ABSTRACT

Making plausible, high quality visual effects, like water splashes or fire, in traditional 2D animation pipelines require an animator to draw many frames of phenomena that are very difficult to recreate manually. We present a technique that uses a database of video clips of such phenomena to assist the animator. The animator has to only input sample sketched frames of the phenomena at particular time instants. These are matched to frames of the video clips and a plausible sequence of frames is generated from these clips that contain the animator drawn frames as constraints. The colour style of the hand-drawn frames is used to render the generated frames, thus resulting in a 2D animation that follows the style and intent of the 2D animator. Our system can also create multi-layered effects animation, allowing the animator to draw interacting mixed phenomena, like water being poured on fire.

Keywords
- non-photorealistic animation; non-photorealistic rendering;
- sketch-based; special effects

1. INTRODUCTION

Complex phenomena like fire, water splashes, rain and smoke need to be convincingly animated as part of many modern animation productions. In common animation parlance, these are referred to as effects animation, and there is often a dedicated team of animators to handle these animation sequences. This is so because creating effects animation requires a lot of skill that is very difficult to acquire.

While these can benefit from physically-based simulation techniques, they are often very time consuming and difficult to control for animators. In this paper we present a method for performing 2D effects animation using data from videos of similar recorded phenomena. It does not matter whether the videos are of real, naturally occurring events or of simulations; we can use them in our method as long as they are similar to what the animator desires to create.

In our data-driven system, the animator provides sketches of a few frames of the phenomena that is to be animated. These sketched frames form constraints for our system as we search for a combination of seamlessly transitioning video clips that can connect the sketched frames at desired time instants. We finally render the animation with colours derived from the sketches, thus matching both the colouring style and intent of the animator. The animator can also mix multiple phenomena in different layers to generate more complex animation with multiple kinds of effects animation. The animator is free to edit their sketches or add new ones, as a part of the process and re-render the final animation, till the desired effect is achieved.

The scope of effects animation is very vast as it covers everything from magic to volcano eruptions. We restrict the scope of our examples to fluid based effects like water splashes, flames, explosions and water streams to make the problem tractable. Our techniques are, however, general.

We cover relevant background literature in Section 2. We start with an overview of our system in Section 3. We follow this up with details about the offline and online parts of our method in Sections 4 and 5. We present results generated using the methods described in this paper in Section 6. We conclude with a discussion of limitations of the current method and avenues for future work in Section 7.

2. BACKGROUND

The complexity of hand drawn effects animation is beautifully explained and illustrated by Joseph Gilland [5]. The fact that he argues that even the most complex hand-drawn or computer generated animation is no match for the real thing as the sheer level of detail is too difficult for the animator to manually create is indicative of the amount of skill involved in making convincing effects animations.

This inherent complexity and skill required in animation has led to development of a number of data driven techniques to animate many phenomena like cloth [7], aerodynamics of rigid objects [10], fluids [2] and fire [13]. None of these, however, attempt to solve the problem of producing artist-directed non-photorealistic effects animation like our method does.

Chuang et al. [3] present a method to animate fluids in images by using stochastic motion textures. In more recent
work, Okabe et al. [11] describe an image based method for animating fluids in real images. In order to do this the user demarcates the fluid region in the image, and then an effect from a best matching video is used to animate the fluid region. The matching is done using the histogram of visual words created from the video database. Our method, however, differs from such works in terms of the non-photorealistic rendering quality of the final animation produced, the generality of the technique and sketch-based guidance given by the animator to the system. Zhu et al. [15] present a system to create non-photorealistic fluid animation based on a multi-layered regional fluid solver. They present an interface where a user can draw pipes, sources, sinks and kinematic objects. This is used to create a hydraulic graph that is solved using the Navier-Stokes equation. The system produces real time flow of fluids in the drawn region. It is a physically-based method and is not as flexible and adaptable to various kinds of animation, as the method presented in this paper.

Zhang et al. [14] present a method that uses a motion graph for creating new simulations from existing simulations. They match parts of existing simulation using a concept called path lines and create seamless transitions between them. They then modify the simulation parameters and re-render a new video. Though we use the concept of motion graphs as well, we use video data to drive forward the animation. We don’t use any simulation and our method for calculating similarity between frames and sketches is different. Schödl et al. [12] present one of the very first works on motion graphs to create infinitely looping videos of a flame. We are inspired by their work in using motion graphs in our system. Motion graphs are also used in character animation synthesis as presented in [8].

We use sketches as input to constrain and create the effects animation. Sketch-based methods have been used extensively for animating characters [6, 2]. A method to control physically-based fluid simulation by sketched strokes was also presented by Churi et al. [4]. However, we do not know of any other work that uses sketch-based inputs to generate non-photorealistic effects animation from a database of videos.

