Stylized Video Cubes

Allison W. Klein[1] **Peter-Pike J. Sloan**[2] **Adam Finkelstein**[1] **Michael F. Cohen**[2]
[1]**Princeton University** [2]**Microsoft Research**

Abstract

We present a new set of non-photorealistic rendering (NPR) tools for processing video. Our approach is to treat the video as a space-time volume of image data. Previous tools to process video for an impressionist effect have painted collections of two-dimensional strokes on each successive frame of video. In contrast, we create a set of "rendering solids." Each rendering solid is a function defined over an interval of time; when evaluated at a particular time within that interval, it provides parameters necessary for rendering an NPR primitive. Rendering solids can be rendered interactively, giving immediate feedback to an artist along with the ability to modify styles in real time.

Benefits of our approach include: a more unified treatment of the video volume's spatial and temporal dimensions; interactive, aesthetic flexibility and control; and the extension of stylized rendering techniques for video beyond the impressionist styles previously explored. We show example styles inspired by impressionist, cubist, and abstract art of the past century.

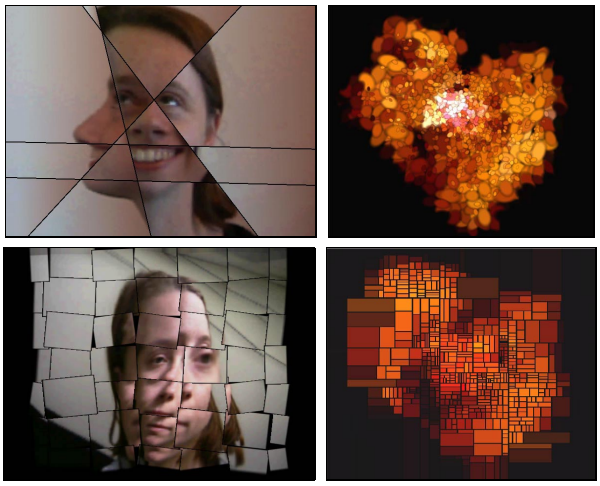

Figure 1: Example NPR video styles created with our system.

1 Introduction

This paper presents a new framework for stylized rendering of video. The cornerstone of our approach is the treatment of video as a *video cube*, a space-time volume of image data. Rather than successively defining a set of two-dimensional, non-photorealistic rendering (NPR) primitives (such as paint strokes) on each video frame, we create a set of parameterized *rendering solids*. A rendering solid is a continuous, vector-valued function defined over an interval of time. When evaluated for a specific frame, this function provides the parameters necessary for rendering an NPR primitive onto the output frame. Rendering solids may be specified automatically from the underlying video, interactively with authoring tools, or through a combination of both interactive and automatic techniques. The evaluation, composition, and final rendering of the solids occur in real time.

Our system for video stylization proceeds in two stages. First, we provide various (offline and interactive) tools for the user to define a collection of rendering solids based on the video volume, for example, growing them along flow trajectories while sampling color data. Second, in an interactive application, each rendering solid is evaluated to provide the rendering parameters for a sprite that is composited into the frame buffer for each output frame. The user can view the resulting video while modifying the set of rendering solids by smoothing their values over time or adjusting their various color and geometric attributes.

Treating the video as a volume offers several advantages over previous NPR video processing techniques. First, with our approach, one can make local choices based on global knowledge from the full video sequence. For example, because rendering solids exist over a stretch of time, they may be constructed to achieve certain artistic goals, such as temporal coherence of strokes, or stroke scale and density distributions based on a local measure of "importance." Rendering solids also enable anticipatory effects, such as automatically stretching the rendered texture sprite in the direction of motion, or localized temporal shifting of the strokes.

The second benefit of our approach is stylistic flexibility under interactive control. The final rendering is done in real time. All images in this paper were rendered in less than $1/30^{th}$ of a second. Rendering solids create a layer of abstraction between the information in the original video sequence and the final rendered appearances of a set of NPR textured sprites. This abstraction enables interactive modification of the mapping between rendering solid parameters and the rendered sprites. Thus, even within a single aesthetic style such as painterly rendering, we are able to provide artists with real-time feedback, enabling them to finely tune the appearance of the output video.

Finally, much of the previous efforts in NPR video processing have focused on filtering video for "impressionist" effects, and we demonstrate that our representation is sufficiently general to capture such effects. However, our main contribution is that rendering solids extend NPR video processing to encompass a host of *new* visual styles (Figure 1). In particular, rather than focusing on simulating any specific traditional artistic medium (e.g., oil paint or pen-and-ink), and translating it into the time domain, we offer tools that allow artists to create entirely new forms of art based on video. Our results are intended as examples of what the rendering

solid abstraction offers, as opposed to demonstrating specific styles.

The remainder of this paper is organized as follows. First, we briefly examine related work. Next, we discuss some general principles of processing the video as a space-time volume and give a more detailed explanation for rendering solids. After briefly looking at a user-interface for defining rendering solids, we then demonstrate how our approach can be used to implement a variety of styles. (Since the work presented here is fundamentally about moving images, the results are best seen on the accompanying video.) Finally, after presenting some performance details, we conclude with proposed areas of future work.

