Chapter 6

Cartoon-style Visual Motion Emphasis from Video

In this chapter we describe the first of three subsystems comprising the Video Paintbox; a framework capable of rendering motion within a video sequence in artistic styles, specifically emulating the visual motion cues commonly employed by traditional cartoonists. We are able to synthesise a wide range of augmentation cues (streak-lines, ghosting lines, motion blur) and deformation cues (squash and stretch, exaggerated drag and inertia) in this single encompassing framework. We also address problematic issues such as camera motion and occlusion handling. Effects are generated by analysing the trajectories of tracked features over multiple video frames simultaneously. Emphasis of motion within video, rather than mitigation against its presence for the purpose of maintaining temporal coherence, is a unique contribution to AR facilitated by the higher temporal level of analysis performed by the Video Paintbox.

6.1 Introduction

The character of an animation is influenced not only by the artistic style in which it is drawn, but also by the manner in which the animated objects appear to move. Although a number of video driven non-photorealistic animation techniques have been proposed to address the former issue, the problem of automatically emphasising and stylising motion within a 2D video sequence has not yet been addressed by the AR community (Section 2.5.3). In this chapter we describe a novel framework capable of rendering motion within a video sequence, and demonstrate the automated synthesis of a wide range of visual motion emphasis cues commonly used by animators[1].

[1]This work has been published in [27] and has been submitted to the journal "Graphical Models". An earlier version of this work was published as [25].

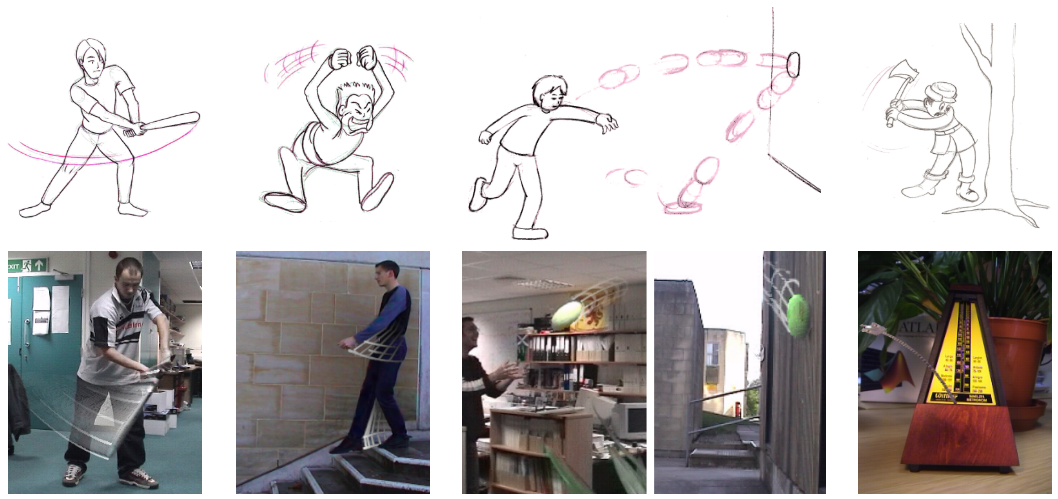


Figure 6-1 Examples of motion cues used in traditional animation (top) and the corresponding cues inserted into a video sequence by our system (bottom). From left to right: two examples of streak-line augmentation cues, the latter with ghosting lines. Two examples of deformation cues; squash and stretch and suggestion of inertia through deformation.

Animators have evolved various ways of visually emphasising the characteristics of a moving object (Figure 6-1). Streak-lines are commonly used to emphasise motion, and typically follow the movement of the tip of the object through space. The artist can use additional "ghosting" lines that indicate the trailing edge of the object as it moves along the streak-lines. Ghosting lines are usually perpendicular to streak-lines. Deformation is often used to emphasise motion, and a popular technique is squash and stretch in which a body is stretched tangential to its trajectory, whilst conserving area [98]. Other deformations can be used to emphasise an object's inertia; a golf club or pendulum may bend along the shaft to show the end is heavy and the accelerating force is having trouble moving it. The magnitude of deformation is a function of motion parameters such as tangential speed, and of the modelled rigidity of the object. In this chapter we develop a system which processes real video to introduce these motion cues; comparative examples of hand drawn and synthesised cues are given in Figure 6-1.

Recall that our aim is to develop tools for animators with which they may interact at a high level, to render artistic effects in video. Users interact with our visual motion emphasis framework by drawing around objects in a single frame and specifying the type of motion emphasis they would like to apply to those objects. This approach makes the problem of tracking features tractable from a Computer Vision point of view, since the general segmentation problem prohibits an automated solution to bootstrapping our feature tracker. It is questionable whether we would prefer to use an automated segmentation technique for this task, were one to exist. Animation by its very nature is driven by an individual's style and creative flair. Although putting traditional animation techniques into practice can be repetitive and tedious work (exhibiting potential for automation), the imagination and direction governing the choice of techniques to apply is better considered to be a function of individual taste. The same idea motivates the well established use of teams of lesser skilled “inbetweeners” in commercial animation. These teams interpolate the key-frames of more experienced animators to cut production times — a practice pioneered back in the 1940s. Thus we find it desirable to allow artistic influence in our system through high level interaction; for example specifying which objects should undergo motion emphasis, and with which techniques. By contrast, automation should free the animator from the mechanics of how, for example, such objects deform — these remain largely constant between animations, as evidenced by the many animator’s handbooks (for example, [169]) which offer guidance on motion emphasis techniques. However, these effects are not rigidly defined; their behaviour can be modulated to fit the animators needs. Likewise, our system allows performance parameters to be set on motion cues, for example constants controlling rigidity in a squash and stretch deformation. We describe these parameters with their respective effects *in situ*.

6.2 Overview of the Subsystem

We now describe the major components of the subsystem responsible for synthesising visual motion cues, leaving detailed explanation to subsequent sections of the chapter. The subsystem has two major components: the Computer Vision component which is responsible for tracking motion of features (e.g. arm, leg, bat or ball), camera motion compensation, and depth ordering of features; and the Computer Graphics component, responsible for the generation of motion cues, and their rendering at the correct depth. We wish for minimal user interaction with the Computer Vision component, which must be robust and general; currently users draw polygons in a single frame to identify features which are then tracked automatically. In contrast, the user is given control over the graphics component via a set of parameters which influence the style in which the motion cues are synthesised.

6.3 Computer Vision Component

The Computer Vision component is responsible for tracking features over the video sequence. A camera motion compensated version of the sequence is first generated; we do not desire camera motion to bias the observed trajectories of subjects. Features are then tracked over this compensated sequence. By analysing occlusion during tracking we determine a relative depth ordering of features, which is later used in the rendering stage to insert motion cues at the correct scene depth.

6.3.1 Camera Motion Compensation

The nature of the motions we wish to emphasise often demands a moving camera to capture them, so necessitating a camera (ego) motion compensation step as the first operation in our subsystem. Ego-motion determination allows the camera to move, to pan, and even compensates for camera wobble. We model inter-frame motion by a homography, and perform compensation by applying a robust motion estimation technique initially proposed by Torr [158].

A number of interest points are first identified within two given images (I and I') using a variant of the Harris corner detector, refined to operate with sub-pixel accuracy (details of this refinement are given in Appendix A.3). Correspondence between interest points is estimated using cross-correlation of local image signal — however this produces many erroneous correspondences. We thus search for an initial approximation to the homography $\underline{\underline{H}}$, between corresponding interest points, using RANSAC [47] to minimise the forward-backward transfer error $E(\underline{\underline{H}})$:

$$E(\underline{\underline{H}}) = \sum_{i=1}^{4} |\underline{\underline{H}}\underline{p}_i - \underline{q}_i| + |\underline{\underline{H}}^{-1}\underline{q}_i - \underline{p}_i| \tag{6.1}$$

where $\underline{\underline{H}}$ is a putative homography proposed by a RANSAC iteration, and $\underline{p}_{[1,4]}$, $\underline{q}_{[1,4]}$ are four pairs of potentially corresponding feature points chosen at random from the complete set of correspondences identified during cross-correlation. On each iteration $\underline{\underline{H}}$ is computed by solving a homogeneous linear system, comprising two linear equations for each inhomogeneous point correspondence $\underline{p}_i = (x_i, y_i)^T \mapsto \underline{q}_i = (x'_i, y'_i)^T$:

$$x'_i(h_7x_i + h_8y_i + h_9) - h_1x_i - h_2y_i - h_3 = 0 \tag{6.2}$$

$$y'_i(h_7x_i + h_8y_i + h_9) - h_4x_i - h_5y_i - h_6 = 0 \tag{6.3}$$

Where $h_{[1,9]}$ represent the nine elements of the matrix transformation for the planar homography (see equation 3.22). The homogeneous system is solved directly in matrix form, using standard linear methods (SVD) to obtain $h_{[1,9]}$:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x'_1x_1 & x'_1y_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y'_1x_1 & y'_1y_1 & y'_1 \\ & \dots & & & & & & \dots & \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x'_4x_4 & x'_4y_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & y'_4x_4 & y'_4y_4 & y'_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \dots \\ h_8 \\ h_9 \end{bmatrix} = 0 \tag{6.4}$$

Having established an estimate for $\underline{\underline{H}}$ using RANSAC, we refine that estimate using a

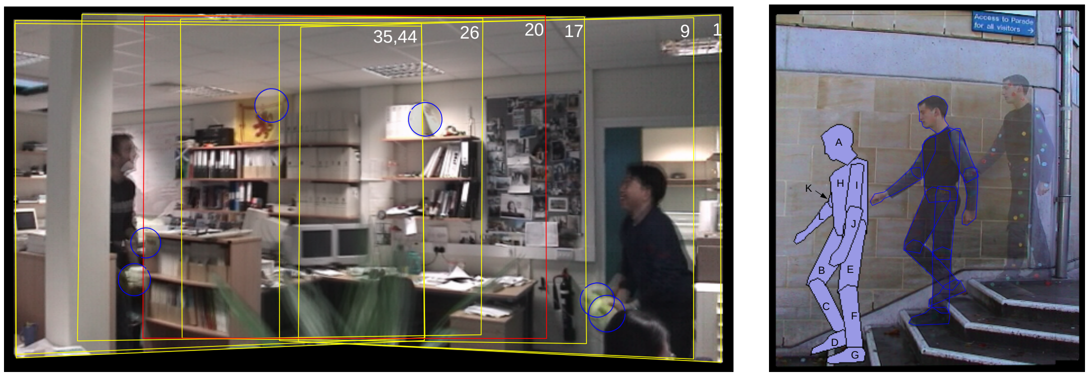


Figure 6-2 Left: The camera compensated *VOLLEY* sequence sampled at regular time intervals. The camera view-port at each instant is outlined in yellow, the tracked feature in blue (see `videos/panorama` in Appendix C). Right: *STAIRS* sequence. (Top) markers are required to track this more complex subject but are later removed automatically (middle). Recovery of relative depth ordering permits compositing of features in the correct order (bottom); labels **A**–**K** correspond to the graph of Figure 6-3.

