time instant, then the view space is an aggregation of view spheres separated in both space and time. The view space shown in Fig. 3 is an example of such a case (with only one view direction for each time instant).
   (c) If there are multiple sets of viewpoints at the same spatial location, sampled at different time instants, then the view space is an aggregation of view spheres separated only in time and not in space.
2. If all the view directions, corresponding to a set of viewpoints sampled at a given time instant, do not intersect at a common point, then the instantaneous view space is not a single sphere. It can be considered as a collection of spheres (one centered at each distinct look-at point). Then the complete view space is an aggregation of such instantaneous view spaces. The view space may have any of the three configurations analogous to the ones described above.

In the work by Rademacher [12] the view sphere formed by view-dependent models is a special case of our view space. Here a convex hull of the viewpoints is computed. This partitions the view space by imposing a triangulation on it. A novel view-dependent model for any new viewpoint is generated by a barycentric blend of the key deformations at the vertices of the triangle in which the new viewpoint lies. This is clearly a special case of our novel view generation strategy on the envelope.

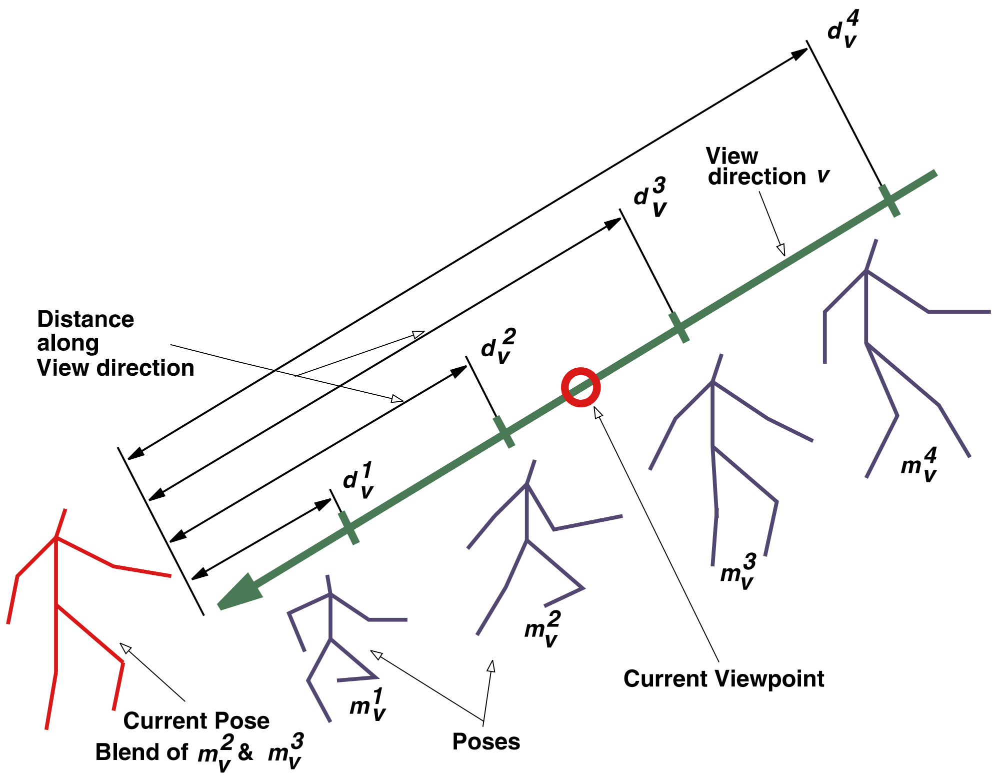

**Fig. 6.** Change of character pose with change of distance of the current viewpoint along a view direction

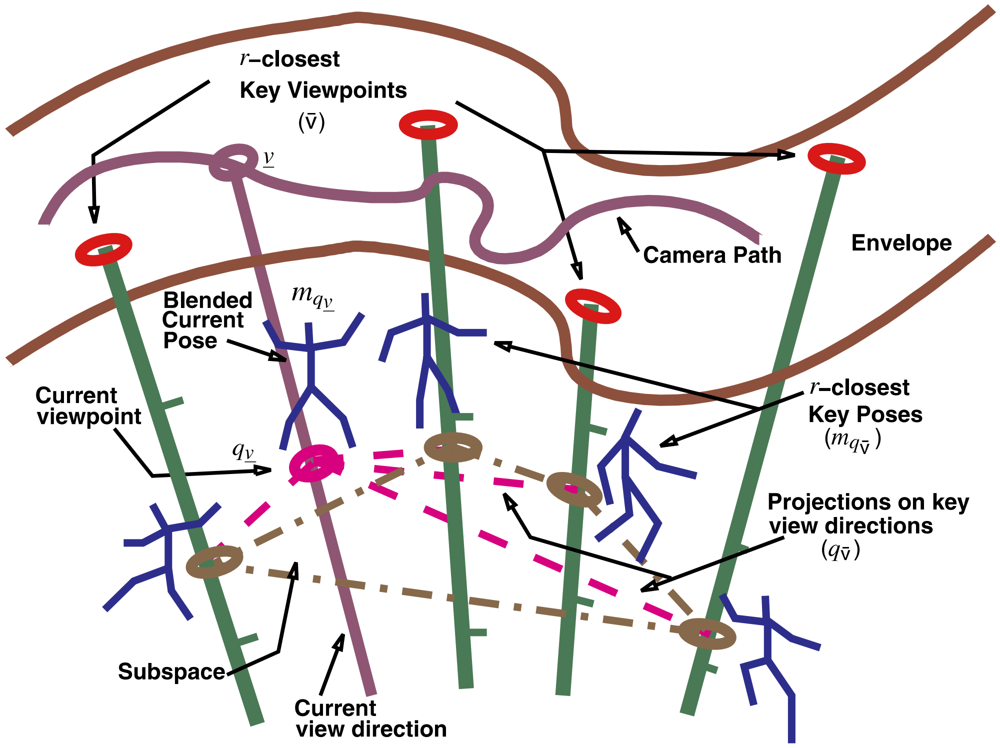

**Fig. 7.** Generating a new character pose for the current viewpoint from key viewpoints after incorporating distance

Here, $r = 3$-closest key viewpoints set up a local barycentric basis for the novel viewpoint. The new character pose associated with this viewpoint is computed as a weighted blend of the key poses at the selected key viewpoints, using the barycentric coordinates of the novel viewpoint as weights. The major limitations of Rademacher's formulation are the following:

– It does not handle the distance of the viewpoint, which is crucial for incorporating zoom effects.
– It cannot handle cases where all the camera view directions do not intersect at a single look-at point (the center of a view sphere) thereby limiting the method considerably.

Our framework can deal with both these conditions.

## 3.2 Distance of the viewpoint

In the previous discussion we developed our framework considering only the view direction without the distance of the viewpoint. Now we add the component of distance to our framework, i.e., we want the character's pose to change as the distance of the viewpoint changes (with or without an accompanying change in the view direction).

We assume that a tuple list $(d_v^l, m_v^l)$ is associated with every view direction, $v$, forming the view space. Here, $d_v$ is the distance of viewing and the associated character pose is $m_v$. The list is sorted on the distance field of each tuple. If the list has $L$ elements, then $1 \leq l \leq L$. So the $m_v^l$'s are the different poses of the character along a view direction at various distances $d_v^l$. As we change the distance, $d : d_v^{l1} \leq d \leq d_v^{l2}$, along a view direction, $v$, the resulting character pose is a blend of the character poses $m_v^{l1}$ and $m_v^{l2}$ (see Fig. 6).

