Mythcomm: A Program for Removing Commercials Quickly

Chris Gray

April 29, 2003

---

1 Introduction: Getting Rid of Commercials

While the advertisements shown by television stations pay for the content that the stations provide, most of us would rather not watch them. This paper describes a system called mythcomm that attempts to make removing them easy through a visualization. We want to do this quickly, since it would be as worthwhile to simply watch the commercials if it took too long.

Mythcomm takes advantage of the visual system's ability to spot anomalous objects quickly from among a large group. To aid the visual system in doing this, it tries to place similar looking objects near each other using clusters.

1.1 Definition of Terms

A *frame* is a picture of the video frozen at a second in time. It can also refer to the amount of time that it takes to display a frame – usually around $\frac{1}{30}$ of a second.

A *scene* is a collection of frames that would be shown in order.

A *program* is the complete video from beginning to end.

A *commercial* is usually an advertisement in a program, though the idea could be extended to any type of scene that the user wants to remove from the program. *Content* is the opposite of a commercial.

When discussing the implementation of the editor, we will use the terms *frame* and *scene* almost interchangeably. This is because in the editor, a scene is represented by its first frame. Also, the term *object* is used to denote either a scene or a cluster.

1.2 Related Work

The field of video editing is certainly not new, and there are many programs with which one can remove commercials from content.

The most common form of video editor is what we will call a "linear editor" (even though they are sometimes referred to as non-linear editors; they act

linearly compared to mythcomm). These present video in the order which it was recorded and give the user a forward key, a backward key, and a mechanism for marking frames. The user then moves through the video using these keys and making marks at the beginning and end of cuts. The system that mythcomm was built upon, mythtv, includes such an editor.

There are also video editors which attempt to remove all the commercials from a program automatically. mythtv has one of these too. In general, these try to detect frames that are completely black or high rates of change between frames. These usually signal the start or end of a commercial. There are also some also try to find the frequency range of the audio, as commercials generally have a range that is much smaller than normal programming in order to seem louder without actually exceeding the allowable volume.

Finally, there is the system described in [1] that uses wavelets to automatically detect areas of high cut rates. While this is not much more interesting than the existing automatic solutions on its own, the method suggests saving some representation of the commercial in something like a hash table, so that it only has to detect each commercial once. This idea distinguishes it from the other automatic solutions.

1.3 Motivation

Given all this previous work, why are we looking for a new solution? Simply put, the existing solutions do not work well enough.

Automatic solutions are plagued by false positives and negatives. For example, a system looking for high cut rates might mistake a music video for a commercial or a slowly moving commercial (such as an infomercial-type commercial) for content. One looking for black frames might think the title cards in Law & Order mark the beginning of a commercial.

With the "linear editors", the false positive problem essentially goes away. However, this comes at the cost of speed. You must search linearly through the entire program at intervals less than the commercial length. Once you see a frame of content followed by a frame of commercial, you must slow down the rate at which you move through the file and essentially perform a binary search until you have found the exact frame where the content stops and the commercial begins. Also, there is no way to take advantage of the self-similarities within programs because of the linear presentation of the video to the user.

The solution from [1] has the flaws of the automatic solutions were mentioned earlier. However, the idea of hashing the video is a good one and will probably be incorporated into mythcomm at some time.

2 Description of Algorithms

We will be describing all the parts of the system, even those not related to the visualization as all of them are necessary for it to work meaningfully and

effectively. However, we will attempt to limit the discussion of those parts that are not visualization related and refer the reader to the bibliography.

2.1 Wavelets

Wavelets are a rather hot topic these days, especially in signal processing circles. Of course, they are not a magic bullet. However, they do provide a convenient metric for measuring the similarity between two frames of video. They are used in two places in mythcomm – as the video is read from the video card, a wavelet transform is done on the frame. If two frames have very different wavelet representations, this usually implies that they are visually very different. This is noted in a file as a scene boundary. Later, when the user starts editing the file, the distance between each pair of scene boundaries is computed in order to cluster the scenes well. More on that later.

The fact that they are wavelets is not all that important, just that there is a metric that somehow captures the notion of how visually similar two frames are. The literature seems to support that wavelets do this well, so they are what we used.

The wavelet transform that we are using is a Haar transform of a 32×32 scaled version of the Y channel of the YUV representation of the frame.