Levenburg-Marquadt iterative search[2] (after [154]) to minimise the mean squared RGB error $E'(\underline{\underline{H}})$ between image pixels which overlap when transformed by $\underline{\underline{H}}$ (we write the subset of overlapping pixels within image I, as X):

$$E'(\underline{\underline{H}}) = \frac{1}{|X|} \sum_{\underline{x} \subseteq X} \left| I'(\underline{\underline{H}}\underline{x}) - I(\underline{x}) \right|^2 \tag{6.5}$$

Frames are projected to a single viewpoint via homography to produce a motion compensated sequence in which the tracking of features is subsequently performed. This process assumes a static background against which camera motion may be judged. This background should contribute over fifty percent of Harris points in the imaged scene; in practice this is rarely a difficult target to obtain, since the background is often reasonably well textured and occupies much of the imaged area.

6.3.2 Tracking Features through the Compensated Sequence

The general problem of tracking remains unsolved in Computer Vision and, like others, we now introduce constraints on the source video in order to make the problem tractable. Users identify features by drawing polygons, which are "shrink wrapped" to the feature's edge contour using snake relaxation [167], in an identical manner to Section 3.4.1. We assume contour motion may be modelled by a linear conformal affine transform (LCAT) in the image plane, a degenerate form of the affine transform consisting of 4 parameters (uniform scale, orientation, and spatial position — represented by the 4-tuple $\underline{h}(\mathrm{t})=[\mathrm{s(t)},\ \theta(\mathrm{t}),\ \mathrm{x(t)},\ \mathrm{y(t)}]$). This allows planar motion plus scaling

[2]The Levenburg-Marquadt algorithm is a general purpose, non-linear smooth function optimiser — described in [123].

of features. Variation of these 4 parameters is assumed to well approximate a second order motion equation over short time intervals:

$$\underline{h}(t+\delta) \approx \underline{h}(t) + \frac{\mathrm{d}\underline{h}}{\mathrm{d}t}\delta + \frac{\mathrm{d}^2\underline{h}}{2\mathrm{d}t^2}\delta^2 \tag{6.6}$$

In homogeneous coordinates, a transform from time t to t' is instantiated by:

$$\underline{\underline{M}}_{tt'}(s,\theta,x,y) = \underline{\underline{T}}(x,y)\underline{\underline{R}}(\theta)\underline{\underline{S}}(s) \tag{6.7}$$

A feature, F, comprises a set of points whose position varies in time; we denote a point in the t$^{\text{th}}$ frame by the column vector $\underline{x}_t = (x,y)_t^T$. In general these points are not pixel locations, so we use bilinear interpolation to associate a colour value, $I(\underline{x}_t)$ with the point. The colour value comprises the hue and saturation components of the HSV colour model; we wish to mitigate against variations in luminance to guard against errors introduced by lighting changes. Although the LCAT can be derived from two point correspondences we wish to be resilient to outliers, and therefore seek the LCAT $\hat{\underline{\underline{M}}}_{tt'}$ which minimises $E[.]$:

$$E[\underline{\underline{M}}_{tt'}] = \frac{1}{|F|}\sum_{i=1}^{|F|} |I(\underline{x}^i_{t'}) - I(\underline{\underline{M}}_{tt'}\underline{x}^i_t)|^2 \tag{6.8}$$

where the tt' subscript denotes the matrix transform from frame t to t'. In a similar manner to camera motion correction (Section 6.3.1), the transformation $\underline{\underline{M}}_{tt'}$ is initially estimated by RANSAC, and then refined by Levenburg-Marquadt search.

Note that this process assumes the animator is able to isolate an unoccluded example of the feature, in a single video frame; a reasonably light restriction given that there is no requirement for all features to be bootstrapped from within the same individual frame.

By default, several well distributed interest points are identified automatically using the Harris [68] operator (see sequences *CRICKET*, *METRONOME*). In some cases a distinctively coloured feature itself may be used as a marker (see *VOLLEY*, Figure 6-2). In more complex cases where point correspondences for tracking can not be found (perhaps due to signal flatness, small feature area, or similarity between closely neighbouring features), distinctively coloured markers may be physically attached to the subject and later removed digitally (see *STAIRS*, Figure 6-2). In these cases the Harris interest points are substituted for points generated by analysis of the colour distribution in a frame (see Appendix A.5) for full details).

Noise and Occlusion

So far we have overlooked the impact of noise and occlusion upon our tracker. The "best" $\underline{\underline{M}}_{tt'}$ measured will be meaningless in the case of total occlusion, and exact in the case of neither noise nor occlusion. The likelihood L_t of the feature being occluded at any time t may be written as a function of detected pixel error:

$$L_t = exp(-\lambda E_{[\underline{\underline{M}}_{tt'}]}) \tag{6.9}$$

where λ is the reciprocal of the average time an object is unoccluded (this is a data dependent constant which can be varied to tune the tracker), and $E[.]$ is the pixel-wise difference defined in equation 6.8. Thus at any time t we know the estimated LCAT $\underline{\underline{M}}_{tt'}$, and the confidence in that estimate $C_t = 1 - L_t$. We have incorporated a Kalman filter [90] into our tracker which processes video frames sequentially, accepting the LCAT estimate and confidence values over time, and outputting a best estimate value for each instant's LCAT. The Kalman filter "learns" a model of the tracked feature's motion over time; this motion model is factored in to the "best estimate" to output at time t. The credence given to the model, versus that given to the observation, is governed by the confidence in our observation at that time, C_t. We give further details of our Kalman tracking approach in Appendix A.4, but summarise that central to the filter is a 3×3 state transition matrix $\underline{\underline{S}}$ which represents a second order motion equation in the 4D parameter space of the LCAT. The coefficients $\underline{c}(.)$ of this equation are continuously updated, according to the confidence of measurements submitted to the filter, to learn the immediate history of the feature's motion. For example, in the case of a zero confidence ($C_t = 0$) input, the filter will produce an output based entirely on this learnt motion:

$$\begin{aligned} \underline{c}(t') &= \underline{\underline{S}}\underline{c}(t) \\ \underline{c}(t') &= \begin{bmatrix} 1 & t'-t & (t'-t)^2/2 \\ 0 & 1 & t'-t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \underline{h}(t) \\ \frac{\mathrm{d}\underline{h}(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}^2\underline{h}(t)}{\mathrm{d}t^2} \end{bmatrix} \end{aligned} \tag{6.10}$$

Observe that if confidence C_t is low, then the object is likely to have been occluded, because we cannot match the template to the image. If $C_t > 0.5$ we say the object is more likely occluded than not. If the likelihood of occlusion is low, $C_t < 0.5$, and the modelled and observed LCAT disagree then the estimate, due to the second order motion equation, is likely to be in error. We can therefore use a low C_t to detect collisions (see Section 6.4.2).

The addition of a Kalman filter to the system permits the tracker to maintain a lock on features moving under occlusion providing that motion, whilst occluded, approximately

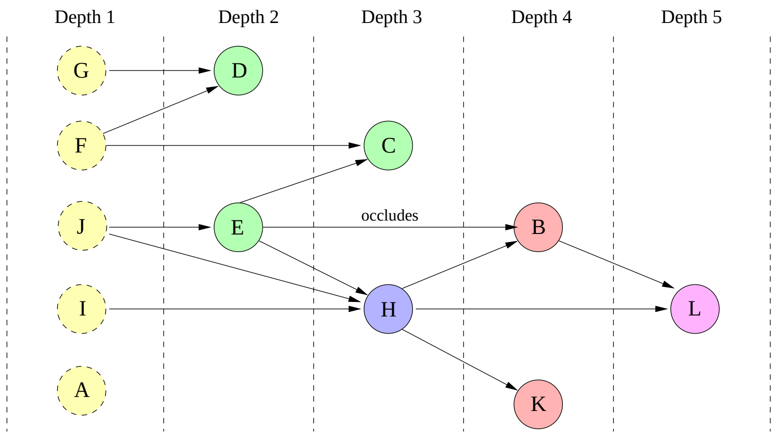


Figure 6-3 An occlusion graph constructed over 150 frames of the *STAIRS* sequence, node letters correspond with Figure 6-2. Dotted lines indicate unoccluded nodes.

follows the second order model estimated by the Kalman filter; this approach has been found sufficiently robust for our needs. However, although this lock is sufficient to re-establish accurate tracking immediately following an occlusion, the predicted LCAT during occlusion is often inaccurate. We refine the tracked motion by deciding at which time intervals a feature is occluded and interpolating between the parameters of known LCATs immediately before and after occlusion. Knowledge of the correct positions of features during occlusion is important when rendering as, although a feature will not be visible while occluded, any applied motion cues may be. Occlusion is also used to determine relative feature depth, and a knowledge of collision events is important later for squash-and-stretch effects.