Now, given a set of key viewpoints, $\boldsymbol{v}$, and the associated tuple lists, $(d_{\boldsymbol{v}}^l, m_{\boldsymbol{v}}^l)$, we want to generate an animation for a camera path $P(\underline{v}, \underline{d}, \underline{t})$. The added parameter $\underline{d}$ is the distance of the viewpoint along the unit view direction $\underline{v}$. The vector $q_{\underline{v}} = \underline{d}\ \underline{v}$ gives the position of the current viewpoint (see Fig. 7). We determine the $r$-closest key viewpoints to $\underline{v}$ on the envelope as before. Now for every key viewpoint, $\bar{v}$, in the $r$-closest set of $\underline{v}$, we project the vector $q_{\underline{v}}$ on $\bar{v}$ and find the length of the projected vector. The projected length, $\underline{d}\ \underline{v} \cdot \bar{v}$, is the distance $\underline{d}$ projected along $\bar{v}$. Find $d_{\bar{v}}^{l1}$ and $d_{\bar{v}}^{l2}$ from the tuple list of $\bar{v}$ such that $d_{\bar{v}}^{l1} \leq \underline{d}\ \underline{v} \cdot \bar{v} \leq d_{\bar{v}}^{l2}$. It is always possible to find a $\beta_{\bar{v}}$ such that $\underline{d}\ \underline{v} \cdot \bar{v} = (1 - \beta_{\bar{v}}) d_{\bar{v}}^{l1} + \beta_{\bar{v}} d_{\bar{v}}^{l2}$. $\beta_{\bar{v}}$ locates a point, $q_{\bar{v}}$, along the corresponding $\bar{v}$ vector. The pose at each $q_{\bar{v}}$ is given by:

$$m_{q_{\bar{v}}} = (1 - \beta_{\bar{v}}) m_{\bar{v}}^{l1} + \beta_{\bar{v}} m_{\bar{v}}^{l2}, \qquad (2)$$

where $m_{\bar{v}}^{l1}$ and $m_{\bar{v}}^{l2}$ are the poses associated with $d_{\bar{v}}^{l1}$ and $d_{\bar{v}}^{l2}$. Then the pose corresponding to the current viewpoint $q_{\underline{v}}$ is given as a weighted blend of the pose at each $q_{\bar{v}}$, as:

$$m_{q_{\underline{v}}} = \sum_{\bar{v}} w_{q_{\bar{v}}} m_{q_{\bar{v}}}, \tag{3}$$

where $w_{q_{\bar{v}}}$ are the weights used for the blending. The process is shown schematically in Fig. 7.

In order to illustrate this concept, we augment the view space, shown in Fig. 3, by adding two more poses for a view direction at different distances. The poses are reconstructed from sketches given by the animator, and the camera center is recovered along with the distance of viewing. Figure 8 shows two camera positions on the left that differ in the distance from the character, and not the view direction. On the right the corresponding character pose is shown, as seen from their associated cameras. Now we trace another path for the rendering camera, specifying $d_p$ for all points on the path, and the required animation is generated as explained above. This also illustrates that there exist other paths that are capable of generating interesting animations. Our framework can generate animation in real time as the animator traces out a path on the view space, thus making it possible for the animator to explore the view space very easily.