3. SYSTEM OVERVIEW

Our method involves a one-time offline phase where all videos in our database are processed to cleanly separate the effect that we are interested in. We then compare all videos belonging to a single effect category with each other. An effects category contains videos belonging to one kind of phenomena like water splashes or explosions. The comparison step is used to compute a similarity matrix, which results in a directed graph between the frames of the compared video clips.

In the online phase, the animator draws sketches on a timeline in our interface. These sketches are then matched to the processed videos in the database. With these matches as constraints, we run a constrained graph search on the graph created in the offline phase. This gives a sequence of frames that are subsequently rendered using the style of the sketches. The animator can add sketches and can repeat this process till the desired effects animation is obtained.

We give further details of the offline and online phases of our system in the following sections.

4. OFFLINE PRE-PROCESSING

We have collected a lot of stock video footage from various sources for water splashes, pouring water streams, open flames and explosions. Examples of these can be seen in Figure 2. These are pre-processed to separate the effect from the rest of the video. This is mostly done manually, as the effects are complex phenomena and are almost always impossible to segment automatically. Pixels in a video frame that do not belong to an effect are marked transparent. Videos belonging to a particular type of effect are labelled with a category label.

4.1 Creating Motion Graphs from Video

For every video category, a motion graph is created by computing a similarity matrix. This matrix captures the similarity between different frames of the videos in a category. The similarity matrix is created using Algorithm 1. The complexity of the algorithm is $O(N_{frames} \times N_{frames} \times n_i \times n_j)$. Here $N_{frames}$ are the number of frames in the videos. $n_i$ and $n_j$ are the number of pixels in frame $i$ and frame $j$, respectively.

Post the computation in Algorithm 1 we move a window across the similarity matrix and save the sum of the values in the window as the final match value. The effect is captured as a filled blob, and the similarity matrix stores the similarity in the shape of these blobs. The sum over a window further accounts for the rate of change in the shape of the blob.

The local minimas of this similarity matrix represent pairs of frames that are very similar to each other.

Each video represents a subgraph in the motion graph. We introduce transition edges between the videos, at the pairs of frames that lie at the local minimas of the similarity matrix. We can adjust the number of transition edges...

![Figure 1: Overview of our method](image)

![Figure 2: Example frames from a (a) splash video and (b) fire video.](image)
Algorithm 1: Algorithm to compute the similarity matrix

1: for $i = 0$ to $N_{\text{frames}}$ do
2: for $j = 0$ to $N_{\text{frames}}$ do
3: Split $\text{frame}_i$ and $\text{frame}_j$ into superpixels of size $20 \times 20$.
4: for each superpixel pair between the two frames do
5: Calculate fraction of superpixel that is filled, $\text{fillfrac}$. This is done by counting the number of non-transparent pixels in it and dividing it by the total number of pixels in the superpixel.
6: if $\text{fillfrac} > 0.3$ then
7: Mark both the superpixels in the pair as filled.
8: end if
9: end for
10: All the filled superpixels in an image form a blob. Align the filled blobs between $\text{frame}_i$ and $\text{frame}_j$ and compute an overlap value, $\text{match}_{ij}$.
11: $\text{similarity}_{\text{matrix}}[i][j] = \text{similarity}_{\text{matrix}}[j][i] = \text{match}_{ij}$.
12: end for
13: end for

in the graph by thresholding the local minimas. Very low thresholds introduce many transitions that are of low quality, while very high cause too few to be introduced, causing issues during graph search. This motion graph is saved to the disk for use during the online phase.

Example similarity matrices can be seen in Figure 3 for various effect categories. Note that a white value shows a better transition point than a black value. The diagonal of the matrix is majorly white as a frame is always similar to itself. It has black patches due to frames in which no blob is detected. For example, the frames before a fire rises or after the fire dies, or when the splash of water disappears under the water surface before rising up again. We assign a similarity value of 0 for such frames by default, in order to avoid transitions on empty (but matching) frames.

5. ONLINE PROCESSING

During online processing, the animator draws the input sketches that are to be used to guide the generated effect animation. These sketches are registered to particular points on the animation timeline. The animator can choose to create a multi-layered effects animation, with different effects in different layers. Each such layer has its own timeline in our system interface. This can be seen in Figure 4.

To start the process the animator specifies the total time of the animation clip to be generated, selects the desired number of layers and the effect category in each layer. The computed motion graphs for the selected categories are loaded into memory. Then the animator clicks at particular instants on a timeline, and draws the key input sketches that depict how the effect in final animation should look at those time instants. These are like keyframes of the animation, except that the sketches themselves will not appear in the final animation. Frames that match the sketch will appear in its place, at the same time instant. After the animator finishes adding all the sketches, each sketch is matched to frames of all the videos in the corresponding category.