6.3.3 Recovering Relative Depth Ordering of Features

We now determine a partial depth ordering for tracked features, based on their mutual occlusions over time. The objective is to assign an integer value to each feature corresponding to its relative depth from the camera. We introduce additional assumptions at this stage: 1) the physical ordering of tracked features cannot change over time (potential relaxation of this assumption is discussed later in Section 6.5) ; 2) a tracked feature can not be both in front and behind another tracked feature; 3) lengthly spells of occlusion occur due to tracked features inter-occluding.

For each instance of feature occlusion we determine which interest points were occluded by computing a difference image between tracked and original feature bitmaps. A containment test is performed to determine whether occluded points lie with the bounding contour of any other tracked features; if this is true for exactly one feature, then the occlusion was caused by that feature. We represent these inter-feature relationships as a directed graph, which we construct by gathering evidence for occlusion over the whole sequence. Formally we have a graph with nodes $G_{1..n}$ corresponding uniquely with the set of tracked features, where $G_i \mapsto G_j$ implies that feature G_i occludes G_j (Figure 6-3). Each graph edge is assigned a weight; a count of how many times the respective occlusion is detected.

This graph has several important properties. First, the graph is directed and *should* also be acyclic (i.e. a DAG) since cycles represent planar configurations which cannot occur unless our previous assumptions are violated or, rarely, noise can cause false occlusion. Second, some groups of polygons may not interact via occlusion, thus the resulting graph may not be connected (a forest of connected DAGs is produced). Third, at least one node G_m must exist per connected graph such that $\forall G_{i=1..n} \neg \exists G_i \mapsto G_m$. We call such nodes "unoccluded nodes", since they correspond to features that are not occluded by any other feature over the duration of the video. There must be one or more unoccluding node per connected graph, and one or more graph in the entire forest.

First we verify that the graph is indeed acyclic. If not, cycles are broken by removing the cycle's edge of least weight. This removes sporadic occlusions which can appear due to noise. We now assign an integer *depth code* to each node in the graph; smaller values represent features closer to the camera. The value assigned to a particular node corresponds to the maximum of the hop count of the longest path from any unoccluded node to that node (Figure 6-3). The algorithm proceeds as follows:

```
1: function $determine\_depth\_coding(G_{1..n})$
2: for $i = 1$ to $n$ do
3:     Find the connected graph $g$ s.t. $G_i \in g$.
4:     Identify the unoccluded nodes $U \subseteq g$.
5:     $code = 0$
6:     for each unoccluded node $u \in U$ do
7:         $code = \mathbf{max}(code, \mathbf{longestpath}(u, G_i))$.
8:     end for
9:     assign depth order of node $G_i = code$
10: end for
```

We note that since connected graphs are encoded separately, depth coding is only consistent within individual connected graphs. By definition, features within disconnected graphs do not occlude each other; thus it is not possible to determine a consistent ordering over all connected graphs using occlusion alone. However since this data is required later only to composit features in the correct order, such consistency is not relevant.

6.4 Computer Graphics Component

In this section we show how to construct *correspondence trails*, *deformation bases*, and *occlusion buffers*, all of which are used when rendering motion cues within the video. We begin with an object, which has been tracked over the full course of a video, as already described. We analyse tracking data and show how to produce an *animated object* which is a re-presentation of the original, subject to the placement of augmentation cues (streak-lines and ghosting), deformation cues (squash and stretch and other distortions), and suitable occlusion handling. We work entirely with two-dimensional data. Each feature has two cels associated with it, one for *augmentation cues* such as streak-lines, the other for *deformation cues* such as squash and stretch. The cels are composited according to the depth ordering of features, from the furthest to the nearest; for a given feature, deformation cels are always in front of augmentation cels (Figure 6-11).

6.4.1 Motion Cues by Augmentation

Augmentation cues, such as streak-lines and ghosting, are common in traditional animation (Figure 6-1). Streak-lines can ostensibly be produced on a per frame basis by attaching lines to a feature's trailing edge, tangential to the direction of motion [80]. Unfortunately, such an approach is only suitable for visualising instantaneous motion, and produces only straight streak-lines. In addition, these "lines" often exhibit distract-

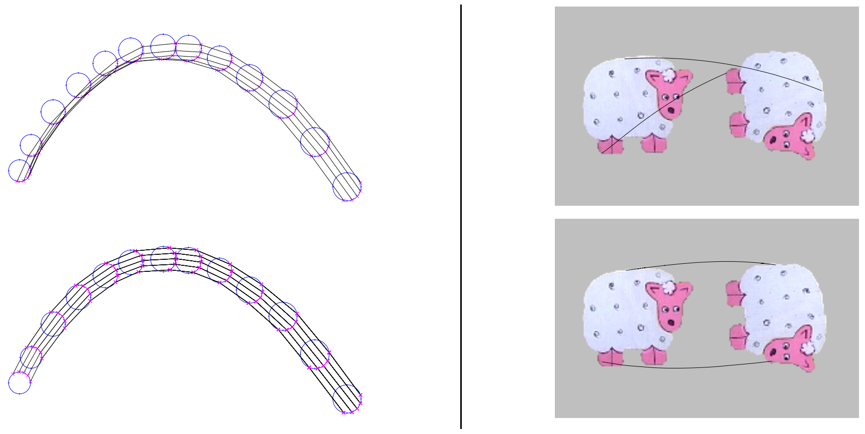


Figure 6-4 Demonstrating that point trajectories do not generally form artistic streak-lines. Corresponding the polygon vertices returned by the tracker for the ball in *VOLLEY* creates unsightly bunching of lines; in fact, in this example such trajectories are ambiguous due to symmetry. However the problem remains even in the case of unambiguous point trajectories, for example that of the sheep (right) where streak-lines converge and cross. We match trailing edges, form correspondence trails, and filter them using heuristics to place streak-lines in a manner similar to that of traditional animators. Our correspondence strategy produces a closer approximation the streak-lines drawn by an animator (bottom-left, bottom-right).

ing wobbles over time due to small variations in a feature's trajectory. By contrast, animators tend to sketch elegant, long curved streaks which emphasise the essence of the motion historically. In Figure 6-4 we show that the trajectories generated by imaged points do not necessarily correspond to an animator's placement of streak-lines. Readily available techniques, such as optical flow [54], are thus infeasible for producing streak-lines for two main reasons: (1) Optical flow algorithms tend to operate over periods of time that are much shorter than typical streak-lines represent. This makes them susceptible to noise, and they fail at occlusions. (2) In general, streak-lines do not trace the motion of a single point on an object (as is the intent with optical flow), but depict the overall "sense-of-motion" of an object.

Streak-line placement is therefore a non-trivial problem: they are not point trajectories, features tend to move in a piecewise-smooth fashion, and we must carefully place streak-lines on features. To produce streak-lines we generate *correspondence trails* over the trailing edge of a feature as it moves, we then segment trails into smooth sections, which we filter to maximise some objective criteria. We finally render the chosen sections in an artistic style.

We begin by sampling the feature boundary at regular spatial intervals, identifying a point as being on the trailing edge if the dot product of its motion vector with the external normal to the boundary is negative. Establishing correspondence between trailing edges is difficult because their geometry, and even connectivity, can vary from frame to frame (as a function of motion). The full LCAT determined during tracking cannot be used, as this establishes point trajectories (which we have observed are not streak-lines); we wish to establish correspondence between feature silhouettes.

We establish correspondence trails by computing the instantaneous tangential velocity of a feature's centroid $\underline{\mu}$. A translation and rotation is computed to map the normalised motion vector from $\underline{\mu}$ at time t to time t'. Any scaling of the feature is performed using the scale parameter of the LCAT determined during tracking. Points on the trailing edge at time t are now translated, rotated, and scaled. Correspondence is established between these transformed points at time t, and their nearest neighbours at time t', this forms a link in a correspondence trail. This method allows the geometry of the trailing edge to vary over time, as required.

Animators tend to draw streak-lines over smooth sections of motion. Therefore the correspondence trails are now segmented into smooth sections, which are delimited by G^1 discontinuities — knots in piecewise cubic curves. This results in a set of smooth sections, each with a pair of attributes: 1) a function $\underline{G}(s)$ where $s \in [0, 1]$ is an arc-

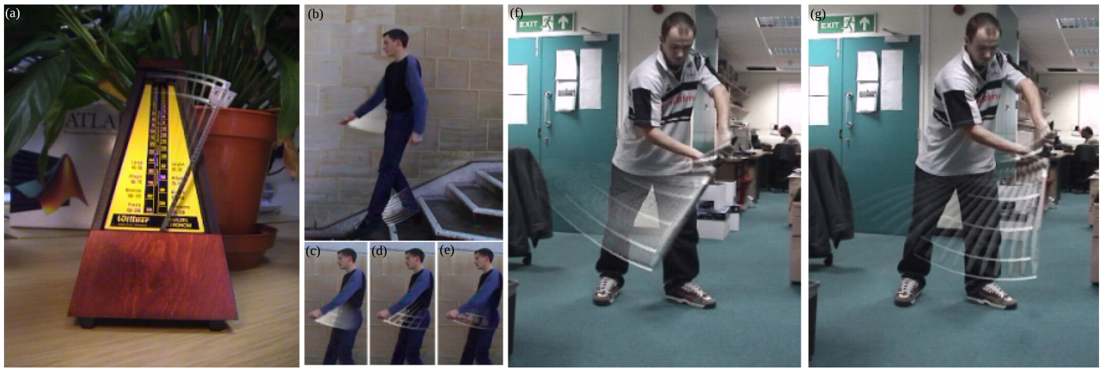


Figure 6-5 A selection from the gamut of streak and ghosting line styles available: Ghosting lines may be densely sampled to emulate motion blur effects (b,c,f) or more sparsely for ghosting (a,d). The feature itself may be ghosted, rather than the trailing edge, to produce Futurist-like echos (e,g). Varying the overlap constant w influences spacing of streak-lines (b,f). The decay parameters of streak and ghosting lines may be set independently (a).