Algorithm 1 Haar wavelet transform

Haar($A, size$)
  if size == 1 **then**
    return
  else
    for $i = 0$ to $size$ **do**
      $A_i = (A_{2i} + A_{2i+1})/2$
      $A_{i+size} = (A_{2i} - A_{2i+1})/2$
    end for
    Haar(A, $size/2$)
  end if

2.2 Clusters

We want to find clusters of scenes so that the user can delete many scenes at a time with a single key stroke. To aid in finding the clusters, we will use random walks on graphs – a method suggested by [2].

A random walk on a graph $G = (V, E)$ with weights $w_{(i,j)}$ is a stochastic process where the probability of transitioning from vertex i to vertex j over edge (i, j) is

$$p_{ij} = \frac{w_{(i,j)}}{d_i}$$

where $d_i = \sum_k w_{(i,k)}$. Now if we put all possible pairs of is and js into a matrix P of probabilities, we can find the probability of visiting j from i in the kth step. It is P^k_{ij} (the (i, j)th element of the kth power of P).

We can use the random walks as a heuristic for finding edges which should separate clusters. We do this by noting that for vertices i and j in the same cluster, the probability of getting from i to k or from j to k in $\leq c$ steps should be roughly the same, whereas if they are in different clusters, this probability should be different. Then, if we use these similarities as weights on a new graph, running the procedure on this new graph should give us an even sharper distinction between vertices which should not be in the same cluster and those that should. The procedure can be iterated a few times, but eventually will go to a steady state.

Algorithm 2 Clustering by Neighborhood Separation

NS(P^1)
  $P^2 = P^1 \times P^1$
  $P^3 = P^2 \times P^1$
  $P^{\leq 3} = P^1 + P^2 + P^3$
  for all i, j **do**
    $R_{ij} = \exp(6 - |P^{\leq 3}_i - P^{\leq 3}_j|) + 1$
  end for
  return R

  for $i = 0$ to 4 **do**
    $Q = NS(Q)$
  end for

The naïve implementation of this algorithm takes $O(n^3)$, but this can be improved if the graph is of constant degree.

So how is this useful to mythcomm? We can make a complete graph where the weights are the similarities between frames as determined by the wavelet metric described above. Then we can run this algorithm on it. Once we have iterated it enough times, we can remove the edges that are below a certain threshold. Then a simple search (BFS or DFS) on the resulting (now disconnected) graph gives us the clusters.

2.3 Layout

Laying out the data collected in a reasonable manner is rather difficult given that the data we are working with only has one meaningful ordinal dimension – time. However, we do have clusters of scenes now and we would like to lay out the clusters so that all scenes in the cluster are close together and so that the clusters overlap each other as little as possible. We would also like to be able to navigate quickly using just the arrow keys, as mythtv is designed so

that it can easily be run on set-top boxes using just a remote-control as input. Finally, the clusters do not really the fit the standard description of things that are layed out by information visualization algorithms. They are trees to some extent (where the depth is always 3), but most tree layout algorithms do not do well in this context. They are by definition not connected in a graph structure because of the way the clusters are found.

2.3.1 A Failed Attempt

Our first try at laying out the clusters involved quadtrees (see [4]). It failed on æsthetic grounds but was rather instructive.

The usual use of quadtrees is to store information about image complexity in graphics. Given an arbitrary quadtree, the layout of the image is completely specified. Thus, specifying a quadtree of the clusters and frames that we want to lay out will automatically lay out the clusters and frames for us in a way that clusters are prevented from overlapping. Also, finding neighbors to respond to the user's requests from the arrow keys is relatively simple using quadtrees.

So what was wrong with it? Suppose the clustering algorithm does a bad job and produces 4 clusters: one with $n-3$ frames and 3 with one frame. Then the best the quadtree layout can do is to have $\frac{1}{4}$ of the window showing $n-3$ frames and the other $\frac{3}{4}$ of the window showing the remaining 3. Obviously, this is a waste of space and looks bad. Also, while clusters are not allowed to overlap, frames will – making it hard to pick out those frames that should be deleted.

2.3.2 Force-based Graph Layout

The general idea of the revised algorithm is to put the clusters back into a graph and then use a force-based graph layout algorithm to lay them out.

The easiest way to think of a force-based graph layout algorithm is to think of each element as electrically charged. The elements that are connected by edges have springs between them. We let these forces (the repulsion of the electricity and the attraction of the springs) act on the elements for a while and then stop them. The resulting layout should be æsthetically pleasing.

Since we are laying out clusters, it is possible for elements to be different sizes. This does not hurt us too much as we can simply think of the *surfaces* as electrically charged and the springs connecting the elements as connecting the centers of the elements.