Thus, in our framework we incorporate both the view direction and the distance of a viewpoint. It is easy to incorporate changes in the character pose with changes in focal length of the camera in a manner similar to the one used for distance of the viewpoint. Since the view space is an abstract representation, it can be easily used with all the view parameters encoded in the form of a camera matrix. We now present our methods for reusing view-dependent animation, using the framework we have developed to synthesize new animations.

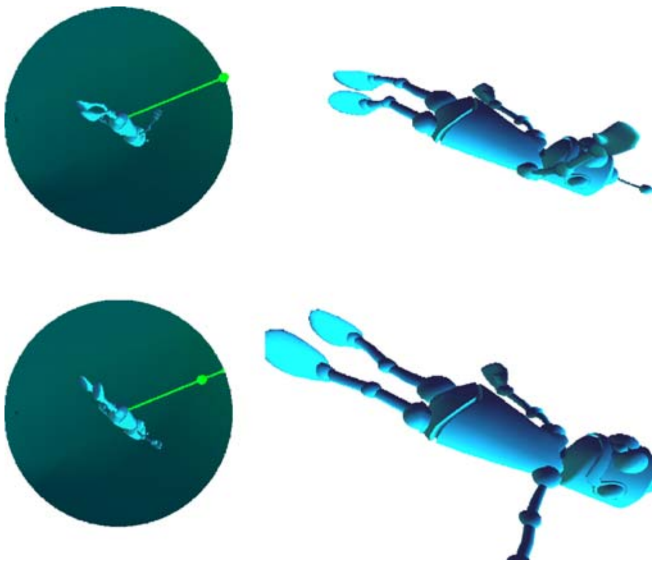
**Fig. 8.** Varying the character pose with change in distance of the viewpoint

## 4 Reusing view-dependent animations

We consider the different view-dependent animations made possible by changing the rendering camera, as variations of each other. We are interested in reusing these variations to synthesize novel animations. We categorize these ways of reusing view-dependent animation into three broad categories. In the subsequent sections, we discuss these categories in terms of their representation in the machinery of our framework.

### 4.1 Animating multiple characters from the same view space

We want to reuse the view-dependent variations of a character to animate multiple characters and create a novel animation.

Let us assume that a camera, $\mathcal{C}_1$, traces a path $P_1(\underline{v}, \underline{d}, \underline{t})$ on the view space, $\mathcal{VS}$. A second camera, $\mathcal{C}_2$, traces another distinct path $P_2(v_p, d_{v_p}, t_p)$ on $\mathcal{VS}$. The animation generated by $\mathcal{C}_1$ can be thought of as an ordered set of $n$ frames $\mathcal{P}_1$, given by $\mathcal{P}_1 = \{\mathfrak{p}_1^i : 1 \le i \le n\}$, where $\mathfrak{p}_1^i$ is the pose of the character in the $i$-th frame. The order implicitly imposed on the set is the temporal sequence of the frames in the animation. Similarly, we have, for the animation generated by $\mathcal{C}_2$, another ordered set of $m$ frames $\mathcal{P}_2$, given by $\mathcal{P}_2 = \{\mathfrak{p}_2^j : 1 \le j \le m\}$. The animations $\mathcal{P}_1$ and $\mathcal{P}_2$ are view-dependent variations of each other, i.e., they are generated from the same view space. The poses, $\mathfrak{p}_1^i \in \mathcal{VS}$ and $\mathfrak{p}_2^j \in \mathcal{VS}$, are view-dependent variations, or instances, of each other.

We then define a novel animation with two characters as an ordered set $\mathcal{Q}$, given by

$$\mathcal{Q} = \Big\{\big\langle \mathfrak{q}_1^k \oplus \mathfrak{q}_2^k \big\rangle : \mathfrak{q}_1^k = \mathfrak{p}_1^k \text{ and } \mathfrak{q}_2^k = \mathfrak{p}_2^k \\ \forall\, k,\ 1 \le k \le \min(n, m)\Big\}, \tag{4}$$

where $\langle \mathfrak{q}_1^k \oplus \mathfrak{q}_2^k \rangle$ indicates that a frame $k$ in $\mathcal{Q}$ consists of two character poses (see Fig. 9). The $\oplus$ operator indicates that the two poses are being composed together to form the synthesized animation frame. The composition can be done in 3D space if the two poses are registered to a common coordinate system. The composition can also be done in 2D image space, by compositing the poses after they have been rendered into the framebuffer. The novel animation has $\min(n, m)$ frames.

In this manner, we can reuse the view-dependent variations of a character to animate multiple characters and create a new animation. As an example of this method of reuse, we create a view space and plan the movement of two cameras on it. Figure 10 shows the final frame generated by compositing the two view-dependent instances of the character. Note that the compositing done is in the image space, i.e., in 2D.