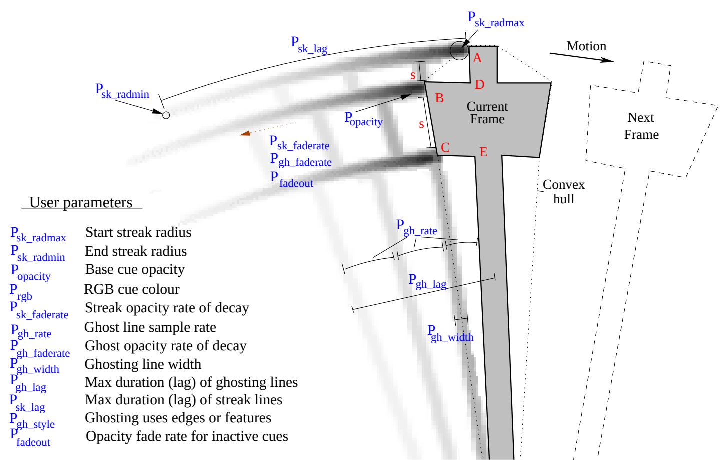
length parameterisation of the curve trajectory; 2) a lookup table $g(.)$ mapping from an absolute time index t (from the start of the clip) to the arc-length parameter, i.e. $s = g(t)$, thus recording velocity along the spatial path $\underline{G}(s)$. The inverse lookup function $g^{-1}(.)$ is also available. Clearly the curve exists only for a specific time period $[g^{-1}(0), g^{-1}(1)]$; we call this interval the duration of the curve.

The association between each smooth section and its data points is maintained. These data points are used to filter the set of smooth sections to produce a subset σ of manageable size, which contains optimal paths along which streak-lines will be drawn.

Our filtering selects curves based on heuristics derived from the practice of animators [130] who favour placement of streak-lines on sites of high curvature and on a feature's convex hull. Long streak-lines and streak-lines associated with rapid motion are also preferred, but close proximity to other co-existing streak-lines is discouraged. We select streak-line curves using a greedy algorithm to maximise the recursive function $H(.)$, on each iteration i adding a new element to σ (initially empty).

$$\begin{aligned} H(0) &= 0 \\ H(i+1) &= H(i) + (\alpha v(x) + \beta L(x) - \\ &\quad \gamma D(x) - \delta\omega(x, \sigma; \eta) + \zeta\rho(x)) \end{aligned} \tag{6.11}$$

where x is the set of spatial points associated with a smooth section. $L(x)$ is the length of a smooth section, $v(x)$ is the "mean velocity" defined as $L(x)/t(x)$ in which $t(x)$ is the temporal duration of the smooth section. $\rho(x)$ is the mean curvature of feature boundary at points in x. $D(x)$ is the mean shortest distance of points in x from the



P_{sk_radmax}	Start streak radius
P_{sk_radmin}	End streak radius
$P_{opacity}$	Base cue opacity
P_{rgb}	RGB cue colour
$P_{sk\_faderate}$	Streak opacity rate of decay
P_{gh_rate}	Ghost line sample rate
$P_{gh\_faderate}$	Ghost opacity rate of decay
P_{gh_width}	Ghosting line width
P_{gh_lag}	Max duration (lag) of ghosting lines
P_{sk_lag}	Max duration (lag) of streak lines
P_{gh_style}	Ghosting uses edges or features
$P_{fadeout}$	Opacity fade rate for inactive cues

Figure 6-6 Left: User parameters used, in conjunction with weights of equation 6.11, to influence augmentation cue placement.

convex hull of the feature. $\omega(x, \sigma; \eta)$ measures the maximal spatiotemporal overlap between x and the set of streak-lines chosen on previous iterations. From each curve we choose points which co-exist in time, and plot the curves with width η returning the intersected area. Constant w is user defined, as are the constant weights $\alpha, \beta, \gamma, \delta$, and ζ; these give artistic control over the streak-line placement (see Figure 6-5). Iteration stops when the additive measure falls below a lower bound.

We are now in a position to synthesise two common forms of augmentation cue; streak-lines and ghosting lines — both of which are spatiotemporal in nature. A streak-line is made visible at some absolute time t and exists for a duration of time Δ. The streak-line is rendered by drawing a sequence of discs along the smooth section with which it is associated, starting at spatial location $\underline{G}(g(t))$ and ending at $\underline{G}(g(max(g^{-1}(0), t - \Delta)))$. The streak-line is rendered by sweeping a translucent disc along the smooth section (backwards in time) which grows smaller and more transparent over time. These decays are under user control.

Ghosting lines depict the position of a feature's trailing edge along the path of the streak-line, and are useful in visualising velocity changes over the course of the streak-line. Ghosting lines are rendered by sampling the trailing edge at regular time intervals as the streak-line is rendered, interpolating if required. The opacity of ghosting lines is not only a function of time (as with streak-lines) but also a function of speed relative to other points on the trailing edge; this ensures only fast moving regions of edge are ghosted. Users may control the sampling rate, line thickness, and decay parameters to stylise the appearance of the ghosting lines (Figure 6-6).

6.4.2 Motion Cues by Deformation

Deformation cues are further tools available to an animator. They are routinely used to depict inertia or drag; to emphasise the line of action through a frame; or to show the construction or flexibility of an object. Features are cut from the current video frame, and motion dependent warping functions applied to render the deformation cue cel for each feature.

Deformations are performed in a curvilinear space, the basis of which is defined by the local trajectory of the feature centroid, and local normals to this curve. This trajectory has exactly the same properties as any smooth section (see Section 6.4.1). A local time-window selects a section of the centroid trajectory, and this window moves with the object. The instantaneous spatial width of this window is proportional to instantaneous centroid velocity. At each instant, t we use the centroid trajectory to establish a curvilinear basis frame. First we compute an arc-length parameterisation of the trajectory: $\underline{\mu}(r)$ in which $r = \int_{i=t}^{t'} \left|\dot{\underline{\mu}}(i)\right| di$. Next we develop an ordinate at an arc-length distance r from the instant; the unit vector $\underline{n}(r)$ perpendicular to $\dot{\underline{\mu}}(t)$. Thus, at each instant t we can specify a point in the world-frame using two coordinates, r and s:

$$\underline{x}_t(r, s) = \underline{\mu}(r) + s\underline{n}(r) \tag{6.12}$$

We can write this more compactly as $\underline{x}_t(\underline{r}) = \underline{C}(\underline{r})$, where $\underline{r} = (r, s)^T$, and maintain the inverse function $\underline{r}_t(\underline{x}) = \underline{C}^{-1}(\underline{x})$ via a look-up table. This mapping, and its inverse, comprise one example of a *deformation basis*.

The above analysis holds for most points on the centroid trajectory. It breaks down at *collision points*, because there is a discontinuity in velocity. We define a collision point as one whose location cannot be satisfactorily predicted by a second order motion equation (constructed with a Kalman filter, see Section 6.3.2). This definition discriminates between G^1 discontinuities in trajectory which are formed by, say, simple harmonic motion, and true collisions which are C^1 discontinuous (see Figure 6-7).

At a collision point we establish an orthonormal basis set aligned with the observed collision plane. We assume the angle of incidence and reflection are equal, and hence the unit vector which bisects this angle is taken to be the ordinate. The abscissa lies in the collision plane. We define this new basis set as an additional instance of a deformation basis, and write the mapping to and from the world frame using notation consistent with the curvilinear basis. In addition, we compute the *impact parameter* of the collision, which we define as the distance (D) from the object's centroid to its

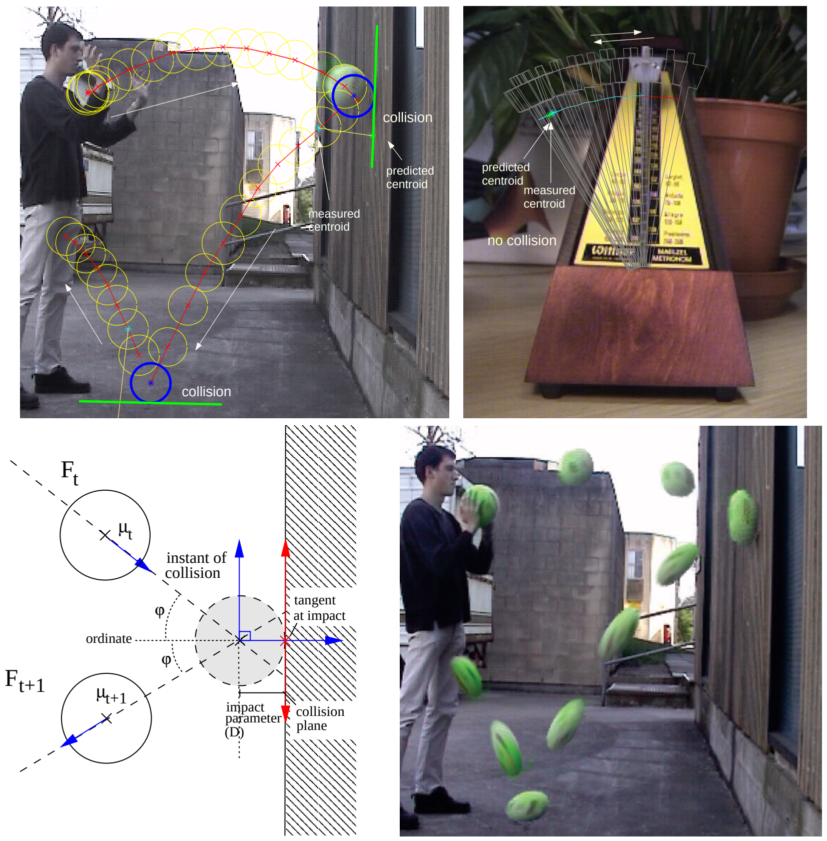


Figure 6-7 Top: bounces are detected as collisions (left), whilst the simple harmonic motion of the metronome is not (right). Predicted motions shown in green, observed position in cyan. Bottom: Collision geometry of the first impact in the *BOUNCE* sequence. The time lapse image (right) shows the resulting deformations (see `videos/bounce_deformationonly`).

boundary, in the direction of the negative ordinate. Note we compensate for temporal sampling around the true collision instant by intersecting extrapolated impact and rebound trajectories (Figure 6-7).