2 Related Work

Researchers have developed a variety of tools to modify two-dimensional images, giving the results a "painterly" or hand-created look in such diverse styles as impressionism [8, 9], pen and ink [17], watercolor [3], and engraving [15]. However, when NPR methods designed for static images are applied to video sequences on a frame-by-frame basis, the results generally contain undesirable temporal aliasing artifacts. To overcome these artifacts, Litwinowicz [14] used optical flow information to push painterly strokes from one frame to the next in the direction of pixel movement. Hertzmann *et al.* [10] created each successive frame of the video by first warping the previous frame to account for optical flow changes and then painting over areas of the new frame that differ significantly from its predecessor. In our work, we construct three-dimensional (space and time) rendering solids using optical flow or other methods, before any rendering is done. The trajectory in time of the rendering solid can then be smoothed or otherwise modified before actually being rendered. In addition, our work is not style-specific and gives the artist greater interactive control than any method designed to simulate a specific artistic medium. Nevertheless, painterly styles can be rendered in our system.

Our system is not the first to treat video as a volume or view time-sequenced imagery from viewpoints other than on the time axis. Fels and Mase [5] presented an interactive system for passing arbitrary cutting planes through the video data volume. One could, for example, view slices of the cube that are not necessarily orthogonal to the time axis (Figure 2). By slicing the video parallel to the time axis, motion (of either scene elements or the camera) is revealed as changes across the scanlines; each scanline contains the time-trace of a single pixel from the original video. In computer vision, such images are sometimes referred to as *epipolar diagrams* [1, 4]. *Multiple-center-of-projection* images [16] and *multiperspective panoramas* [18] may also be considered two-dimensional (though non-planar) slices of a video in which the camera is moving. One of our contributions is to apply the underlying ideas of a video volume and non-standard cutting planes towards non-photorealistic rendering of video.

Finally, this work has been greatly inspired by the aesthetics of the cubist and futurist art movements of the early 20^{th} century. Just as these artists mixed space and time within an image, we leverage the ability to slice the input video in non-standard ways to mix space and time. Nonetheless, it is not the goal of this paper to produce an automated video filter for reproducing the style of Picasso, Duchamp, or any other specific artist. Instead, we wish to provide a framework for creating many different styles.

3 Processing Video as 3D Data

We begin this section by introducing *video cubes*, and explain the relationship between video cubes and rendering solids. Next, we give a high-level overview of the framework for defining, modifying, and interactively rendering the rendering solids to create stylized video. We then look in more detail at the construction of the rendering solids and conclude by providing a number of example styles based on these rendering solids.

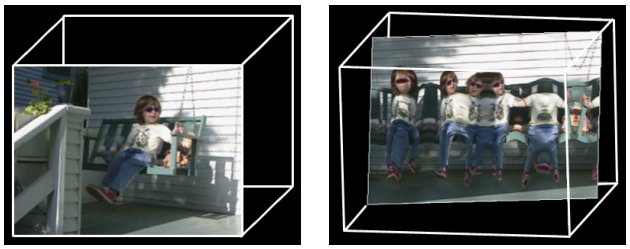

Figure 2: Cutting a video cube. Left: a frame of a video sequence. Right: an image generated by a non-axis-aligned cutting plane.

3.1 Video Cubes and Rendering Solids

A video is a sequence of images that has been discretized in 2D space (pixels) and in time (frames). Because there is a large body of work in computer graphics devoted to discretized 2D image processing techniques, it seems only natural that most NPR video processing work to date has consisted of running these techniques on each individual frame. The major concession to the third dimension, time, has been the use of optical flow [10, 14]. One of our main goals is to treat time in a more unified way.

Our approach is to treat the video as a *video cube*, a three-dimensional volume of pixel data (Figure 2). Rather than defining two-dimensional strokes successively on each video frame, we create a set of parameterized "rendering solids" that exist within the space-time volume of the video cube. A rendering solid is a continuous, vector-valued function defined over an interval of time. When evaluated at a particular time t, a rendering solid provides the parameters necessary to render an NPR primitive. Each rendering solid may be thought of as defining a solid region within the video cube. Depending on the rendering style, the set of rendering solids may either intersect, fully tile the volume, and/or leave parts of the cube empty.

More formally, a rendering solid $\mathcal{S}$ is a function $\mathcal{S} : \Re_{[t_1,t_2]} \mapsto \Re^m$ that maps time t in the interval $[t_1, t_2]$ to an m-dimensional vector expressing properties such as color, position, and other parameters for rendering a (possibly textured) sprite that will be composited onto the output frame. The use of rendering solids transforms the discrete representation of the video volume into a collection of continuous functions of time. These continuous functions are beneficial in that they may be sampled as finely as desired for different playback speeds, or smoothed for temporal coherence. We should note that playback time T, the rendering direction, typically aligns with the conventional time dimension in the original source video; however, for some artistic effects, "time" might align differently.

3.2 Defining Rendering Solids

Defining a set of rendering solids proceeds through a combination of automated and artist-initiated, interactive methods. The video is first pre-processed to add additional information including measures of color gradients and *importance*, a measure of local variation in space and time. Pixel color plus these derived values are then used as input to a variety of automated and interactive tools to define the rendering solids. We next describe three of many possible choices for types of rendering solids.