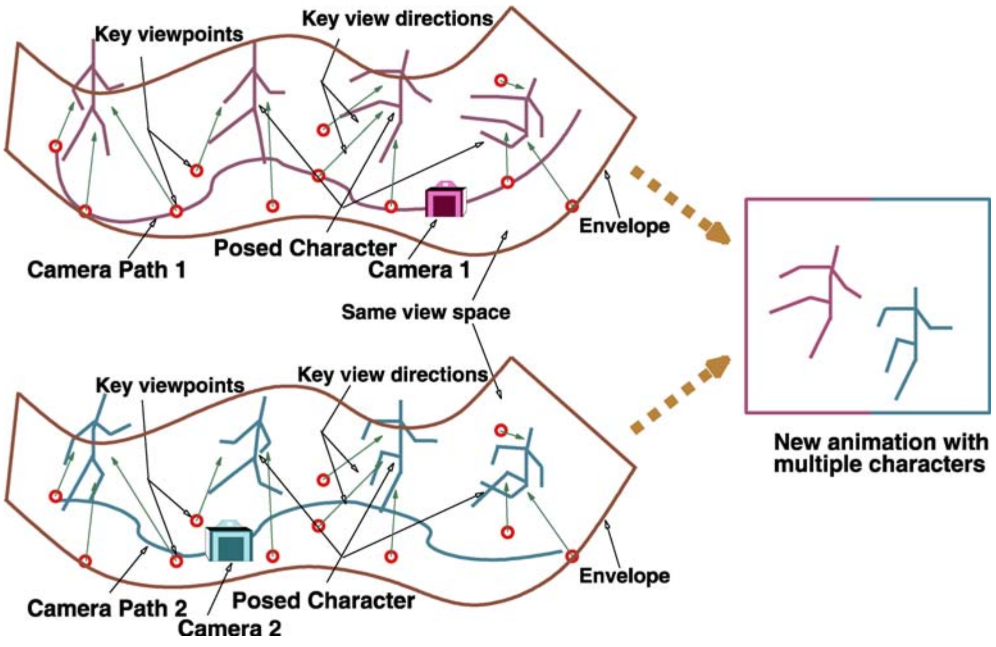

**Fig. 9.** Animating multiple characters from the same view space

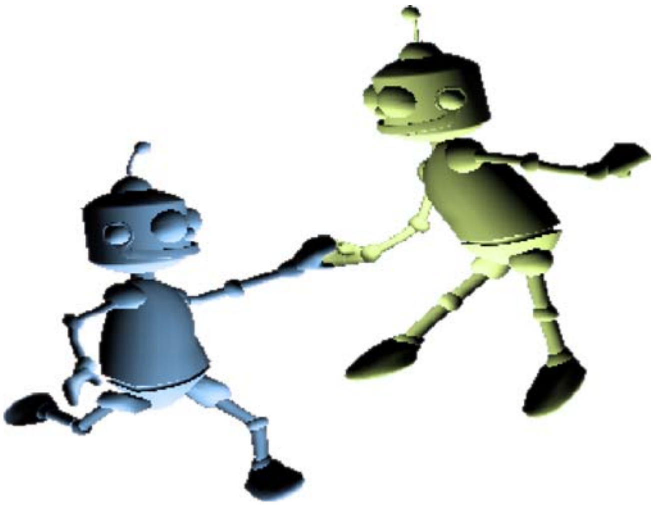

**Fig. 10.** Two view-dependent instances composited together

## 4.2 Animating multiple characters from multiple view spaces

The reuse strategy presented in Sect. 4.1 uses multiple instances of the same character, each from the same view space. We want to further expand this idea and look at animating groups of distinct characters together.

Consider that we have $N$ distinct characters and we have constructed a view space for each. Then we can generate the distinct animations, $\mathcal{P}_r$, with $1 \leq r \leq N$. Note that the generated $\mathcal{P}_r$'s are distinct even if the path traced on each view space is the same, because the character in each is distinct. Each $\mathcal{P}_r$ is an ordered set of $n_r$ frames and is given by $\mathcal{P}_r = \{\mathfrak{p}_r^i : 1 < i \leq n_r\}$. We can now construct a new animation of a group of these distinct characters as

$$\mathcal{Q} = \left\{ \left\langle \bigoplus_{l=1}^{N} \mathfrak{q}_l^k \right\rangle : \mathfrak{q}_l^k = \mathfrak{p}_l^k \;\; \forall\, k,\;\; 1 < k \leq \min_{l=1}^{N}(n_l) \right\}, \tag{5}$$

where the $\bigoplus$ operator indicates that $N$ poses are being composed together to form the synthesized animation frame.

We now look at the problem of how to control the paths that we want to trace on the $N$ distinct view spaces. Let a camera be associated with every view space. We call this camera the *local* camera for the corresponding view space. Let the path traced by this camera be $P_r(\underline{v}_r, \underline{d}_r, \underline{t}_r)$. Now, we define a single *global* camera and the path traced by this camera as $\mathfrak{P}$. The path $\mathfrak{P}$ is not a path on any view space, but is the trajectory of the global camera in 3D space, defined in the global coordinate system.