An animated object is, in general, a deformed version of the original. Squash and stretch tangential to instantaneous motion leads to visually unattractive results; it is better to not only squash and stretch, but also to bend the object along the arc of its centroid trajectory. Let $\underline{x}(t)$ be any point in the object. We transform this point with respect to a single deformation basis into a new point $\underline{y}(t)$, given by

$$\underline{y}(t) = \underline{C}(\underline{A}_t[\underline{C}^{-1}(\underline{x}(t))]) \tag{6.13}$$

where $\underline{A}_t[.]$ is some deformation function at time t. In the case of squash and stretch the deformation is an area-preserving differential scale that depends on instantaneous

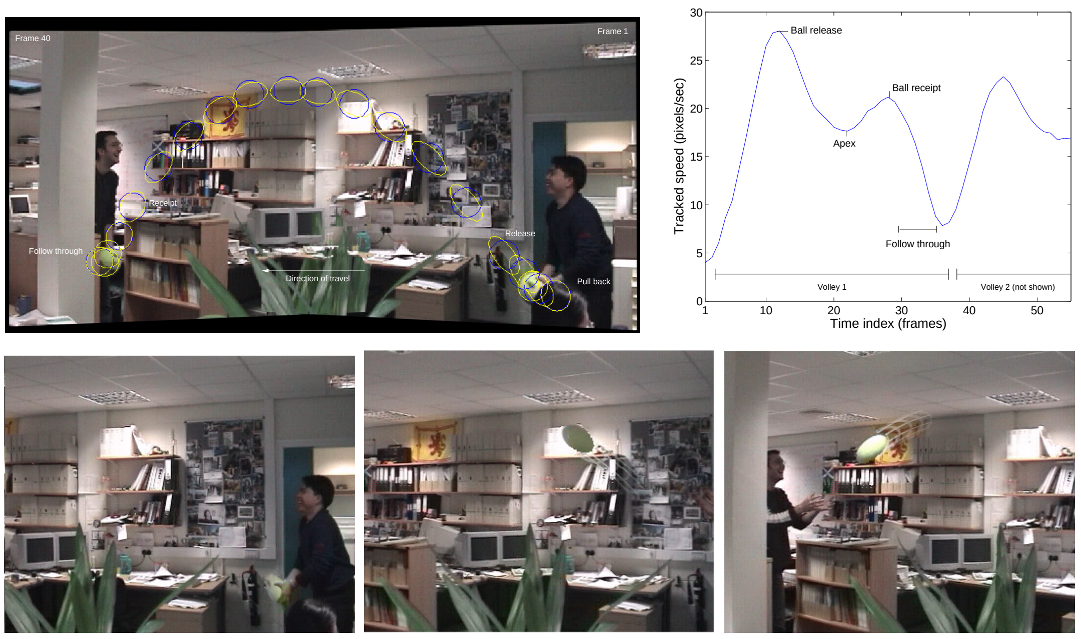


Figure 6-8 Top left: Illustrating the squash and stretch effect; eccentricity varies as a function of tangential speed (right). Bottom: Frames from the *VOLLEY* sequence exhibiting squash and stretch. Observe the system handles both large scale camera motion, and lighting variation local to the ball (`videos/volley_streaks`).

speed $|\underline{\dot{\mu}}(t)|$:

$$\underline{\underline{A}} = \begin{bmatrix} k & 0 \\ 0 & \frac{1}{k} \end{bmatrix} \tag{6.14}$$

$$k = 1 + \frac{K}{2}(1 - cos(\pi \frac{v^2 + 1}{2}))$$

$$v = \begin{cases} 0 \text{ if } |\underline{\dot{\mu}}| < V_{min} \\ 1 \text{ if } |\underline{\dot{\mu}}| >= V_{max} \\ (|\dot{\mu}| - V_{min})/(V_{max} - V_{min}) \text{ otherwise} \end{cases} \tag{6.15}$$

However during collision animators tend to vary the amount of "stretch" in proportion to acceleration magnitude, rather than speed. In addition, the deformation due to impact takes place in the deformation basis defined about the collision site, as discussed earlier in this Section.

We linearly interpolate between the deformation caused by a "standard" deformation basis with that caused by a "collision" deformation basis. Suppose $\underline{p}(t) \mapsto \underline{q}(t)$ and $\underline{p}(t) \mapsto \underline{q}'(t)$, respectively, then:

$$\underline{r}(t) = f(d)\underline{q}(t) + (1 - f(d))\underline{q}'(t) \tag{6.16}$$

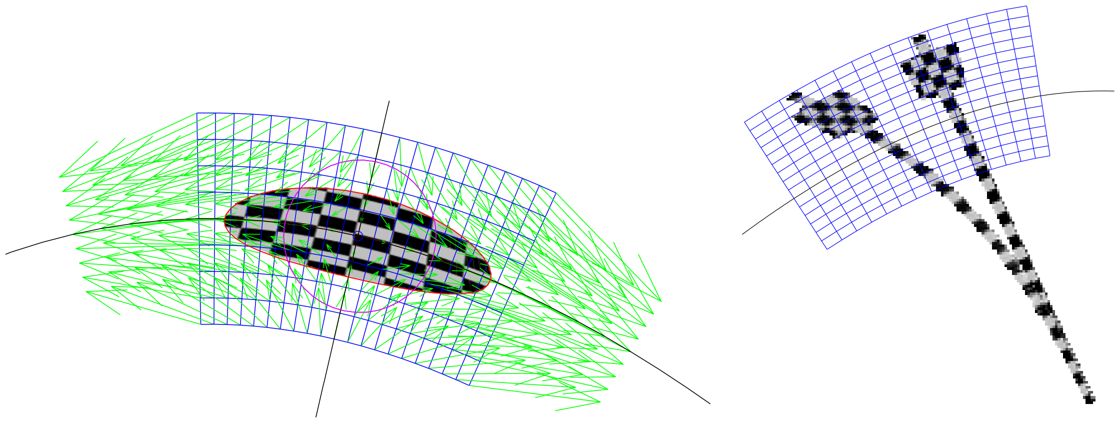


Figure 6-9 Examples of curvilinear warps used to create deformation cues in the *VOLLEY* (left) and *METRONOME* (right) sequences.

where $f(d) = sin(\frac{\pi}{2}\min(1, d/(sD))$ in which D is the impact parameter of the collision (defined previously), and s is a user parameter which controls the spatial extent over which collision influences the deformation. This transfer function gives a suitably smooth transition between the two deformation bases. As a note, the mapping to $\underline{q}'(t)$ not only has to scale the object but also translate it toward the impact point, so that the edge of the deformed object touches the collision plane. This translation vector is expressed within the collision basis as:

$$\begin{bmatrix} 0 \\ D(1 - \frac{1}{k}) \end{bmatrix} \tag{6.17}$$

Non-linear deformations are also possible, and these can be used to create bending effects. We can form warping functions which depend on each point's velocity and acceleration as well as its position. We write $x' = \underline{C}(\underline{A}[\underline{C}^{-1}(\underline{x}); \underline{\dot{x}}, \underline{\ddot{x}}])$, where $\underline{A}(\underline{r}; .)$ is a functional used, for example, to suggest increased drag or inertia. A typical functional operates on each component of $\underline{r} = (r_1, r_2)^T$ independently; to create effects suggesting drag we use:

$$r_1 \quad \leftarrow \quad r_1 - F(\frac{2}{\pi} arctan(|\underline{\dot{x}}_i|))^P sign(\underline{\dot{x}}_i) \tag{6.18}$$

F and P are user defined constants which affect the appearance of the warp. For example, large F give the impression of heavy objects and P influences the apparent rigidity of objects. By substituting acceleration for velocity, and adding rather than subtracting from r_1 we can emphasise inertia of the feature (see Figure 6-10):

$$r_1 \quad \leftarrow \quad r_1 + F(\frac{2}{\pi} arctan(|\underline{\ddot{x}}_i|))^P sign(\underline{\ddot{x}}_i) \tag{6.19}$$

