Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

Cartoon-style Rendering of Motion from Video

J. P. Collomosse[1], D. Rowntree[2] and P. M. Hall[1]

[1] Department of Computer Science, University of Bath, Bath, England.
[2] Nanomation Ltd., 6 Windmill Street, London, England.

Abstract

The contribution of this paper is a novel non-photorealistic rendering (NPR) system capable of rendering motion within a video sequence in artistic styles. A variety of cartoon-style motion cues may be inserted into a video sequence, including augmentation cues (such as streak lines, ghosting, or blurring) and deformation cues (such as squash and stretch or drag effects). Users may select from the gamut of available styles by setting parameters which influence the placement and appearance of motion cues.

Our system draws upon techniques from both the vision and the graphics communities to analyse and render motion and is entirely automatic, aside from minimal user interaction to bootstrap a feature tracker. We demonstrate successful application of our system to a variety of subjects with complexities ranging from simple oscillatory to articulated motion, under both static and moving camera conditions with occlusion present. We conclude with a critical appraisal of the system and discuss directions for future work.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

Keywords: Video processing, Motion cues, Non-photorealistic animation

1. Introduction

This paper presents a novel non-photorealistic rendering (NPR) system capable of rendering motion within a 2D image sequence in artistic styles. The user may stylise the rendering through a parameterised framework encompassing a diverse gamut of motion cues commonly used in animation; augmentation cues such as streak lines, ghosting and blurring are available, as are deformation cues such as the squash and stretch techniques used to emphasise motion in cartoons. The algorithm draws upon techniques from both the vision and the graphics communities to analyse and render motion. Aside from minimal interaction when bootstrapping a feature tracker, the system is entirely automatic.

Our work is motivated by a desire to render 2D image sequences in cartoon-like styles, a problem that decomposes into two separable sub-goals: 1) producing temporally coherent shading effects in the video; 2) emphasising motion in the image sequence. Whilst we do address the former issue, this paper is primarily concerned with the latter issue of visually depicting motion through artistic rendering. To the best of our knowledge we believe that artistic rendering of motion within a video sequence is a novel contribution to NPR, and one that implies interesting new application areas for Computer Vision.

The majority of research in non-photorealistic animation focuses upon the synthesis of 2D artistically rendered sequences from 3D geometries [1, 2]. Some progress has been made in processing 2D video sequences into non-photorealistic styles, for example the animated painterly effects proposed by Litwinowicz [3], and later Hertzmann [4]. Such algorithms focus primarily upon the task of extending static NPR techniques to moving video whilst maintaining temporal coherence (avoiding flickering) in the image sequence. Whilst such methods seek to mitigate against the effects of motion for the purposes of coherence, the literature is relatively sparse concerning the emphasising and rendering of motion within the image sequence itself.

In an early paper [5], Lasseter highlights many of the motion emphasis techniques commonly used by animators for the benefit of the computer graphics community, though presents no algorithmic solutions. Streak lines, anticipation and deformation for motion emphasis are discussed. Recent

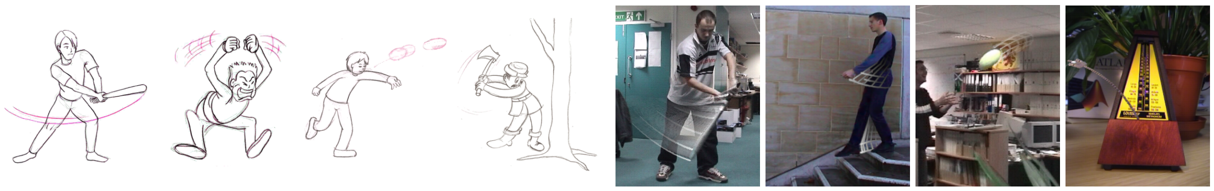

Figure 1: *Examples of motion cues used in traditional animation (left) and the corresponding cues inserted into a video sequence by our system (right). From left to right: two examples of streak line augmentation cues, the latter with ghosting lines. Two examples of deformation cues; squash and stretch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

work addresses one of these techniques by applying a squash and stretch effect to spheres and cylinders in object space prior to ray-tracing [6]. Strothotte *et al* [7], after Hsu *et al* [8], also identify depiction of motion as important, though the former are concerned primarily with the effect of motion cues on temporal perception. In both studies streak lines are generated via user-interactive processes. Earlier work by the authors renders motion within an image sequence by composing salient features to produce paintings reminiscent of Cubist art [9].