We can define a *path-mapping* function $f_r$, $P_r = f_r(\mathfrak{P})$, $1 \leq r \leq N$. The function $f_r$ maps the global path to the corresponding local path on the view space. The function $f_r$ is a coordinate system transfer function, from the global coordinate system to the local coordinate system of each view space. In order to create the novel animation, the animator has to plan the camera trajectory only for the global camera and define the various $f_r$'s. Then moving the global camera along $\mathfrak{P}$, will cause each of the local cameras to move along the corresponding $P_r$ on

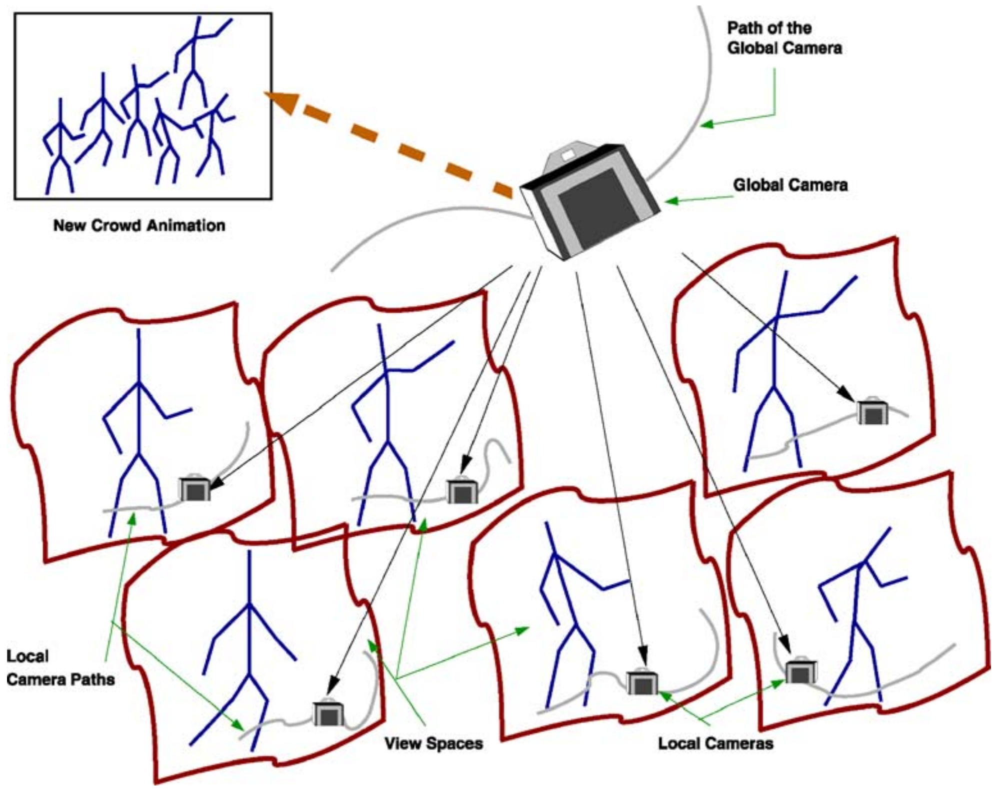

**Fig. 11.** Animating multiple characters from multiple view spaces

their respective view spaces. This will generate the distinct $\mathcal{P}_r$'s. These can be composited together to generate the final animation (see Fig. 11). A straightforward choice for the compositing method is to render the various poses as they appear when viewed through the global camera. This technique automatically composites them in the rendered frame. The animator, however, can use any other compositing method as required for the animation. Before starting the animation process, the animator has to place the various characters in the global coordinate system as a part of global scene definition. Hence, the animator already knows the coordinate system transfer function, $g_r$, from the global coordinate system to the local coordinate system of each character. The mapping from the local coordinate system of the character to the coordinate system of the view space, $h_r$, is easily recovered during view space construction. Thus, we have $f_r = g_r \circ h_r$ (where $\circ$ represents function composition).

We used this reuse technique to animate a crowd of characters in the *Mexican Wave* animation. In this example, the same character is replicated many times to generate a crowd (Fig. 12b). Each character has a local view space as shown in Fig. 12a. The local key viewpoints are shown in blue and red, while the current local camera is shown in green. Moving this local camera on the path shown (in green) causes the single character's pose