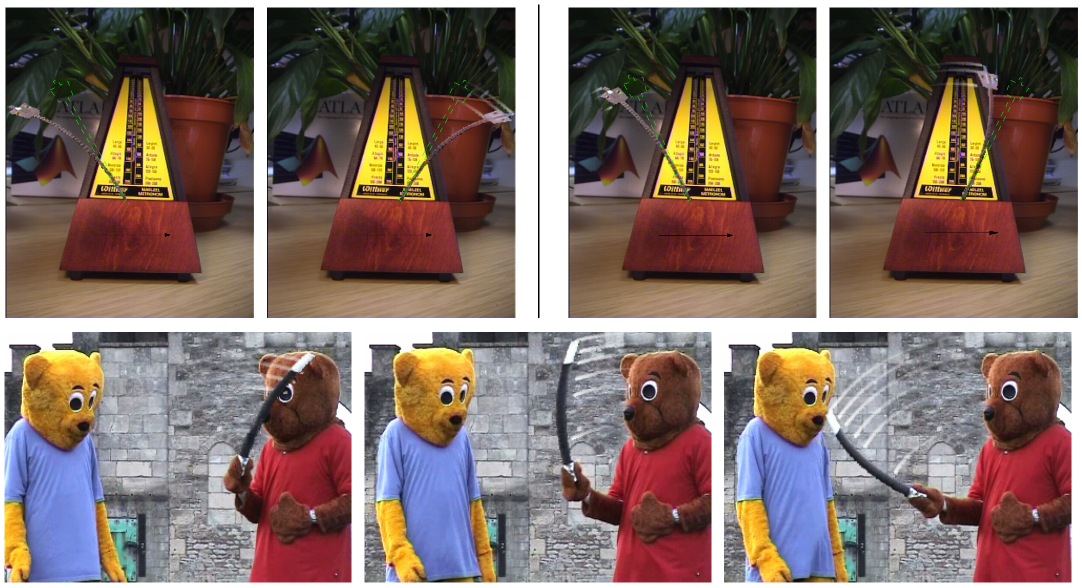


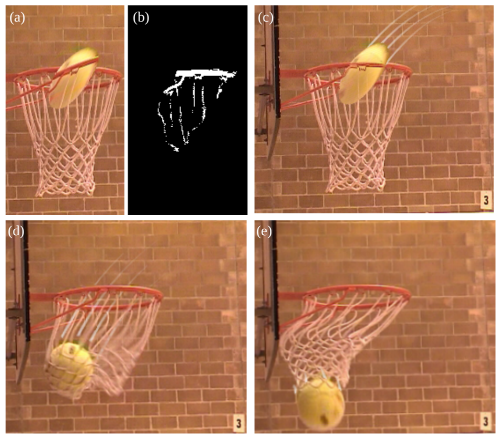
Figure 6-10 Top: Frames from *METRONOME* suggesting inertia (left) and drag (right) effects through deformation, original feature outline in green. Bottom: Deforming a rigid wand in the *WAND* sequence, using a velocity based "drag" effect — streak-lines are warped along with the object, adding to the effect (`videos/wand_cartoon`).

Finally, we ensure visual consistency by deforming not only features, but also their associated augmentation cue cels containing streak-lines or ghost lines.

6.4.3 Rendering in the Presence of Occlusion

In any real video a tracked feature may become occluded by other elements of the scene. Naïvely, all pixels inside the bounding contour of a feature will be included in that feature and so are subject to deformation. Consequently, it is easy to mistakenly warp parts of occluding objects; we now explain how to avoid producing these unwelcome artifacts.

We begin our explanation with an illustrative example. Consider the *BASKETBALL* video clip of Figure 6-12, in which a fast moving ball passes behind a hoop. We may identify pixels as belonging to an occluding object, in this case the ring, by forming a difference image (using the hue and saturation components in HSV space, again to affect simple invariance to illumination changes) between the feature region in the current frame and the feature itself. This yields a weighted mask of occluding pixels which are re-textured by sampling from feature regions in neighbouring unoccluded frames. An approximation to the unoccluded feature is thus reconstructed, and after deformation occluding pixels may be recomposited to give the illusion of the ball passing 'behind' occluding objects (Figure 6-11d).



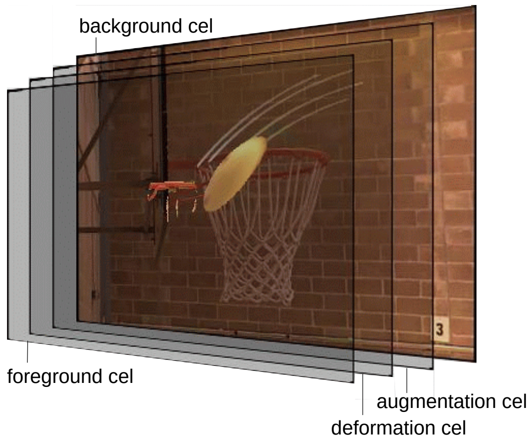


Figure 6-11 Left: (a) naïve deformation creates artifacts, but our method does not (c). An occlusion buffer is constructed over time (b), allowing augmentation cues to be handled in the wake of the feature. The system breaks down (d) after impact erratic movement of the netting causes the occlusion buffer to empty and cues to be incorrectly drawn in front of that netting. Right: Frames are rendered as layers, in back-to-front order. In this case a single object (the ball) is tracked to produce augmentation cues (streak-lines) and deformations (squash and stretch). Occluding pixels are overlaid in the foreground to produce the illusion of the ball deforming behind the hoop (`videos/basket_rendered`).

The situation becomes more complicated when one considers that a feature may be deformed (stretched, for example, as part of squash and stretch) outside its original polygonal boundary. Similarly, augmentation cues should pass behind occluding objects; these cues are also drawn outside of the original boundary. Fortunately we can make progress, since these cues traverse an identical path to that of the feature itself. We construct an *occlusion buffer* by examining a small temporal window of frames centred upon the current time instant, summing the difference images generated whilst handling occlusion in the manner described in the preceding paragraph. Using this buffer we may determine which pixels will occlude cues such as streak-lines, so long as those pixels do not change in the time interval between the real feature passing and the augmentation cues being drawn. The occlusion buffer contains a difference image for occlusion and the RGB value of each pixel. Pixels in the buffer are deleted when the difference between the stored colour for a pixel, and the measured colour of that pixel at the current time is significant. In this case the occluding pixel has moved, obsoleting our knowledge of it.

This algorithm works acceptably well over short time intervals, but in general we can hope only to mitigate against artifacts generated by occluding objects which we have not explicitly tracked. For example, Figure 6-11d demonstrates how streak-lines are able to correctly pass behind complex netting, whilst that netting is stationary. The system breaks down in Figure 6-11e when the netting moves erratically after ball impact, causing the occlusion buffer to quickly empty and streak-lines to be incorrectly drawn in front of the netting.

6.4.4 Compositing and Rendering

The graphics component composits cels to create each frame of the animation. To render a particular frame, a *background cel* is first generated by subtracting features from the original video; determining which pixels in the original video constitute features by projecting tracked feature regions from the camera-compensated sequence to the original camera viewpoint. Pixels contributing to feature regions are deleted and absent background texture sampled from locally neighbouring frames in alternation until holes are filled with non-feature texture. This sampling strategy mitigates against artifacts caused by local lighting changes or movement.

Once augmentation and deformation cels have been rendered for each feature, cels are composited to produce an output video frame. Cels are projected by homography to coincide with the original camera viewpoint, and composited onto the background in reverse depth order. Finally, a *foreground cel* is composited, which contains all pixels that occlude identified objects. These pixels are taken from the occlusion buffer described in Section 6.4.4. Thus motion cues appear to be inserted into video at the correct scene depth (Figure 6-11).

6.5 Summary and Discussion

We have described and demonstrated a subsystem, within the Video Paintbox, for the artistic rendering of motion within video sequences. The subsystem can cope with a moving camera, lighting changes, and presence of occlusion. Aside from some (desirable) user interaction when boot-strapping the tracker, and the setting of user parameters, the process is entirely automatic. Animators may influence both the placement and appearance of motion cues by setting parameters at a high conceptual level; specifying the objects to which cues should be attached, and the type of motion emphasis desired. Parameters may also be set on individual motion cues to stylise their appearance, for example streak-line spacing or squash and stretch rigidity.

We have shown that through high level analysis of features (examining motion over blocks of video with large temporal extent, rather than on a per frame basis) we may produce motion cues closely approximating those used in traditional animation (Figure 6-1). For example, we have shown that streak-lines are generally poorly represented by point trajectories [80], and should instead be rendered as smooth curves capturing the essence of a trajectory over time. Although the former are obtainable via temporally local processing, the latter require analysis which necessitates a higher level of temporal analysis for trajectory analysis. Likewise deformation effects such as squash and stretch demand detection and localisation of collisions, the analysis of which requires examination large temporal window around the collision instant. The spatiotemporal occlusion buffer is another example of higher level temporal analysis performed by the motion emphasis subsystem.

Further developments could address the relaxation of some of the assumptions made in the design of the subsystem. For example, violations of depth assumptions in Section 6.3.3 are detectable by the presence of heavily weighted cycles in the depth graph. It may be possible to segment video into a minimal number of chunks exhibiting noncyclic depth graphs, and in doing so recover relative feature depth under more general motion. The tracker could also benefit from more robust, contemporary methods such as CONDENSATION [85], although this may involve imposing a more restrictive motion model than Kalman filtering requires, which is why we opted for the latter. We can imagine a system that uses CONDENSATION tracker for common objects, such as people walking, but defaults to a Kalman filter in more general cases. Further developments might evaluate the robustness of the algorithm using both ground truth comparisons for measures such as velocity, as well as processing sequences exhibiting distinctly non-planar motion.

Although these incremental changes would serve to enhance robustness, the most interesting extensions of this work lie in an expansion of the gamut of motion cue styles, and the incorporation of motion emphasis with the coherent artistic shading of video. These two developments are documented in Chapters 7 and 8 respectively.

A selection of source and rendered video clips have been included in Appendix C.