Animators have evolved various ways of emphasising characteristics of a moving object (Figure 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 which 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 [5]. 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 paper we process real video to introduce these motion cues; examples are given in Figure 1.

An attempt to automatically render motion cues presents the following challenges:

1. Motion cues tend to emphasise fast, large-scale feature motions, the filming of which often requires the camera to move. Features may also become occluded during motion. How shall we track features through a video sequence in these circumstances, and what constraints must we impose to make the tracking problem tractable?
2. How will motion cues be generated and attached to tracked features? How will such cues adapt to variations in velocity and acceleration? How will motion cues be embedded coherently in the existing image sequence?
3. How can we render the final scene to produce coherent artistic shading styles?

The first point falls within the bounds of Computer Vision, the latter two are primarily Computer Graphics issues.

The remainder of the paper is organised as follows. In Section 2 we give an overview of the system. In Section 3 we discuss the vision algorithms for camera motion correction and tracking. In Section 4 we describe the algorithms for generating motion cues and inserting them into video. We conclude in Section 5 with a discussion of future work.

2. Overview of the system

We now describe the major components of the system, leaving detailed explanation to subsequent sections of the paper. The system 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.

3. The Computer Vision component

The Computer Vision component is responsible for the tracking of features over the video sequence. A camera motion compensated version of the sequence is first generated, thereby ensuring that camera motion does not influence the observed trajectories. Features are then tracked over the sequence using standard techniques. By analysing occlusion during tracking we determine a relative depth ordering of features, later used in the rendering stage to insert motion cues at the correct scene depth.

We compensate for camera motion using a robust motion estimation technique initially proposed by Torr [10]. Harris interest points [11] are identified in adjacent video frames, and RANSAC [12] used to produce an initial estimate of the homography between frames. This estimate is then refined using a Levenburg-Marquadt iterative search [13]. Frames are

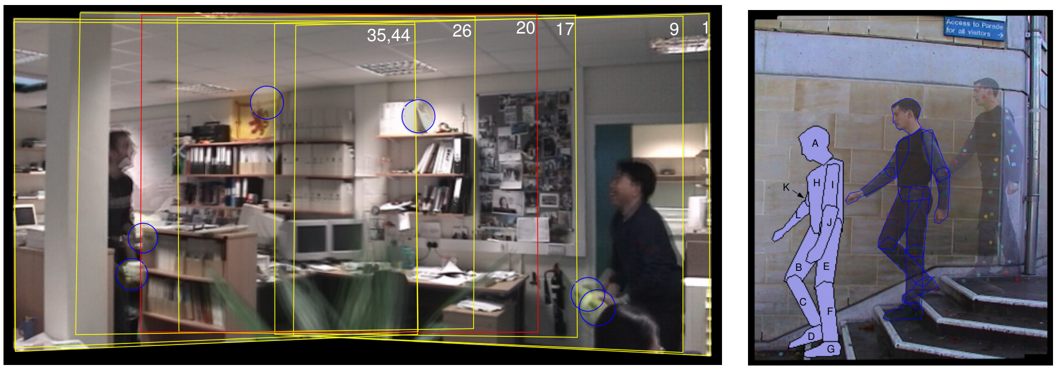

Figure 2: *Left: The camera compensated* VOLLEY *sequence sampled at regular time intervals. The camera viewport at each instant is outlined in yellow, the tracked feature in blue. Right:* STAIRS *sequence. (Top) markers are required to track this more complex subject but are later removed automatically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

projected via their homographies to produce a motion compensated sequence in which the tracking of features is subsequently performed.

3.1. Tracking features over 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 [14]. We assume contour motion may be modelled by a linear conformal affine transform (LCAT) in the image plane, which has 4 parameters (scale, orientation, and spatial position). Variation of these parameters is assumed to well approximate a second order motion equation over short time intervals.

