Chapter 8

Stroke Surfaces: Temporally Coherent Artistic Animations from Video

In this chapter we describe the third and final subsystem comprising the Video Paintbox: a framework capable of creating artistically shaded animations directly from video. We demonstrate that through automated analysis of the video sequence at a higher spatiotemporal level — as a block of frames rather than on a per frame, per pixel basis as with current methods — we are able to generate animations in a wide variety of artistic styles, exhibiting a uniquely high degree of temporal coherence. In addition to rotoscoping, matting and novel temporal effects unique to our method, we demonstrate the extension of "traditional" static AR styles to video including painterly, sketchy and cartoon shading effects. We demonstrate how our coherent shading subsystem may be combined with the earlier motion emphasis subsystems (Chapters 6 and 7) to produce complete cartoon-styled animations from video clips using our Video Paintbox.

8.1 Introduction

In this chapter we propose a solution to the long-standing problem of automatically creating temporally coherent artistically shaded animations from video[1]. As observed in Chapter 5, this problem has been sparsely researched. Existing video driven AR methods [75, 96, 103] address only the problem of producing painterly effects in video, and typically produce animations exhibiting poor levels of temporal coherence. AR techniques are predominantly stroke based, and temporal incoherence occurs principally when either:

[1]This work appeared in [26] and an overview presented at the BMVA Symposium on Spatiotemporal Processing (March 2004). This work has also been submitted to BMVC 2004.

1. the motion of strokes, and so motion within the resulting animation, does not agree with the motion of content within the source video sequence.

2. the visual attributes of strokes fluctuate rapidly, creating flicker in the animation.

The manifestation of temporal incoherence is as an uncontrolled motion and rapid flickering in the animation, termed *"swimming"*. Swimming severely damages the aesthetics of an animation, and tends to produce perceptual cues which distract from the content of the image sequence itself. This observation is supported by psychophysical research. For example the Gestalt "common fate" cue [95], describes how objects moving in a similar manner become grouped. Conflicts between the motion of phantom objects perceived due to grouping, and physical objects, contribute to the distracting nature of swimming. Wong *et al* [172] observe that rapidly flickering dots are perceived to "pop-out" from their neighbours; explaining the confused sense of depth apparent in an animation with poor stroke coherence. Unfortunately, we observe that this pop-out effect manifests most strongly at around 6Hz, close to the aesthetically optimal frame rate for coherent painterly animations determined by Hertzmann and Perlin [75].

Swimming in AR animations is therefore a significant practical problem, and one that can be solved only by smoothly moving strokes in a manner consistent with motion in the scene. Numerous object-space AR techniques based upon this principal have been published in recent years, and are capable of creating coherent AR animations from 3D models [32, 65, 108, 111]. Broadly speaking, object-space methods operate by fixing strokes to surfaces in the 3D geometry which move coherently when the object moves relative to the camera (see Chapter 2 for details of specific approaches). As we observed in Chapter 1, the problem statements of object-space and video driven AR are thus somewhat different. With the former there is no requirement to recover structure and motion prior to stroke placement, since scene geometry is supplied. With the latter, we must analyse pixel data to recover missing structure and motion prior to rendering.

All existing automatic 2D video AR algorithms [75, 96, 103] largely disregard spatial structure by moving brush strokes independently, and attempt motion recovery using inter-frame comparisons; motion is estimated from one frame to the next, and brush strokes translated accordingly. Both per frame optical flow [96, 103] and frame differencing [75] approaches to motion estimation have been applied to AR (Section 2.5.2 contains details), however both approaches fall far short of producing temporally coherent animations. We have argued (Chapter 5) that there are in-principal difficulties with analysing video on a temporally local, per frame progressive basis, when attempting to produce a globally coherent AR animation. These difficulties include the rapid accumulation and propagation of error over subsequent frames due to poor motion estimates, and the limitations of the motion estimate techniques employed (especially

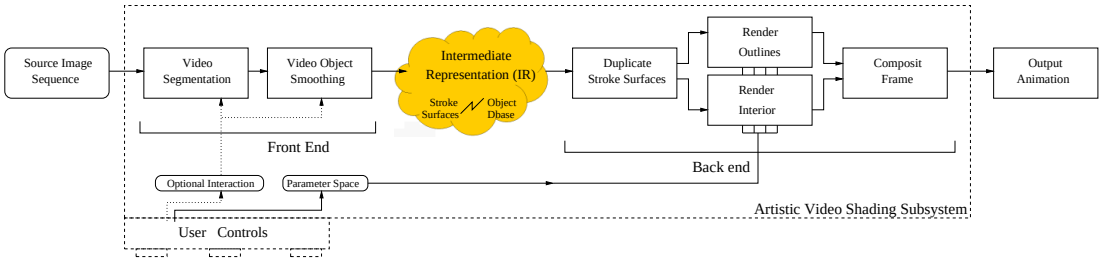

Figure 8-1 Illustrating the rendering pipeline of the artistic rendering subsystem. Video is parsed into an intermediate representation (IR) by the front end, using Computer Vision techniques. This representation is then rendered by the back end, under the influence of user parameters which may be varied to stylise the output according to the animator's wishes.

when operating upon flat textured objects, or on objects undergoing occlusion). If one were processing video for interaction or real-time animation then a frame by frame approach would be justified (an example is Hertzmann's "Living Painting" [75]). However the motivation of the Video Paintbox is primarily to create a tool for animators, with which they are able to process video for post-production effects. A global analysis over all frames available during offline rendering seems a more promising avenue for producing temporally coherent animations.

We therefore argue for a higher level of spatiotemporal analysis than that employed by existing automatic techniques. Spatially, we operate at a higher level by segmenting images into homogeneous regions, which correspond well with individual objects in the scene. Using the novel approach we describe in this Chapter brush stroke motion is guaranteed to be consistent over entire regions — contradictory visual cues do not arise, for example where stroke motion differs within a given object. Temporally we work at a higher level, automatically corresponding regions over time to carve smooth trajectories through the video, and smoothing region attributes, such as colour, over blocks of adjacent frames to mitigate swimming; this is in contrast to all existing AR video methods. We believe the paradigm of automatically processing video at this higher spatiotemporal level to be a novel and valuable approach to the problem of synthesising AR animations from video.

The remainder of this chapter describes our novel framework for the production of temporally coherent AR animations from video, which comprises the third and final subsystem with the Video Paintbox. Our approach is unique (among automated AR video methods) in that we treat the image sequence as a spatiotemporal voxel volume; a stack of sequential frames in which time forms the third dimension. The interpretation of video as a volume, rather than as a disjoint set of frames, is a powerful abstractive technique proposed as far back as the early eighties [86] simplifying analysis and

exploitation of frequency patterns in both the spatial and temporal dimensions. Applications of spatiotemporal processing have been identified both within the field of Computer Vision, for example in motion estimation [139] and content based image retrieval [118], and in Computer Graphics for interactive video editing [6, 93] and more commonly for visualisation [34, 128, 173]. We demonstrate that by manipulating video in this representation we are able to synthesise a wide gamut of artistic effects, which we allow the user to stylise and influence through a parameterised framework. The diversity of artistic style, and level of temporal coherence, exhibited by our animations further evidence our central argument for a higher level of spatiotemporal analysis in image-space AR.

8.1.1 Overview and Capabilities of the Subsystem

In a similar manner to the motion emphasis subsystems (Chapters 6 and 7), the artistic shading subsystem consists of a single rendering framework which may be broken into a front and back end. The front end (Section 8.2) is responsible for parsing the source video to create an "intermediate representation" (or "IR", Section 8.3), and is automated through application of Computer Vision techniques. This abstracted video representation is then passed to the back end (Section 8.4), where it is rendered in one of a range of artistic styles. The user is given control over the back end of the system via a set of high level parameters which influence the style of the resulting animation (Figure 8-1).

The artistic shading subsystem operates in the following manner. We begin by segmenting video frames into homogeneous regions, and use heuristics to create semantic associations between regions in adjacent frames. Regions are thus connected over time to produce a collection of conceptually high level spatiotemporal "video objects". These objects carve sub-volumes through the video volume delimited by continuous isosurface patches, which we term "Stroke Surfaces". The video is encoded by a set of such boundaries and a counter-part database containing various properties of the enclosed video objects. The surfaces and database respectively form the two halves of the IR, which is passed to the back end for rendering. To render a frame at time t the back end intersects the Stroke Surfaces with the plane $z = t$, to generate a series of splines corresponding to region boundaries in that frame. By manipulating the IR (for example, temporally smoothing the Stroke Surfaces), the back end is able to create temporally coherent animations in a range of artistic styles, under the high level direction of the animator.

Although our spatiotemporal framework was originally motivated by our goal of creating coherent, flat-shaded cartoons from video, it now encompasses many artistic styles.

Figure 8-2 Top: Stills from the hand-illustrated music video to A-Ha's "Take On Me" [Barron, 1985]. A woman enters a world inside a comic strip and appears non-photorealistic (sketchy), whilst interacting with a number of photorealistic and non-photorealistic beings and objects. Bottom: Our automatic video AR framework is capable of similar "mixed media" effects (left, we show photorealistic people against AR background; middle, vice versa), as well as many other artistic styles such as oil paint, watercolour, flat shaded cartoon (right), and can create a range of novel temporal effects too. Motion emphasis cues from the previous two chapters may be readily combined with this subsystem to create complete cartoon animations from video (right).

In addition to novel temporal effects unique to our framework, we demonstrate the extension of numerous static AR styles to video including oil and watercolour paint, sketchy, cartoon shading effects, as well the ability to create "mixed media" effects (Figure 8-2). An application to rotoscoping and video matting is also identified, in fact rotoscoping and stroke based AR techniques (such as painterly rendering) are unified under our framework. A potential application to abstracted, low bandwidth transmission of video content is also identified, resulting from the compact, continuous vector representation of the IR. Furthermore, we are able to combine our coherent shading framework with our earlier motion cue work (Chapters 6 and 7) to produce polished cartoon-styled animations from video clips using the completed Video Paintbox.

8.2 Front end: Segmenting the Video Volume

In this section we describe the video segmentation process of the front end. The front end is responsible for the smooth segmentation of the video volume into sub-volumes in a voxel representation, which describe the trajectories of features. These volumes are

then encoded in our "IR", which is passed to the back end for rendering. We describe the nature of this IR, and the encoding process, in Section 8.3.

There are three stages to the video segmentation algorithm, each of which we overview briefly here, and describe in detail in subsections 8.2.1, 8.2.2, and 8.2.3 respectively. We begin by independently segmenting video frames into connected homogeneous regions using standard 2D Computer Vision techniques (we describe these momentarily). The second stage of processing creates associations between segmented regions in each frame, to regions in adjacent frames. A filtering process removes spurious associations. The result of this second step is a set of temporally convex sub-volumes carved from the spatiotemporal video volume; we introduce the term *"video objects"* to describe these sub-volumes. These video objects are associated over time in a graph structure, which we refer to as the *"object association graph"*. The third and final stage performs a coarse temporal smoothing of the boundaries of video objects. This smoothing, combined with the filtering process the second stage, mitigate temporal incoherence in the video segmentation. The trajectory of a single feature through the video volume is represented by a collection of one or more associated video objects; we describe the union of video objects, which comprise such a trajectory, as a *"feature sub-volume"*.

8.2.1 Frame Segmentation

We now explain the first step of our process, which segments each video frame into homogeneous regions. The criterion for homogeneity we have chosen for our system is colour (after [38]). Many segmentation techniques also subscribe to this approach [5, 39, 42, 164], under the assumption that neighbouring regions are of differing colour. Each frame is independently segmented to create a class map of distinct regions. Associations between regions in adjacent frames are later created. Choice of segmentation algorithm influences the success of this association step, as segmentations of adjacent frames must yield similar class maps to facilitate association. Robustness is therefore an important property of the segmentation algorithm chosen to drive our subsystem: given an image I, a robust segmentation process $S(.)$, and a resulting class map of regions $S(I)$, small changes in I should produce very similar class maps $S(I)$. Although most published 2D segmentation algorithms are accompanied by an evaluation of their performance versus a ground truth segmentation, to the best of our knowledge a comparative study of algorithm robustness, as we have defined it, is not present in the literature. Consequently, we investigated the robustness of several contemporary 2D segmentation algorithms, with an aim of selecting the most robust algorithm to drive our video segmentation process.

We evaluated the robustness of five contemporary algorithms on twenty short clips

(around 100 frames each) of artificial and natural scenes. For our purposes, a natural scene is one of complex, cluttered content (such as Figure 8-5, *SOFA*), whilst an artificial scene is typically an uncluttered, low complexity scene with few homogeneous colour regions (such as Figure 8-5, *POOHBEAR*). In all clips the footage was of a static scene, with a gradual change of camera viewpoint over time. We tested the following algorithms in both RGB and HSV spaces:

1. Recursive histogram split (RHS) [145] A colour histogram of the image is built, and the colour which corresponds to the largest peak in the histogram is identified. The largest connected region of that colour is isolated, and "cut" from the image to form one segmented region. This process is iterative, and "cut" regions do not contribute to the histogram on subsequent iterations. The process terminates when the height of the histogram peak falls below a threshold.

2. Split and Merge (SM) [78] A two stage algorithm, comprising a "split" and a "merge" step. A tree T is constructed, initially with the whole image as the only node (root). We iterate through each node, splitting the node's region into quarters (creating four children) if that region is not "sufficiently homogeneous". The "split" step terminates when all nodes have been tested, and will split no further. Each leaf in T is a homogeneous region, though the image may be over-segmented. The "merge" step mitigates the over-segmentation by examining each leaf node in T, combining regions which are both homogeneous and spatially connected.

3. Colour Structure Code (CSC) [127] A form of split and merge technique which operates upon the image using small neighbourhoods with a hexagonal topology. Hierarchies of hexagonal "islands" are grown and merged iteratively to form homogeneous regions.

4. EDISON [19] A synergistic approach to segmentation which fuses boundary information from: 1) homogeneous regions produced by a colour based mean shift segmentation [29]; 2) edges detected in the luminance channel of the image.

5. JSEG [42] A technique which first performs colour quantisation, and then proceeds to grow pixel clusters to form regions of homogeneous colour texture. These clusters must exhibit low "J values"; these values are a novel texture variance descriptor introduced in the paper.

Algorithms (1) and (2) are popular, classical segmentation techniques. The remainder are comparatively recent segmentation techniques, which are reported by their authors [19, 42, 127] to perform well upon general images without the imposition of additional models or constraints.

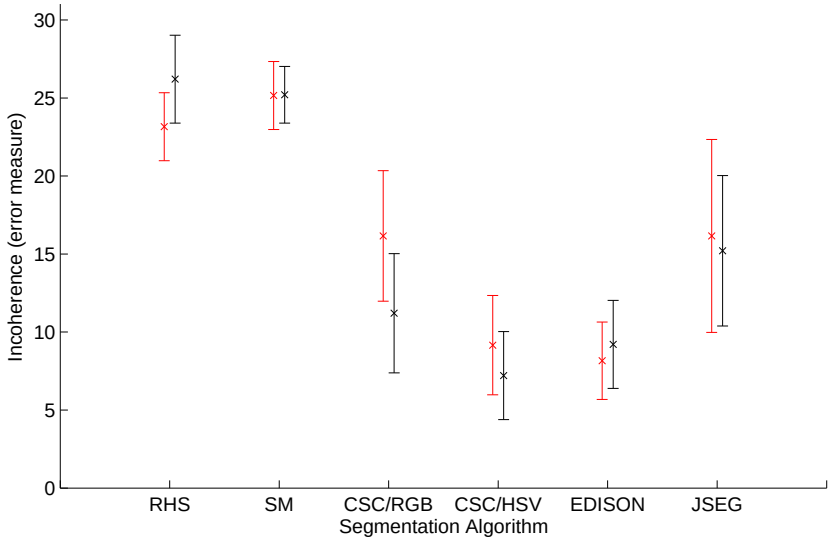

Figure 8-3 Results of comparing the robustness of segmentation algorithms. The lower the error measure on the ordinate, the more robust the algorithm. All algorithms operated in the RGB colour space unless otherwise stated. This graph shows the six most robust scenarios. Red and black plots indicate the results of processing ten "natural" and "artificial" scenes respectively. The crosses indicate the mean error measure for each scenario over all ten clips, the error bars indicate one standard deviation.

We measured robustness between adjacent frames in the following manner. Distance transforms [145] were computed for both frames' class maps. This resulted in two rasters whose individual pixel values correspond to the shortest distance from that pixel to a region boundary in the respective class map. We then computed the absolute difference between these distance transforms. The mean pixel value in this difference map was used the metric for robustness of segmentation between the two frames. The mean value of this inter-frame metric, computed over all consecutive frame pairs, was used as the metric for measured robustness of segmentation over the entire sequences. In practice the pixel values of the distance transform were thresholded at 30 pixels distance to prevent large errors from skewing the mean value (such errors emerged in cases of gross disparity between temporally adjacent class maps).

All segmentation algorithms operated in RGB colour space, but we also experimented with processing in HSV colour space — although this produced worse results for the most-part, results improved in the case of the CSC algorithm. Figure 8-3 summarises the means and variances of the robustness error measure for the six most robust scenarios, computed over ten "natural" and ten "artificial" source clips. Results indicate EDISON to be preferable in the case of natural scenes, and that artificial scenes were best segmented using the CSC algorithm operating in HSV space. However the associated error bars imply that there is little significant difference in one algorithm's performance over the other.

In our Video Paintbox we have chosen to use EDISON by default, but enable the user to switch to CSC/HSV if results of the segmentation are poor. The chosen algorithm is then independently applied to each frame in the source video clip to generate a set of segmented regions in each frame. Associations are then created between regions in adjacent frames to produce spatiotemporal video objects. We explain this process in the subsection 8.2.2, but first briefly justify our decision to opt for this associative approach to volume segmentation.

Choice of Video Segmentation Strategy: 2D plus time or 3D?

We have opted for a 2D segmentation followed by a temporal association step; this approach is commonly referred to as a "2D plus time" ($2D+t$) technique in Computer Vision literature. However, our initial development work centred upon an alternative methodology — performing a colour segmentation via a three dimensional flood-fill approach, and so processing the video as a single $3D$ volume (similar to [40]). Although a volumetric methodology is, perhaps, more in keeping with our spatiotemporal approach, there are a number of reasons that we opted for a $2D + t$ heuristic driven, associative approach over a $3D$ segmentation:

- Attributes such as the shape, colour or shading of a region are permitted to evolve gradually over time by the heuristics of our $2D + t$ approach. Such variation is difficult to accommodate within the framework of a single 3D segmentation (without introducing complicated 3D models and constraints, encapsulating the expected evolution of these attributes over time).

- Small, fast moving objects may form disconnected volumes in $3D$, resulting in temporal over-segmentation. However these discontinuities do not arise with our proposed $2D+t$ association scheme between frames, providing an upper threshold on search distance (parameter Δ, described later in Section 8.2.2) is set correctly.

- For pragmatic reasons. The problem of 2D image segmentation has received extensive study from the Computer Vision community, in contrast to 3D segmentation (exceptions lie within medical imaging, but do not deal with the problem domain of video imagery). However the modular nature of our framework (Figure 8-1) is such that the rendering process is loosely coupled with the segmentation technology used; thus we allow for substitution of segmentation algorithms as novel, improved technologies become available in the literature.

Alternative $2D+t$ approaches which associate contours over time have been described in the literature [53, 55, 92]; however we differ in that we create temporal association using region based properties rather than edges alone (the unconstrained problem of associating edge contours over time produces poor tracking solutions, and often requires

restrictive motion models to be imposed [84, 85]). We now explain the temporal region association process in detail.

8.2.2 Region association algorithm

The problem of associating sets of regions, each within multiple frames, is combinatorial in nature and an optimal solution can not be found through exhaustive search for any practical video. We propose a two step heuristic solution to the association problem, which we have found to perform well (that is, results in a locally optimal solution where associated objects exhibit an acceptable level of temporal coherence) and has quadratic complexity in the number of regions per frame. First, for each frame we generate associations between regions in that frame and those in frames adjacent to it. These associations are made according to heuristics based on mutual colour, area, spatial overlap, and shape. Second, the resulting chains of associated regions are filtered using a graph based search which removes sporadic associations. Association is complicated by the fact that objects may merge or divide in the scene. For example, a ball passing behind a post might appear to split into two regions, and then recombine. In our system it is satisfactory to represent a single occluded object as multiple imaged regions since, as we will describe, these regions become linked in a graph structure as a product of the region association process.

We observe that in a robust video segmentation: 1) properties of regions such as shape, colour, and area are typically subject only to minor change over short periods of time. The exceptions are the instants at which regions merge or divide; 2) although regions may appear or disappear, merge or divide over time, such events should be for the relative long-term (given a video frame rate of 25 frames per second) and not be subsequently reversed in the short-term. The first observations influences the choice of heuristics for the first stage of processing (region association), whilst the second observation governs the operation of the second stage (filtering).

Step 1: Iterative Association Algorithm

Consider a single region $r \in R_t$, where R_t denotes the set of segmented regions in frame t. We wish to find the set of regions in adjacent frames with which r is associated. We compute this by searching sets R_{t-1} and R_{t+1} independently — examining potential mappings from r to R_{t-1}, and then from r to R_{t+1}. Thus r could potentially become associated with zero or more regions in adjacent frames (Figure 8-4a). The suitability for two regions in adjacent frames $r \in R_t$ and $\rho \in R_{t\pm1}$, to be associated may be evaluated using an objective function $E(r, \rho)$. We describe this function momentarily, but first complete our description of the association algorithm.