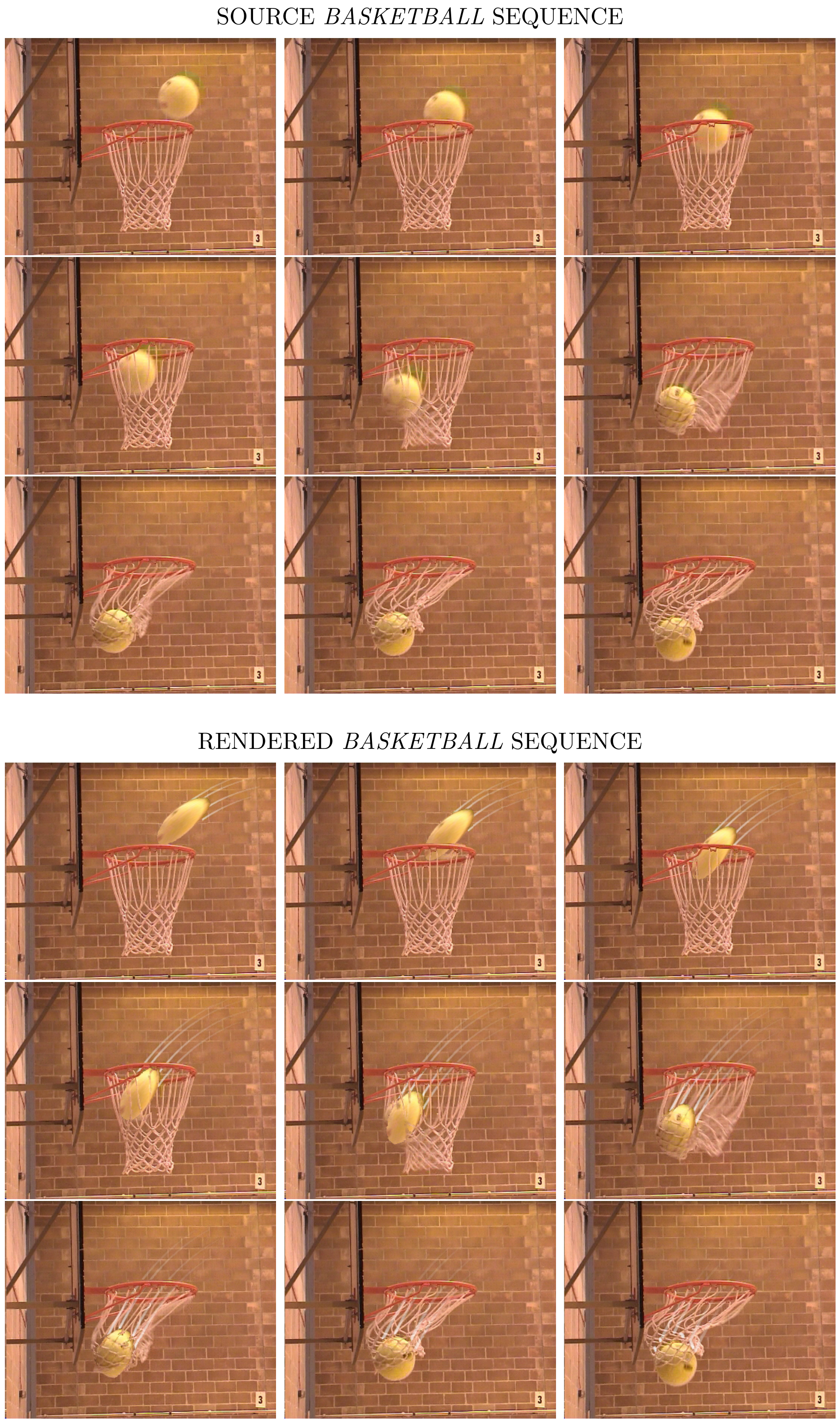


Figure 6-12 Stills from the original (`videos/basket_source`) and rendered (`videos/basket_rendered`) versions of the *BASKETBALL* sequence. This sequence not only demonstrates the use of squash and stretch deformations, streak-lines and ghosting, but also serves to illustrate our occlusion handling.

RENDERED *BOUNCE* SEQUENCE

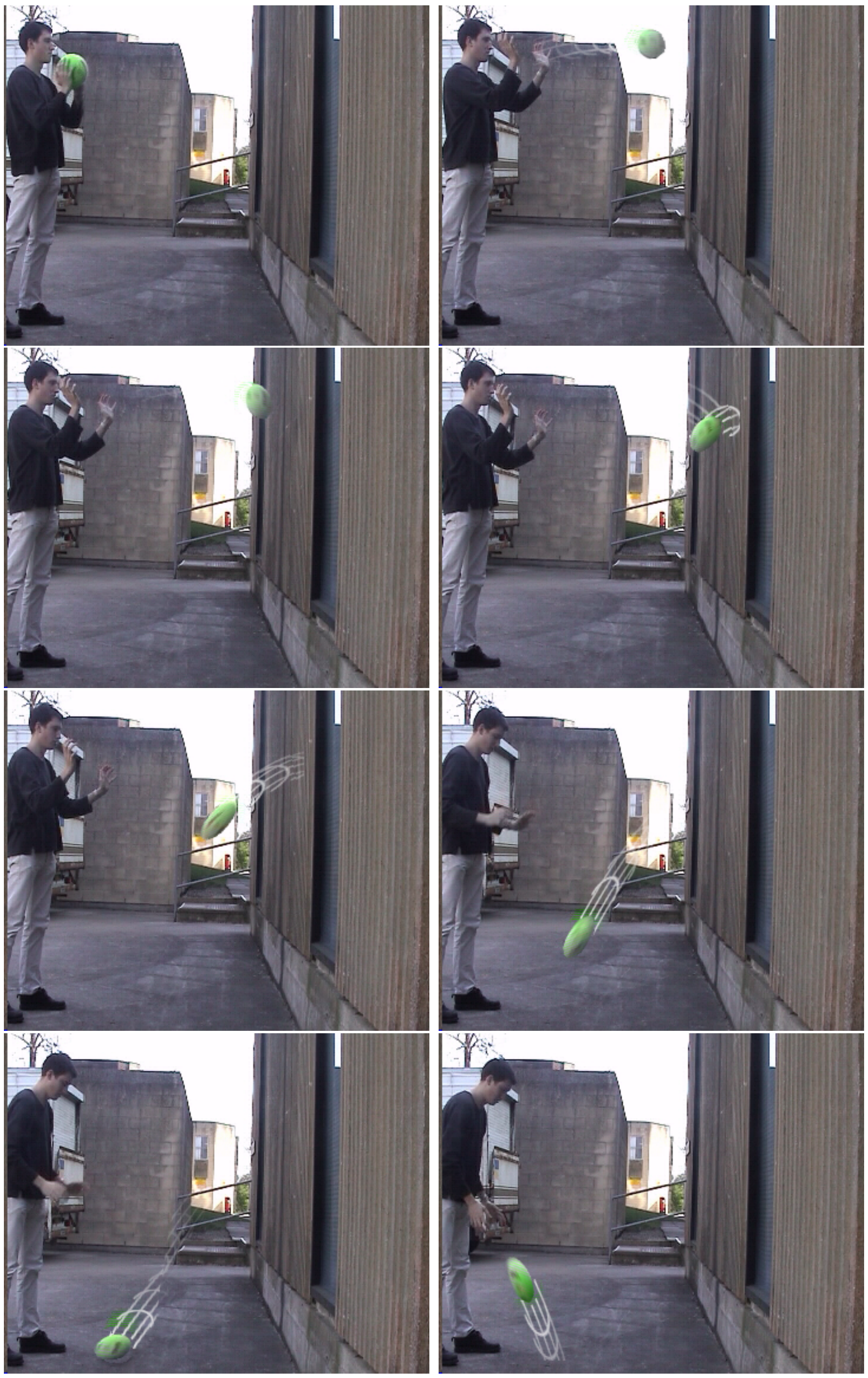


Figure 6-13 Stills from the rendered *BOUNCE* sequence, demonstrating both augmentation cues and our squash and stretch deformation effect (see `videos/bounce_motiononly`). This sequence demonstrates the correct detection and handling of collision events.