The LCAT is a degenerate form of the affine transform consisting of a uniform scale s, orientation θ, and a translation $(u,v)^T$. In homogeneous coordinates we have a product of matrices:

$$\underline{\underline{M}}(s,\theta,u,v) = \underline{\underline{T}}(u,v)\underline{\underline{R}}(\theta)\underline{\underline{S}}(s) \qquad (1)$$

A feature, F, comprises a set of points whose position varies in time; we denote a point in the tth frame by the column vector $\underline{x}_t = (x,y)_t^T$. In general these points are not pixel locations, and 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 ignore variations in luminance to mitigate 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 $E[.]$:

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

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

By default, several well distributed interest points are identified automatically using the Harris [11] detector (see sequences *CRICKET*, *METRONOME*). In some cases a distinctively coloured feature itself may be used as a marker (see *VOLLEY*, Figure 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 2). In these cases the Harris interest points are substituted for points generated by analysis of the colour distribution in a frame.

So far we have overlooked noise and occlusion. The 'best' $\underline{\underline{M}}_{tt'}$ will be meaningless in the case of total occlusion, and exact in the case of no noise or occlusion. The likelihood L_t of the feature being occluded at any time t may be written as a function of detected pixel error:

$$L_t = exp(-\lambda E_{[\underline{\underline{M}}_{tt'}]}) \qquad (3)$$

where λ is the reciprocal of the average time an object is unoccluded. At any time t we know the estimated LCAT $\underline{\underline{M}}_{tt'}$, and the confidence in that estimate $C_t = 1 - L_t$ which we pass through a Kalman filter to obtain our best estimate value for the LCAT. The state transition matrix in the filter represents a second order motion equation in the 4D parameter space of the LCAT, trained over the immediate history of contour motion. In the case $C_t = 0$ results will be predicted entirely by this second order motion equation, in case $C_t > 0$ the measured LCAT holds greater influence over the predicted motion. We may threshold C_t at 0.5 to make a binary decision as to whether, at a given frame, a feature is occluded or not (see Section 3.2).

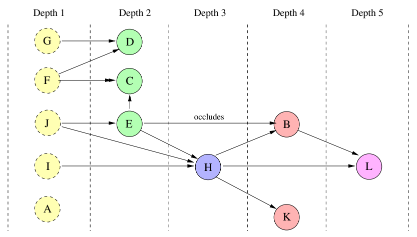

Figure 3: *An occlusion graph constructed over 150 frames of the* STAIRS *sequence, node letters correspond with Figure 2. Dotted lines indicate unoccluded nodes.*

The ability of the algorithm to re-establish tracking following occlusion has been found sufficiently robust for our needs, providing occluded motion approximately follows the second order model estimated by the Kalman filter. However 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 may not be visible, any attached motion cues may be. Occlusion is also used to determine relative feature depth.

3.2. Recovering relative depth ordering of features

We now determine a partial depth ordering for tracked features, based on their mutual occlusion 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 in Section 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 if 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 of 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 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 *should be* acyclic since cycles represent planar configurations which cannot occur unless our previous assumptions are violated or, rarely, noise has cause false occlusion. Second, groups of polygons may not interact via occlusion, thus the resulting graph may not be connected (a forest). Third, at least one 'unoccluded node' G_m must exist per connected graph such that $\forall G_{i=1..n} \neg \exists G_i \mapsto G_m$.

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 3). 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 superfluous to our needs.

4. The Computer Graphics component

The graphics component composits cels to create each frame of the animation. 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.

4.1. Motion cues by augmentation

Augmentation cues such as streak lines and ghosting, are common in traditional animation (Figure 1). Streak lines can be produced on a per frame basis by attaching lines to a feature's trailing edge, tangential to the direction of motion [8]. Such an approach is only suitable for visualising instantaneous motion, and produces only straight streak lines. In contrast animators tend to sketch elegant, long curved streaks which emphasise motion historically. For the same reasons, optical flow cannot be used to create streak lines.