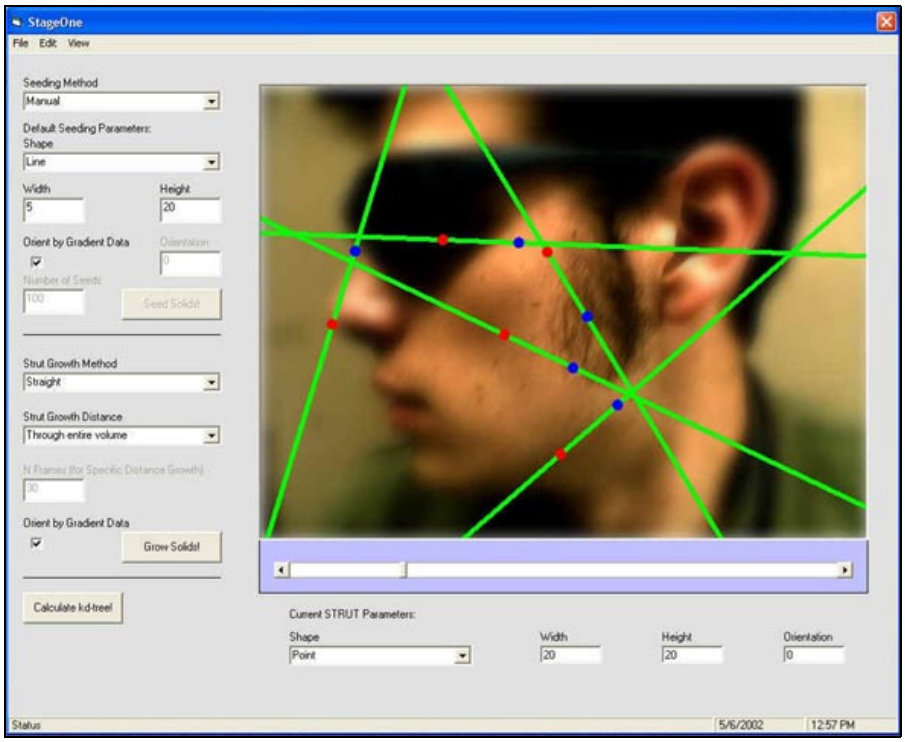

Figure 3: Our interface for defining rendering solids. Here we see a user interactively creating a set of cutting lines; each cutting line is defined by two points (specified by mouse clicks). The cutting lines will be interpolated between key frames to describe swept surfaces over time, dividing the volume into video shards, which are described in section 4.3.

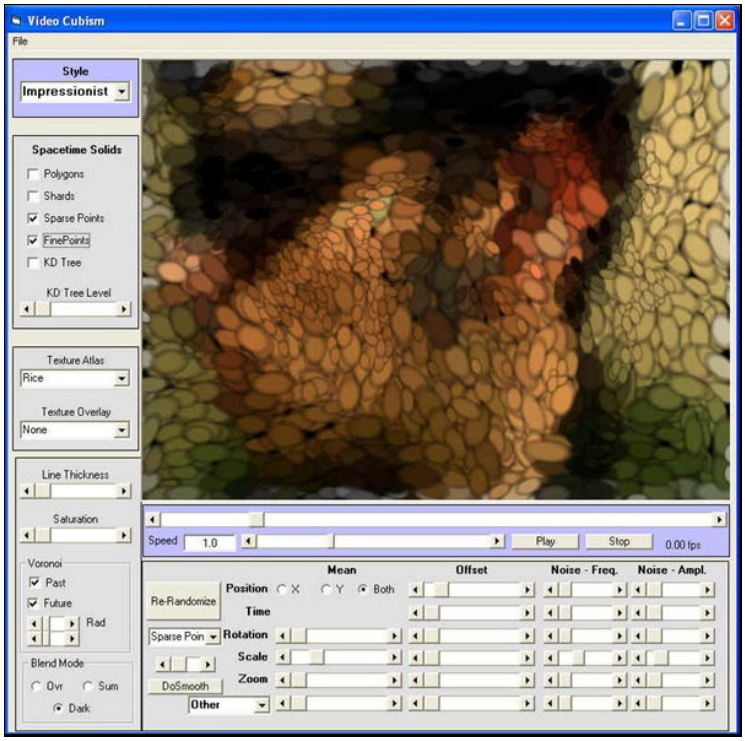

Figure 4: Our interface for the interactive rendering session. The artist has a variety of interactive controls over the final look of the video, for example: choosing the texture atlas; drawing more or fewer rendering solids to control how "busy" the image appears; and scaling, shifting, smoothing, or adding noise to the parametric curves of the rendering solids

Shard-like Rendering Solids

The first type of rendering solid subdivides the video volume into a number of shard-like solids. An artist interactively creates a set of swept surfaces, each of which slices the video volume into two pieces. Each swept surface is created by interpolating a series of cutting lines placed on key frames. To create a cutting line, the user specifies the line by placing two points (two left clicks) in the current frame. The first control point is the line's *center point* and the second control point specifies the *direction* of the line through the center point. (See Figure 3.) The user then scrolls through the input video to a different key frame, selects the line, and modifies either its center point or direction. Each frame in which the cutting line is manipulated is considered to be a key frame. Between key frames, the values for the two control points are linearly interpolated. For frames before the first key frame or after the last key frame, the control lines simply use the closest key frame values. The outer boundaries of the video volume are also considered to be cutting surfaces separating the video from the space outside.

The union of the set of interactively constructed cutting surfaces (and video boundaries) defines a set of shard-like rendering solids. Each shard is assigned an ID made up of a bit code indicating whether it lies on the left or right side of each cutting surface. Note that a single shard might occupy two disjoint portions of the video volume since it may vanish and then later reappear as the swept surfaces cross and then uncross. The shard ID is then used to anchor interactively modified rendering parameter values at runtime.