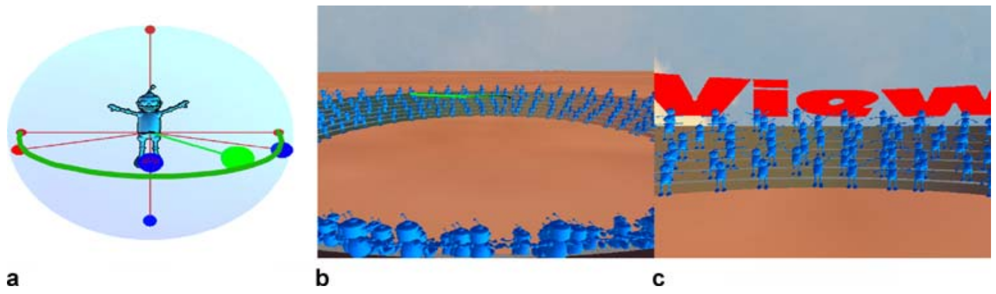

**Fig. 12.** The Mexican wave animation

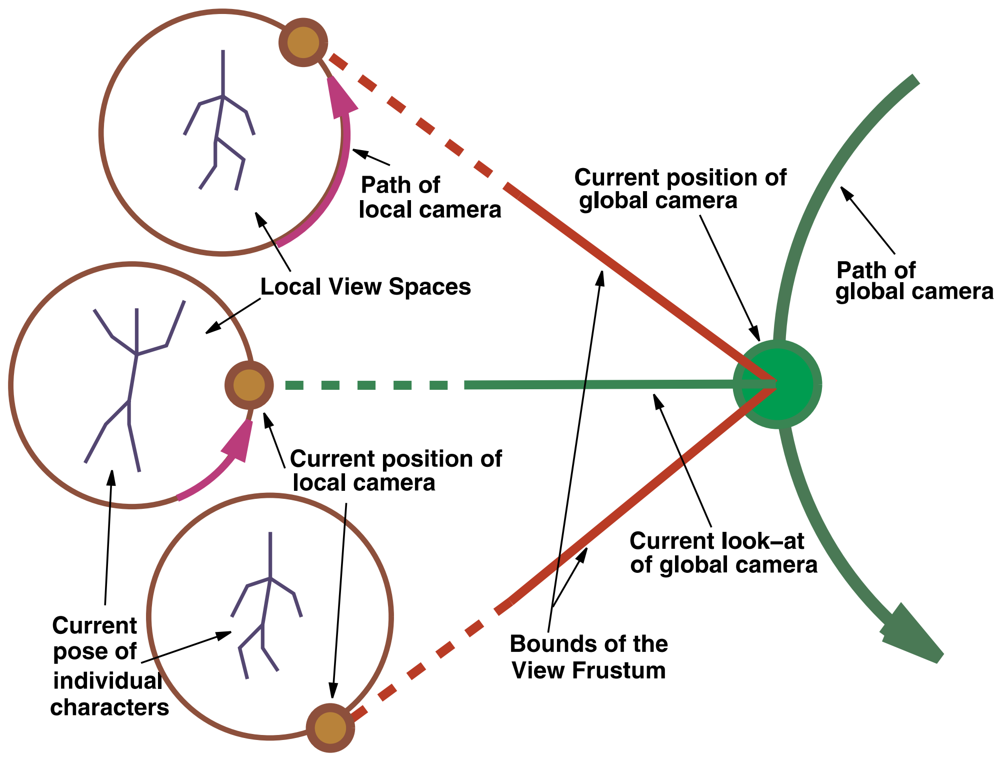

**Fig. 13.** Mapping the movement of global camera to the local cameras

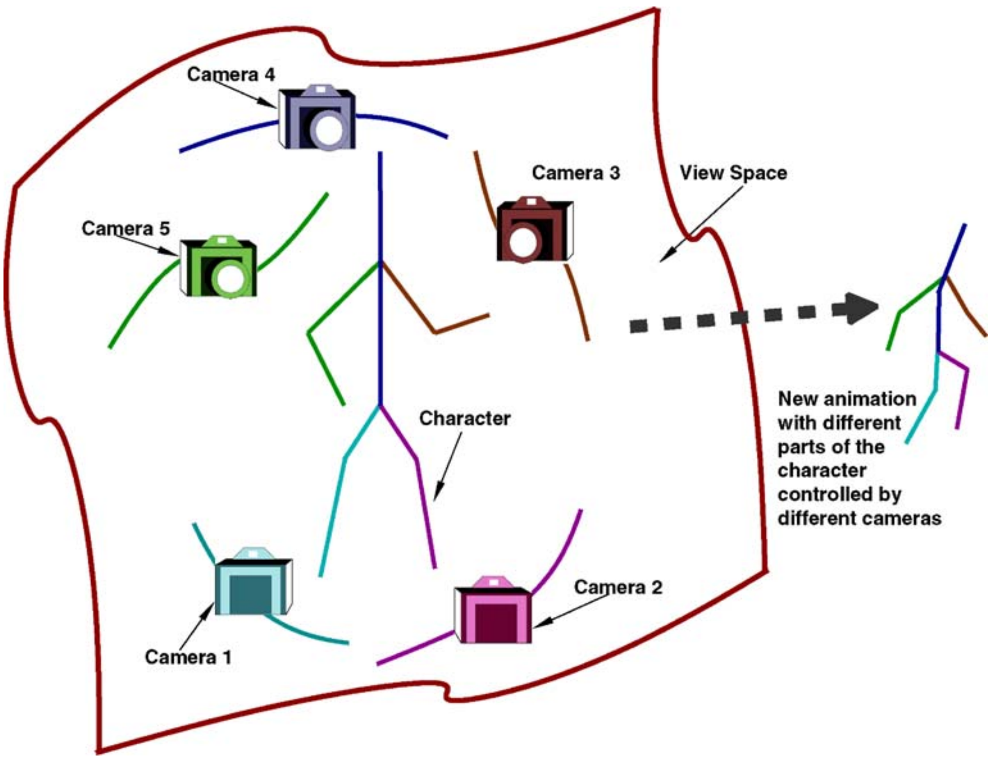

**Fig. 14.** Animating different parts of a single character from a single view space

change as it is supposed to change during the crowd animation. The movement of the global camera is mapped to each of these view spaces to move the corresponding local cameras, which generates the final animation. The path of the global camera and current look-at is shown in green in Fig. 12b. Note that the crest of the Mexican wave is in front of the current camera look-at. We also perform conservative view culling to efficiently render the crowd. A final frame of the animation is shown in Fig. 12c.

In this example, finding the path-mapping function, $f_r$, which will generate the wave in the crowd, for a specific movement of the global camera is not difficult. Figure 13 shows the position of the local cameras in their respective local view spaces for a given position of the global camera. The mapping ensures that the local cameras in local view spaces outside the bounds of the current view frustum do not move. This mapping function can be intuitively constructed. For a general case, however, designing a path mapping function to obtain a desired animation may not always be easy.