We use the blob shape matching method already presented in Algorithm 1 for this purpose too. The top five matching frames for every sketch are shown to the animator, from which she selects the ones most appropriate to her purpose. Each of these matching frames are a node in the motion graph. Now we have to find a path in the graph between each pair of nodes, such that the length of the path is equal to the time difference between the pair of frames on the timeline. This is so because videos are recorded at a particular frame rate and therefore, number of frames directly correspond to the length (time) of the video.

5.1 Constrained Graph Search

We have to find a path between each pair of input sketch
matches. The path length should be equal to the distance between the sketches on the timeline. In general, searching for a path of a specified length in a graph is NP-hard. We employ a combination of search techniques to find the path that is closest to desired the length. This is explained in Algorithm 2.

Algorithm 2 Algorithm to search for a path in a graph closest in length to a given length.

1: for all pairs of consecutive input sketches do
2: Calculate distance $d$ between the pair of sketches.
3: Find the single source shortest path distance, $s_d$ on the graph starting from the first node in the pair.
4: if $s_d \geq d$ then
5: return the shortest path.
6: else
7: Consider a window of size $k$, such that $d-k > s_d$.
8: repeat
9: Use a graph $d$-length path search algorithm to find a path, $p$, for which $d-k \leq \text{length}(p) \leq d+k$.
10: if such a path $p$ is found then
11: return the path $p$.
12: else if $d-k < s_d$ then
13: $k = 2 \times k$
14: else
15: return the shortest path.
16: end if
17: until a path is found.
18: end if
19: end for

The graph $d$-length path search algorithm used for Step 9 in Algorithm 2 is a variant of a standard DFS on a graph. This is presented in Algorithm 3.

Algorithm 3 Algorithm for $d$-length path search in a graph.

1: Let the current search window be $d-k$ to $d+k$.
2: START a DFS on the graph from source node.
3: while all paths of length less than $d+k$ from source node have node been traversed once do
4: if on any path, depth $> d+k$ and the destination has not been reached then
5: backtrack and continue DFS.
6: else if the destination is reached and path length is in search window then
7: return path
8: else
9: backtrack and continue DFS.
10: end if
11: end while

The complexity for Algorithm 2 is $O(V + E)$. For us, it is $O(N_{frames} + E) = O(N_{frames} + N_{frames})$, since only a few additional transitions are added in the graph. Therefore, the final complexity is $O(N_{frames})$, i.e., roughly linear in the number of nodes.

The complexity for Algorithm 3 is $(N_{sk}) \times ((N_{frames} \times N_{frames}) \text{ for single source shortest path}) + (\text{complexity of Algorithm 2}) = (N_{sk}) \times \log_2(s_d - d))$. This equals $N_{sk} \times (N_{frames} \times N_{frames}) \text{log}_2(s_d - d)$, which is $O(N_{sk} \times N_{frames} \times N_{frames})$ since $N_{frames} >> \log_2(s_d - d)$ always. Here $N_{sk}$ is the number of input sketches.

As an option and if the animator wants, we can also invoke a timeout and stop the search after 10 seconds to maintain interactivity of the system. If in this interval no other suitable path has been found, the shortest path is used. After we’ve obtained matches for all the input sketches and have found a path between each pair of matched frames, the motion graph is traversed and the in-between frames are generated. An example of a motion graph is shown in Figure 3. The particular path traversed is shown in red - with the red nodes being the nodes from the original videos and the red lines the transition edges. The frames corresponding to original frames in the videos are re-rendered, while frames corresponding to transition edges added during motion graph construction are generated on the fly by morphing between pairs of similar frames. Both these are explained in the next section.

5.2 Rendering

The colour the animator uses to sketch the effect in the input sketches is used to render the generated frames. In particular, we replace the hue and brightness of the pixels that constitute the effect with that of the colour from the sketch. For the saturation, an average value of the saturation value from the existing colour and sketch colour is used. This preserves lighting details from the video, while renders the frames in the colour style used by the animator. This is done for each layer for a multi-layer effect animation.

All layers of the generated frames are aligned automatically with the location of the effect in the sketched frame, as the location of the effect in the input videos may be different. This lends simple translation invariance to our method.