The cutting surfaces defining the shards also help define "flow" within the video volume[1]. For any point P in a video frame i, we determine the flow vector by:

1. Initialize the flow vector to $(0, 0)$.
2. Find the cutting lines that form the polygonal cross-section of the shard enclosing P.
3. For each line segment L of the shard polygon, with center point C and direction D:
   (a) Distance = P's perpendicular distance to L in frame i.
   (b) Weight = (length of L in frame i) / (Distance + ϵ). The ϵ is used to prevent division by zero. Note: Weight = 0 for boundary lines.
   (c) Displacement_velocity = (L's center point in frame $i+1$) — (L's center point in frame i).
   (d) Rotation_vector = (L's direction in i) $\times$ (L's direction in $i+1$)
   (e) Angular_velocity = Rotation_vector $\times$ (the vector from P to L's center point in frame i).
   (f) Current_velocity = Displacement_velocity + Angular_velocity
   (g) Flow_vector += Current_velocity $\cdot$ Weight
4. Normalize Flow_vector by the sum of all weights

KD-Tree-Based Rendering Solids

An automated method for defining cutting line segments uses importance values to construct a kd-tree defining a set of rendering solids whose rectangular cross-sections tile any time slice. Kd-trees are typically used to hierarchically decompose space into a small enough number of cells such that no cell contains too much complexity. For similar purposes, we use kd-trees to subdivide the video volume into sub-volumes, each containing approximately the same amount of importance, while also attempting to maintain an even aspect ratio. The kd-tree is constructed of alternating vertical and horizontal swept surfaces. The first level of the kd-tree defines a vertical surface that cuts the entire video volume in two. Each subsequent level (alternating horizontal and vertical) slices each subvolume. The position of the vertical and horizontal kd-tree partitions change through time, thereby producing curved swept surfaces. Figure 5 shows a simplified, didactic image of two levels of such a kd-tree within the video volume. The shaded plane shows the kd-tree decomposition for the current time t. The curved lines through the video volume show the edges of the swept surfaces created by the smoothly moving kd-tree partitions over time.

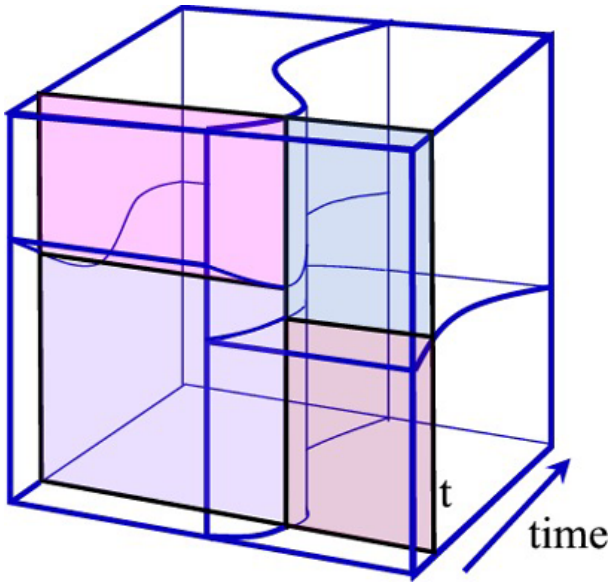

Figure 5: A simplified, didactic image of our time-varying kd-tree within the video volume. The shaded plane shows the kd-tree decomposition for the current time t. The curved lines through the video volume show the edges of the swept surfaces created by the smoothly moving kd-tree partitions over time.

To construct these k-d trees, we first build a 3D summed-area-table[2] of importance. The summed-area-table provides the values needed to quickly evaluate the integral of importance over any rectilinear subregion of the video. Kd-tree construction then proceeds by first determining a vertical swept surface that, at each point in time, has approximately equal total importance on each side. More specifically, the horizontal position, $x(t)$, is determined such that

$$\int_{t-0.25}^{t+.25}\int_{0}^{yres}\int_{0}^{x} I(x,y,t)dtdydx = \int_{t-0.25}^{t+.25}\int_{0}^{yres}\int_{x}^{xres} I(x,y,t)dtdydx$$

In other words, to create a smoothly varying vertical surface, the horizontal position of the surface is determined so that a slab one half second thick in time, centered about each time value, is (vertically) divided equally in terms of importance. In a similar fashion, each half of the volume is divided by a horizontal swept surface. Each of these quarters is then divided vertically, and so on, until six levels of kd-tree are defined. Each of the rendering solids defined by the kd-tree is annotated with its average color.

[1] Automatic optical flow methods could also be used, however, we found that these methods were not very robust and did not provide the artist any controls.

Worm-like Rendering Solids

A third type of rendering solid is defined by a point trajectory through time, in other words a curved line that does not bend back on itself in the time dimension. At each time t, the (x, y) location of the trajectory will guide the placement of textured sprites at runtime. The union of these sprites for a single trajectory implicitly defines a worm-like solid structure. Parameters such as color or orientation (based on color gradient information) over time are recorded along these trajectories and added to their respective rendering solids to be used at runtime.

Automatic and interactive tools are provided to generate a set of position trajectories. The simplest trajectory is one that has a constant value, in other words a single position that stays still through time. More interesting trajectories can be created interactively by interpolating points placed at key frames. Finally, large sets of trajectories can be automatically generated that follow the flow defined by the shards described earlier.

To create a set of evenly spaced rendering solid trajectories that follow the flow, we have adapted the 2D streamline placement algorithm of Jobard *et al.* [12]. The strategy is to divide the volume into a regular grid, and then grow curves (streamlines) by stepping through the grid following flow. Whenever a curve advances to a new grid cell, it can efficiently determine if it is too close to nearby curves simply by checking neighboring cells; if the current streamline is closer than a pre-determined distance threshold, or if it reaches the edge of the volume, the curve terminates. At this point the algorithm walks along the curve checking for a place to seed a new curve, again by testing nearby cells for vacancy. In adapting the Jobard algorithm to 3D, we had to address the efficiency challenge of maintaining the entire voxel grid in memory and then walking that memory with little coherence. Fortunately, our source of data (flow) provides a solution to this problem: the flow at every point has a component in the time dimension. Thus, we modify the algorithm to start from the first frame and proceed forward through time, considering one small slice of the volume at any time. In that slice, we keep track of many curves at once, and advance all of them to the next slice in unison. As we advance, some curves are terminated, and new curves are born in the interstices. The results of this process are a set of discretized curves that are the (x,y,t) trajectories of a set of rendering solids. Finally we fit continuous cubic B-splines to these values. This reduces the amount of data, provides a means to efficiently evaluate position at any point in time, and also provides simple methods to smooth the trajectories by reducing the number of control points.