For the purposes of illustration, let us consider the creation of associations between r and regions in the set R_{t+1}. The area (in pixels) of r is first computed, to give an "area-count". A potential set of associating regions in R_{t+1} is identified, whose centroids fall within a distance Δ of the centroid of r. These regions are sorted into a list in descending order of their score $E(.)$. Next a cumulative sum of their area counts is computed, working from the start of the list and storing the cumulative sum with each region. The area-count of r is subtracted from each cumulative area term in the list. The resulting set associated regions extends down this list until either the score $E(.)$ falls below a lower bound, or the area measure becomes less than or equal to zero. It is therefore possible for no associations to be created to past or future regions; in such circumstances a feature appears or disappears in the video, perhaps due to occlusion either by other objects or by the edge of the frame. The process is repeated for each frame t independently. The final set of associations for the sequence is taken to be the union of associations created for all frames.

Associated regions are thus semantically linked over time to create connected feature sub-volumes such as that in Figure 8-5c,d. These sub-volumes are broken into, possibly many, temporally convex video objects. Note we consider only the exterior boundary of these objects, disregarding "holes" in a volume produced by other nested objects; these are represented by their own external boundaries. A property of the temporally convex representation is that two separate objects will merge to produce one novel object, and an object division will produce multiple novel objects (see Figure 8-4b, Figure 8-5, bottom left). This representation simplifies later processing. The associations between video objects are also maintained; this mesh graph structure is also useful in later processing stages, as we refer to it hereafter as the "object association graph".

Heuristics for Association

We made use of a heuristic score $E(r, \rho)$ in our association algorithm, which determines the suitability of two regions in adjacent frames $[r \in R_t, \rho \in R_{t\pm1}]$ to be associated. This score may be written as a weighted sum of terms:

$$E(r,\rho) \quad = \quad \begin{cases} 0 \text{ if } \delta(r,\rho;\Delta) > 1 \\ w_1\sigma(r,\rho) + w_2\alpha(r,\rho) - ... \\ w_3\delta(r,\rho;\Delta) - w_4\gamma(r,\rho) \text{ otherwise} \end{cases} \tag{8.1}$$

The function $\delta(.)$ is the spatial distance between the region centroids as a fraction of some threshold distance Δ. The purpose of this threshold is to prevent regions that are far apart from being considered as potentially matching; $E(.)$ is not computed unless the regions are sufficiently close. We have found $\Delta = 30$ pixels to be a useful threshold. Constants $w_{1..4}$ are user defined weights which tune the influence of each

of four bounded ($[0,1]$) heuristic functions. It is through variation of these constants that the user is able to "tune" the front end to create an optimal segmentation of the video sequence; in practice less than an order of magnitude of variation is necessary. We have found $w_1 = 0.8, w_2 = 0.6, w_3 = 0.6, w_4 = 0.4$ to be typical values for the videos we present in this thesis. $\gamma(.)$ is the the Euclidean distance between the mean colours of the two regions in *CIELAB* space (normalised by division by $\sqrt{3}$). $\alpha(.)$ is a ratio of the two regions' areas in pixels. $\sigma(.)$ is a linear conformal affine invariant shape similarity measure, computed between the two regions. We now describe this final, shape similarity measure in greater detail.

We wish to compare the shape of two regions A and B. This comparison should be invariant to rotation, translation and uniform scale, since such transformations are common in imaged regions caused by an object undergoing motion relative to the camera. Regions are first normalised to be of equal area (affecting uniform scale invariance). We compute $\sigma(A, B)$ by analysis of the external regions' boundaries in the following manner.

We begin by computing the Fourier descriptors [31] of the "angular description function" of each boundary. The angular description function discretises a region's boundary into n vectors of equal arc-length $\underline{v_1}, \underline{v_2}, ..., \underline{v_n}$, and encodes that boundary as a series of variations in the angles between adjacent vectors. We write the angular descriptions for each region, A and B, as the scalar functions $\Theta_A(s)$ and $\Theta_B(s)$, where $sin[0, n]$ is a continuous dummy parameter which iterates around the boundary. Observe that $\Theta(.)$ is periodic, and invariant to translation of the region. We compute the Fourier transform of each $\Theta(.)$, to obtain spectral representations $F[\Theta_A(.)]$ and $F[\Theta)B(.)]$. Shape similarity is inversely to proportional to Euclidean distance between the magnitude vectors of these Fourier descriptors (disregarding phase for rotational invariance):

$$\sigma = 1 - \frac{1}{N} \sum_{\nu=1}^{N} \left| \frac{|F[\Theta_A(\nu)]| - |F[\Theta_B(\nu)]|}{(\sum_{s=1}^{n} (\Theta_A(s) - \Theta_B(s))^2)^{\frac{1}{2}}} \right| \tag{8.2}$$

where the summation is over the N lowest frequency components of the signal $|F[\Theta(.)|$. We have found that only the first eight components are desirable for inclusion in the summation ($N = 8$); the remaining high frequencies principally describe sampling errors due to the discrete, raster origin of the boundary descriptions.

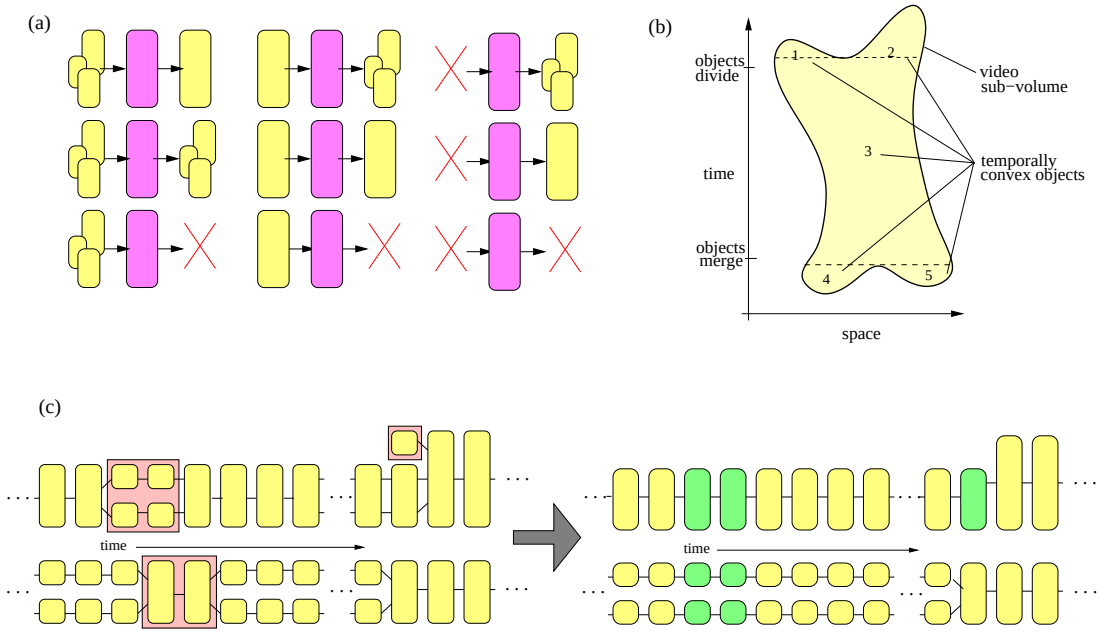

Figure 8-4 Illustrating the region association and filtering process (a) Nine cases of region association. Associations are created between a region in the current frame and potentially many regions in adjacent frames, through iterative evaluation of equation 8.1. (b) Example of a single video-sub-volume split into five temporally convex objects (c.f. Figure 8-5d). (c) An object association graph before and after graph filtering. Sporadic associations (red) are removed, and object boundaries interpolated (green) from neighbours.

Step 2: Filtering Sporadic Associations

Sporadic associations are sometimes incorrectly created between regions due to noise. We have observed that associations maintained over short time intervals may be categorised as noise, and filter out these artifacts by examining the object association graph's structure.

Since new objects are created for every merge or divide encountered in a feature sub-volume, one can identify sporadic merges or divisions by searching the graph for short-lived objects. We specify a short-lived object as an object which exists for less than a quarter of a second ($\leq$ 6 frames). This constant (as well as the search parameter Δ in equation 8.1) may be adjusted according to the assumed maximum speed of objects in the video. Short-lived objects deemed to correspond to false associations are removed by "cutting" that object from the graph and filling the gap by extrapolating an intermediate object from either neighbour (by duplicating the nearest neighbour, see Figure 8-4c). A serendipitous effect of this process is that poorly segmented areas of the video exhibiting high incoherence, tend to merge to form one large coherent object. This is subsequently rendered as a single region, abstracting away detail that would otherwise scintillate in the final animation.

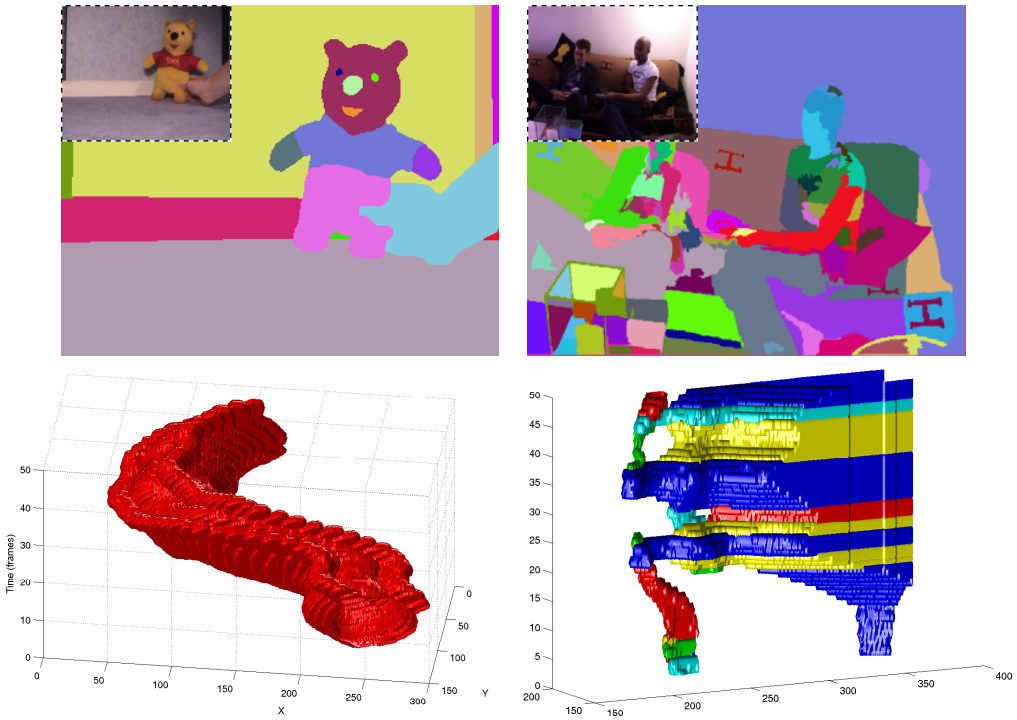

Figure 8-5 Above: Sample segmented frames from the *POOHBEAR* and *SOFA* sequences with original footage inset. Below: Two visualisations from the *POOHBEAR* video volume, corresponding to the head and the right hand section of the skirting board. Observe that whilst the head produces a single video object, the skirting board is repeatedly occluded by the bear's hand during movement causing division of regions. This resulting volume consists of a connected structure of several temporally convex video objects (coloured individually for clarity of illustration).

8.2.3 Coarse Temporal Smoothing

The segmentation and association processes result in an object association graph, and a number of video objects in a voxel representation. It is desirable that the boundaries of these video objects flow smoothly though the video volume, thereby creating temporally coherent segmentations of each video frame. Therefore, as a final step in our video segmentation algorithm we perform a temporal smoothing of video objects.

We fit constrained, smooth surfaces around each video object, and then re-compute the sets of voxels bounded by these surfaces to generate a new set of smoothed video objects. We describe this process in the remainder of this subsection. Note that this surface fitting process is a means to smooth the voxel video objects, and quite distinct from the surface fitting operations we introduce in Section 8.3 to encode the results of the front end in the IR.

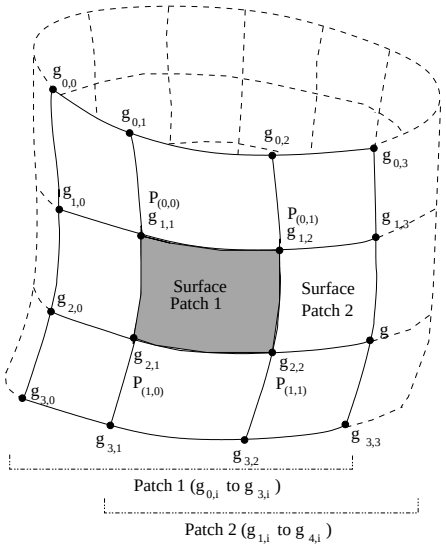

Figure 8-6 Coarse smoothing: A piecewise bi-cubic surface is fitted around each temporally convex video object. Multiple bi-cubic Catmull-Rom patches are stitched together with overlapping control points, to form a C_2 continuous bounding surface [14, 51].

Surface fitting by relaxation

Each video object is temporally convex, and as such its external boundary may be described by a continuous 2D surface — disregarding its"end caps", i.e. planes of constant time. These "end caps" are represented implicitly by the minimum and maximum temporal coordinates of the bounding surface's control points.

We fit a constrained 2D parametric surface to each segmented video volume, which we describe piecewise with a series of bi-cubic Catmull-Rom patches [14]. Each individual patch may be represented in a 2D parametric form $\underline{P}(s,t)$, where s and t are the spatial and temporal parameterisations, as:[2]

$$\underline{P}(s,t) = \underline{t}^T \underline{\underline{M}}^T \begin{pmatrix} \underline{g}_{0,0} & \underline{g}_{0,1} & \underline{g}_{0,2} & \underline{g}_{0,3} \\ \underline{g}_{1,0} & \underline{g}_{1,1} & \underline{g}_{1,2} & \underline{g}_{1,3} \\ \underline{g}_{2,0} & \underline{g}_{2,1} & \underline{g}_{2,2} & \underline{g}_{2,3} \\ \underline{g}_{3,0} & \underline{g}_{3,1} & \underline{g}_{3,2} & \underline{g}_{3,3} \end{pmatrix} \underline{\underline{M}}\underline{s} \tag{8.3}$$

where $\underline{t} = \left(t^3, t^2, t, 1\right)^T$, $\underline{s} = \left(s^3, s^2, s, 1\right)^T$, and $\underline{\underline{M}}$ denotes the Catmull-Rom cubic blending matrix [14] (given in equation 7.27). This class of spline function was chosen since it both interpolates all control points $\underline{g}_{i,j}$ (allowing greater local control over the

[2]Equation 8.3 represents a mild abuse of notation (used in the text of Foley *et al.* [51]) to avoid introduction of tensor notation. The 4×4 matrix of control points should be read as containing sixteen vector valued points $\underline{g}_{i,j}$ — rather than as an expansion of those points to create a 12×4 matrix of scalars.

surface during the fit), and ensures C_2 continuity between adjacent patches, i.e. whose control points overlap. We introduce the notation $\underline{Q}(s,t)$ to describe the entire 2D bounding surface, consisting of multiple piecewise bi-cubic patches (see the example of Figure 8-6).

The C_2 continuity of the Catmull-Rom spline class is important to our application since surface discontinuities in the first and second derivatives with respect to time ($\frac{\partial Q}{\partial t}$, $\frac{\partial^2 Q}{\partial t^2}$) have been found to produce jerky and incoherent motion in resulting animations. However, we note that spatial discontinuities in derivatives ($\frac{\partial Q}{\partial s}$, $\frac{\partial^2 Q}{\partial s^2}$) may be desirable when, for example, representing sharp corners. Fitting is performed via a generalisation of 1D active contours [91] to 2D surfaces (after [21]):

$$E_{surf} = \int_0^1 \int_0^1 E_{int}(\underline{Q}(s,t)) + E_{ext}(\underline{Q}(s,t))dsdt \tag{8.4}$$

the internal energy is:

$$E_{int} = \alpha \left| \frac{\partial \underline{Q}}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 \underline{Q}}{\partial s^2} \right|^2 + \gamma \left| \frac{\partial \underline{Q}}{\partial t} \right|^2 + \zeta \left| \frac{\partial^2 \underline{Q}}{\partial t^2} \right|^2 \tag{8.5}$$

and the external energy is:

$$E_{ext} = \eta f(\underline{Q}(s,t)) \tag{8.6}$$

Function $f(.)$ is the Euclidean distance of the point $\underline{Q}(s,t)$ to the closest voxel of the video object. Hence constant η controls the influence of the data in the fit (we present this to unity). The pairs of constants $[\alpha, \beta]$, and $[\gamma, \zeta]$ bias the internal energy toward certain spatial and temporal characteristics respectively. Reducing constants α and γ cause more elastic spacings of control points, whilst reducing constants β and ζ allow greater curvature between control points. We preset α and β at 0.5 and 0.25 respectively, to permit high curvatures (for example, sharp corners), and so close fits in the spatial dimension. γ is set to 0.5 to maintain temporal spacing of control points. Constant ζ is the most interesting, since this parameter dictates the energy penalties associated with high curvatures in the temporal dimension. Recall that we observe high curvatures in this dimension correlate strongly with temporal incoherence. Thus by varying this parameter the we may vary the level of temporal smoothing in the sequence; we have chosen to make the value of this temporal constraint available as a user variable. In this manner the animator may tailor the degree of temporal smoothing to the speed of motion in the sequence, or even chose to retain some of the noise present as a consequence of the segmentation process; we draw analogy with presence

of film grain in a movie.

Surfaces are fitted within the video volume using an adaptation of Williams' algorithm to locate surface points [167] through minimisation of E_{surf} (equation 8.4). Note that this algorithm relaxes penalties imposed due to high magnitudes in the second derivative during the final iterations of the fitting process. We inhibit this behaviour in the temporal dimension to improve temporal coherence. Initial estimates for surface control points are obtained by sampling the bounding voxels of the object at regular intervals. The result of the process is a fitted 2D surface about each video object. The video volume is then re-segmented using these fitted surfaces to create a set of smoothed video objects in a voxel representation. At this stage, the fitted 2D surfaces are discarded, since they have served their purpose in smoothing the video objects.

In practice, smoothing the surface of each video object independently is problematic. If the position of the surface bounding a video object is adjusted, voxel "holes" may appear in the video volume, that is, some voxels are no longer assigned to any video object. Alternatively, a single voxel might now fall within the fitted, bounding surfaces of more than one video object; we term these voxels "duplicates". We take the following simple steps to remove "holes" and "duplicates". First, any "duplicate" voxels are relabelled as "holes". Second, "holes" are filled by repeatedly "growing" video objects via binary morphological dilation; only voxels marked as "holes" are overwritten during this process. Video objects are dilated in random order to promote an unbiased, and even filling of holes. The process terminates when all holes are filled.

8.3 Front end: Building the Representation

The results of the video segmentation process (Section 8.2) are a set of segmented, smooth, video objects in a voxel representation, and an object association graph describing how video objects are connected over time. From this information we generate an intermediate representation (IR) of the video, to be passed to the back end. This IR consists of a series of "Stroke Surface" patches and a counter-part database. In this Section we describe each of these components in turn, and explain how they are created.

In our IR, the spatiotemporal locations of objects are represented in terms of their interfacing surfaces. This is preferable to storing each object in terms of its own bounding surface as boundary information is not duplicated. This forms a more compact, and more manipulable representation (which is useful later when we deform object boundaries, either for further fine smoothing, or to introduce temporal effects) see Figure 8-10. By manipulating only these interfacing surfaces, we do not create "holes" as before.

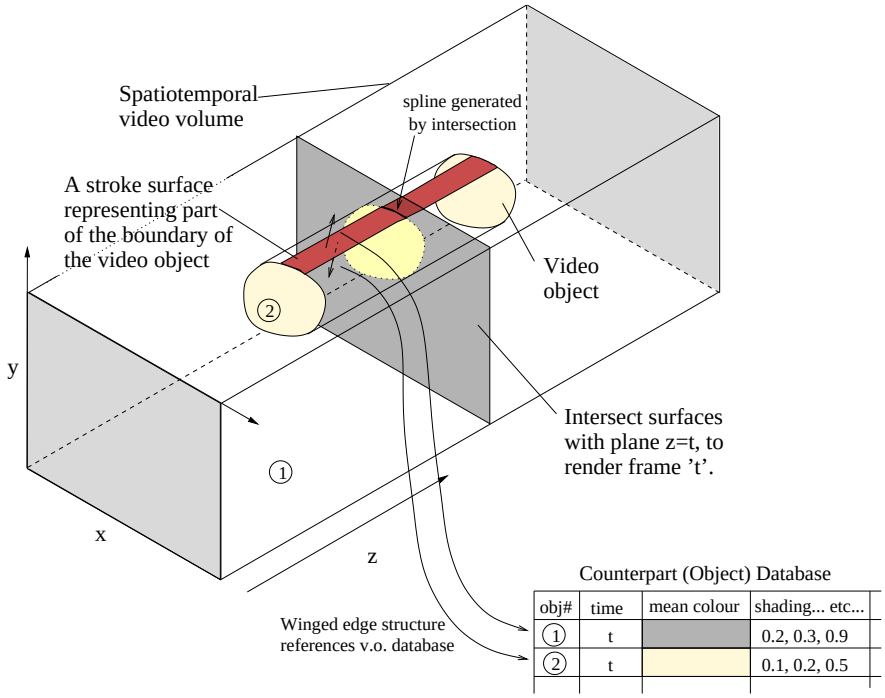

Figure 8-7 Illustrating the IR of the artistic shading subsystem, and the means by which it is rendered. Video is parsed by the front end, and represented as a collection of Stroke Surface patches — each accompanied by a winged edge structure which references the counter-part database. The attributes of video objects are stored in this database during parsing, here we demonstrate two objects, a yellow blob and a grey background. The back end renders the IR by intersecting a time plane with Stroke Surfaces; intersected surfaces form splines (holding lines) which bound regions. A region is rendered according to the graphical database attributes referenced by its bounding Stroke Surface(s).