Streak line placement is 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 sample the feature boundary at regular 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 point ordering can vary from

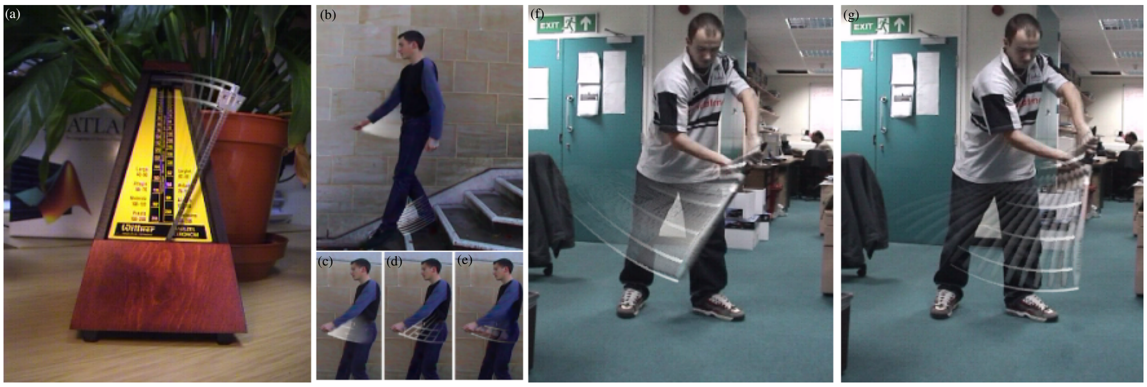

Figure 4: *A selection from the gamut of streak and ghosting line styles available: Ghosting lines may be densely sampled to emulate motion blur effects (b,c,f) or more sparsely for traditional ghosting (a,d). The feature itself may be ghosted, rather than the trailing edge, to produce Futurist-like 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).*

frame to frame, as well as shape (and even connectivity). 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 an link in a correspondence trail. This method gives independence on point ordering, and 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, delimited by sharp changes in trajectory. Such motion is usually caused by rapid translational, rotary or projectile motion in a scene, resulting in simple linear or curved trajectories. We have therefore chosen to model these trajectories using a subset of conics; elliptic curve fragments, parabolic curve fragments (which permit linear forms as a degeneracy).

We use a greedy algorithm to fit curves: (1) begin at the start of the correspondence trail; (2) iterate forward capturing points and fitting elliptical [15] and parabolic curves along the way using least-squares; (3) when no curve fits sufficiently well (below a threshold), or average velocity falls too low, the smooth section is terminated and the best fitting curve used as a model for that section.

We fit piecewise smooth models to every correspondence trail, as just described. This results in a set of smooth sections, each with a pair of attributes: 1) a function $\underline{G}(s)$ where s is an arclength parameterisation of the spatial trajectory of the fitted curve $s = [0,1]$; 2) a lookup table $g(.)$ mapping from an absolute time index t to the arclength parameter, i.e. $s = g(t)$, thus recording velocity along the spatial path $\underline{G}(s)$. The inverse lookup function $g'(.)$ is also available. Clearly the curve exists only for a specific time period $[g'(0), g'(1)]$; 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 managable size, which contains optimal paths along which streak lines will be drawn.

Our filtering selects curves based on heuristics derived from the practice of traditional animators 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, on each iteration i adding a new element to σ (initially empty) to maximise the recursive function $H(.)$

$$\begin{aligned} H(0) &= 0 \\ H(i+1) &= H(i) + (\alpha v(x) + \beta L(x) - \\ &\quad \gamma D(x) - \delta\omega(x,\sigma;w) + \zeta\rho(x)) \end{aligned} \tag{4}$$

where x is the set of 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 duration of x. $\rho(x)$ is the mean curvature of feature boundary at points in x. $D(x)$ is the mean shortest distance of points in x from the convex hull of the feature. $\omega(x,\sigma;w)$ measures the maximal spatio-temporal 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

w 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 4). 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 spatio-temporal 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 $\underline{G}(g(argmax(g'(0), t - \Delta)))$. 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. Secondary streak lines may be generated at small spatio-temporal offsets to produce sketchy or turbulent effects.

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. 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 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 appearence of the ghosting lines (Figure 4).

4.2. Motion cues by deformation

Our framework offers the facility to emulate effects such as 'squash and stretch', or to suggest inertia or drag through deformation. Features are cut from the current video frame, and motion dependent warping functions applied to render the deformation cue cel for each feature.