The exact equations for the magnitude of the force are

$$f_a = \frac{d^2}{len}$$

for the attractive force and

$$f_r = \frac{len^2}{\max(d, \varepsilon)}$$

for the repulsive force. The value d is the shortest between the boundaries of the clusters and len is the desired length between clusters. Notice that all forces cancel out when all clusters are len apart.

To go from a fairly random set of clusters to a laid out graph, we first connect arbitrary clusters into a graph similar to Figure 1. Then we make sure that no two clusters overlap in the obvious manner. Then we run the force-based algorithm described above.

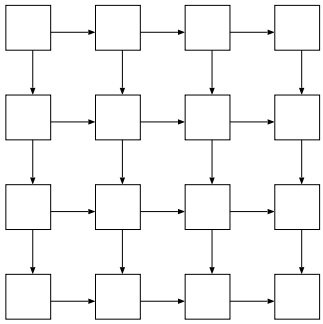

Figure 1: Graph Layout

Since we do not want the frames to overlap, they are laid out on a simple grid. They are easier to lay out than the clusters because they are all the same size.

This layout method is nice for a few reasons. First, finding neighbors is extremely simple. Each node in the graph simply has to keep track of the four neighbors that it has and the ordering should not change when the force-based algorithm is run. Secondly, it is designed to be æsthetically pleasing. Third, and maybe most importantly, it is space efficient. Larger clusters get more room than smaller clusters.

2.3.3 Cut Indicator

Because of the very non-linear nature of the way the data is laid out and the fact that the user can not tell how long each scene is just by looking at the first frame, we need to give the user some indication of the effects of deleting a scene (or cluster). This is achieved by having a bar at the bottom of the screen that encodes which parts of a program have been deleted. Whenever the user deletes an object, the parts of the program that the user has deleted change from blue to red in the bar. Thus the user can test the effects of an action before it is actually performed. This is useful if the scene detector does a bad job and leaves a very long scene that started during a commercial. In this case, the user can see that deleting this scene has a big impact and undo his action.

2.4 Interaction

As mentioned previously, mythtv is designed for interaction by remote-control, so the number of keys used must be kept to a minimum and the mouse must

not be used. Thus, mythcomm uses only the arrow keys, D, C, and the space bar. As expected, the arrow keys change the currently selected object to the neighbor in the direction of the key. The space bar toggles between selecting clusters and selecting frames. The D key deletes the current object and the C key clears the previous deletion (it is essentially an undo operation).

Initially, the user is presented with a screen showing the layed out clusters where the top left frame of the top left cluster is selected. The selection is shown by drawing a red border around the frame. Top left was chosen to take advantage of natural biases based on reading habits. The user can then navigate inside the cluster using the arrow keys, delete frames, or toggle to the cluster level. If he switches, a green box is drawn around the cluster and he can then move between clusters.

When the user deletes an object, it is removed from the screen. If the user requests for it to be replaced (using the clear operation), it is put back. Otherwise, he continues navigating as before.

After deleting an object, the graph layout algorithm is rerun on the remaining graph. This makes the graph more compact than it had been before. To maintain the user's idea of where objects are, the objects undergo an animated transition.

3 Results / Evaluation

The current implementation is already very usable. Times taken to edit certain programs (by the author) are shown in Table 2.

Program	mythcomm	linear
Simpsons	38 s.	52 s.
Tonight Show	1 m. 7 s.	1 m. 16 s.
The Daily Show	1 m. 50 s.	1 m. 37 s.

Figure 2: Usage times to edit programs

The times in table 2 do not include the preprocessing time for mythcomm because all preprocessing can be done once before the user requests it and saved for later. This cuts an average of 30 seconds off mythcomm's time.

With mythcomm, there are sometimes commercials that get missed and you are more likely to see the first and last few seconds of every commercial break than with a linear editor. Thus, mythcomm usually takes slightly less time while slightly sacrificing quality. As mentioned in the introduction, though, time is fairly critical in this application.

It should be noted that the time taken in mythcomm seems proportional to the number of scenes found – a number which is highly dependent on the wavelet module. On the other hand, the linear editor always takes time proportional to the length of the video. Thus, properly tweaking the wavelet module can substantially improve the time taken.

3.1 Problems

One problem with the layout approach taken is that the angles between connected clusters are not preserved. Thus, pressing the right arrow key might move the selection up or down a large amount. In some sense, this is good because it allows the representation to be more compact, but it could obviously be confusing to the user. A couple of options we are considering to ameliorate this are drawing all the edges of the graph or simply drawing the edges from the currently selected cluster. This second option could also be done if the edges deviate more than a threshold from their expected angle.