However, in this representation it is much harder to vary the topology of the scene, for example the adjacency of the video objects. This motivated the coarse smoothing step of subsection 8.2.3.

8.3.1 Stroke Surface Representation

When two video objects abut in the video volume, their interface (in planes non-parallel to z) is represented in our IR in piecewise form by a collection of one or more disconnected surface patches. We introduce the term "Stroke Surface" to describe these individual patches.

As in Section 8.2.3 we use a piecewise Catmull-Rom bi-cubic form to represent a Stroke Surface, and a 2D parameterisation $\underline{Q}(s,t)$ to describe points upon it. Each Stroke Surface is contiguous. However, under certain interface geometries it is possible that internal holes may exist within the surface's external boundary — as such, regions of surface's parameter domain ($s,t \in [0,1]$) may be invalid. We address this repre-

sentational problem in a manner similar to binary CSG [51]. Each Stroke Surface is described by a continuous, externally bounded "primary" surface, from which a set of zero or more continuous secondary (or "hole") surfaces are subtracted[3] (Figure 8-8). We specify these holes as 2D polygonal regions in the parameter space $[s,t] \in \Re^2$.

To further illustrate the concept of Stroke Surfaces, consider the following two examples:

1. Consider two plasticine bricks (A and B). If the bricks were pressed together around a long metal pole, A and B would touch on either side of the pole, but the pole would "split" the interfacing surface between A and B into two discontinuous surfaces. The pole, A, and B are analogous to video objects. The interface between A and B would be represented by two Stroke Surfaces.

2. Now consider two new plasticine bricks (C and D), which are pressed together to completely surround a third object E. In this scenario, only one Stroke Surface is required to represent the interface between C and D. However, that surface will contain an internal "hole" where C touches E, rather than brick D. In fact the C-E and D-E interfaces will be represented by two further Stroke Surfaces.

8.3.2 Fitting Stroke Surfaces

Stroke Surfaces are fitted to the voxel video objects in the following manner. Each video object is systematically tested against each of its peers to determine whether the pair are spatially adjacent, that is, if there is an interface non-parallel to the z plane (temporal adjacency is not encoded via Stroke Surfaces, as this information is already stored in the object association graph). If an interface exists between the two objects, then that interface must be encoded in the IR by one or more Stroke Surfaces. We now describe the process by which we fit a single Stroke Surface between two example voxel video objects; we write the coordinate sets of voxels comprising these video objects as $\mathcal{O}$ and $\mathcal{P}$.

Fitting the primary surface

We begin by fitting the "primary" surface. This is accomplished in a manner similar to Section 8.2.3, using Williams' relaxation algorithm to fit a 2D parametric surface to the voxels which represent the interface between the two video objects. Internal parameters α', β, γ, ζ and η are set as described in Section 8.2.3, as we desire similar geometric properties in our fitted surface. However a new function $f(.)$ is substituted for the external energy term (see equation 8.6), which determines error between the

[3]Read "subtraction" in the context of binary constructive solid geometry (CSG). For example if two objects are represented by sets of voxels A and B, then A subtract B may be written as $A \setminus B$.

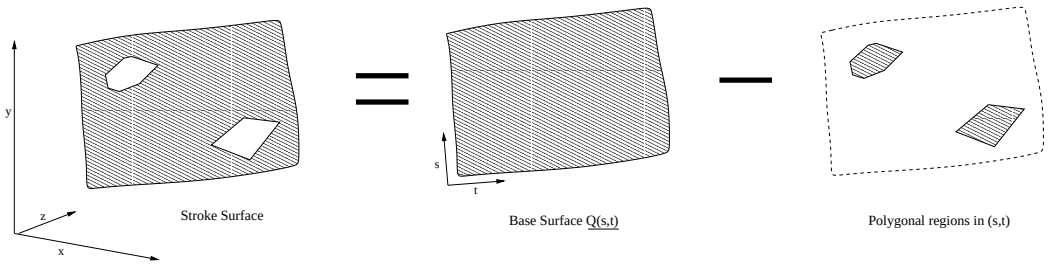

Figure 8-8 Stroke Surfaces are represented as a primary parametric surface $\underline{Q}(s,t) \in \Re^3$, minus a collection of one or more polygonal regions in space $[s,t] \in \Re^2$ which describe "holes" in that surface.

fitted surface and the data:

Given two voxel video objects $\mathcal{O}$ and $\mathcal{P}$, we compute a distance field $|\vec{\mathcal{OP}}| \in \Re^3$ from each voxel in $\mathcal{O}$ to the nearest voxel in $\mathcal{P}$. Likewise we compute field $|\vec{\mathcal{PO}}| \in |\Re^3$ from each voxel in $\mathcal{P}$ to the nearest voxel in $\mathcal{O}$. We generate a field $F = \min(|\vec{\mathcal{OP}}|, |\vec{\mathcal{PO}}|)$. The manifold $F = 0$ now represents the interface between $\mathcal{O}$ and $\mathcal{P}$, and points in field F encode distance from this interface. We wish to fit our surface to the manifold $F = 0$. Thus F forms the basis of the external function $f(.)$, which measures distance between points on the surface, and the interface to which that surface is to be fitted. The distance field F is also useful later to identify "holes" within this fitted, primary surface (see next subsection).

All that remains is to define an initial estimate for the surface control points. We threshold field F at a distance of one voxel, and apply standard morphological thinning [182], to obtain a field F'. The binary voxel map that results is a good approximation to the manifold $F = 0$. We sub-sample the voxels upon this manifold at equal temporal instants, and at equal spatial arc-length increments, to establish initial estimates for the control points. An optimal configuration of surface control points is then sought using Williams' algorithm, and so the primary surface $\underline{Q}(s,t)$ is fitted. We re-parameterise s and t to be normalised arc-length parameters, using standard methods (linear approximation, see [51]).

Fitting holes within the primary surface

We wish to find the set of polygons in space $[s,t] \in \Re^2$, which describe holes within the interior of the primary surface $\underline{Q}(s,t)$. Recall that these are the regions which do not lie on the interface between video objects $\mathcal{O}$ and $\mathcal{P}$. We sample the field $F(\underline{Q}(s,t))$ to create a 2D scalar field which denotes distance to the video object interface at point (s,t); i.e. the error in the fit of the primary surface $\underline{Q}(s,t)$. In cases where (s,t) does

not lie close to the interface of video objects $\mathcal{O}$ and $\mathcal{P}$, then $F(\underline{Q}(s,t))$ will be large. We threshold this field to obtain a collection of connected binary components which correspond to regions of $\underline{Q}(s,t)$ which do not lie on the interface. We obtain the chain codes [52] of the external boundaries of each component, and use a standard chain code vectorisation technique [119] to obtain a polygonal description of each region.

Each Stroke Surface is thus represented as a single function $\underline{Q}(s,t)$ (encoded in piecewise bi-cubic form), and a series of 2D polygonal regions which describe "invalid regions" of the parameter domain $[s,t] \in \Re^2$. We store the complete set of Stroke Surfaces for the video as one half of the IR. Each stroke surface holds an additional *winged edge structure* which contains two pointers corresponding to the two objects which it separates (one for each normal, Figure 8-7). These pointers reference records in the counter-part database, as we now describe.

8.3.3 Counter-part Database

A database is maintained as the second half of the IR, containing one record per object in the video volume. This counterpart database is referenced by the pointers held in the Stroke Surfaces' winged edge structure, and encapsulates the object association graph created by the region association process (Section 8.2.2), as well as various graphical attributes about each object at each frame of its existence. We now briefly summarise the information stored in the counterpart database, describing how the database is populated with this information in the next subsection (Section 8.3.4).

For each temporally convex video object (referenced by the Stroke Surfaces), the database maintains a record containing the fields:

1. **Born:** The frame number (B) in which the video object comes into existence.
2. **Died:** The frame number (D) in which the video object determines.
3. **Parent object list:** A list of video objects which have either split, or merged with other video objects, at time B to form the current video object. If this list is empty, the video object "appeared" as a novel feature in the sequence — all video objects in frame 1 will have be born in this manner, as will any regions which were segmented in a frame but were too different to be associated to any regions in previous frames (these often appear just after an occlusion).
4. **Child object list:** A list of video objects which the current video object will become one of (possibly many) parents for at time D. If this list is empty, the video object simply disappears (possibly due to occlusion, or because the end of the video sequence has been reached).

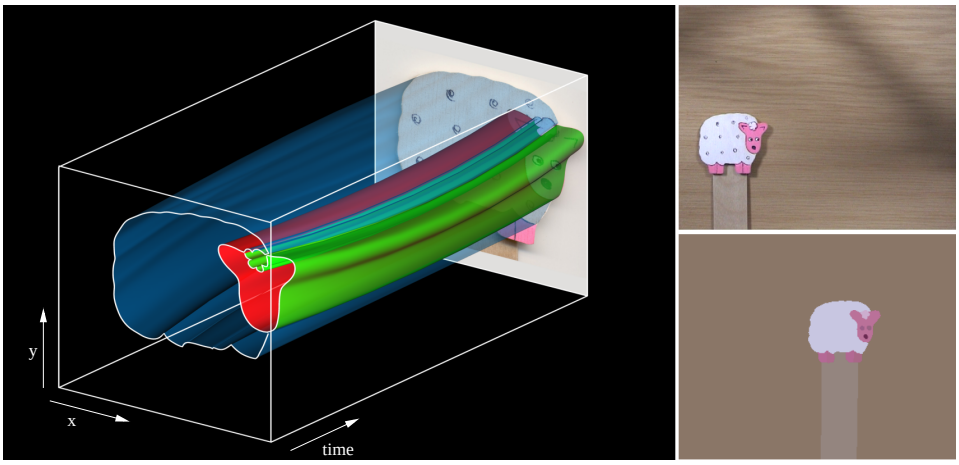

Figure 8-9 A visualisation (left) of the Stroke Surfaces generated from the *SHEEP* sequence (upper right), which when intersected by the a time plane generate a coherent segmentation (lower right). Observe how a single video object, for example the sheep's body, may be described by multiple Stroke Surfaces; in the example of the sheep's body these are the blue surface meeting the background object, and the red surface meeting the head object (see `videos/sheep_source` and `videos/sheep_flatsegment`).

5. **Colour list:** A list of $B - D + 1$ RGB colour vectors which encode the mean colour of the region in the original video, during each frame of the video object's existence.

6. **Homography list:** A list of $B - D + 1$ 3×3 matrices which describe the projective transform mapping the texture within the region, at each frame of the video object's existence, to the texture within the region in the previous frame. The first element in this list is the identity matrix, and is redundant.

7. **Shading list:** A list of $B - D + 1$ triples, containing the linear gradient shading parameters described in Section 8.3.4.

Attributes (1-4) encode the object association graph generated by the region association process of Section 8.2.2). Attributes (5-7) encode graphical information for each time instant t ($B \leq t \leq D$) useful to the back end during rendering. This signal in these graphical attributes is low-pass filtered (smoothed) over period $[B, D]$ by the front end to improve temporal coherence within the animation. For clarity of reading, we have deferred justification of this smoothing process to Section 8.4.2 of the back end, where the benefits of smoothing are more easily demonstrated in the context of cartoon flat-shading.

8.3.4 Capturing Interior Region Details in the Database

We begin by computing the mean colour of each region for each frame of the video. This information is stored in the respective video object record of the counterpart

database. At this stage in our explanation, each feature within the video is therefore stored in our representation only in terms of a series of linked, coherent spatiotemporal object boundaries and their mean interior colours. This is sufficient to recreate a basic, flat shaded animation, but can prove insufficient to render some artistic styles — the internal texture that has been abstracted away often forms perceptually important visual cues (for example, the cross-hatching marks sketched by artists to represent illumination, and so depict depth in a scene). We now describe two approaches we have developed to introduce internal detail into the animation in a temporally coherent manner.

First, we fit a linear shading gradient to each object on a per frame basis. The gradient at time t over an object may be described as a triple $\underline{G_t} = [g_0, g_1, \theta]$, where g_0 and g_1 are the start and end shading intensities respectively, and θ specifies the direction of shading over the region (as an angle). An optimal $\underline{G_t}$ is computed by a search [114] aiming to minimise the error $E(.)$:

$$E(\underline{G_t}, F_t) = \frac{1}{|P|} \sum_{p \in P} |I(\underline{G_t}) - F_t|^2 \tag{8.7}$$

Where $I(\underline{G_t})$ is a image created from the gradient triple $\underline{G_t}$, using the hue and saturation components of the object mean colour from the database and varying the luminance as defined by $\underline{G_t}$. F_t is the video frame at time t, and $\mathcal{P}$ is the set of pixels inside the object region at time t. The application of this gradient alone, when rendering, can dramatically improve the sense of depth in an image (Figure 8-14).

Second, images of the original object in a small temporal window around t are differenced with the optimal $I(\underline{G_t})$; the result is a difference image containing the detail thus far abstracted away by our representation. We pass these images through our salience measure (described in Section 3.2), which correlates salience and rarity; these maps indicate the salient marks in the interior of the region. We form a local motion estimate for the object over the temporal window by assuming the object to be approximately planar, and so its motion relative to the camera to be well approximated by a homography. Object regions over the window are projected to the reference frame of the object at time t. A initial degenerate estimate of the homography is obtained by taking the 2nd order moments of the two regions. This estimate is then refined using a Levenburg-Marquadt iterative search to minimise mean square pixel error between the interiors of the regions, using an identical optimisation approach to that described in Section 6.3.1. The computed salience maps are projected by homography to the reference frame at t, and averaged to form a final map. This results in the suppression of sporadic noise and reinforcement of persistent salient artifacts (under

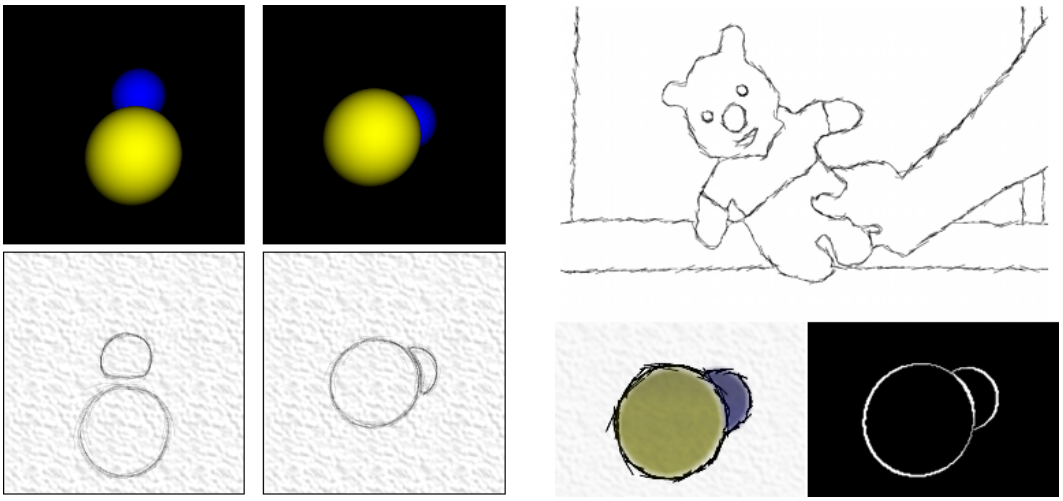

Figure 8-10 Left: Early experiments applied to a synthetic (bouncing spheres) test scene represented each video object individually in the IR by means of the individual bounding surfaces. This led to duplication of surface information, causing boundaries to no longer appear coincident following surface manipulation (for example after creating the sketch rendering effect described in Section 8.4.4). This led to unappealing artifacts such as the double edges between spheres (2^{nd} image set on left). Right: Our IR now represents video objects by means of their interfacing surfaces (Stroke Surfaces), which do not create artifacts such as double edges or holes when they are manipulated in the video volume.

the assumption that noise obeys the central limit theorem). We threshold the map, and apply morphological thinning. Stroke Surface patches are then fitted around each disconnected artifact as before. Finally, the patches are transformed to their original reference frames via the inverse homographies used to generate the map.

One shading gradient triple $\underline{G_t}$ and a homography are stored in the supplementary database, per frame of the video object. The homographies are of additional use later for rotoscoping and stroke based rendering. The new interior Stroke Surfaces are added to those of the existing representation, but with both sides of their winged edge structure set to point to the video object in which they originated. Thus interior edge markings are also encoded in our representation.

8.4 Back end: Rendering the Representation

We now describe how the IR is rendered by the back end, to produce animations in a variety of artistic styles.

Stroke Surfaces are first duplicated to produce two sets; one for the interior shading stage, and one for the line rendering stage, of the back end (Figure 8-7). These sets of surfaces may then be manipulated in some manner; for example, fine scale temporal smoothing may be applied to boost temporal coherence, or novel frequency components

introduced to the surfaces to create "coherent wobbles" (Section 8.4.1).

To render a particular frame at time t, the sets of Stroke Surfaces embedded in the video volume ($[x, y, z] \in \Re^3$) are intersected with the plane $z = t$. The frame is then rendered in a two stage process comprising: 1) shading of the object's interior region (processing the first set of surfaces); 2) rendering of the object's outline (referred to by animators as the *holding line*) and any interior cue lines also present (processing the second set of surfaces) . This separation allows us to create many novel video effects, such allowing interior shading to spill outside of the holding lines.

The remainder of this section is organised as follows. We first describe how Stroke Surfaces may be smoothed, or otherwise manipulated in frequency space, to create a number of temporal effects (subsection 8.4.1). We then describe how interior regions bounded by these surfaces are rendered, by drawing on data in the graphical attributes in the database component of the IR (subsection 8.4.2). Rotoscoping, matting and stroke based AR styles are also obtainable within this single framework (subsection 8.4.3). Finally, we describe the mechanism for rendering holding and interior lines in the animation (subsection 8.4.4).

8.4.1 Surface Manipulations and Temporal Effects

The representation of video as a set of spatiotemporal Stroke Surfaces simplifies manipulation of the image sequence in both spatial and temporal domains, and enables us to synthesise novel temporal effects which would be otherwise difficult to produce on a per frame basis. In this subsection we describe a method for manipulating the geometry of Stroke Surfaces in frequency space. We show that by applying a low-pass filter in the temporal dimension we may further improve temporal coherence, and by introducing novel frequency components we can produce coherent "wobbles" in the video reminiscent of popular commercial cartoons, for example "Roobarb and Custard" [Grange Calveley, 1972].

Planar offset parameterisation

We now describe our mechanism for manipulating a Stroke Surface $\underline{Q}(s,t)$ to produce small scale deformations, and so create spatiotemporal effects in the animation. We subsample the fields $s \in [0,1], t \in [0,1]$ at equal, user defined intervals, to obtain a grid of well distributed points on the manifold $\underline{Q}(s,t)$. By creating a simple 3D triangular mesh using these regular samples as vertices (see the geometry of Figure 8-11), we are able to create a piecewise planar approximation to $\underline{Q}(s,t)$, which we write as $\underline{P}(s,t)$.

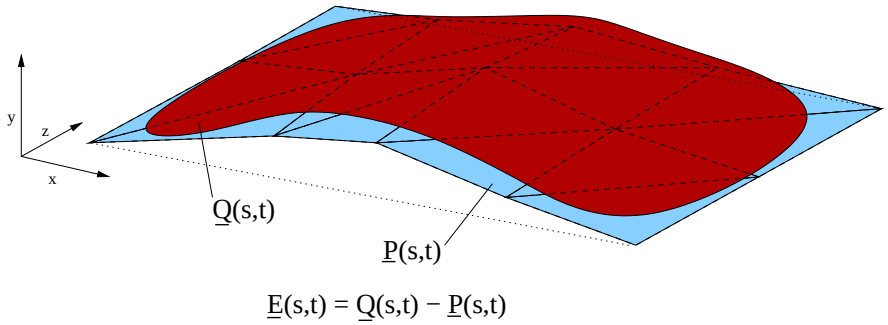

Figure 8-11 The planar offset parameterisation $\underline{E}(s,t)$ is derived by differencing corresponding points on the Stroke Surface $\underline{Q}(s,t)$ with those on a piecewise planar approximation to the surface $\underline{P}(s,t)$. $\underline{P}(.)$ is created by sub-sampling and linking points on the manifold of $\underline{Q}(.)$.

The error (offset) of this approximation is:

$$\underline{E}(s,t) = \underline{Q}(s,t) - \underline{P}(s,t) \tag{8.8}$$

Suppose we now hold $\underline{P}(.)$ constant, and allow manipulation of the offset field (we write the modified field as $\underline{E}'(.)$). We obtain a modified Stroke Surface:

$$\underline{Q}'(s,t) = \underline{P}(s,t) + \underline{E}'(s,t) \tag{8.9}$$

In our framework we allow manipulation of $\underline{E}'(.)$ through variation of a 2D scalar field $S(.)$:

$$\underline{E}'(s,t) = S(s,t)\frac{\underline{E}(s,t)}{|\underline{E}(s,t)|} \tag{8.10}$$

The identity in this framework is $S(.) = |\underline{E}(.)|$. However alternative $S(.)$ are possible, and variation of this field forms the basis for temporal effects in our framework.