Summary

After creating rendering solids using the techniques described above, we can use these rendering solids at runtime to interactively render NPR frames. Each rendering solid can be evaluated at any point in time, returning a shape (a polygon for each shard, a rectangle for each kd-tree node, or a point for each worm-like solid), plus values for color, orientation, or other properties needed at runtime. We do not consider these three types of rendering solids to be a complete set. Far from it, we see these as just a few examples of the kinds of representations that are possible. Depending on the style chosen and the associated parameter settings, a wide variety of visual representations can be created in real-time from the rendering solids. We will explore a few of these next.

4 Stylized Rendering

In this section, we discuss a number of styles we used to explore the concept of rendering solids. Since each style provides a number of parameters that can be modified in real time, the artist is given wide latitude to interactively explore different visual results. All styles we describe below are rendered in less than a thirtieth of a second per frame.

The specific examples we show were inspired by 19^{th} and 20^{th} century paintings and styles – paintings by Monet (impressionism), Picasso (cubism), Duchamp (futurism), Mondrian (abstract), and Hockney (photo mosaics). However, the algorithms described here are not intended to automate or replicate works by those artists. Indeed, it is difficult to imagine an algorithmic representation of the genius of these artists. Nor is it clear how any of these artists would translate his work for time-changing imagery, even if it were possible. Our goal is to provide tools for people to be able to create new forms of video-based art, and the examples we present were inspired by the works of those masters. The styles described below are only a few point samples in a very wide space of possibilities that open up by considering the video as a data volume. Therefore, these styles and their particular implementation details are presented more as examples of what one can do within this space rather than as a comprehensive coverage of the possibilities.

In each style, output time, T, is a function of the input time, t, allowing the artist to modify the speed or produce other effects. Each rendering solid may also be interactively repositioned in time, thus any single output frame may draw information from many points in time in the original video. The inverse function $t(T)$ provides the value to sample the rendering solids for each output frame. We will simply refer to this sampling time as t.

To render each frame, we evaluate each rendering solid at the current time t to extract rendering parameters such as position, scale, and orientation. These rendering parameters plus time can also be interactively modified. For example, the rendering solid can be shifted in space and time, rotated, and/or scaled. Color values can be scaled for effects such as increased saturation. If the rendering calls for extracting a texture from the underlying video, the texture coordinates can also be shifted and/or scaled. (See Figure 4.) The fact that rendering solids form an abstraction separating the structure of the input video from its final, non-photorealistic rendering enables this interactive exploration of stylistic choices.

4.1 Paint Strokes

"Impressionist" imagery is, in general, constructed of short strokes of color that, taken together, produce the final image. We use the worm-like rendering solids to position a set of colored sprites that create our "impressionist" imagery (e.g., Figure 6). Parameters that are recorded along the trajectories, such as color or orientation (based on color gradient information [8]), are also extracted from the rendering solids.

A final input to the stroke-based style is a *texture atlas*. The texture atlas contains one or more opacity masks for strokes. Multiple masks can be indexed by parameters such as size, orientation, time, or simply randomly to generate desired variability across individual strokes. Alternatively, a single mask can be scaled (possibly anisotropically) and rotated at runtime as it is rendered. Finally, the mask is colored by the values in the rendering solid and optionally modified by an additional texture to create darkened edges or a stippled effect.

At runtime a plane is passed through the video cube. Each resulting frame is constructed on the fly by compositing one sprite for each rendering solid intersecting the plane. The artist has a variety of interactive controls over the final look of the video, for example: choosing the texture atlas; drawing more or fewer rendering solids to control how "busy" the image appears; and scaling, shifting, smoothing, or adding noise to the parametric curves of the rendering solids.

Smoothing, in particular, is achieved as follows. Each parametric

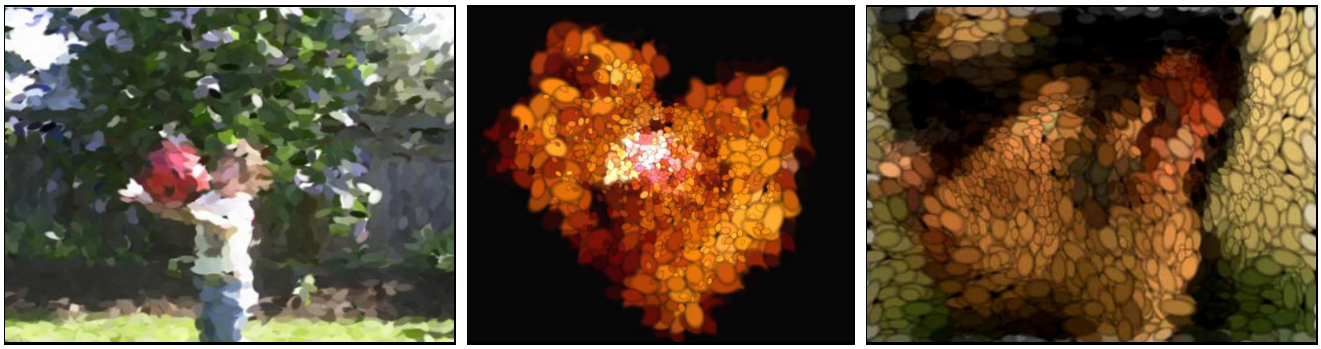

Figure 6: Stroke-based images. Left: standard painterly brush strokes. Center and Right: hard-edged strokes used to render an exploding fireball and a face in profile, respectively.