RENDERED *VOLLEY* SEQUENCE

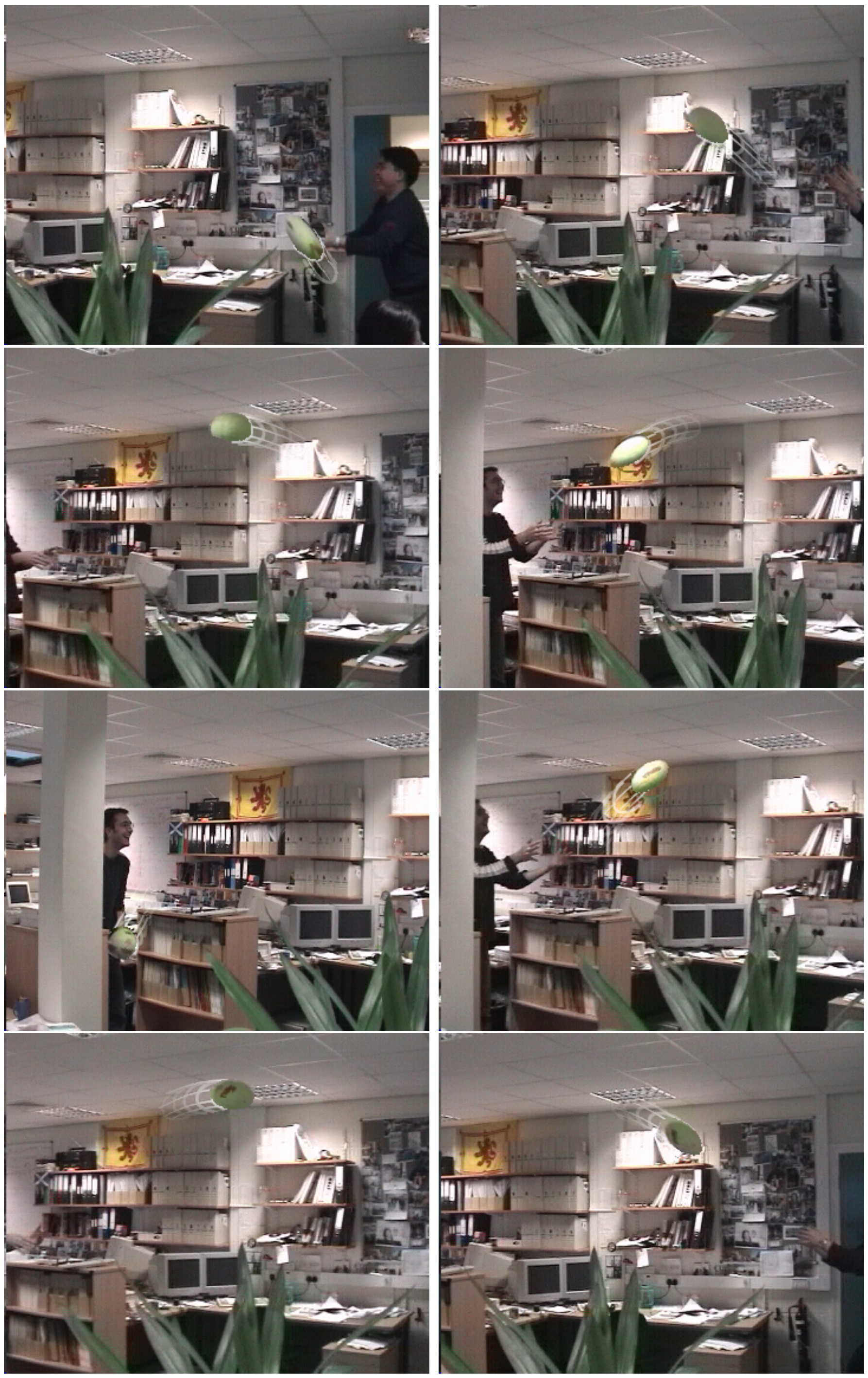


Figure 6-14 Stills from the rendered *VOLLEY* sequence, demonstrating both augmentation cues and our squash and stretch deformation effect (see `videos/volley_motiononly`). This sequence demonstrates that the system is able to compensate for large scale camera motion, and handle local lighting changes (on the ball).

RENDERED *BALLET* SEQUENCE

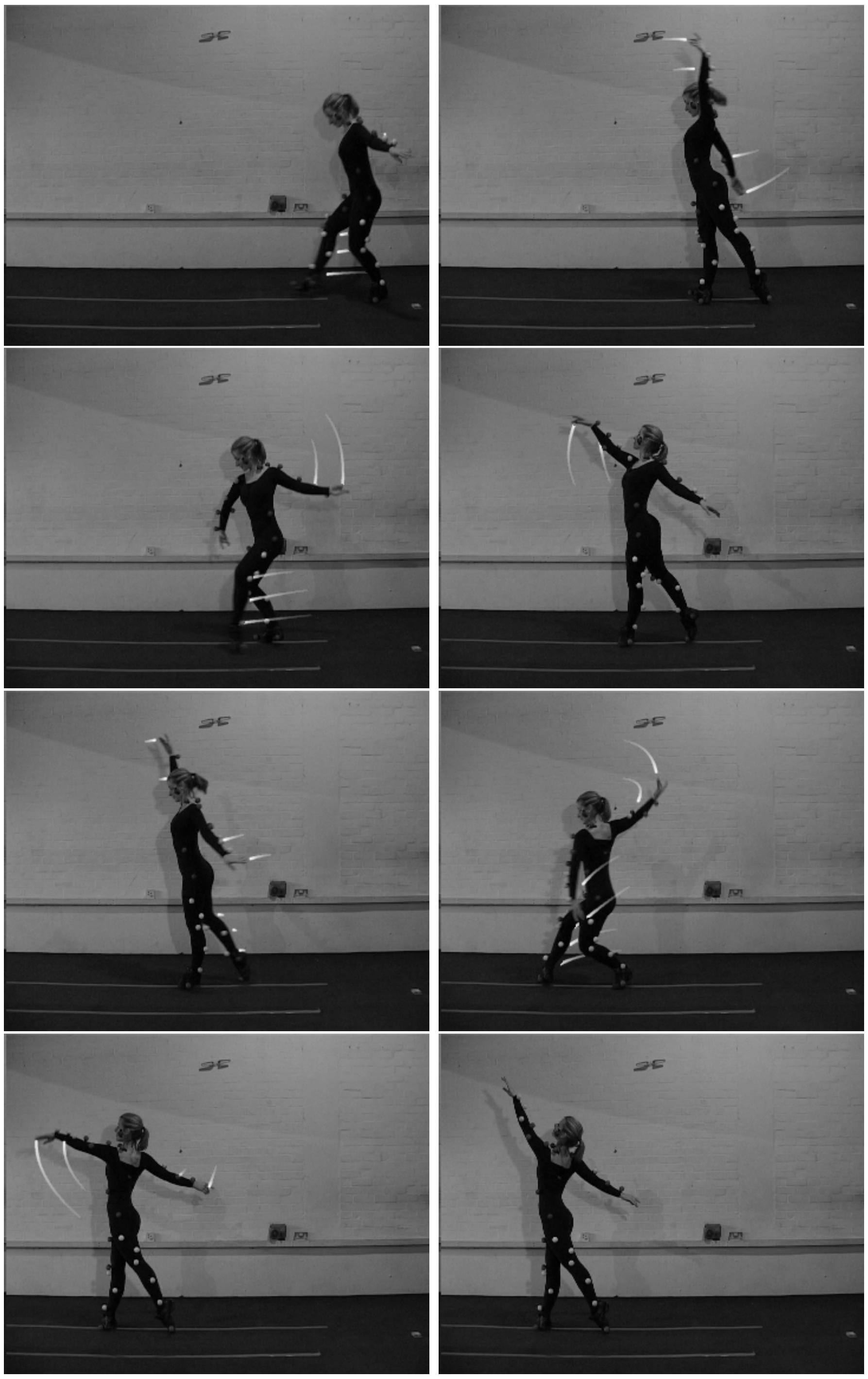


Figure 6-15 Stills from the rendered *BALLET* sequence, demonstrating the insertion of augmentation cues in to the sequences at the correct scene depth (`videos/ballet_streaks`).

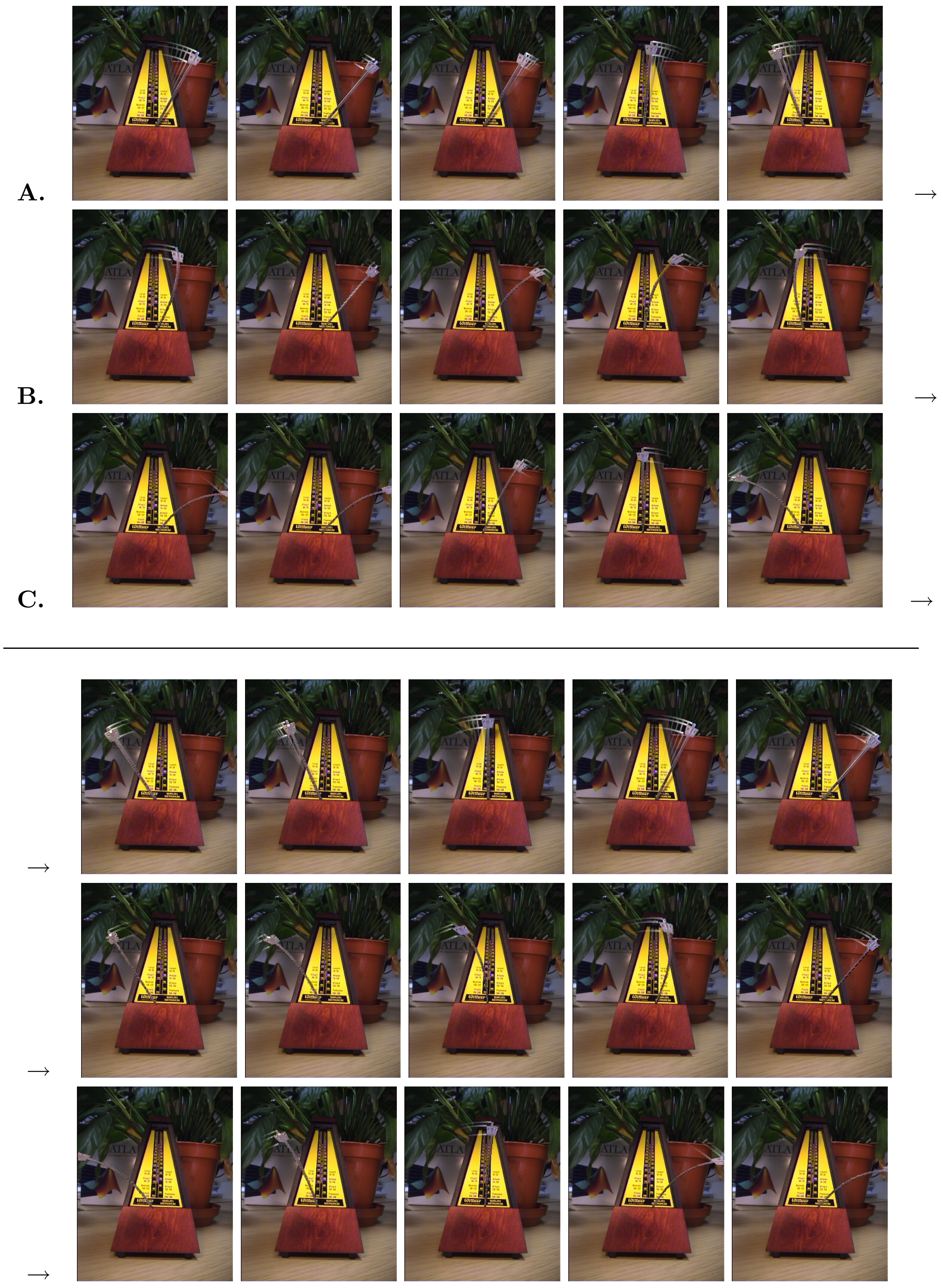


Figure 6-16 Ten stills taken simultaneously (time runs from left to right) over three different renderings of the *METRONOME* sequence: (A) Augmentation cues only (`videos/metro_streaks`); (B) Deformation emphasising drag (velocity dependent warp) with streak lines (`videos/metro_warp_veloc`); (C) Deformation emphasising inertia (acceleration dependent warp) with streak lines (`videos/metro_warp_accel`). Approximately one still per five frames.