To maintain continuity between neighbouring Stroke Surfaces we must reduce the magnitude of any deformations local to the surface's boundaries (both the external surface edges where either s or t are close to 0 or 1, and around the edges of any internal hole within the Stroke Surface). We define a 2D field of weights in (s,t) space by producing a binary map of these internal and external boundaries and performing a 2D distance transform. We then raise values in this field to a user defined power φ and normalise to obtain the weight field, which we write as $W(s,t)$. Parameter φ may increased by the animator to decrease the apparent rigidity of the Stroke Surface during these deformations. We revise equation 8.9 to weight any change in $\underline{Q}(s,t)$ by $W(s,t)$:

$$\underline{Q}'(s,t) = \underline{P}(s,t) + (1 - W(s,t))\underline{E}(s,t) + W(s,t)\underline{E}'(s,t) \tag{8.11}$$

Frequency manipulation of the offset field

Manipulation of the 2D field $S(s,t)$ is performed in the frequency domain by defining a complex transfer function $\mathcal{T}(\mathcal{C}^2)$ such that:

$$S'(.) = F^{-1}[\mathcal{T}(F[S(.)])] \tag{8.12}$$

where $S'(.)$ is the modified scalar field, $F[.]$ and $F^{-1}[.]$ specify the standard and inverse discrete Fourier transforms respectively, and $\mathcal{T}(.)$ is a general 2D transfer function which maps a 2D complex field to some other 2D complex field. The user is free to develop custom transfer functions, however we now give two useful examples:

1. **Temporal Smoothing:**

   We can apply a low-pass filter $S(.)$ to remove any high frequency undulations in the temporal dimension of the surface. A standard method to achieve this is through convolution with a Gaussian of user defined size σ, i.e. the following multiplicative transfer function in the frequency domain:

   $$\mathcal{T}_{smooth}(F;u,v) = \frac{F(u,v)}{\sqrt{2\pi}\sigma} e^{-\frac{v-d}{2\sigma^2}} \tag{8.13}$$

   where $F[u,v]$ specifies a component in the 2D Fourier domain, and d is a scalar such that the line $v = d$ intersects the coordinates of the d.c. component in $F[u,v]$. Depending on the implementation of the discrete Fourier transform, d may be located at zero or at the centre of the v axis.

2. **Coherent Wobbles:**

   We are able to introduce controlled wobbles and distortion effects by perturbing Stroke Surfaces, producing a distinctive pattern of incoherence in the animation. Such effects are easily generated by introducing several impulses in the frequency domain. The i^{th} impulse is defined by a randomly generated triple $[f_i, a_i, \omega_i]$; f_i is the frequency of the wobble effect; a_i is the amplitude of the effect; ω_i is the angular speed of the effect (the rate at which crests and troughs in wobble appear to "rotate" around a region's perimeter over time). We write the modified signal as:

   $$\mathcal{T}(F[\underline{x}]) = F[\underline{x}] + J(\underline{x}) \tag{8.14}$$

   where $F[\underline{x}]$ represents a the frequency component $\underline{x} = (u,v)^T$ in the 2D Fourier domain, and $J(\underline{x})$ the field containing impulses:

   $$J(\underline{x}) \quad = \quad \sum_{i=1}^{n} \begin{cases} a_i \text{ if } |\underline{x} - (\underline{D} + \underline{\nu}_i)| = 0 \\ 0 \text{ otherwise} \end{cases} \tag{8.15}$$

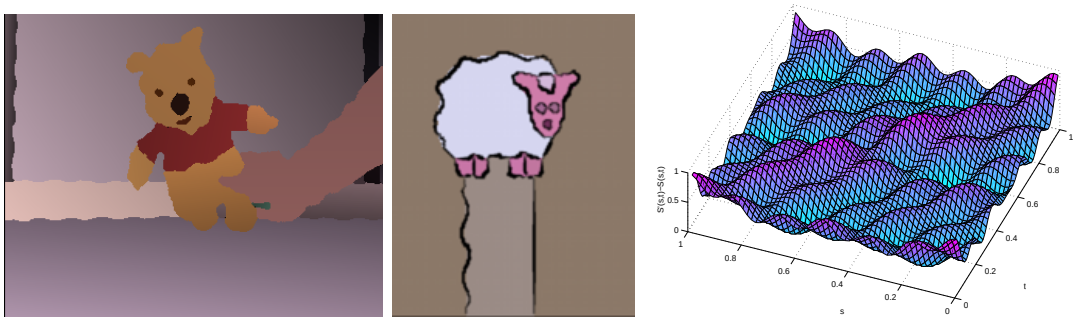

Figure 8-12 Not only can we enforce temporal coherence, but also introduce novel forms of incoherence. Left: example of a coherent wobbling effect in the gradient shaded *POOHBEAR* animation (`videos/pooh_wobblygradshade`). Middle: A example still from *SHEEP* demonstrating how the animator may transform one set of Stroke Surfaces (used to render holding and interior lines), yet leave the surfaces for interior regions unaffected (`videos/sheep_wobblycartoon`). Right: A visualisation of the random displacement field $(S'(.) - S(.))$ for a single Stroke Surface from *POOHBEAR*.

introducing the notation $F[\underline{D}]$ to specify the d.c. component in the Fourier domain, and n as the total number of impulses. Inside the summation, the term $\underline{\nu_i}$ is defined as:

$$\underline{\nu_i} = \begin{bmatrix} \cos\omega_i & -\sin\omega_i \\ \sin\omega_i & \cos\omega_i \end{bmatrix} \begin{bmatrix} f_i \\ 0 \end{bmatrix} \tag{8.16}$$

This mechanism creates smooth, periodic displacements in the 2D signal $S(.)$. This in turn produces smoothly varying undulations upon the Stroke Surface. As a result the wobbles of region boundaries appear to move coherently over time in the animation.

Since adjacent objects are represented in terms of their interfacing surfaces, residual temporal incoherences in those boundaries may be dampened by smoothing the surfaces. There is no danger of introducing "holes" into the video volume as with the coarse smoothing step (Section 8.2.3) — if volume is lost from one video object, it is gained by surrounding video objects. Similar consistency applies to all temporal effects created by this mechanism.

Figure 8-12 contains stills from animations produced using the second of these transfer functions, demonstrating that a simple periodic displacement function over Stroke Surface patches can produce dramatic and novel temporal effects in the animation. The random displacement field $S'(.) - S(.)$ for a single Stroke Surface in the *POOHBEAR* sequence has has also been visualised.

8.4.2 Rendering the Interior Regions

The process of intersecting the plane $z = t$ with the spatiotemporal Stroke Surfaces in the IR produces a series of splines which are scan converted into a buffer. Bounded regions in this buffer correspond to the interiors of video objects. In this section we describe how these interior regions are rendered to form part of the animation.

For a given region, we begin by determining exactly which video object that region is a "slice" of. This information is obtained by examining the the winged edge structures attached to the bounding Stroke Surfaces. Thus each region inherits a pointer referencing a record in the database component of the IR, which contains all the rendering information about that region at time instant t. This information may be used to render regions in a number of artistic styles.

Cartoon-style Flat Shaded Animations

Arguably the most straightforward strategy for rendering a region interior is to flat shade with a single, mean colour computed over that region's footprint in the original video. Recall that the front end recorded this colour information (at each time instant) in the database component of the IR. The cartoon-like "flat shading" effect that results goes some way to satisfying the second (shading) sub-goal of our original motivation — the automated generation of cartoons from video. However as video objects divide and merge over the course of the clip, the mean colour of their imaged regions can change significantly from frame to frame (perhaps due to shadows). This can cause unnatural, rapid colour changes and flickering in the video (see the left hand skirting board in Figure 8-13).

Thus, it is not enough to obtain temporally smooth segmentation (Stroke Surface) boundaries in an animation. One must also apply shading attributes to the bounded regions in a temporally smooth manner. The flickering of region colour we demonstrate is symptomatic of the more general problem of assigning the graphical attributes stored in the IR database, to regions in a coherent way. We draw upon our spatiotemporal representation to mitigate against this incoherence — specifically, by smoothing database attributes over time, as previously alluded in Section 8.3.

Smoothing Database Attributes

Recall that objects are associated via a graph structure; pointers to each video object's child and parent objects are stored in the database component of the IR. We analyse this graph to obtain a binary voxel map describing the union of all video objects within

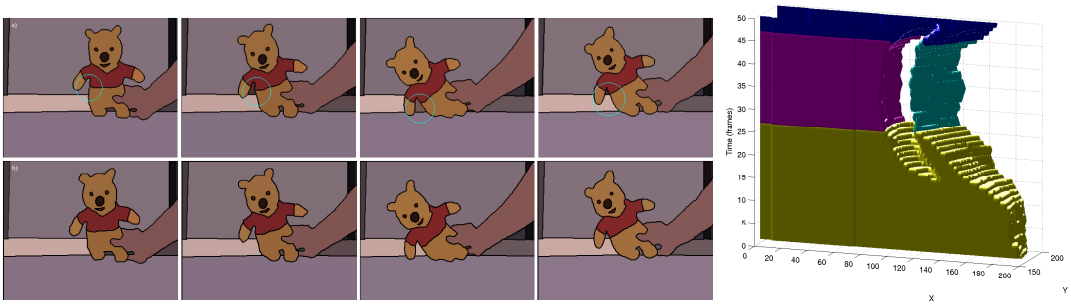

Figure 8-13 Demonstrating the temporal coherence of our cartoon-style flat shading. Top: Temporal incoherence (highlighted in blue), in the form of flickering colour, is caused by shading on a per frame basis (`videos/pooh_incoherentshade`). Bottom: Our spatiotemporal representation allows us to mitigate against these incoherences by smoothing attributes, such as colour, over the video volume (`videos/pooh_coherentshade`). Right: A volume visualisation of the left hand skirting board, comprised of a four associated video objects. Smoothing attributes with respect to this, and other volumes in the sequence, improves temporal coherence.

the subgraph containing the video object corresponding to the region being rendered. By averaging graphical database attributes (5-7, see Section 8.4.2), such as colour, over the volume we can create a smooth transition of those attributes over time (even if objects appear disjoint in the current frame but connect at some other instant (in the past or future). Such coherence could not be obtained using the per frame sequential analysis performed by current video driven AR methods [75, 96, 103].

The size of the temporal smoothing window is defined by the user; the larger the window size, the smoother the transitions of attributes over time; but consequently, rapid motions or lighting changes may not be reproduced faithfully in the animation. The animator thus varies the temporal window size until they are satisfied with the result — typical window sizes range between five and ten frames.

Gradient Shaded Animations and Morphological effects

We observed in Section 8.3.4 that the highly abstracted nature of a flat shaded video can be unappealing for certain applications; artists often make use of shading and cue marks to add a sense of lighting and depth to a scene. We can augment the flat shaded regions by rendering the gradient shading attributes fitted earlier, smoothing the parameters in a similar manner to colour to ensure coherence (Figure 8-14). Interior line cues may also be added by rendering the interior Stroke Surfaces of the object (Figure 8-17) although the rendering of such cues occurs later in the line rendering stage (Section 8.4.4).

We are also able to apply morphological operators to interior regions, prior to rendering. Figure 8-18 demonstrates a water-colour effect (combined with a sketchy outline — see

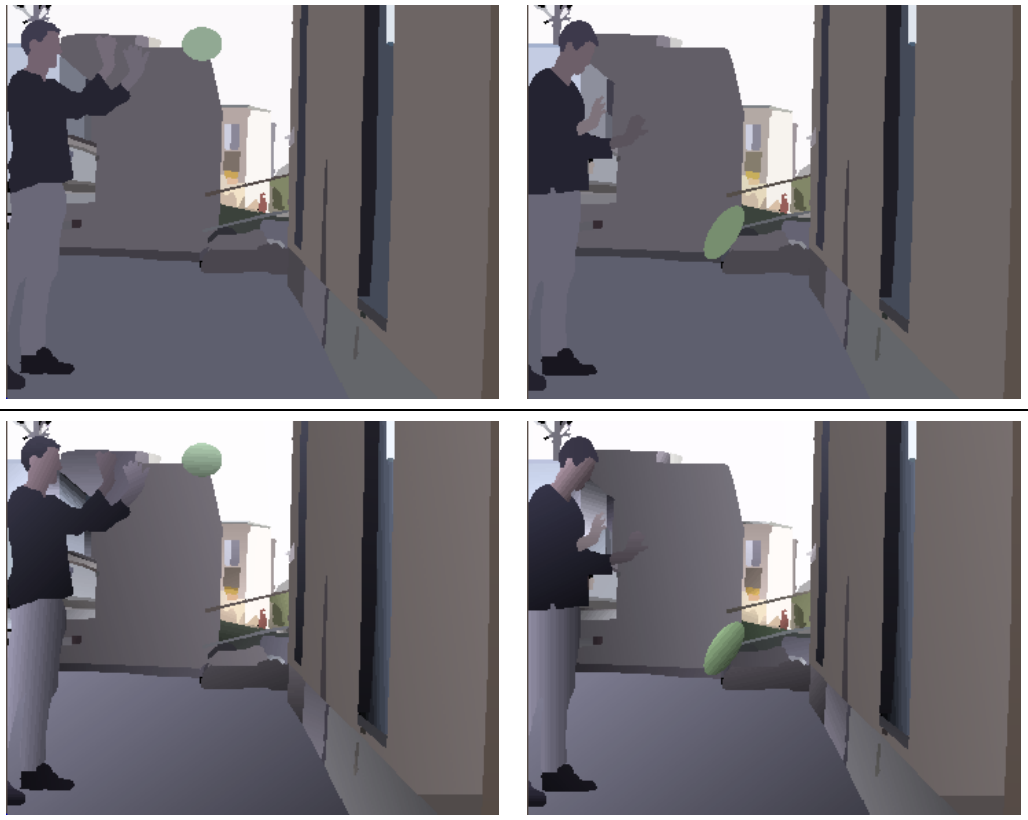

Figure 8-14 Examples of the temporally coherent flat shaded (top) and linear gradient shaded (bottom) interior rendering styles available in our framework (see `videos/bounce_flatshade`, `videos/bounce_gradshade`).

Section 8.4.4 for details), in which we have applied an erosion operator to the region prior to rendering; this gives the aesthetically pleasing impression of brush strokes stopping "just short of the edges". The effect is coherent over time (see Appendix C, `videos/pooh_watercolourwash`). The water-colour wash texture was produced via multiplication of pixel values with pre-supplied texture map.

8.4.3 Coherent Reference Frames and Stroke Based Rendering

The labour saving technique of "rotoscoping" was pioneered by cartoonists in the late 1930s, and was instrumental in the production of Disney's first feature length animated movie "Snow White" [Disney, 1937] — we refer the reader to Section 2.5.2 for further examples of rotoscoping. Traditionally, rotoscoping is the manual tracing over photographs, or film stills, to produce the stylised cels of an animation [169].

As with rotoscoping, AR algorithms draw over photorealistic footage to produce stylised output. In this sense, we argue that image-space static AR methods (for example stroke based renderers [27, 71, 103] or adaptive textures [65]) may be considered a form of modern day rotoscoping. This parallel between AR and rotoscoping is of further rele-

vance to video driven AR when one considers that both share a common objective; to produce stylised animations from video in which motion is coherent with the motion of the underlying source image sequence.

In this subsection we demonstrate that rotoscoping, video matting, and static AR can be unified in a single framework to produce temporally coherent animations from video. Our system creates temporally coherent motion estimates of image regions, requiring only a single key-frame to produce rotoscoping effects. The same framework may be used to place coherently moving brush strokes on image regions, enabling us to produce AR animations automatically from video.

Reference Frames and Rotoscoping

Recall the local motion estimate for video objects computed in Section 8.3.4, and stored in the database component of the IR. This estimate models inter-frame motion as a homography, and was previously used to create correspondence between region texture in adjacent video frames to enable recovery of internal edge cues. However this same motion estimate may be used to to implement automated rotoscoping in our framework. Animators draw a design, and attach it to a key-frame in the original footage. The inter-frame homography stored in the IR database is then used to automatically transform the design from frame to frame — the design remains static within the reference frame to which it is attached; it appears to be rigidly attached to the video object. Figure 8-15 demonstrates the attachment of this rigid frame using a synthetic chequerboard pattern. There are many useful applications for this automated rotoscoping; for example, to add personality to an animation by rotoscoping an expression on to a face (Figure 8-15, bottom). Such rotoscoping is particularly useful in our Video Paintbox, since small scale features such as eyebrows or mouths sometimes fail to be captured by the region segmentation and association processes of the front end.

Video Matting

As a special case of rotoscoping, we may set the inter-frame homography estimates for a region to the identity matrix (implying no region motion). By supplying these regions with video as a texture, rather than hand-drawn animations, we are able to replace shaded video objects with alternatively sourced video footage. This facilitates video matting within our framework, as we demonstrate in Figure 8-15 (top right) by substituting a novel background into the *SHEEP* sequence. We can also use the source video footage itself as a texture, and so reintroduce photorealistic objects from the original video back into the non-photorealistic animation. This technique produces the "mixed media" effects demonstrated in Figure 8-2 (`videos/bounce_mixedmedia`).

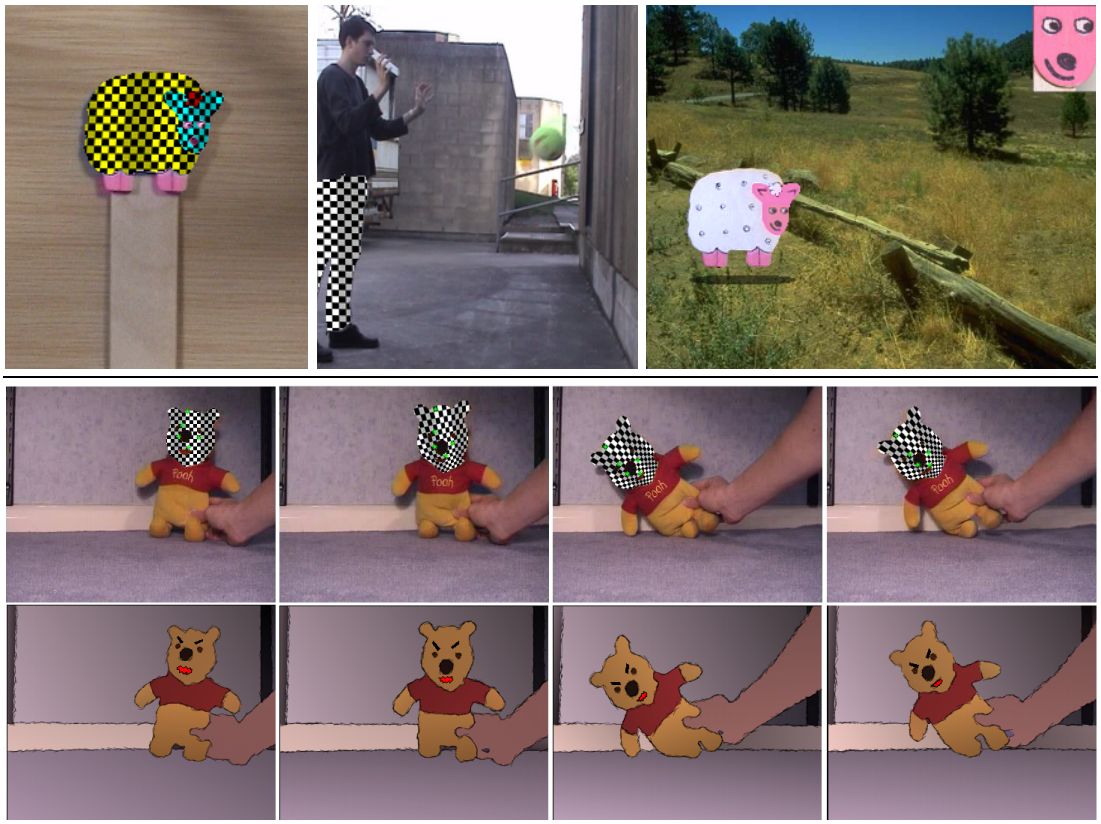

Figure 8-15 Rotoscoping and matting effects are possible within our AR video framework. Top left, middle: Rigid reference frames are attached to individual video objects via inter-frame homography (demonstrated here by a synthetic chequerboard pattern which moves coherently over time — `videos/sheep_reframe`, `videos/bounce_reframe`). Top right: The attached reference frame allows us to place markings such as illustrations, upon an object, and have those markings appear to move with the object: a form of automated rotoscoping. We may also substitute video objects (for example the background) for alternative textures to create matting effects `videos/sheep_rotomatte`. Bottom: A further example of rotoscoping, applying a new facial expression in the *POOHBEAR* sequence (`videos/pooh_angrybear`). Observe the green crosses on the chequerboard pattern; these symbolise the paint strokes which we may also attach to reference frames to produce coherent painterly effects (`videos/pooh_reframe`). Rotoscoping and stroke based AR are closely related in our rendering framework.

Note that all the effects described in this subsection require external information in addition to that stored in the IR — specifically, access to the relevant video source, image, or animator supplied illustration depending on the effect desired.