## 4.3 Animating different parts of a single character from a single view space

In the previous sections we looked at the problem of synthesizing a novel animation with multiple characters using view-dependent variations of one or many characters. Now we draw inspiration from *cubist* paintings, which portray the parts of the same character in a painting from different perspectives. Many such paintings by Pablo Picasso are perfect examples of a scene that can be visually thought of as broken into disjoint parts that are viewed from different perspectives and then patched back together. Similarly, we want to generate a new animation, where different parts of the same character are controlled by separate cameras. All the cameras move on the same view space. The final animation will have the character with each separately animated body part blended together.

In order to do this we consider a pose, $\mathfrak{p}$, to be made up of a union of $M$ body parts, $\mathfrak{b}_u$, i.e., $\mathfrak{p} = \bigcup_{u=1}^{M} \mathfrak{b}_u$. We assume that there is no overlap between the body parts, i.e., they are distinct and mutually exclusive. Now, we associate a camera $\mathcal{C}_u$ with a body part $\mathfrak{b}_u$. Each camera traces a path, $P_u(\underline{v}_u, \underline{d}_u, \underline{t}_u)$, on the view space. The synthesized animation of $n$ frames, $\mathcal{Q}$, is then given by

$$\mathcal{Q} = \{\mathfrak{q}^i : \mathfrak{q}^i = \mathfrak{p}^i : 1 \leq i \leq n\}. \qquad (6)$$

At any point, $p_u$ on a camera path, the configuration of the corresponding body part, $\mathfrak{b}_u$, is computed by using a process analogous to pose computation at $p_u$ for a normal view-dependent animation as given in Sect. 3. We can also associate other parameters, like the scaling of each body part, with the respective cameras. We can then vary these parameters when the corresponding cameras move. The various body parts are then composited together to form the final pose (see Fig. 14). The compositing method used is the animator's prerogative.

We present two variations of this reuse technique as examples. In the first, different body parts are viewed from their respective cameras and the views are composited in 2D image space to generate a multi-perspective image. This compositing technique is similar to the one given by Coleman and Singh [6]. We associate six *body* cameras, one each, with the head, torso, two arms and two legs. We explicitly associate the cameras with the bones of the embedded skeleton for each body part, which automatically groups the mesh vertices into various body parts as each mesh vertex is uniquely contained in a control lattice cell, which in turn is associated to exactly one bone of the embedded skeleton. We also associate scaling parameters of the various body parts with the position of their respective body cameras. Since each body camera is at a different position, each body part is scaled differently, in addition to having a different perspective. We then composite the view from each to get the final image shown in Fig. 15. In this image the head of the character is seen from the right, the torso from the front, the left hand from the top, the right hand from the left-bottom, the left foot from the front, while the right foot is seen from the right side. This may be thought of as the view-dependent analogue of cubist paintings.

In the second variation, we again associate six body cameras with the various body parts. The composition of the body parts is done in object space in 3D. This is done

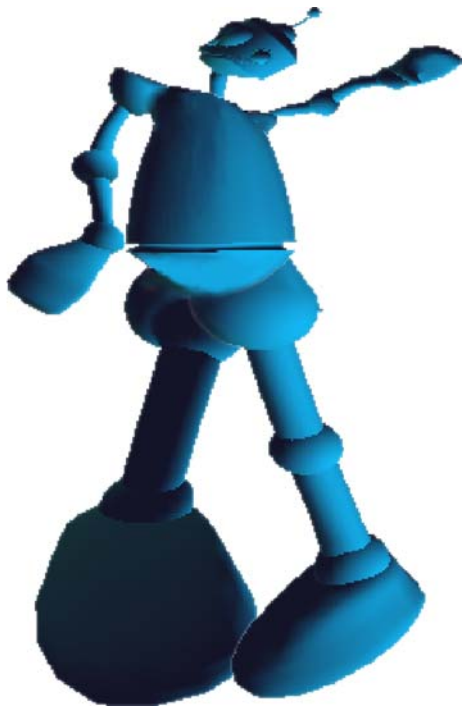
**Fig. 15.** A multi-perspective image

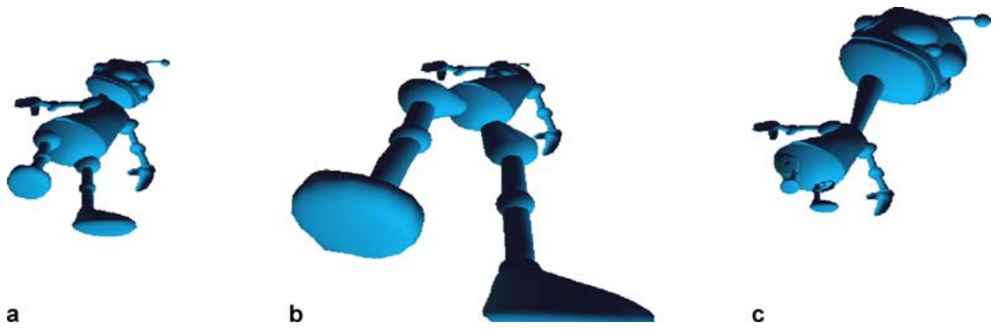

**Fig. 16.** Compositing in object space and rendering from the master camera

by taking one model of the character and by posing the various body parts as per the associated camera. The connectivity of the body parts is not disturbed, and hence they can be blended in object space. The rendered viewpoint is that of a master camera. The body cameras follow the movement of the master camera. Figure 16 shows frames from three animations that we generated using this technique, each with a different set of scaling parameters for the various body parts. Figure 16a has no scaling, Figure 16b has scaling that exaggerates the perspective effect, i.e., the part closer to the camera appears very big, while the part farther away appears very small. This effect can be seen in the legs and the head, as the camera moves from below the character to the top. As the camera moves, the scaling associated with the body parts changes to maintain the exaggerated perspective effect. The hands and the torso are not scaled. Figure 16c has scaling, which counters the perspective effect (i.e. the head appears larger).