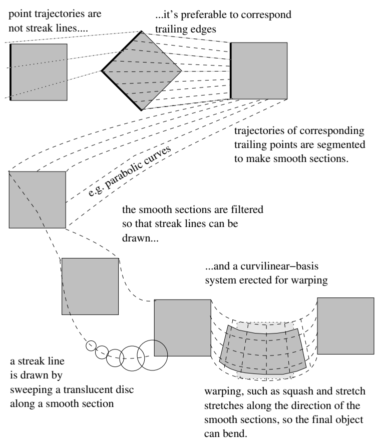
point trajectories are not streak lines....
...it's preferable to correspond trailing edges
trajectories of corresponding trailing points are segmented to make smooth sections.
e.g. parabolic curves
the smooth sections are filtered so that streak lines can be drawn...
...and a curvilinear–basis system erected for warping
a streak line is drawn by sweeping a translucent disc along a smooth section
warping, such as squash and stretch stretches along the direction of the smooth sections, so the final object can bend.

Figure 5: *Summarising the generation of motion cues.*

Deformations are performed by forming a curvilinear space, the basis of which is defined by the local trajectory of the feature centroid, and lines normal to this curve; this trajectory has exactly the same properties as any smooth section. Two parameters s and r are use to locate a spatial point, $\underline{x} = \underline{G}_c(s) + r\underline{n}(s)$, where $\underline{n}(s)$ is the unit normal to centroid trajectory $\underline{G}_c(.)$, at s. This curvilinear space is translated in so that its origin is coincident with the feature centroid. Whilst these bases may not exactly coincide with correspondance trails, they form a close approximation sufficient for our purposes.

For convenience we denote the transform from parameter space $\underline{r} = (s, r)^T$ to rectilinear space by $\underline{x} = U(\underline{r})$. The inverse transform $\underline{r} = U^{-1}(\underline{x})$ is maintained in a look-up table.

The squash and stretch effect conserves area whilst stretching the object in the direction tangential to motion. The complete transformation may be expressed as:

$$\underline{x} \leftarrow U(\begin{bmatrix} k & 0 \\ 0 & \frac{1}{k} \end{bmatrix} U^{-1}(\underline{x})) \tag{5}$$

where k varies eccentricity of the squash and stretch effect as a function of tangential speed $|\dot{\mu}|$ and user specified constants V_{min} and V_{max}:

$$\begin{aligned} k &= 1 + \frac{K}{2}(1 - cos(\pi\frac{v^2+1}{2})) \\ v &= \begin{cases} 0 \text{ if } |\underline{\dot{\mu}}| < V_{min} \\ 1 \text{ if } |\underline{\dot{\mu}}| >= V_{max} \\ (|\underline{\dot{\mu}}| - V_{min})/(V_{max} - V_{min}) \text{ otherwise} \end{cases} \end{aligned} \tag{6}$$

User defined constants V_{max}, V_{min} define a velocity window for the effect, whilst K limits the eccentricity of the squash and stretch. Squash and stretch uses speed of the centroid as the parameter to the warping function. More generally we can form warping functions which depend on a point's local velocity and acceleration as well as its position. We write $x' = U(T(U^{-1}(\underline{x}), \dot{\underline{x}}, \ddot{\underline{x}}))$, where T is a functional used, for example, to suggest increased drag or inertia. A typical functional operates on each component of $\underline{r} = (r_1, r_2)$ independently, to create effects suggesting drag we use:

$$r_1 = r_1 - F(\frac{2}{\pi}atan(|\dot{\underline{x}}_i|))^P sign(\dot{\underline{x}}_i) \tag{7}$$

where F is a function of suggested mass, and P influences the apparent rigidity of the feature. By substituting acceleration for velocity, and adding rather than subtracting from r_1 we can emphasise inertia of the feature.