Coherent Stroke Based Rendering

Using a similar principal to rotoscoping, we are able to extend static, image-space AR methods to operate over video with a high level of temporal coherence. We observed (Section 8.1) that temporal coherence requires that:

1. Strokes should move in a manner consistent with the video content.

2. Strokes should not exhibit rapid fluctuations of their visual attributes, for example colour.

The first of these criteria may be met by rigidly attaching strokes to the local reference frames of video objects. From the viewpoint of the user, this causes stroke locations to exhibit motion matching that of the object to which they are attached. In this manner, strokes move coherently over time (see the green stroke centres in Figure 8-15). The reference frame attached to a video object defines a 2D plane of infinite extent, on which we place brush strokes. We subsample the plane at regular spatial intervals, to create potential locations for brush stroke centres. Strokes are instantiated at locations upon the sub-sampled plane when a potential stroke centre moves within the "footprint" of a video object — the region generated by intersecting the video object with the time plane at a given instant. We have found that, in contrast to rotoscoping, the full eight degrees of freedom of the homography are superfluous to needs when produced stroke based AR animations. A minimum least-squares error affine approximation to the homography produces aesthetically superior results.

The second criterion may be met by smoothing the visual attributes of strokes over time, in a similar manner to how visual database attributes in the IR are smoothed. Visual attributes, such as colour, are assigned to instantiated strokes; as with static AR, these attributes are functions of pixel data in the neighbourhood of the stroke's centre. These attributes are updated from frame to frame by re-sampling image data, causing the visual attributes of strokes to adapt to the video content. However, the state of attributes are also functions of their state in past and future time instants. The nature of this function is to constrain the values of attributes to evolve smoothly over time, so mitigating against stroke flicker. Once instantiated, a stroke is never destroyed — a record of the stroke's attributes persists for the remainder of the video sequence, even if the stroke moves out of the bounds of the footprint and becomes invisible. This retention of stroke attributes helps reduce flicker caused when strokes move repeatedly in and out of a footprint. However, note that visibility itself is one of many attributes assigned to strokes, and is therefore smoothed such that strokes do not rapidly alternate their visibility, but appear and disappear over lengthy time periods.

We now give an illustrative example of our coherent, stroke based rendering process by extending our pointillist painterly rendering technique of Chapter 3 to video.

Recall that each brush stroke in the painterly method of Chapter 3 formed a z-buffered conic of superquadric cross-section. Each stroke has seven continuous visual parameters:

1. α – Superquadric form factor

2. a – Aspect ratio of stroke
3. b – $1/a$
4. θ – Orientation of stroke
5. h – Height of stroke in z-buffer (implicit ordering of stroke)
6. $\underline{c}$ – Colour of stroke (an RGB triple)
7. r – Scale factor of superquadric base

In addition to these parameters each stroke also has two further attributes:

8. $(x, y)^T$ – the 2D location of the stroke's centre, relative to the video object's reference frame.
9. v – a continuous "visibility" value $v = [0, 1]$, where a value of $v \geq 0.5$ implies that the stroke is visible (i.e. should be painted)

The collection of attributes (1-9) represents the complete state of a stroke (written as a vector, $\underline{S}_t$) at some time offset t from that stroke's creation. Upon instantiation (at $t = 0$) we construct $\underline{S}_0$ as follows. Attributes 1-7 are determined from the footprint texture, in exactly the same manner as described in Chapter 3. Attribute 8 (location) is determined by the stroke's location on the reference frame, and is constant relative to this reference frame for the duration of the video. Attribute 9 (visibility) is set to 0.5.

At later instants ($t > 0$) we produce a vector of updated stroke attributes $\underline{S}_t$ in a similar manner. Attributes 1-6 are sampled from the footprint texture at time t. A value "ρ" for attribute 7 (scale of base) is determined as before, but modified to take into account any scaling applied to the reference frame. This is necessary, since if the reference frame is scaled up by a large factor, then the fixed, regular spacing of stroke could cause unpainted gaps to appear in the canvas. The scale component "s" of the affine transformation applied to the reference frame can be obtained using SVD (s is a product of the eigenvalues, computed for the transformation's linear component). The value r for attribute 7 is simply $r = \rho s$. Attribute 8 is a constant, and so simply copied from time S_{t-1}. Attribute 9 is set to 0 if the stroke's centre is outside the footprint, or 1 if it is inside.

The state of each stroke $\underline{S}_t$ is computed for all t, prior to the synthesis of any animation frames. To mitigate against stroke flicker, for example small, rapid changes in colour or visibility due to video noise, we smooth each stroke's attributes over time by independently low pass filtering each component of the signal $\underline{S}_t$. By default we use a Gaussian with a standard deviation of 3 frames, but allow the user to override this if

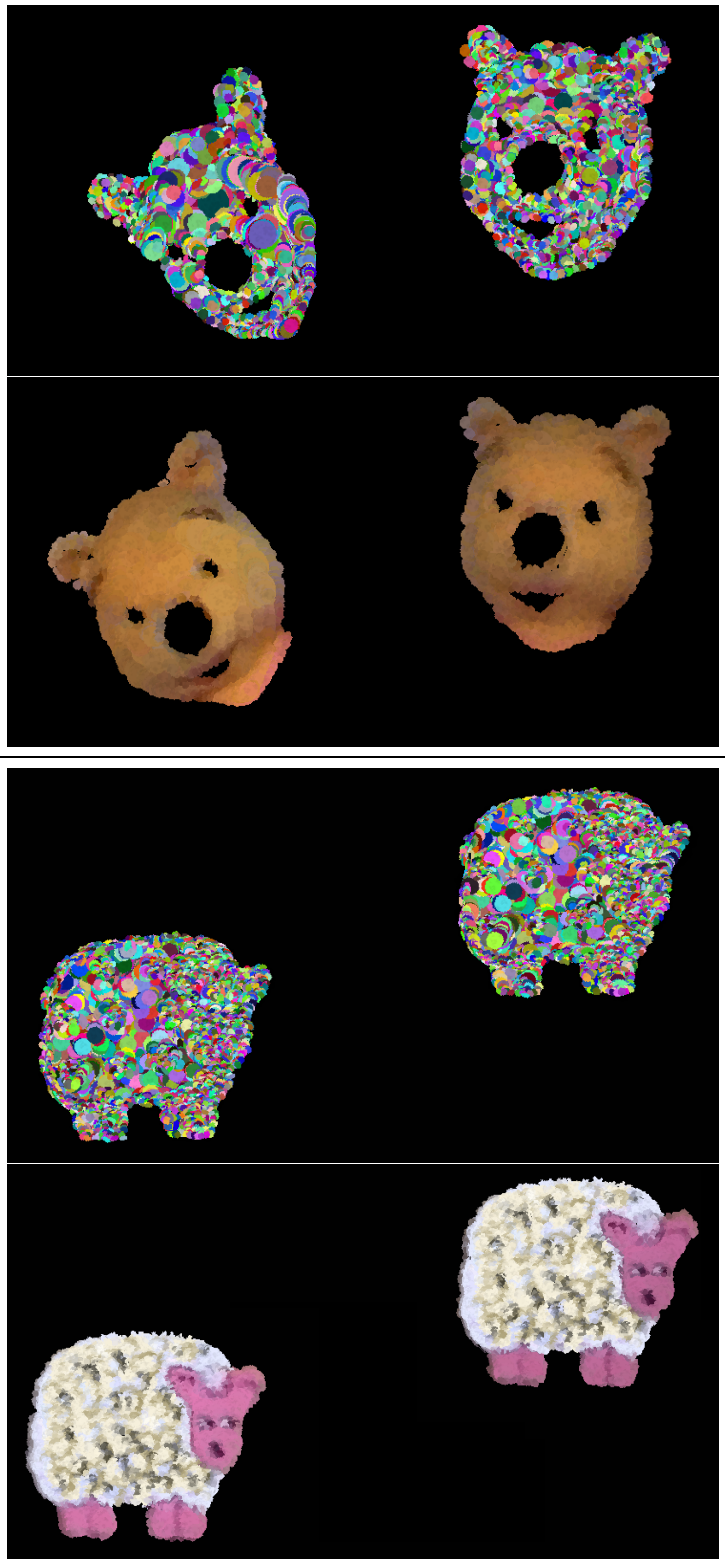

Figure 8-16 Illustrating the coherent painterly effect produced by extending our pointillist-style painting algorithm (Chapter 3) to video. We have used false colour to illustrate the coherence of individual strokes within the animation, however the reader is referred to Appendix C where complete animations demonstrating this effect have been included.

desired. For example, smaller scale filters could be used where stroke visual attributes must change at a faster rate — in these cases, rapid attribute changes are due to video content rather than video noise.

Implicit in our decision to perform temporal smoothing on stroke attributes is the assumption that video noise obeys the Central Limit Theorem. As the temporal window over which we integrate increases, signals will be reinforced and sporadic noise will be cancelled out. Similar assumptions regarding video noise have been by others; for example Capel [13] registers and averages video frames to achieve noise reduction in video surveillance images.

Rendering strokes with their smoothed attributes yields painterly animations exhibiting a uniquely high level of temporal coherence. Figure 8-16 gives examples of painted output generated by our system, including visualisations of strokes in false colour, where attribute 6 (colour $\underline{c}$) was fixed at a random, constant value for the duration of the sequence. The reader is referred to the source videos and corresponding painted animations available in Appendix C which demonstrate the temporal coherence of our algorithm. We compare our approach with the state of the art in video painting [103], in Section 8.6.

8.4.4 Rendering the Holding and Interior Lines

Having concluded our explanation of interior region rendering, we now describe the edge rendering process of the back-end. Recall the surface intersection operation by which we determine the Stroke Surfaces to be rendered a particular frame. The splines which result from this intersection form trajectories along which we paint long, flowing strokes, which are stylised according to a user selected procedural AR brush model. This produces attractive strokes which move with temporal coherence through the video sequence; a consequence of the smooth spatiotemporal nature of the Stroke Surfaces. We have used an implementation of Strassman's hairy brush model [150] (Section 2.2.1) to render our splines, producing thick painterly strokes to depict the exterior (holding) lines of objects (Figure 8-17, left).

Brush models (for example, Strassman's model) often invoke a stochastic process to simulate effects such as brush bristle texture. Without due care this non-determinism can cause swimming in the animation. Such models require only the *illusion* of randomness for aesthetics, and we have found that seeding the pseudo-random number generator with a hash of the unique ID number of a stroke surface is a convenient way of creating "reproducible" randomness for the duration of a Stroke Surface patch; a simple manifestation of a "noise box" [51].

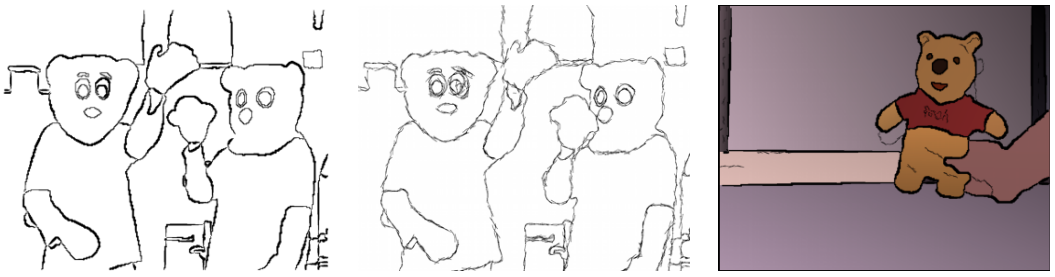

Figure 8-17 Examples of various line rendering styles available in our system. Left: Thick painted brush strokes generated by sweeping Strassman's [150] brush model over spline trajectories produce by intersecting a time plane with Stroke Surfaces in the *WAVE* sequence. Middle: A sketchy effect applied to the same video, generated by duplicating, shattering and jittering Stroke Surfaces prior to intersection. Each resulting splines has been rendered using a fine brush tip. Right: Combining the thick brush tip style on exterior cue (holding) lines (Stroke Surfaces with heterogeneous winged edge pointers) and a thin brush tip style on interior cue lines (Stroke Surfaces with homogeneous winged edge pointers).

The Stroke Surfaces whose intersection results in the spline strokes may be manipulated, prior to rendering, to produce a number of effects. In Section 8.4.1 we described a coherent wobbling effect that may be applied to both outline and interior Stroke Surface sets. However, effects specific to line rendering may also be applied, as we now explain.

Sketchy stroke placement

Artists often produce sketchy effects by compositing several light, inaccurate strokes on canvas which merge to approximate the boundary of the form they wish to represent. We may apply a similar, coherent, sketchy effect to video using a similar, two stage, technique applied to our Stroke Surfaces.

Stroke Surfaces are first shattered into many smaller individual Catmull-Rom [14] bi-cubic patches (Figure 8-18). The spatial and temporal intervals for this fragmentation are user parameters through which the appearance of the final animation may be influenced; small spatial intervals create many small sketchy strokes, and very large temporal intervals can create time lag effects. Each of these bi-cubic patches becomes a stroke in its own right when later intersected and rendered.

Second, each bi-cubic patch is subjected to a small "jitter" — a random affine transformation $\underline{\underline{M}}$ — to introduce small inaccuracies in the positioning of patches. Specifically:

$$\underline{\underline{M}} = \underline{\underline{T}}(\underline{\tau})\underline{\underline{T}}(-\underline{c})\underline{\underline{S}}(\sigma)\underline{\underline{R}}(\rho)\underline{\underline{T}}(\underline{c}) \tag{8.17}$$

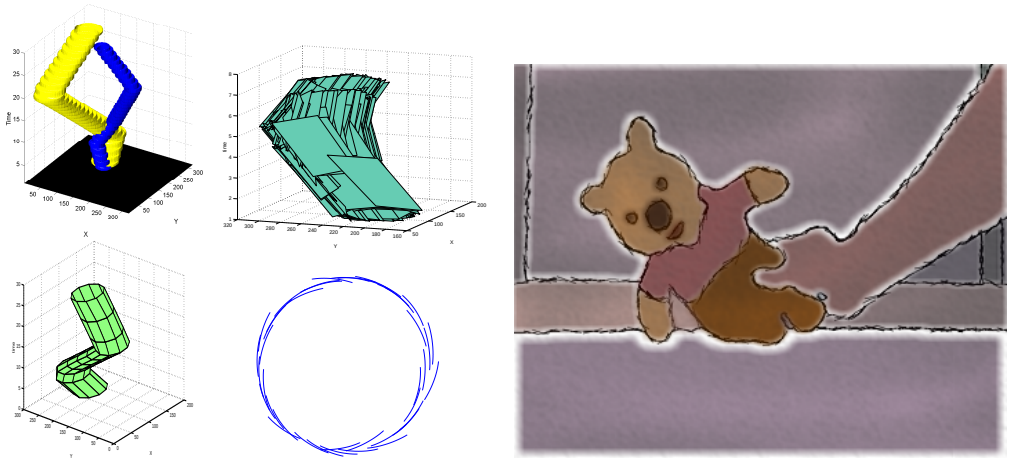

Figure 8-18 Left: A test sequence of bouncing spheres is segmented (top left). We visualise the Stroke Surface about one sphere, prior to (bottom left), and following (top right) surface shattering; we approximate surfaces here with piecewise linear patches for ease of visualisation. When shattered patches are intersected, coherent sketchy effect is formed (bottom right). Right: A still from a coherent animation produced from the *POOHBEAR* footage. Here the sketch effect on the holding line has been combined with a watercolour wash on the interior (`videos/pooh_watercolourwash`).

where:

$$\underline{\underline{T}}(\underline{x}) = \begin{bmatrix} \underline{\underline{I}} & \underline{x} \\ 0 & 1 \end{bmatrix}, \quad \underline{\underline{R}}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \underline{\underline{S}}(s) = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8.18)$$

$\underline{c}$ is the centroid of the patch, and $\underline{\tau}, \sigma, \rho$ are normal variates which control the jitter, and are typically of low magnitude (the user is given control over their limits). Figure 8-18 gives a visualisation of the resulting, perturbed patches.

The bi-cubic patches are intersected with a time plane ($z = t$) as before, and the resulting splines rendered with a fine tipped brush to yield a sketchy effect. Each stroke is guaranteed to be coherent over its temporal extent, since it results from intersection with a smooth bi-cubic patch embedded in the spatiotemporal volume. Finally, the density of sketching may be increased by further duplicating the Stroke Surfaces prior to shattering and jittering in the manner described.

It is a matter of artistic taste whether sketch lines should be re-sketched at a constant rate for the duration of the video, or whether they should only be re-sketched as the object moves. Currently our framework subscribes to the former point of view, but could be easily modified to produce the latter. By substituting the normal variates τ, σ, ρ for values picked from a noise-box parameterised by spatiotemporal location,

strokes would appear to be "re-sketched" only when the location of a Stroke Surface changes i.e. during object motion.

Other line styles

We do not concern ourselves unduly with the stylisation of individual strokes. We defer the problem of artistic media emulation to the literature, being concerned primarily with stroke placement rather than stroke rendering. However we have found that interesting results can be obtained by varying line weight according to some transfer function. Some successful transfer functions we have experimented with vary the line weight in proportion to:

- the maximum of the speed of the two objects it bounds.
- the maximum area of the two objects.
- the intensity gradient between the two objects.

The latter suggestion helps to mitigate against artifacts produced when a feature has been over-segmented, leading to, say, a face broken into two features divided by a thick black line. If there is little evidence for an edge in the image at that boundary, then the stroke may be omitted.

Note that interior cue lines generated in Section 8.3.4 are naturally accommodated into this framework, since they are simply further examples of Stroke Surfaces. Exterior and interior lines can be easily discriminated if desired, by examination of the Stroke Surfaces' winged edge structure, and may be rendered in differential styles (see Figure 8-17).

8.5 Interactive Correction

Feature sub-volumes may become over-segmented in the video volume, producing two distinct graphs of video objects where one would suffice. This situation arises when the initial segmentation algorithm consistently over-segments a feature over several video frames, often because of local illumination variance, to the extent that the region association process does not recombine the over-segmented regions. Since Computer Vision is unable to provide a general solution to the segmentation problem, such errors are unavoidable in our system.

We therefore provide an interactive facility for the user to correct the system by merging video objects as required. This operation takes place directly after the region

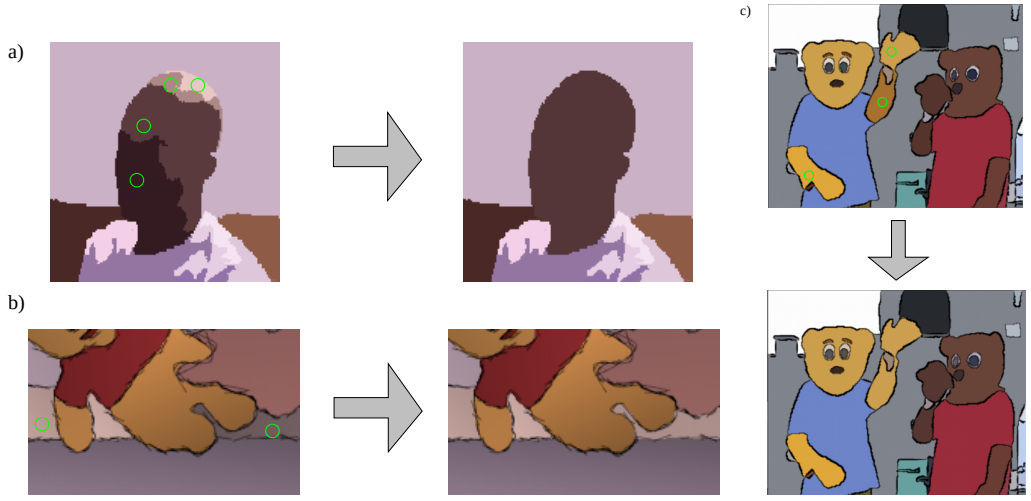

Figure 8-19 Users may create manual associations between objects to tune the segmentation or composition. (a) User creates a physical link between over-segmented objects, a single object replaces four. (b,c) User creates a semantic link between two objects. The objects remain distinct, but associations are created in the object association graph; during rendering attributes are blended between regions to maintain temporal coherence.

association and filtering process. Objects are linked by point-and-click mouse operations in a single frame, and those changes propagated naturally through to all other frames in which the object exists (since objects are spatiotemporal in their nature). We tend to bias the parameters to the EDISON [19] segmentation algorithm (see front end, Section 8.2.2) slightly toward over-segmentation, since over-segmentation is much more easily resolved via merging objects, than under-segmentation (which introduces the complicated problem of specifying a spatiotemporal surface along which to split video objects). The user may form two types of corrective link:

1. **Soft Link:**

   The animator semantically links two objects by creating edges in the object association graph. Objects remain as two distinct volumes in our representation, but the graph is modified so that any smoothing of graphical attributes (such as region colour) occurs over all linked objects (see Figure 8-19b,c). This type of link is often used to encode additional, semantic knowledge of the scene (for example, the continuity of the skirting board in Figure 8-19b).

2. **Hard Link:**

   The animator physically links two adjacent video objects by merging their voxel volumes. The objects to be merged are deleted from the representation and replaced by a single object which is the union of the linked objects (see Figure 8-19a). This type of link is often used to correct over-segmentation due to artifacts such as shadow, or noise, and is preferable in this respect to "soft" linking, since

the merged volume will undergo subsequent coarse smoothing as a single video object.

Interactive correction can also be used to control focus in the composition, for example by coarsening the scale of chosen region in the video; this is similar to the process driven automatically by our salience measure in the painterly rendering of Chapter 3. In the future we would also like to drive this video process automatically using a perceptual model.

Extensive correction of a sequence is rarely required, and correction itself takes little time since one click can associate spatiotemporal objects over multiple frames. Similarly, we allow the user to selectively chose rendering parameters for specific objects — for example, in Figure 8-15 (top right) specifying a region should be photorealistically matted rather than animated using the global specified settings. Again, such interaction requires only a couple of mouse clicks to modify parameters for video objects, and is a level of interaction which we wish to retain in our Video Paintbox to allow creative direction of the artistic process by the animator.