As a final example of the elegance of our reuse technique, we stylize the *Hugo's High Jump* animation by associating different cameras with different body parts of the character. In this animation, as Hugo jumps, his limbs stretch and his head becomes larger. This is made possible by the scaling parameters associated with the various moving body cameras.

## 5 Conclusion

We have formulated the concept of a view space of key views and associated key character poses as a framework for representing view-dependent animations. It captures all the information about the views and character poses efficiently and concisely. The animator can trace camera paths on the view space and the corresponding animation is generated in real time. The ability to understand and explore view-dependent animation using our framework gives us an insight into the reuse of view-dependent animation.

We have formalized the concept of reusing view-dependent animations in terms of our framework. We have presented three novel reuse strategies. In the first we have shown how to animate multiple characters from the same view space. Next, we have shown how to animate multiple characters from multiple view spaces. We used this technique to animate a crowd of characters. Finally, we have drawn inspiration from cubist paintings and created their view-dependent analogues by using different cameras to control various body parts of the same character. We have thus shown that reusing view-dependent animation is possible using our framework and it can be used to synthesize a variety of interesting stylized animations.

For future work we would like to analyze under what conditions a suitable mapping function can be designed, given any desired combination of paths of the global and local cameras.

**Acknowledgement** Hugo's mesh is courtesy of Laurence Boissieux, INRIA.

## References

1. Agrawala, M., Zorin, D., Munzner, T.: Artistic multiprojection rendering. In: Proceedings of the Eurographics Workshop on Rendering Techniques, pp. 125–136 (2000)
2. Arikan, O., Forsyth, D.A., O'Brien, J.F.: Motion synthesis from annotations. ACM Trans. Graph. **22**(3), 402–408 (2003)
3. Bregler, C., Loeb, L., Chuang, E., Deshpande, H.: Turning to the masters: Motion capturing cartoons. In: Proceedings of SIGGRAPH, pp. 399–407 (2002)
4. Chaudhuri, P., Jindal, A., Kalra, P., Banerjee, S.: Stylistic reuse of view-dependent animations. In: Proceedings of Indian Conference on Vision, Graphics and Image Processing, pp. 95–100 (2004) Online version is available at http://www.cse.iitd.ac.in/ ~parag/ pubs/icvgip2004_preprint.pdf
5. Chaudhuri, P., Kalra, P., Banerjee, S.: A system for view-dependent animation. Comput. Graph. Forum **23**(3), 411–420 (2004)
6. Coleman, P., Singh, K.: Ryan: Rendering your animation nonlinearly projected. In: Proceedings of NPAR, pp. 129–156. ACM Press, New York (2004)
7. Glassner, A.S.: Cubism and cameras: Free-form optics for computer graphics.

Technical Report (MSR-TR-2000-05) Microsoft Research. Cited Jan. 2000
8. Hays, J., Essa, I.: Image and video based painterly animation. In: Proceedings of NPAR, pp. 113–120. ACM Press, New York (2004)
9. Kovar, L., Gleicher, M., Pighin, F.: Motion Graphs. In: Proceedings of SIGGRAPH, pp. 473–482 (2002)
10. Martín, D., García, S., Torres, J.C.: Observer dependent deformations in illustration. In: Proceedings of the NPAR, pp. 75–82. ACM Press, New York (2000)
11. Meier, B.J.: Painterly rendering for animation. In: Proceedings of SIGGRAPH, pp. 477–484 (1996)
12. Rademacher, P.: View-dependent geometry. In: Proceedings of SIGGRAPH, pp. 439–446 (1999)
13. Singh, K.: A fresh perspective. In: Proceedings of Graphics Interface, pp. 17–24 (2002) Online version is available at http://www.graphicsinterface.org/proceedings/2002/152/

PARAG CHAUDHURI is a post-doctoral researcher at MIRALab, the University of Geneva. He received his PhD from the Indian Institute of Technology Delhi, India, in 2006. He received the outstanding PhD award from IBM IRL for 2006. His primary research interests include all types of computer animation. He is also interested in character animation, motion capture, real time computer graphics, and computer vision (geometric and active).

PREM KALRA is a professor in the Department of Computer Science and Engineering at the Indian Institute of Technology, Delhi. Earlier, he was at MIRALab, the University of Geneva (Switzerland). He obtained his PhD in computer science from the Swiss Federal Institute of Technology, Lausanne, in 1993. His research interests include computer vision based modeling and rendering, 3D visualization and animation, and image/video super-resolution.

SUBHASHIS BANERJEE received a BE degree from Jadavpur University, Calcutta, in 1982, an ME degree in electrical engineering and a PhD from the Indian Institute of Science, Bangalore, in 1984 and 1989, respectively. Since 1989 he has been on the faculty of the Department of Computer Science and Engineering at IIT Delhi, where he is currently a professor. His research interests include computer vision and real-time embedded systems.