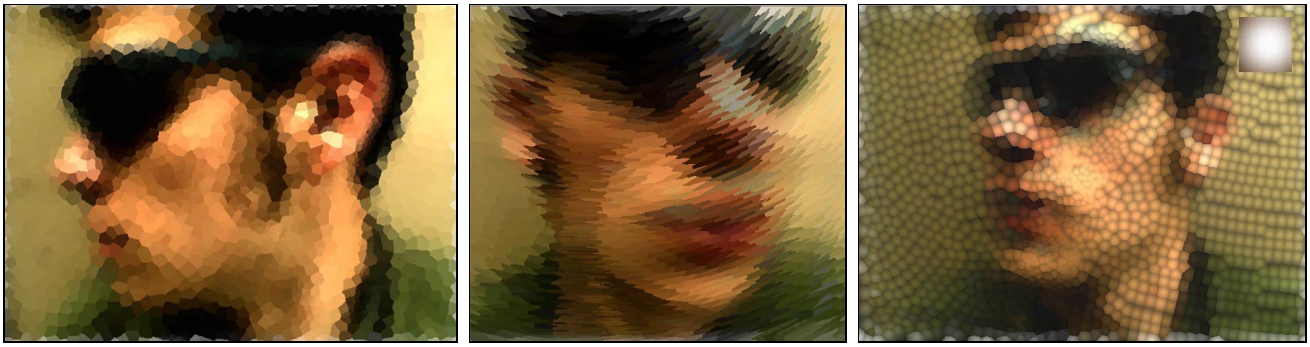

Figure 7: Voronoi rendering results. The center image has Voronoi seed movement extended both forwards and backwards in time. The right-most image uses a second bubble-like texture (inset in upper-right corner) to modify each cell.

function over time is fit to a cubic B-spline with uniform knot spacing. Since all rendering solids are fit to the same B-spline basis set, a single matrix is inverted to fit the data. Thus, to evaluate these curves at a particular frame, the system only has to compute four B-spline coefficients per frame and reuse these for every parameter of every rendering solid. To smooth one parameter we fit it to a B-spline basis set with fewer knots and then reproject back to the original (higher) number of control points. Optionally noise can be added to any parameter using the same strategy.

Figure 6 shows a variety of results from this style. The left-most image of a girl catching a ball is a standard painterly rendering. The center image of an exploding fireball shows the "paint" strokes rendered with a rounded texture sprite and hard edges. Notice the small strokes clustered in areas of higher spatio-temporal importance. Finally, the right-most image shows the interactive user interface during a session manipulating textured strokes of a man's profile.

4.2 Voronoi Cells

One of the hallmarks of early cubist and futurist imagery was the decomposition of a scene into geometric shapes. Taking this as inspiration, we have investigated stylized video based on a 3D Voronoi decomposition of the video cube. Many image manipulation programs provide the ability to generate tiled images from input images. The underlying algorithm is based on work by Haeberli [8] in which tiles are actually 2D Voronoi tiles, and the tile colors are sampled from the Voronoi seed point locations. If one were to apply this algorithm to successive frames of video with a fixed collection of seed points, the resulting imagery would look as if the video were being viewed through a shower door. On the other hand, if one were to apply the algorithm to successive frames of video with a different set of seed points for every frame, the resulting video would be "flickery" and have very little temporal coherence.

To address these problems, we generalize the process to 3D. The 3D Voronoi cells are grown around the trajectories of the worm-like rendering solids. These trajectories are the same ones used to render the impressionist style. In addition to providing a means for decomposing our volume into geometric shapes, a significant benefit to this approach is that 3D Voronoi cells yield substantial temporal coherence without suffering from the "shower door" effect. As the plane passes through a given cell, its shape varies smoothly because the trajectory that defines the cell is continuous in time.

To render the Voronoi style interactively, we never explicitly construct the 3D Voronoi cells. Computing the full 3D Voronoi diagram of the curves would be computationally prohibitive. Rather, for each rendering solid intersecting the current frame, we sample its trajectory, extracting position and velocity values. This defines a tangent line that represents a local approximation to the trajectory. For each line, we draw a cone representing the distance function between the image plane and the line. After drawing all cones, the z-buffer then leaves intact just the region of the plane closest to each rendering solid. Hoff *et al.* [11] describe the algorithm in detail. While in theory, the cone drawn for each rendering solid covers the entire image, in practice a much smaller cone will suffice, allowing us to improve rendering performance. By manipulating a slider, the user can increase or decrease the radius of the rendering cone.

Using the Voronoi diagram of a line rather than a point causes each Voronoi cell to extend in the direction in which its respective rendering solid is moving. Again, by means of a slider, we allow the user to accentuate the direction of movement (forwards, backwards, or both) through time. Alternately, the user can disable this movement, generating a standard Voronoi diagram.