8.6 Comparison with the State of the Art

The majority of artistic styles that may be synthesised by the Video Paintbox have no parallel in the existing video driven AR literature; these techniques address only the single, specific problem of producing temporally coherent painterly animations. This improved diversity of style is one of the principal contributions of the Video Paintbox. However, we also wish to demonstrate the contribution made due to the improvements in temporal coherence available through our framework.

We begin by recapitulating and expanding on our description of existing video-driven painterly algorithms, which make use of either inter-frame differencing (Section 8.6.1) or inter-frame optical flow (Section 8.6.2) to improve temporal coherence. We then perform a comparison (Section 8.6.3) between these and the technique we developed in Section 8.4.3, which applied the coherent stroke based rendering component of our proposed framework to produce painterly animations from video.

8.6.1 RGB Differencing

Hertzmann and Perlin [75] proposed a video-driven painterly technique which computes the RGB difference between adjacent frames of video to locate regions of high motion magnitude. Their contribution was to paint the first frame of video using Hertzmann's static technique [71], and then update, or "paint over", regions with high motion magnitude in the next frame. Recall that animations swim if each frame is

repainted individually. Hertzmann and Perlin thus attempt to localise swimming to only the "moving" regions of the animation.

There is one clear advantage that this method holds over optical flow driven techniques (Section 8.6.2). Real-time frame rates (reportedly up to 6 frames per second) are possible on standard PC hardware, facilitating interactive applications. Without specialised hardware, reliable optical flow estimation algorithms are still too slow to facilitate real-time rendering. The same disadvantage applies to our artistic rendering subsystem, primarily because of its high level temporal nature (examining both "past" and "future" events in the video volume, rather than rendering on a per frame basis).

However there are three disadvantages to the frame differencing methodology:

1. First, although expedient to process, RGB differencing is a very poor method of determining the motion magnitude of imaged objects in a scene. The approach fails under varying lighting conditions, and assumes neighbouring objects are highly contrasting. Slow moving objects are hard to detect, since the threshold used to exclude small inter-frame differences caused by video noise, also exclude low speed motions.

2. Second, there is no internal consistency of motion within an object, since motion estimates are made on a per pixel rather than a per region basis — no higher level spatial grouping is used to assist motion estimation (in contrast to the higher level spatial analysis of our spatiotemporal method). As a result, moving flat-textured objects tend to be detected as collections of moving edges with static interiors. In Section 8.6.2 we will see that similar objections hold for optical flow.

3. Third, there is the problem discussed previously regarding the per frame sequential nature of processing. Errors in motion estimation accumulate and propagate through subsequent frames. For example, a region of the video containing an arm may move, but be misclassified as stationary by the differencing operation. This leaves a "phantom image" of the arm at one location in the painting which persists throughout the animation until something moves over the phantom arm's location causing "paint over" to occur again. The reduction in flicker is typically at the cost of an inaccurate and messy painting.

Temporal coherence of painterly animations is improved using Hertzmann and Perlin's method, but only within static regions. Temporal coherence is not attainable within moving regions, since these regions are repainted with novel strokes, laid down in a random order. Unlike optical flow based methods [96, 103] it is not possible to translate strokes with the video content, because frame differencing produces estimates only for motion magnitude, rather than both magnitude and direction. Recognising

the improved accuracy (but non-interactive frames rates) of optical flow, Hertzmann and Perlin describe how to adapt their system to use optical flow at the end of their paper. This enables their system to be adapted for offline video processing.

8.6.2 Optical Flow

Litwinowicz [103] proposed the first optical flow driven painting algorithm for video[4]. The algorithm paints short, linear brush strokes to produce a painterly effect on the first frame of video. Brush strokes are then translated from frame to frame to produce subsequent frames of animation. The vector field for the translation of strokes is determined by an inter-frame motion field, computed using an optical flow technique [7]. As discussed in Chapter 5, the per frame sequential processing of video causes the accumulation of large positional errors over time. This error is manifested either by a rapid flickering, or by the motion and painted content of the animation becoming non-representative of the underlying video.

Translation can cause dense bunching of strokes, or the density of strokes to thin causing holes to appear within the canvas (Figure 8-20, middle). Thus after the stroke translation stage there is a further *stroke density regulation* stage. Strokes in areas deemed too dense are culled at random, until the density falls below a preset "acceptable" threshold. New strokes are inserted into sparsely populated regions until the stroke density becomes acceptable. The order in which strokes are painted is preserved between frames to mitigate against flickering. Any new strokes inserted into the sequence are "mixed in" with old strokes by inserting them at random into this ordering.

The principal advantage of the optical flow methodology is that the strokes move with the content of the underlying video sequence — rather than moving regions being repainted afresh, as with frame differencing. This yields a substantial improvement in temporal coherence. However, there are also a number of disadvantages of this methodology:

1. First, the non-determinism of the stroke insertion strategy (due stroke density regulation) causes a significant level of flickering (Figure 8-20).

2. Second, the temporal coherence of the animation is only as good as the underlying optical flow algorithm and, in general, optical flow algorithms perform poorly on scenes exhibiting flat textures, occlusion, or lighting changes. No accurate, general, optical flow algorithm exists since the task demands solution of an underconstrained "correspondence problem". The resulting errors accumulate over

[4]Kovacs and Sziranyi [96] published a similar optical flow driven algorithm some years later in the Computer Vision literature (see Section 2.5.2 for a comparison of this and Litwinowicz's algorithm).

Figure 8-20 Identical frames four seconds into the *BOUNCE* sequence rendered with our technique (right) and SoA (left, middle). The errors that have built up in the cumulative optical flow estimate at this stage cause large scale distortion and flickering in the image. We have omitted the stroke density regulation stage (SoA-) in the left-hand image; this improves temporal coherence. However large holes are created in the canvas (red circle), and elsewhere tight bunching of strokes causes the scene to tend back toward photorealism. The reader is referred to the videos in Appendix C where the improved aesthetics and temporal coherence of our method are clearly demonstrated (see `videos/bounce_painterly_SoA`, `videos/bounce_painterly_SoA-` and `videos/bounce_painterly_ourmethod` for left, middle and right respectively).

time, as a consequence of the per frame sequential processing model employed. After only a short while (around 1 or 2 seconds depending on video content) this causes a gross disparity between source video and animation in terms of both content and motion.

3. Third, as with RGB differencing the motion of each brush stroke is estimated individually. No higher level spatial grouping of strokes into semantic regions is performed which could be exploited to improve temporal coherence (for example, to ensure consistency of stroke motion within a single object).

8.6.3 Comparison Methodology

The intended application of our Video Paintbox is as a post-production video tool; real-time frame rates are not required. Thus for our application, the more accurate optical flow methodology (rather than RGB differencing) is regarded as the current state of the art. We have implemented Litwinowicz's optical flow driven algorithm [103] (hereafter referred to as "state of the art", or "SoA") and now draw comparisons between the temporal coherence of this, and our proposed, painterly technique[5].

Recall our two criteria for a temporally coherent animation:

1. The locations and visual attributes of rendering elements should not vary rapidly

[5]For reference, the full paper detailing Litwinowicz's method is included in Appendix C.

(flicker) over time. Assessment of this criterion requires measuring the rate of change of these properties in the *rendered* image sequence.

2. The motion of strokes should agree with motion of content with the source video. Assessment of this criterion requires measurement, and comparison, of the motion fields of both *source* and *rendered* image sequences.

The second criterion is problematic since it demands an error-free means to determine the magnitude and direction of motion in the *source* video — if such a technique was known, we would have already built it into our Video Paintbox! Thus we require an externally supplied ground-truth motion field to evaluate this criterion.

We now describe two experiments which test each of these criteria respectively. We document the results of these experiments separately in Section 8.6.4, and draw conclusions from the results.

Experiment 1: Measurement of Flicker in the Animation

Our first experiment quantifies the level of stroke flicker in the rendered animation. Stroke flicker is a contributor to temporal incoherence, and is manifested as:

1. Rapid fluctuations in stroke location

2. Rapid fluctuations in stroke visual attributes

In both SoA and our algorithm, a stroke's location is defined by a 2D vector; we write $\underline{L}_t = (x, y)^T$ to specify the normalised coordinates (x, y) of a brush stroke's centre within the frame. Likewise, a stroke's visual attributes may be defined as a point in a high-dimensional space. Each stroke in our painterly method has seven visual attributes (see Chapter 3), in SoA there are two: colour and orientation. For the purposes of comparison we specify visual attributes as points in a 4D space $\underline{A}_t = (r, g, b, \theta)^T$ where the triple $(r, g, b \in [0, 1])$ specifies stroke colour and $\theta \in [0, \pi]$ specifies a rotation from the vertical (strokes are symmetrical about their principal axes) in both algorithms. We concatenate both location and visual parameter vectors to form a single vector representing the stroke state: $\underline{V}_t = [\underline{L}_t^T \;\; \underline{A}_t^T]^T$. We normalise the space of $\underline{V}_t$ so that distance $|(0, 0, 0, 0, 0, 0)^T - (1, 1, 1, 1, 1, \pi)^T|$ is unity; this normalises our flicker measure to range [0,1].

For a single stroke, the absolute rate of fluctuation of $\underline{V}_t$ at time t may be computed using a finite difference scheme:

$$R(t) = \big||\underline{V}_t - \underline{V}_{t-1}| - |\underline{V}_{t+1} - \underline{V}_t|\big| \tag{8.19}$$

This measure quantifies the absolute rate of change of visual state for a single, existing stroke. For strokes that are created or destroyed due to the stroke density regulation step of SoA, we set $R(t) = 1$. Note that this density equalisation step does not form part of our painting algorithm.

We write $\hat{R}(t)$ as the absolute rate of change of visual state averaged over all strokes present in frame t — hence quantifying the level of flicker present in frame t. Our objective measure of the average level of stroke flicker over all n frames of the animation is the mean of $\hat{R}(t)$ over all t.

Additionally, we may obtain a qualitative estimate of the level of flicker by visual inspection of the animation.

Experiment 2: Measuring the similarity between motion in the source and rendered sequences

The smaller the error between the motion vector fields of the source video and target animation, the greater the temporal coherence of the animation. We therefore produce a source image sequence, for which we know the ground truth motion, and render that image sequence using the algorithms under evaluation. The motion vector field for the target animation is obtained from the motion vectors of the brush stroke centres. We then perform a comparison of the ground truth and brush stroke motion vector fields.

A number of synthetic source test sequences are used for this experiment; examples of both flat and textured image regions, which are independently subjected to translation, rotation and scaling transformations. The ground truth motion vector field at time t, which we write as $\underline{S}_t(i,j) = (r(t), \theta(t))$, is compared with the motion field of the target animation at the identical instant $\underline{T}_t(i,j) = (r(t), \theta(t))$. These motion fields are in polar form ($r(.)$ is the magnitude of displacement, $\theta(.)$ is the direction of displacement — both $r(.)$ and $\theta(.)$ are normalised to range between zero and unity). The mean squared error (MSE) between the two fields is computed to obtain a measure of the similarity between the two motion fields at time t; we write this measure as $C(t)$:

$$C(t) = \frac{1}{xy} \sum_{i=1}^{x} \sum_{j=1}^{y} |\underline{S}_t(i,j) - \underline{T}_t(i,j)|^2 \tag{8.20}$$

where x and y are the frame width and height respectively. Our objective measure $\mathcal{C}$ of the average level of motion dissimilarity for the animation is the mean of $C(t)$ over all time. As with experiment one, this measure can be verified by qualitative visual inspection of the source and animated image sequences. In this manner qualitative

	SoA	SoA-	Our method
SPHERES	0.62	0.45	0.05
BOUNCE	0.79	0.65	0.11
SHEEP	0.78	0.61	0.13

Table 8.1 Table summarising the "flicker level" (see experiment 1) present in an animation produced by each of three algorithms (horizontal), over each of three source video sequences (vertical). Flicker level ranges from zero (no flicker) to unity (severe flicker).

estimates may be made for real source video sequences, for which there is no available ground truth.

8.6.4 Results and Discussion

Both experiments were conducted to compare the temporal coherence of animations produced by both our algorithm and SoA. We now discuss the results.

Experiment 1: Flicker in synthesised animation

We applied both algorithms to one synthetic sequence (*SPHERES*) and two real sequences. Of these two real sequences, one (*SHEEP*) contained regions of predominantly flat texture and the other (*BOUNCE*) was a natural scene containing moderately complex texture. We also tested SoA with, and without, the stroke density regulation step (we refer to SoA without this regulation step as SoA-). Table 8.1 summarises the levels of flicker measured within the resulting animations. Note that this measure of flicker quantifies the rate of change of stroke properties in the animation. By this definition, all animations should exhibit some degree of "flicker". However for a single video, this measure will be comparatively large in the case of frequent, rapid changes in stroke attribute and position (flicker), and lower in cases of smooth, less sporadic motion caused by movement of content in the video.

Our results indicate that SoA exhibits about half an order of magnitude greater stroke flicker than our method, on both real and synthetic video sources. All algorithms appear to perform consistently regardless of the level of texture in the video. This can be explained, since although optical flow tends to produce worse motion estimates in cases of flat texture, this experiment measures only flicker in the animation, not accuracy of motion (this is addressed by experiment 2). Omission of the stroke density regulation stage (SoA-) does reduce the level of flicker, but damages the aesthetics of the animation as "holes" appear upon the canvas.

We conclude that our painterly approach produces animations exhibiting significantly less flickering than the state of the art. Visual inspection of the resulting painterly

animations verifies these results (see Appendix C). The reduction in flicker can be explained by:

1. Our use of a robust motion estimation technique, which takes advantage of spatial grouping (segmentation) to move all strokes attached to objects in a similar way — rather than moving each stroke in isolation of all others. Error in motion estimation is spread over the entire region, making the approach more tolerant to noise that per stroke optical flow techniques.

2. The absence of a stochastically driven "stroke insertion" step in our algorithm. Unlike SoA, our approach does not require a "stroke insertion stage" since there are potentially an infinite number of strokes rigidly attached to the planar reference frame that moves with video objects. The state of strokes on the plane persists even whilst a stroke is not visible in a frame of the animation. This is not true with SoA — when strokes disappear, for example due to occlusion, they are deleted (as strokes bunch together), and then reinitialised after the occlusion (triggered by strokes spreading too thinly) without taking into account their previous state. The strategy of inserting new strokes at a random order was chosen in SoA to prevent regular artifacts from appearing in the video sequence during stroke insertion. We observe that similar use of non-determinism has been used elsewhere in Computer Graphics (for example the noise introduced by Cook's [30] distributed ray tracing), and also in many static AR algorithms to mask the regular, machine origins of the artwork produced. However non-determinism must be used sparingly in the context of AR animations, since it introduces flicker. Since our method does not require a stroke insertion step, stochastically driven or otherwise, the temporal coherence of the resulting animations is greatly improved.

3. In our system the visual attributes of strokes, for example orientation and colour, are sampled from each frame in the video, but smoothed over time under the assumption that noise obeys the Central Limit Theorem. In SoA these attributes are sampled from the video frame, and directly applied to strokes without taking into account the state of those attributes in previous or future frames. Point-sampling is highly susceptible to video noise (especially when sampling from derivative fields — such intensity gradient to obtain orientation). With SoA the sporadic fluctuations in sampled values result in the unsmoothed sporadic fluctuation of visual stroke attributes. This is not so with our approach.

4. The thresholded Sobel edges used to clip strokes in SoA are prone to scintillation over time. This in turn causes the lengths of strokes to fluctuate causing increased flickering. This clipping mechanism is not a component of our approach, and so introduces no such difficulty. Instead, we smoothly vary the continuous "visibility" attribute of strokes over time.

Experiment 2: Similarity between motion fields of source and animation

Experiment two measures the degree of similarity between the motion fields of both source and rendered image sequences. We have used only synthetic source sequences for this experiment, since we require an accurate ground truth motion estimate for the input video. Both our algorithm and SoA were modified to output a vector field corresponding to stroke motion in the animation, rather than an animation itself. We tested both flat (*SMILE*) and textured (*SPICES*) image regions under translation, rotation and scaling transformation, comparing the motion fields in both source and rendered footage.

Figures 8-21 to 8-26 present the results of this experiment. In all figures the top row shows the original and transformed images (i.e. the first and second frames of the video sequence). The second row shows the ground truth motion field, and a colour visualisation of motion magnitude. The third row shows the dense motion field used to move strokes in our system. The fourth row shows the optical flow generated motion field used by SoA. In all cases, we observe that the motion field generated by our method closely matches the ground truth. The MSE (equation 8.20) between our field and the ground truth is approximately 0.05 for all transformation classes. The MSE between the optical flow derived stroke motion field and the ground truth varies between 0.5 and 0.7; an order of magnitude less accurate.

We draw attention to the false negative readings returned by optical flow for flatly textured regions (regions of near constant intensity) within the images. By contrast optical flow estimates around most of the edges appear to reasonably accurate. The results of this error can be seen most clearly in the resulting painterly animations, where brush strokes in the centres of flat regions remain static, but strokes around the edges move. This conveys contradictory motion cues to the viewer; strokes around the edges of an object appear to move in the opposite direction relative to those in the middle of the object. By contrast the homography based motion field generated by our method is computed over the entire image region. This produces an accurate motion estimate even within flatly textured regions, and ensures that stroke motion is consistent within individual regions in the video.

In summary, the higher spatial level of processing performed by our technique performs motion estimation on a per object, rather than per pixel basis. Errors in motion estimation are thus distributed over the entire object, rather than individual strokes. Similarly the higher level of temporal processing performed by our technique smooths stroke attributes over time. Measurements of stroke attributes in adjacent frames combine to reinforce each other and cancel out noise. Robustness to such errors, produced

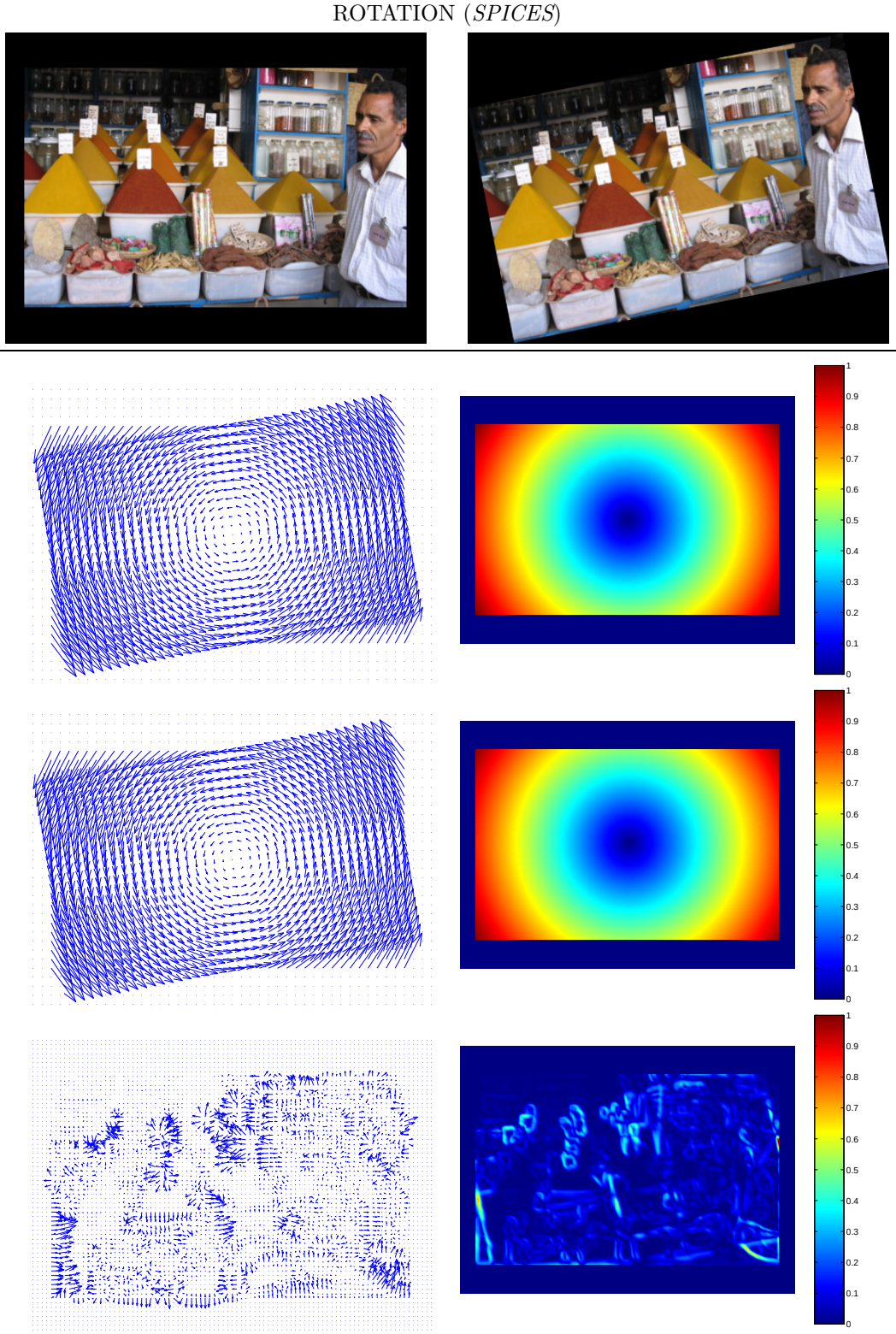

Figure 8-21 Rotation of a highly textured example *SPICES* (12° anticlockwise). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