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

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

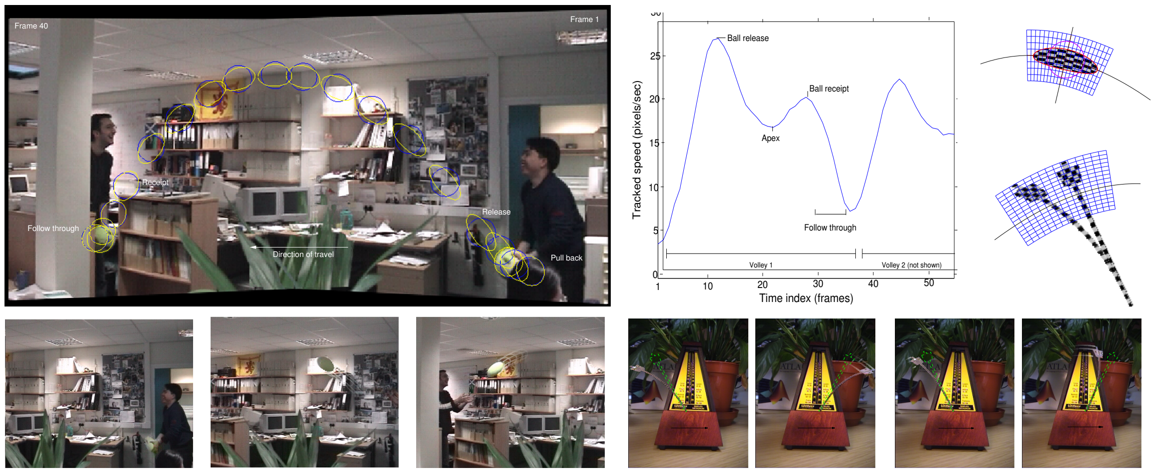

Figure 6: *Top left: Illustrating the squash and stretch effect; eccentricity varies as a function of tangential speed. Bottom: Frames from the* VOLLEY *sequence exhibiting squash and stretch. Observe the system handles both large scale camera motion, and lighting variation local to the ball. Frames from* METRONOME *suggesting drag and inertia effects through deformation, original feature outline in green. Top right: Curvilinear space used to deform features in* VOLLEY *and* METRONOME.

bounding contour of a feature will be included in that feature and so are subject to deformation. Consequently, it is easy to warp parts of occluding objects; we now explain how to avoid such unwelcome artefacts.

We identify pixels as belonging to an occluding object by forming a difference image (using the hue and saturation components in HSV space) between the feature region in the current frame and the feature itself. This yields a template of occluding pixels which can be retextured by sampling from feature regions in neighbouring unoccluded frames. 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 7).

Similarly, augmentation cues should pass behind occluding objects. Fortunately these cues traverse an identical path to that of the feature in prior frames. We construct an *occlusion buffer* over time, summing the difference images generated whilst handling occlusion for deformation cues (Figure 7d). Using this buffer we may determine which pixels will occlude the streak lines, so long as those pixels do not change in the time interval between the 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 (Figure 7e).

4.4. Compositing and Rendering

We generate a background for each video frame 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 viewpoint. Pixels contributing to feature regions are deleted and absent background texture is reprojected from locally neighbouring frames in alternation until holes are filled with non-feature texture. This sampling strategy mitigates against artefacts 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. Thus motion cues appear to be inserted into video at the correct scene depth (Figure 7g). Additionally, temporally coherent NPR shading effects may be generated by rigidly attaching strokes or texture to each feature; strokes are subjected to the identical LCAT motion and deformations as their respective feature (Figure 7f).

5. Conclusion

We have described and demonstrated a system for the artistic rendering of motion within video sequences. The system can cope with a moving camera, lighting changes, and presence of occlusion. Users may stylise both the placement and appearance of motion cues using the parameterised framework described in Section 4. Novel effects may be produced by interpolating between points in this parameter space, gradually changing the appearance of motion cues over time.