Figure 7 shows some results from rendering the 3D Voronoi

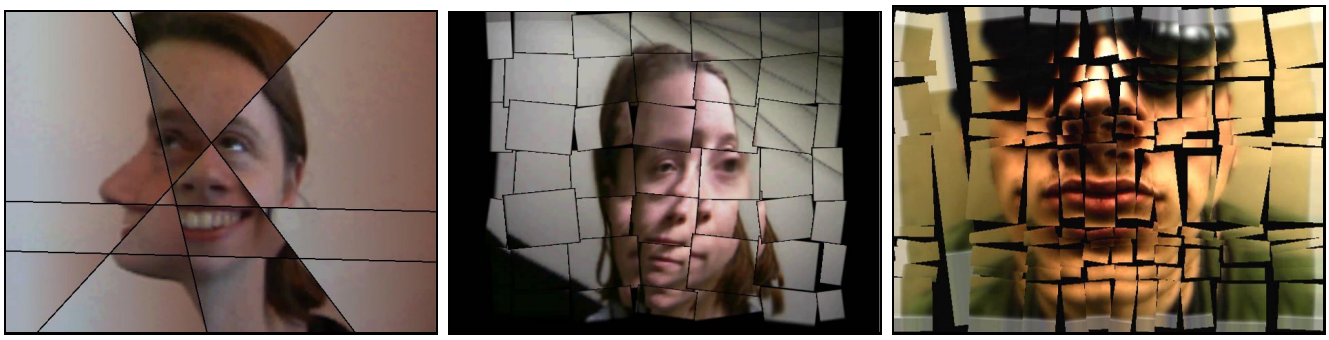

Figure 8: Shards rendering results. The left image uses two videos captured simultaneously. All three images show multiple points in time.

cells. In all three images, rendering solids were scattered based on the importance function to emphasize local detail. The center image has Voronoi seed movement extended both forwards and backwards in time, while the right-most image uses a second bubble-like texture (inset in upper-right corner) to modify each cell.

4.3 Video Shards and Mosaics

In another style inspired by early cubism, we use the shard-like rendering solids. As in the case of the voronoi style, the 3D shards are not explicitly evaluated. At runtime, the cross-section of the individual shards are determined on the fly in the following manner: for each frame, shard vertices are found at cutting line crossings, which break the cutting lines into line segments. By always turning left at intersections, the segments are then linked together to form convex polygons. An efficient edge-based structure can be computed very quickly [7]. The original video is used to texture each shard.

At runtime, we provide the artist with a number of ways to modify each shard, including: zooming each shard texture coordinates; modifying its time association with the input video; and modifying the video texture by multiplying it with a second texture. (Zooming defines the ratio of the size of the source video texture to the output region. A small source texture mapped to a larger output region will have an effect like a magnifying glass.)

To interactively texture the output frame with shard areas from multiple frames in the input video, we have implemented a ring buffer. Given a maximum of N frames for the time perturbation, holding the current frame plus the N frames before and after the current frame in texture memory provides all the source textures for any output frame. As each frame from the original video falls more than N frames in the past, its texture in the ring buffer is replaced with a new frame drawn from N frames in the future. In addition, to maintain better memory locality, each frame in the ring buffer is looped through and used to texture any rendering solids that touch it, rather than looping through the rendering solids and finding the appropriate frame to draw the texture from. Because these shard areas do not overlap, compositing order is irrelevant.

Figure 8 shows some results of using swept surfaces to decompose the image. Each shard has been scaled and shifted in time as a function of the shard size. For the left-most image, we have also used two videos, both of which were captured simultaneously. Within this image, the left-hand shards show a woman in profile. The right-hand shards show the same woman, but facing forward.

Using a grid-like placement of the cutting lines creates images reminiscent of the photo mosaics of pop-artist David Hockney. Again, we can spatially and temporally offset the individual image "tiles," as well as rotating and zooming them. The center and right images in Figure 8 shows some results from this photo-mosaic style. Note the multiple points in time, as well as rotations and scales in the individual tiles. In the center image, we first processed the underlying video to show multiple points of view at a single time. The original video stream was captured by a camera moving relative to the woman's head.

4.4 Abstract Tiles

Motivated by Mondrian's abstract paintings, we seek to turn video into a mobile series of colorful, rectangular compositions. To achieve this goal we use the kd-tree defined rendering solids.

As we traverse the video cube at runtime, each kd-tree cell is colored with a constant color representing that cell's average color. At runtime, the artist can interactively specify a number of parameters, including thickness of dividing lines, and color remapping to increase saturation. The artist also specifies the depth of the kd-tree traversal. A traversal closer to the leaves yields smaller blocks, while a traversal closer to the root yields fewer, larger blocks. Only nodes at the chosen level are rendered except during transitions between levels. Dropping one level in the tree subdivides each node into two new ones. As new cells are added to (or removed from) the output, they are smoothly transitioned in (or out) from their children (or parent) to avoid popping.

Figure 9 shows some results of applying this abstract geometric style to videos of a talking woman and an exploding fireball. (Both the center and left images also have increased color saturation.) The center image is reminiscent of a Mondrian painting, while the more detailed image on the left is less abstract and more recognizable as a human face. The video shows the smooth variation from the center image to the left one by sequentially selecting deeper tree levels. We find the transformation from abstract images to more representational ones to be visually intriguing.

5 Notes on performance

Each of the images in this paper and all the examples in the accompanying video were rendered in real time on a 1.7GHz Pentium 4 PC with 512MB of RAM and an Nvidia GeForce3 graphics card. Depending on the specific style and particular settings chosen, the frames of the NPR video are rendered at between 50 and 200 frames per second. However, the timings on many of the styles can be seriously degraded by some of the choices possible at runtime. Clearly, by including too many rendering solids in any of the styles, the system can become polygon bound. More commonly, the system slows down due to limitations in fill rate. For example, by setting the radius too large for the cones associated with each seed point of the Voronoi style creates a high depth complexity and thus a fill rate too high for interactive speeds.