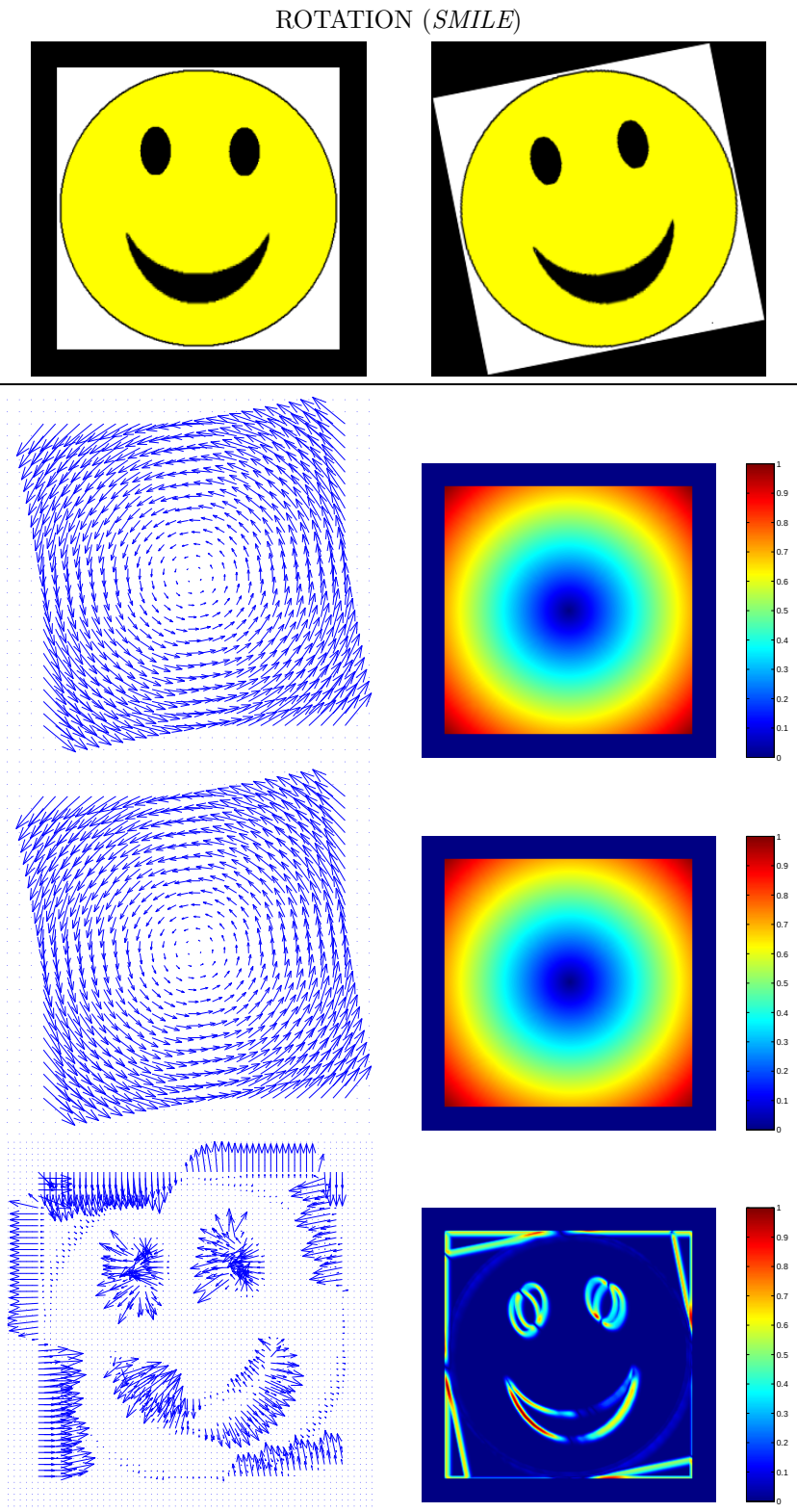

Figure 8-22 Rotation of an example with predominantly flat texture *SMILE* (12° anti-clockwise). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

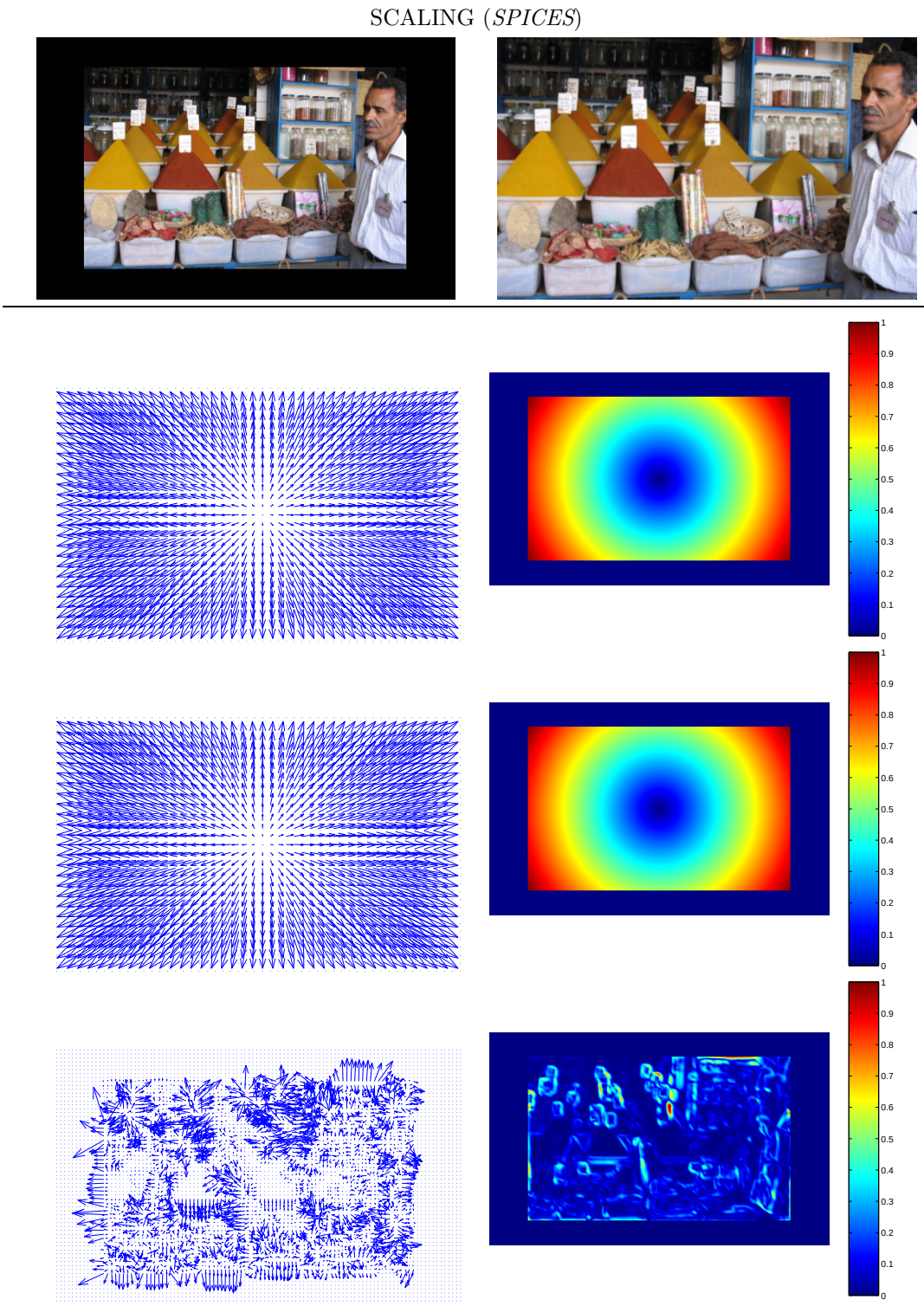

Figure 8-23 Uniform scaling of a highly textured example *SPICES* (scale factor 1.3). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

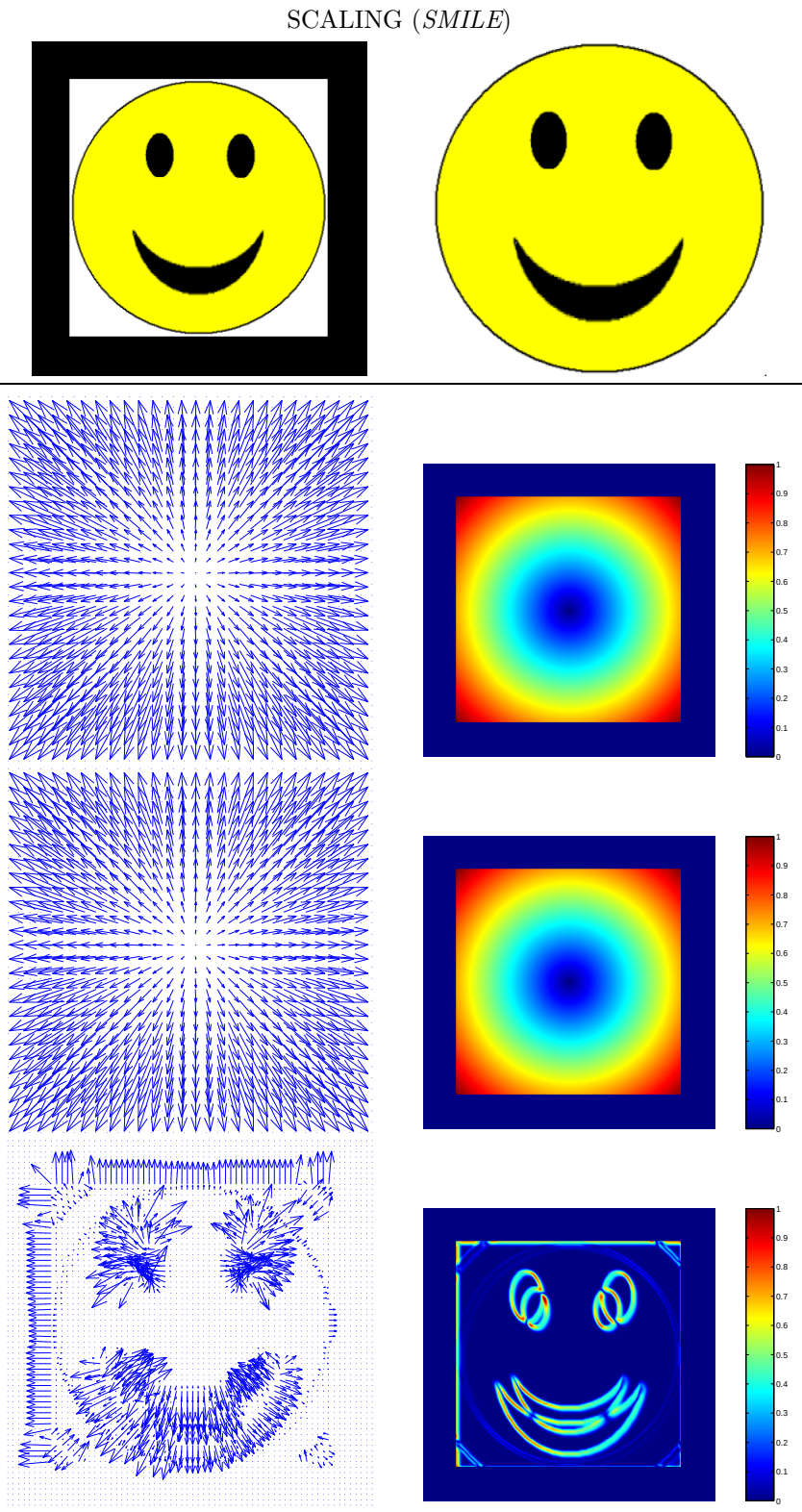

Figure 8-24 Uniform scaling of an example with predominantly flat texture *SMILE* (scale factor 1.3). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

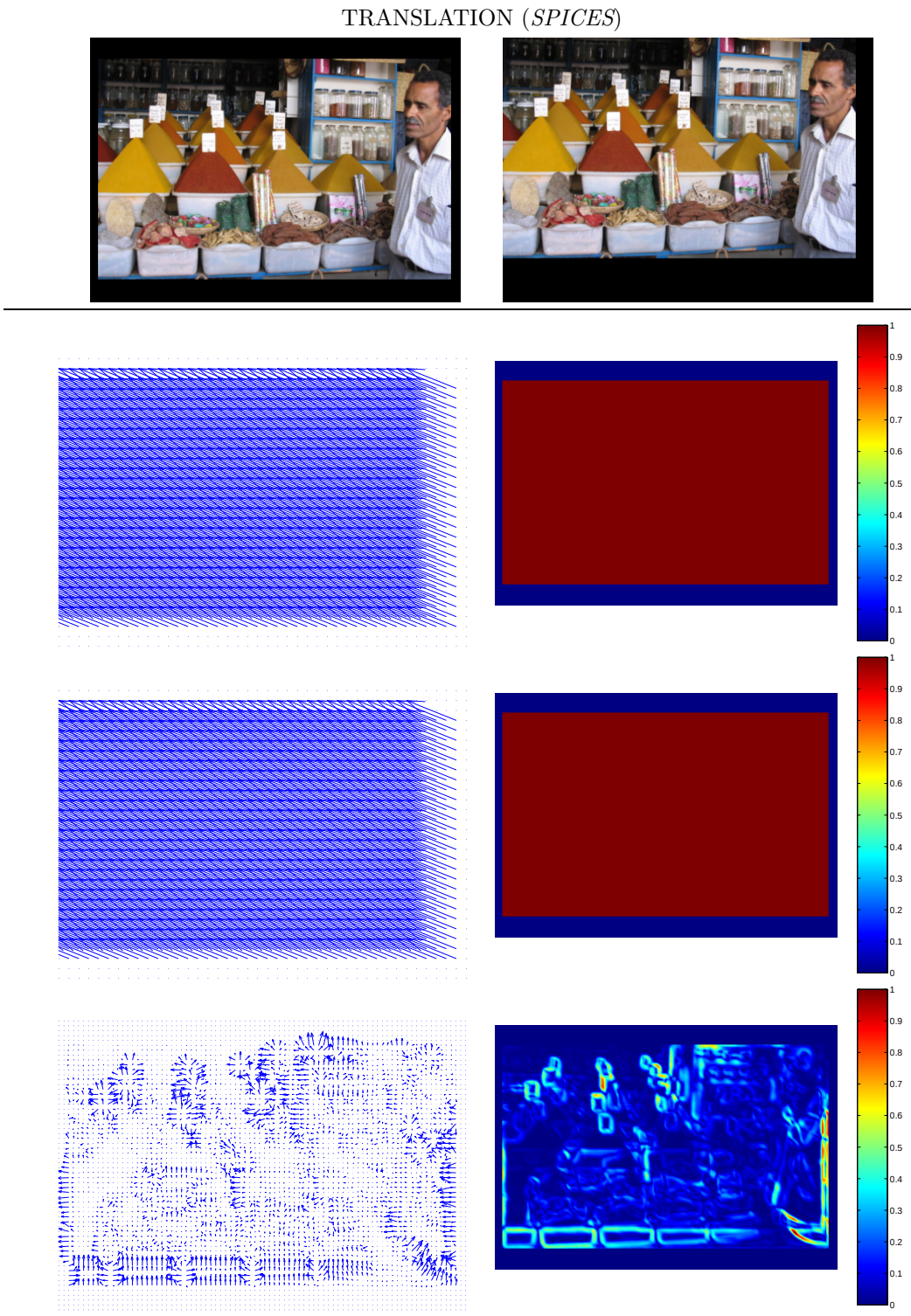

Figure 8-25 Translation of a highly textured example *SPICES* (shift by $(-20, -50)^T$). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

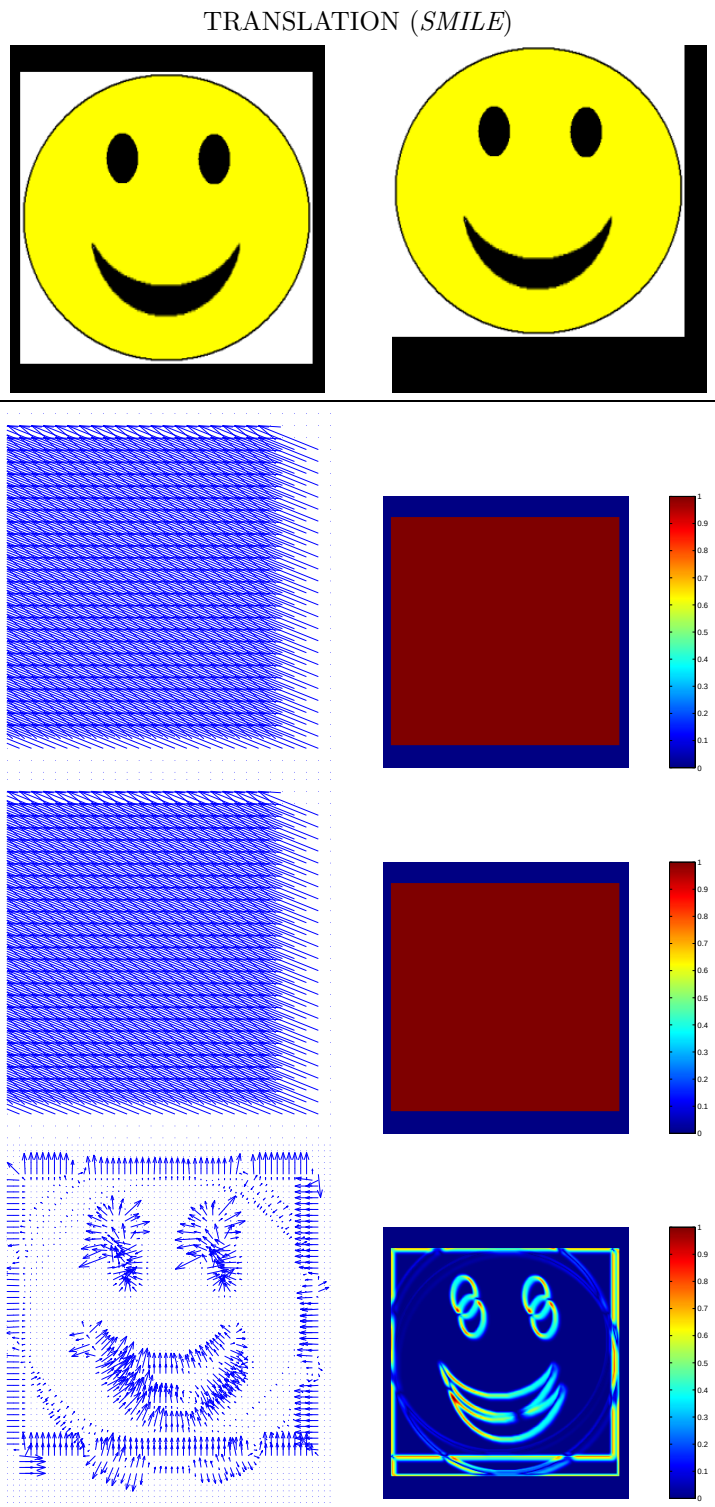

Figure 8-26 Translation of an example with predominantly flat texture *SMILE* (shift by $(-20, -50)^T$). First row: source and transformed image. Second row: ground truth motion vector field (left) and colour temperature visualisation of ground truth vector magnitude (right). Third row: Estimated vector field using our method, and visualisation of vector magnitude. Fourth: Estimated vector field using optical flow, and visualisation of vector magnitude.

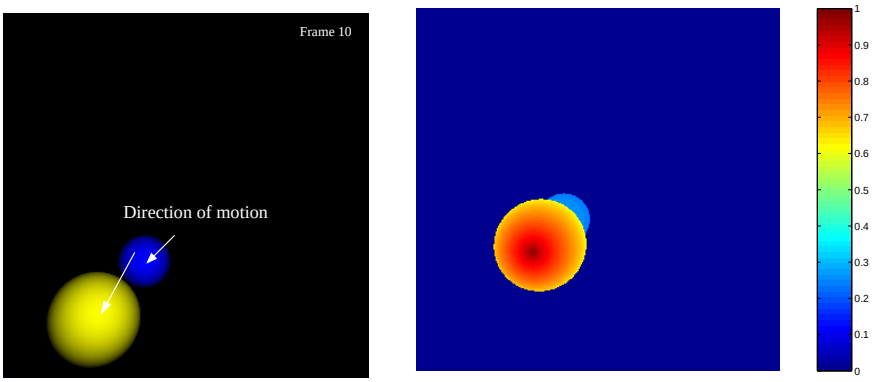

Figure 8-27 A synthetic test sequence (*SPHERES*, `videos/spheres_source`) consisting of two front lit, Lambertian shaded spheres (left). The spheres are of identical size but at differing scene depths (relative proximity to camera is illustrated by the depth map on the right). The spheres translate in planes parallel to the image plane, inter-occluding and casting shadows; a ground truth motion vector field is also projected to the camera plane for reference. We use this sequence to illustrate the limitations of the homography based motion model (see text).

for example by camera noise, is especially important with modern day equipment such as DV cameras whose compression algorithms often create artifacts in stored images.

Limitations of our technique

As with optical flow, our painterly rendering technique is governed by a number of assumptions. These are principally:

1. that the video to be painted is segmentable (some video, for example crowd scenes or water, are difficult to segment)

2. that the change of viewpoint of an object over time is well modelled by a homography (plane to plane transformation) between imaged regions of that object

Violation of assumption (1) will prevent a video being rendered by the artistic rendering subsystem. Violation of assumption (2) is non-fatal, since the video may be still processed into a painterly form, but one that exhibits a lesser level of temporal coherence. However, as we now show in a final experiment, this reduced level of temporal coherence can still represent a significant improvement over the coherence afforded by optical flow based painting techniques.