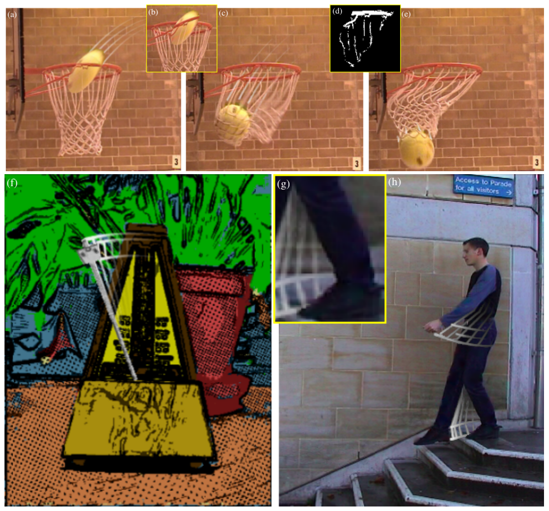

Figure 7: *Above: three frames from* BASKETBALL*, demonstrating occlusion handling. The system fails when the netting moves erratically after impact, causing the buffer to empty and streak lines to be drawn in front of the netting (7e). Below left: A coherent cartoon effect created by applying Q-maps to cels* [16]*; Below right: Streak lines are inserted at the correct scene depth; cues attached to the far leg pass behind the near leg.*

Further developments could address the relaxation of some of the assumptions made in the design of the system. For example, violations of depth assumptions 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 non-cyclic depth graphs, and in doing so recover relative feature depth under more general motion. As regards rendering, we might extend the possible gamut of motion cues; investigating whether additional techniques in Lasseter's paper [5] such as anticipation, can be rendered automatically.

We have shown that through high-level analysis of features (examining motion over the entire video, rather than on a per frame basis) we may produce motion cues closely approximating those used in traditional animation (Figure 1). We believe the most productive avenues for future research will not be in incremental refinements to the current system, but rather will examine alternative uses for higher-level spatio-temporal analysis of video with applications to NPR.

A selection of rendered video sequences are available online at **http://www.cs.bath.ac.uk/∼vision/cartoon**.

References

1. B. Meier, "Painterly rendering for animation", in *Proceedings Computer Graphics (ACM SIGGRAPH)*, pp. 447–484, (1996).
2. E. Daniels, "Deep canvas in disney's tarzan", in *Proceedings Computer Graphics (ACM SIGGRAPH, Abstracts and Applications)*, p. 200, (1999).
3. P. Litwinowicz, "Processing images and video for an impressionist effect", in *Proceedings Computer Graphics (ACM SIGGRAPH)*, pp. 407–414, (1997).
4. A. Hertzmann and K. Perlin, "Painterly rendering for video and interaction", in *Proceedings NPAR Symposium*, pp. 7–12, (2000).
5. J. Lasseter, "Principles of traditional animation applied to 3D computer animation", in *Proceedings Computer Graphics (ACM SIGGRAPH)*, vol. 21, pp. 35–44, (July 1987).
6. S. Chenney, M. Pingel, R. Iverson, and M. Szymanski, "Simulating cartoon style animation", in *Proceedings NPAR Symposium*, (2002).
7. T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey, "How to render frames and influence people", in *Proceedings Computer Graphics Forum (Eurographics)*, vol. 13, pp. C455–C466, (1994).
8. S. C. Hsu and I. H. H. Lee, "Drawing and animation using skeletal strokes", in *Proceedings Computer Graphics (ACM SIGGRAPH)*, pp. 109–118, (1994).
9. J. P. Collomosse and P. M. Hall., "Cubist style rendering from photographs." IEEE TVCG, in press.
10. P. H. S. Torr, *Motion segmentation and outlier detection*. PhD thesis, University of Oxford, (1995).
11. C. J. Harris and M. Stephens, "A combined corner and edge detector", in *Proceedings 4th Alvey Vision Conference*, (Manchester), pp. 147–151, (1988).
12. M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography", *Communications of the ACM*, **24**(6), pp. 381–395 (1981).
13. R. Szeliski, "Image mosaicing for tele-reality applications", Tech. rep., Digital Equipment Corp., (1994).
14. D. Williams and M. Shah, "A fast algorithm for active contours and curvature estimation", *CVGIP: Image Understanding*, **55**(1), pp. 14–26 (1992).
15. A. W. Fitzgibbon and R. B. Fisher, "A buyer's guide to conic fitting", in *Proceedings BMVC*, (Birmingham), (1995).
16. P. Hall, "Non-photorealistic rendering by Q–mapping", *Computer Graphics Forum*, **1**(18), pp. 27–39 (1999).