The operations carried out during rendering solid definition are mostly interactive. Manually placing points or lines within the volume, or randomly seeding the volume with points is real time. However, filling the volume with many small, worm-like rendering

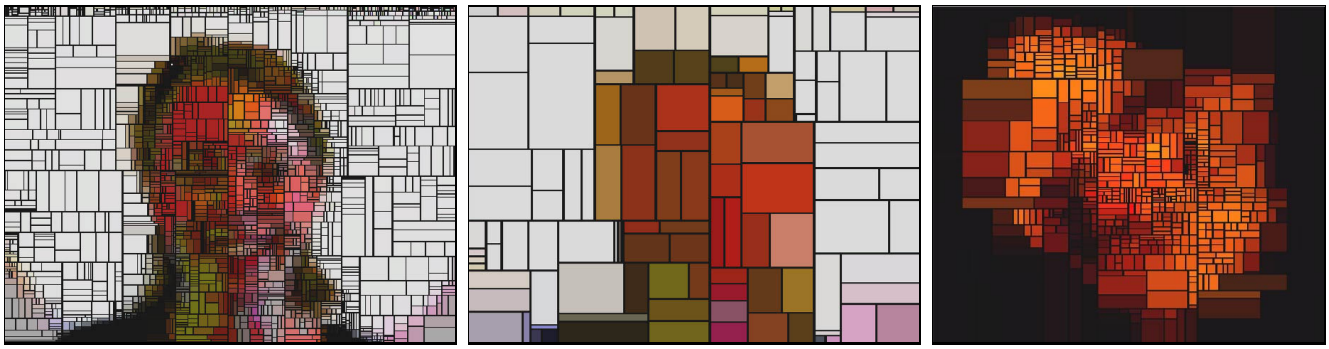

Figure 9: Our abstract geometric style applied to videos of a talking woman (left and center) and an exploding fireball.

solids that grow along flow lines can take about 2 minutes of computation on a 640x480 video of 300 frames. Pre-computing the image orientation and importance values takes on the order of 20 minutes for a 300 frame video. However this must only be done once for a given video. Finally, computing the summed-area-table to guide the kd-tree divisions in the abstract tiling style takes only a few seconds.

6 Conclusion and Future Work

This paper presents a new framework for stylized rendering of video. Our approach is to treat the video as a space-time volume of image data when designing NPR rendering methods for video. Rather than working with collections of two-dimensional strokes on successive video frames, we create rendering solids which, when evaluated at a specific frame time, provide the parameters necessary for rendering an NPR primitive. There are three key benefits to this approach. First, one can make local choices based on global knowledge of both the input video and the resulting rendering solids. Second, our approach provides flexible, interactive control to the user. Rendering solids create a layer of abstraction between the information in the original video sequence and the final rendered appearance of a rendered NPR primitives. This abstraction enables interactive modification of the mapping between the rendering solid parameters and the rendered primitives. Finally, our main contribution is that rendering solids extend NPR video processing to encompass a host of new visual styles. Rather than focusing on simulation of a specific artistic medium, we offer tools that allow artist to create entirely new forms of art-based video.

This work suggests a number of areas for future investigation:

- **Better support for multiple simultaneous video cubes** Other than the single example shown in Figure 8, we have not explored interaction with multiple simultaneous video cubes. Such exploration could provide even greater expressiveness and flexibility for artists.
- **Applying rendering solids to data visualization** The idea of growing rendering solids through a volume could be applied to other volumetric data sets. For example, given a volume of density data, one could grow rendering solids through the volume along fields of constant density. Other data values within the same volumetric data set, such as temperature, could be sampled and mapped to rendering parameters (such as color or texture) during the run-time rendering stage.
- **Applying rendering solids to higher-dimensional approximations of the plenoptic function** Video can be though of as a 3D slice of the plenoptic function. Lumigraphs/light fields [6, 13] are 4D approximations to the plenoptic function. By applying rendering solids to lumigraphs/light fields , one could achieve artistic walk-throughs with frame-to-frame coherence. This idea could also be applied to time-changing approximations of the plenoptic function.

References

[1] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55, 1987.

[2] F. C. Crow. Summed-area tables for texture mapping. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):207–212, 1984.

[3] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin. Computer-generated watercolor. *Computer Graphics (Proceedings of SIGGRAPH 97)*, 31:421–430, 1997.

[4] O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[5] S. S. Fels and K. Mase. Techniques for interactive video cubism. In *Proceedings of ACM Multimedia*, pages 368–370, Oct 2000.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. *Computer Graphics (Proceedings of SIGGRAPH 96)*, 30:43–54, 1996.

[7] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[8] P. E. Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24:207–214, 1990.

[9] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. *Computer Graphics (Proceedings of SIGGRAPH 98)*, 32:453–460, 1998.

[10] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. *Computer Graphics (Proceedings of NPAR 2000)*, pages 7–12.

[11] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics (Proceedings of SIGGRAPH 99)*, 33:277–286, 1999.

[12] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. *Proceedings of Eighth Eurographics Workshop on Visualization in Scientific Computing*, April 1997.

[13] M. Levoy and P. Hanrahan. Light field rendering. *Computer Graphics (Proceedings of SIGGRAPH 96)*, 30:31–42, 1996.

[14] P. Litwinowicz. Processing images and video for an impressionist effect. *Computer Graphics (Proceedings of SIGGRAPH 97)*, 31:407–414, 1997.

[15] V. Ostromoukhov. Digital facial engraving. *Computer Graphics (Proceedings of SIGGRAPH 99)*, 33:417–424, 1999.

[16] P. Rademacher and G. Bishop. Multiple-center-of-projection images. *Computer Graphics (Proceedings of SIGGRAPH 98)*, 32:199–206, 1998.

[17] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. *Computer Graphics (Proceedings of SIGGRAPH 97)*, 31:401–406, 1997.

[18] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin. Multiperspective panoramas for cel animation. *Computer Graphics (Proceedings of SIGGRAPH 97)*, 31:243–250, 1997.