The synthetic *SPHERES* sequence represents a situation where assumption (2) is violated. The sequence contains two diffuse shaded, spheres (Figure 8-27) which undergo translation in planes parallel to the image plane. We rendered this sequence using both our method and SoA to obtain two painterly animations (see `spheres_painterly_SoA` and `spheres_painterly_ourmethod` in Appendix C). We obtained motion fields for

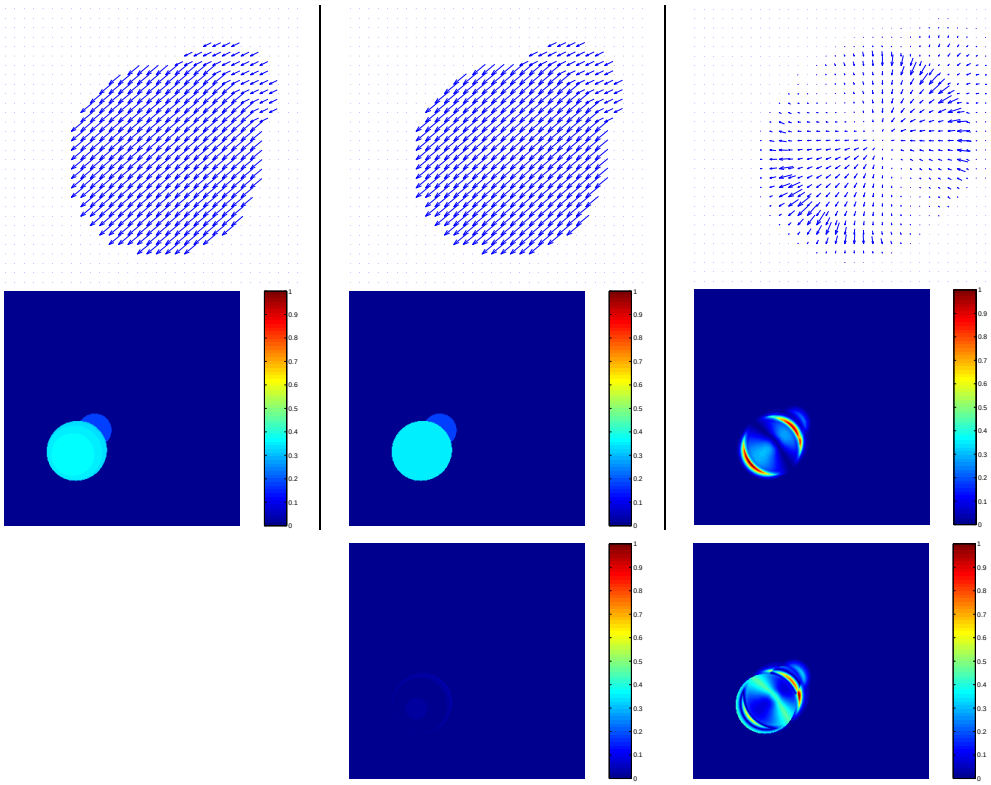

Figure 8-28 Top: Motion vector fields between frames one and two of *SPHERES*. From left to right; ground truth field (true motion of 3D objects projected to the camera plane); field for stroke motion determined via our method; field determined via optical flow [7]. Middle: Visualisation of motion magnitude, ordered as (top) and using identical colour temperature scales. Bottom: Difference between ground truth motion magnitude and that of our (left) and optical flow (right) using identical colour temperature scales for the purposes of comparison. Observe optical flow performs poorly in interior regions, whereas our method does not. Our method exhibits around half an order of magnitude less error than optical flow. We have normalised (left) in Figure 8-29 to illustrate the distribution of estimation error over the region.

brush strokes in the two animations, and modified our ray tracer to output a ground truth motion field of the spheres, projected to the 2D image plane.

The resulting source and rendered motion fields, corresponding to the imaged spheres, are shown in Figure 8-28 (first column). Observe that in the ground truth motion field, distant points exhibit lesser motion magnitudes due to parallax.

Optical flow performs poorly when recovering this motion field (Figure 8-28, second column). The computed fields suggest that the flat textured interiors of the spheres do not move at all, whilst the edges perpendicular to the direction of motion are deemed to have moved. Edges parallel to the direction of movement are not deemed to have moved, since the spatially local nature of the optical flow algorithm can not determine such motion due to the "aperture problem" [145].

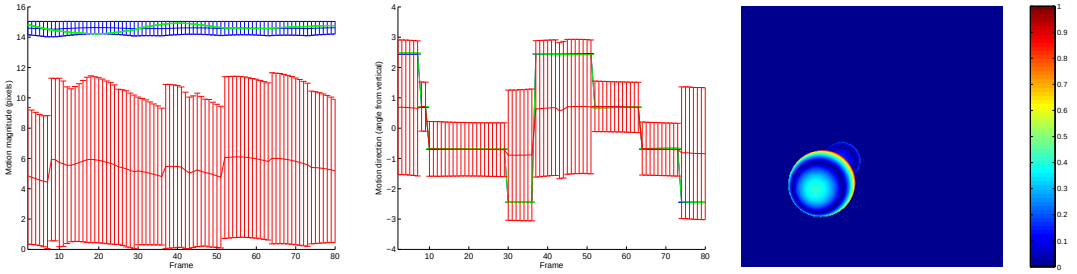

Figure 8-29 Plots illustrating errors in both our, and optical flow's, motion estimate for the yellow sphere. The left graph shows the mean value of motion vector magnitude within the yellow sphere region, at each time instant. The error bars show one standard deviation. Blue is the ground truth, green is our motion estimate, red is the optical flow estimate. Observe that our method produces a consistent, uniform vector field estimate that closely matches the mean of the ground truth. The optical flow motion estimate is inconsistent over the region, as evidenced by high standard deviation. Similar observations apply to the middle plot, which shows mean direction of motion over time. The right-hand figure is a normalised absolute difference map between ground truth motion magnitude, and motion magnitude estimated by our system. Our method produces a single motion estimate for each region, distribution estimation error over that region. In the case of the yellow sphere, the result is a motion estimate corresponding to around middle distance on the sphere; approximately the mean of all motion vectors.

Our method interprets the imaged spheres as translating planar discs, with a radial shading pattern upon their surfaces. The resulting motion field is thus a uniform set of vectors specifying a translation approximately equal to the mean of the ground truth vectors (Figure 8-28, third column). Although this uniform vector field is erroneous, the residual error due to our method is much lower than that of optical flow (see Figure 8-28). The internal consistency of motion vectors generated by our method is also much closer to that of the ground truth, whereas there is very little internal consistency within the sphere according to optical flow (observe the error bars depicting standard deviation in Figure 8-29, and Figure 8-28 top right).

8.7 Integration with the Motion Emphasis Subsystems

Our video shading subsystem meets the aims of the Video Paintbox's second sub-goal, generating temporally coherent artistic animations from video. We may combine this subsystem with the earlier motion emphasis work of Chapters 6 and 7, to meet our original aim of producing full, cartoon-like animations from video. Figure 8-30 contains a schematic of the entire Video Paintbox, illustrating how we combine the shading and motion emphasis work. We now describe the complete Video Paintbox rendering process:

A camera motion compensated version of the video clip is first generated (Section 6.3.1).

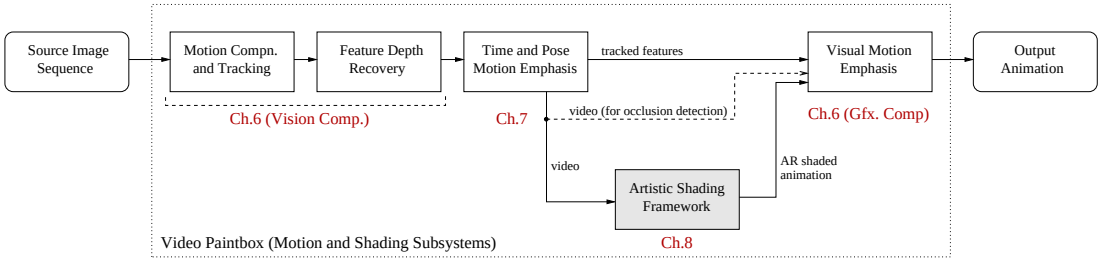

Figure 8-30 Schematic of the complete Video Paintbox, showing where the artistic rendering subsystem, described in this Chapter, fits into the rendering framework.

The user identifies features for the application of motion cues, and these features are tracked through the camera compensated sequence. Their locations and relative depths are recorded (Section 6.3.3). This processing is performing by the Computer Vision component of Chapter 6.

Time and pose cues (Chapter 7) are then applied, synthesising a novel video (for example, exhibiting anticipation) and altering the recorded locations of any tracked features. At this stage we discard the source video clip, and use this "time and pose" emphasised video as input to subsequent stages.

The "time and pose" emphasised video is passed to the artistic shading subsystem, which synthesises an AR version of the video in the requested artistic style. We now have three pieces of information: the tracked feature data, a photorealistic version of the "time and pose" emphasised video, and an artistically rendered version of the "time and pose" emphasised video.

As a final step we pass both the tracked feature data, and the AR version of the video, to the Computer Graphics component of the visual motion cue subsystem (Section 6.4). In our original description of that subsystem we stated that the Computer Graphics component accepts as input:

1. tracker data output by the Computer Vision component
2. the original, photorealistic video

We have altered the rendering pipeline, so that the Computer Graphics component accepts an artistically rendered version of the video sequence in place of (2) — see Figure 8-30. The final output is an artistically shaded animation which also exhibits motion cues. Stylisation of both the shading and motion cues is controlled by the animator at a high level (by requesting particular rendering styles and motion effects, and by setting parameters upon those special effects).

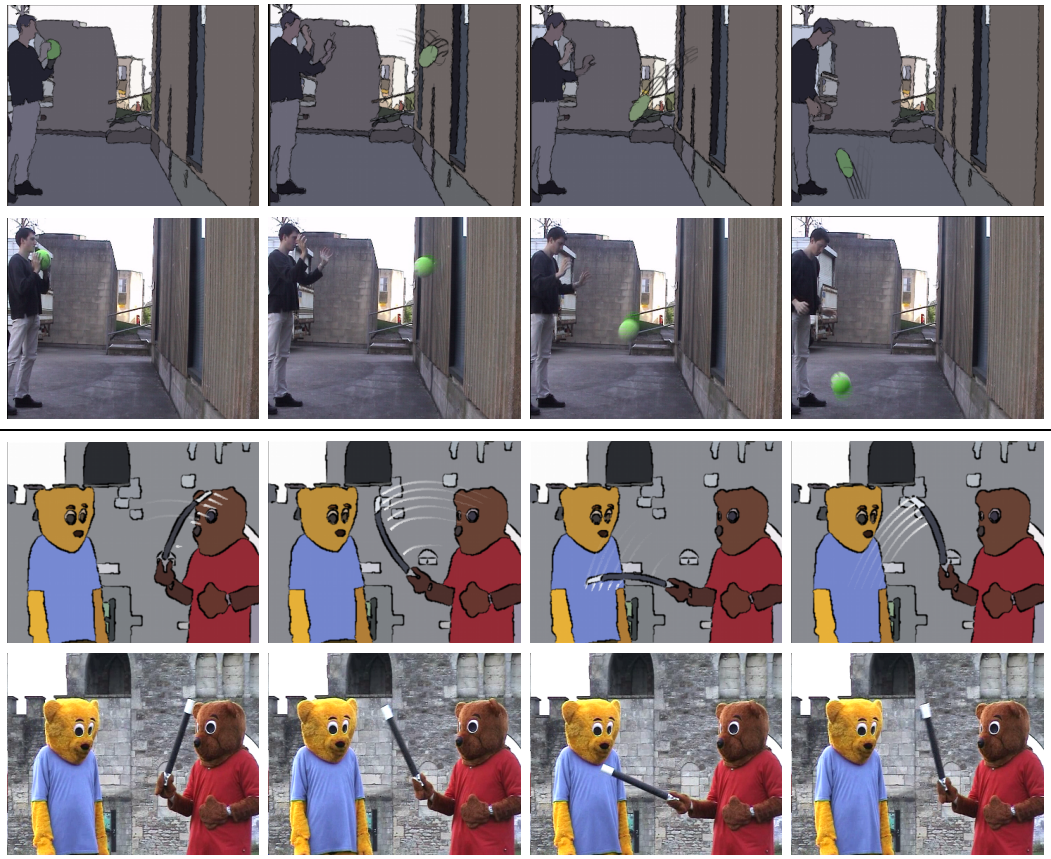

Figure 8-31 The artistic rendering subsystem is combined with the motion emphasis work of the previous two chapters, to produce complete cartoon animations from video (see `bounce_fullcartoon` and `wand_cartoon`).

As a practical note, we have observed the occlusion buffer system (Section 6.4.3) to operate with markedly less precision when using the non-photorealistic video in the manner described. We therefore make a small modification to the rendering pipeline, as described, which allows the occlusion buffer system access to the photorealistic version of the video sequence to perform its inter-frame differencing operations. This access is represented by the dashed arrow of Figure 8-30.

8.8 Benefits of an Abstract Representation of Video Content

Recall the basic architecture of the artistic rendering subsystem — the Computer Vision driven front end parses source video into an intermediate representation (IR), which the back end subsequently renders into one of many artistic styles. The IR therefore encodes an abstracted video representation, encapsulating semantic video content but not instantiating that content in any particular artistic style.

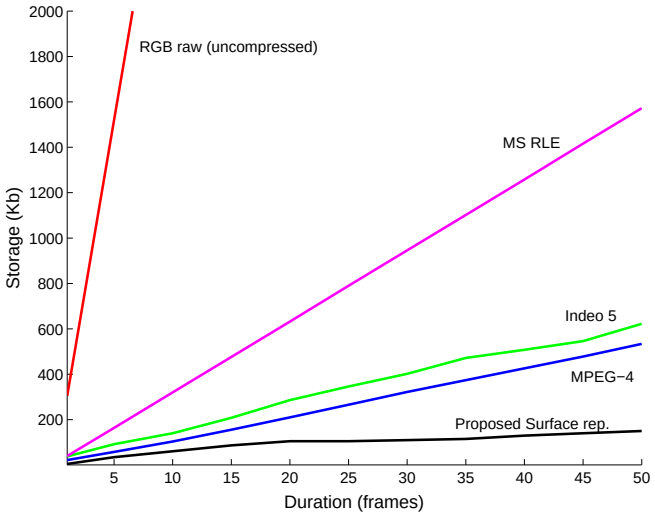

Figure 8-32 Demonstrating the comparatively low storage requirements of the surface representation when transmitting cartoons. Our test comparison uses up to 50 frames of the gradient shaded *POOHBEAR* sequence.

We draw an analogy with our IR and the popular XML data format. XML also divorces content from presentation style, deferring the problem of visualising information to the client who must process the XML content against an XSLT style-sheet to produce a legible document. This separation of information (XML) and stylisation (XSLT) creates a compact, and more manipulable format for information storage and exchange. Likewise, the IR is a highly compact representation of video content. Video objects are represented by a set of continuous spatiotemporal vector surfaces, and a small supplementary database. These properties present a number of interesting directions for future development of our IR and of the Video Paintbox system.

Hand-held mobile devices are no longer fundamentally limited by speed of CPU, or by storage constraints. The primary challenge for the next generation of wireless (3G) devices, at the time of writing, is to achieve a suitably high bandwidth wireless connection over which video content may be delivered. We suggest that, with further development, our IR could provide interesting new perspectives on both the issue of bandwidth, and upon the nature of the content delivered over the network. Figure 8-32 summarises details of a brief comparative investigation, contrasting the storage requirements of the IR with those of common video compression technologies, to store a cartoon. Approximately 150KB were required to store 50 frames of the *POOHBEAR* sequence. The compact nature of the IR can be seen to compare favourably with the other video compression algorithms tested; although we note that the spatiotemporal nature of our representation prohibits real-time encoding of video. The caveat is that the video must be abstracted and stylised in a cel animated fashion. However, the IR may be rendered into a wide gamut of artistic styles once downloaded, creating a novel level of

abstraction for video in which the server (implementing the front end) determines the video content, whilst the client-side (implementing the back end) determines the style in which that video is rendered. Splitting the responsibilities of video content provision and content visualisation between the client and server is a promising direction for development of our Video Paintbox architecture.

Aside from the benefits of compact representation and abstraction, also of interest is the continuous spatiotemporal nature of the Stroke Surfaces in the IR. This provides a highly manipulable vector representation of video, akin to 2D vector graphics, which enables us to synthesise animations at any scale without pixelisation. Indeed many of the figures in this Chapter were rendered at a scale factor greater than unity to produce higher resolution images than could be captured from a standard PAL video frame. Future developments might investigate the use of temporal scaling to affect the frame rate of animations.

8.9 Summary and Discussion

In this Chapter we have described a novel framework for synthesising temporally coherent non-photorealistic animations from video sequences. This framework comprises the third and final subsystem of the "Video Paintbox", and may be combined with the previously described motion emphasis work to produce complete cartoon-style animations from video.

Our rendering framework is unique among automated AR video methods in that we process video as a spatiotemporal voxel volume. Existing automated AR methods transform brush strokes independently between frames using a highly localised (per pixel, per frame) motion estimate. By contrast, in our system the decisions governing the rendering of a frame of animation are driven using information within a temporal window spanning instants before and after that frame. This higher level of temporal analysis allows us to smoothly vary attributes such as region or stroke colour over time, and allows us to create improved motion estimates of objects in the video. Spatially, we also operate at a higher level by manipulating video as distinct regions tracked over time, rather than individual pixels. This allows us to produce robust motion estimates for objects, and facilitates the synthesis of both region based (e.g. flat-shaded cartoon) and stroke based (e.g. traditional painterly) AR styles. For the latter, brush stroke motion is guaranteed to be consistent over entire regions — contradictory visual cues do not arise, for example where stroke motion differs within a given object. We have shown that our high level spatiotemporal approach results in improved aesthetics and temporal coherence in resulting animations, compared to the current state of the art.

Much of the discussion of the relative merits of our approach over optical flow can be found in Section 8.6.

We have demonstrated that automated rotoscoping, matting, and the extension of many "traditional" static AR styles to video, may be unified in a framework. Although we have experimented only with the extension of our own pointillist-style painterly method (Chapter 3) to video, we believe this framework to be sufficiently general to form the basis of a useful tool for the extension of further static stroke based AR techniques to video. The application of our framework to other static AR styles is perhaps the most easily exploitable direction for future work, though does not address the limitations of our technique, which we now discuss.

Perhaps the most limiting assumption in our system is that video must be segmented into homogeneous regions in order to be parsed into the IR (and so subsequently rendered). As discussed in Section 8.6, certain classes of video (for example crowd scenes, or running water) do not readily lend themselves to segmentation, and so cause our method difficulty. Typically such scenes are under-segmented as large feature subvolumes, causing an unappealing loss of detail in the animation. This is not surprising; the segmentation of such scenes would be a difficult task even for a human observer. Thus although we are able to produce large improvements in the temporal coherence of many animations, our method is less generally applicable than optical based flow methods, which are able to operate on all classes of video — albeit with a lower degree of temporal coherence. The problem of compromising between a high level model for accuracy, and a lower level model for generality, is an issue that has repeatedly surfaced in this thesis, and we defer discussion of this matter to our conclusions in Part IV. However we summarise that as a consequence we view our method as an alternative, rather than a replacement, for optical flow based AR.

The second significant limitation of our system stems from the use of homographies to estimate inter-frame motion from an object's internal texture. We assume regions to be rigid bodies undergoing motion that is well modelled by a plane to plane transformation; in effect we assume objects in the video sequence may be approximated as planar surfaces. There are some situations where lack of internal texture can cause ambiguities to creep in to this model; for example if an object moves in-front of an untextured background, is that background static and being occluded, or is that background deforming around the foreground object? Currently we assume rigid bodies and so search for the best homography to account for the shape change of the background. The worst case outcome of poor motion modelling is a decrease in the temporal coherence of any markings or brush strokes within the interiors of objects. Other artistic styles (such as

sketchy outlines or cartoon-style rendering) do not use the homography data in the IR, and so are unaffected. As a work-around we allow the user to set the motion models of video objects to be "stationary" if they deform in an undesirable manner. This single "point and click" corrective interaction is necessary to introduce additional knowledge into an under-constrained system, and is in line with the high level of creative interactive we desire with the animator. Future work might examine whether the planar surface assumption could be replaced by an improved model; perhaps a triangulated mesh, or replacement of the linear bases which form the plane with curvilinear bases (adapting the recent "kernel PCA" technique of [137]). However, many of the video sequences we have presented contain distinctly non-planar surfaces which nevertheless create aesthetically acceptable animations, exhibiting superior levels of temporal coherence than the current state of the art. We therefore question whether the additional effort in fitting more complex models would pay off in terms of rendering quality.

We did not set out to produce a fully automated system — not only do we desire interaction with the Video Paintbox for creative reasons (setting high level parameters, etc.) but also, rarely, for the correction of the Computer Vision algorithms in the front end. The general segmentation problem precludes the possibility of segmenting any given video into semantically meaningfully parts. However we have kept the burden of correction low (Section 8.5). Users need only click on video objects once, for example to merge two over-segmented feature sub-volumes in the video, and those changes are propagated throughout the spatiotemporal video volume automatically. In practical terms, user correction is often unnecessary, but when needed takes no more than a couple of minutes of user time. This is in contrast to the hundreds of man hours required to correct the optical flow motion fields of contemporary video driven AR techniques [61].

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