Now if a transition edge is traversed in the selected path computed earlier, we morph the frames over a window using the Beier-Neely morphing method 1. The window size is the same as used in computation of the similarity matrix and thus every transition edge represents the number of frames in the window. In order to do the morph, we create a morphed image halfway between every corresponding pair of frames in the window. We automatically place feature lines in the blob region representing the effect pixels by considering a bounding box for the blob and considering its edge bisectors as feature lines. Since our images have completely transparent and completely opaque pixels, some of the morphed pixels have an intermediate value of transparency which produces ghosting effects in the morphed frame. We thus post-process these frames to remove the semi-transparent pixels.

Finally for multi-layer effect animations, if we simply draw all layers on top of each other, the boundaries between the layers look out of place in the animation. Thus, we merge the boundaries of the layers with each other. Even in case of single layer animations, we merge the layer boundaries with the background layer. This is done using a gradually varying alpha value in transparency masks and the layers are alpha-composited using these masks.

6. RESULTS

We present examples of rendered frames from the result effect animations that we have generated using our system. The first of these is shown in Figure 6. The first row shows the two input sketches drawn by the animator on the timeline. Eight rendered frames from the generated effects animation can be seen below.

Since we have used stock video footage from various sources,
sometimes these sources have watermarks which we have not removed due to copyright reasons. The final result animations therefore also have some of these watermarks.

The animation results and a recorded session of our system can been seen in the supplementary video submitted with this paper.

A multi-layer effects animation is shown in Figure 7. The inputs are two fire sketches in the fire timeline and a water pouring sketch in the pouring timeline, drawn by an animator. The two bottom rows show frames from an effect animation where the fire is being extinguished by pouring water on it. Note that the shape of the water and fire is taken from the video, while the colours match that of the sketch drawn by the animator. Next we show a water splash animation result in Figure 8 shows the result for a water splash animation. The input sketches show an object falling into a bed of still water, followed by the shape of the splash at various points in the animation. The generated frames are shown below.

Our method is discriminative enough to distinguish between different classes of the same effect. When the animator draws a larger object falling into the water bed and a larger, differently shaped splash, our system generates appropriate animation frames to match the animator’s intent (see Figure 9).

Since our system can work with any kind of video, we show a dam break animation result in Figure 10 that is generated from a set of video clips that were rendered using a physically-based fluid simulations system. There are four input sketches drawn by the animator. The rendered animation includes matches for these inputs as can be seen in
6.1 Timing Results

The time taken for various steps in creating each of the results described above can be seen in Table 1. This is the time taken on a laptop machine with 1.7GHz Intel i5 processor with 8GB of DDR3 RAM and a NVIDIA GeForce 820M graphics card. We parallelized our implementation using OpenMP in many places. The time needed for the one-time generation offline processing phase varies from about 1 minute to 40 minutes, depending on the number of videos in the database. As can be seen from the timing results, most of the time is taken in the matching steps. It varies based on the number of videos of the selected effect category as well as the number of filled cells in each of those frames. An effect such as dam break with lots of water particles takes more time to match than a splash effect, with almost equal number of videos for both of these categories. It can also be seen that our system is very efficient and an animator can generate complex effects animation at near interactive rates.

7. CONCLUSIONS
Table 1: Time taken for individual steps for each result

<table>
<thead>
<tr>
<th>Effect</th>
<th>#Input</th>
<th>#Videos</th>
<th>Matching</th>
<th>Path Finding</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explosion</td>
<td>2 frames</td>
<td>10 videos</td>
<td>21.784 sec</td>
<td>0.003 sec</td>
<td>40.996 sec for 140 frames</td>
</tr>
<tr>
<td>Pouring water on fire</td>
<td>3 frames</td>
<td>6 videos</td>
<td>46.301 sec</td>
<td>0.001 sec</td>
<td>53.037 sec for 300 frames</td>
</tr>
<tr>
<td>Splash</td>
<td>3 frames</td>
<td>14 videos</td>
<td>171.959 sec</td>
<td>0.078 sec</td>
<td>43.525 sec for 357 frames</td>
</tr>
<tr>
<td>Dam Break</td>
<td>4 frames</td>
<td>13 videos</td>
<td>429.248 sec</td>
<td>24.042 sec</td>
<td>82.591 sec for 714 frames</td>
</tr>
</tbody>
</table>

Figure 10: Animation showing a dam break. The first row shows the input sketches, while the bottom two rows show example frames from the rendered animation.

We have presented a method for creating effect animation from pre-processed video data. Our system is intuitive, fast and easy to use and bridges an important gap between hand-drawn special effects animation and computer-generated animation. Our system can handle a variety of effects and requires minimal user input from the animator, while still allowing sufficient control to the animator.

The system has some obvious limitations. It cannot generate any effect for which video data is not present in the database. We plan to perform a thorough user survey of our system to ascertain its usefulness to novice and expert animators. We would also like to extend it to cover other more complex effects.

8. REFERENCES