Also, some of the features mentioned in this paper do not actually exist in the implementation yet – for example, the force based layout scheme is very challenging to implement well and efficiently and seems rather delicate. Thus, an approximation of it is currently being used.

4 Examples

In Figure 3, we see the original layout used for the Simpsons episode whose time was mentioned in the evaluation section. It looks fairly standard; all the frames and clusters fit fairly easily.

In Figure 4, the layout is shown after editing the program. The cut indicator has been circled. Looking closely, one might see that there is a frame that does not look like The Simpsons. In fact, this frame comes just at the end of the commercial and was undeleted after the cut indicator showed that deleting it would cause a major change.

5 Conclusion

This tool actually solves a specific problem at a level at least on par with most of its competitors, though there are still some issues (such as the fact that the scope of the implementation does not yet match the scope of this paper) to be worked out.

That said, we do not see a major future for this tool anywhere besides commercial editing. The automatic scene detection does not give enough control to a video editor to make cuts precisely where he wants them. It might be useful as a starting point, but a linear editor will always be necessary in the video industry. However, in the field of commercial editing, where absolute control is not really needed (seeing 20 seconds of commercial in an hour long program is worth it to save 2 minutes editing time), this program does show promise.

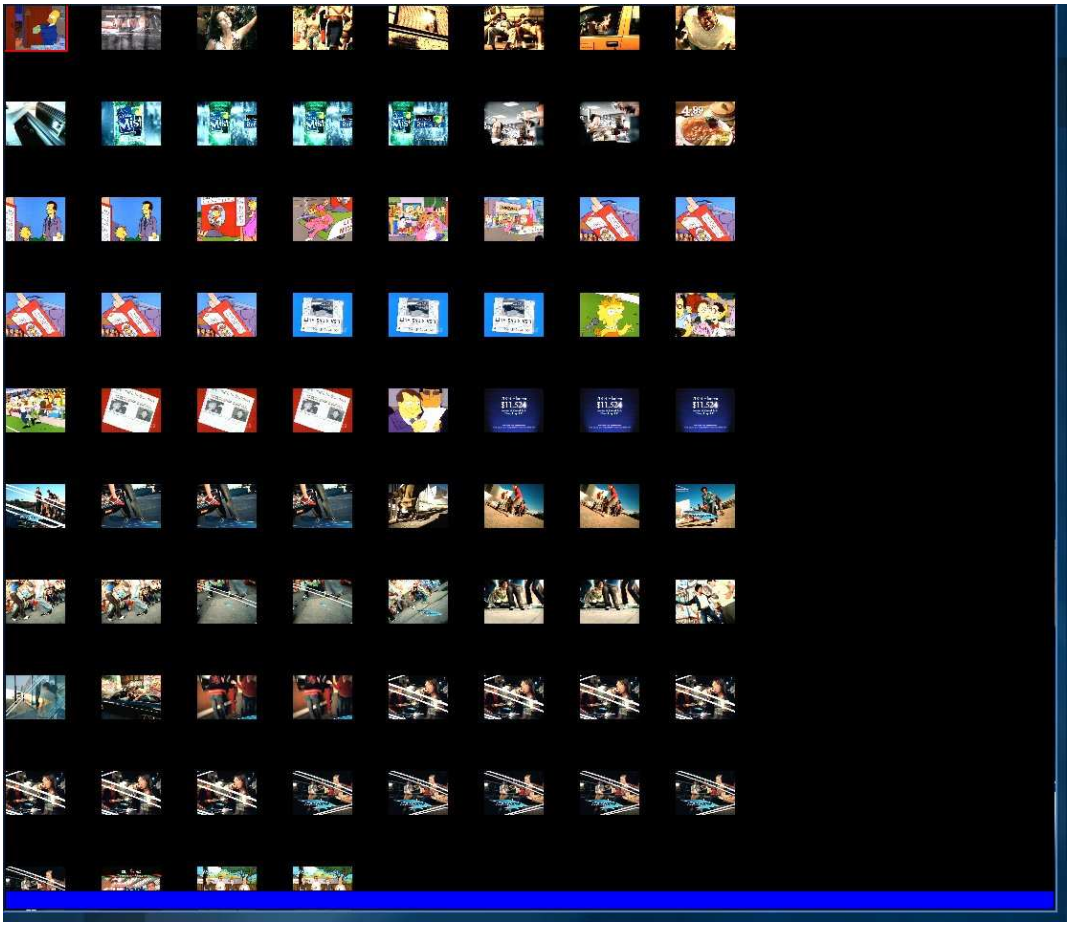

Figure 3: The Original Layout

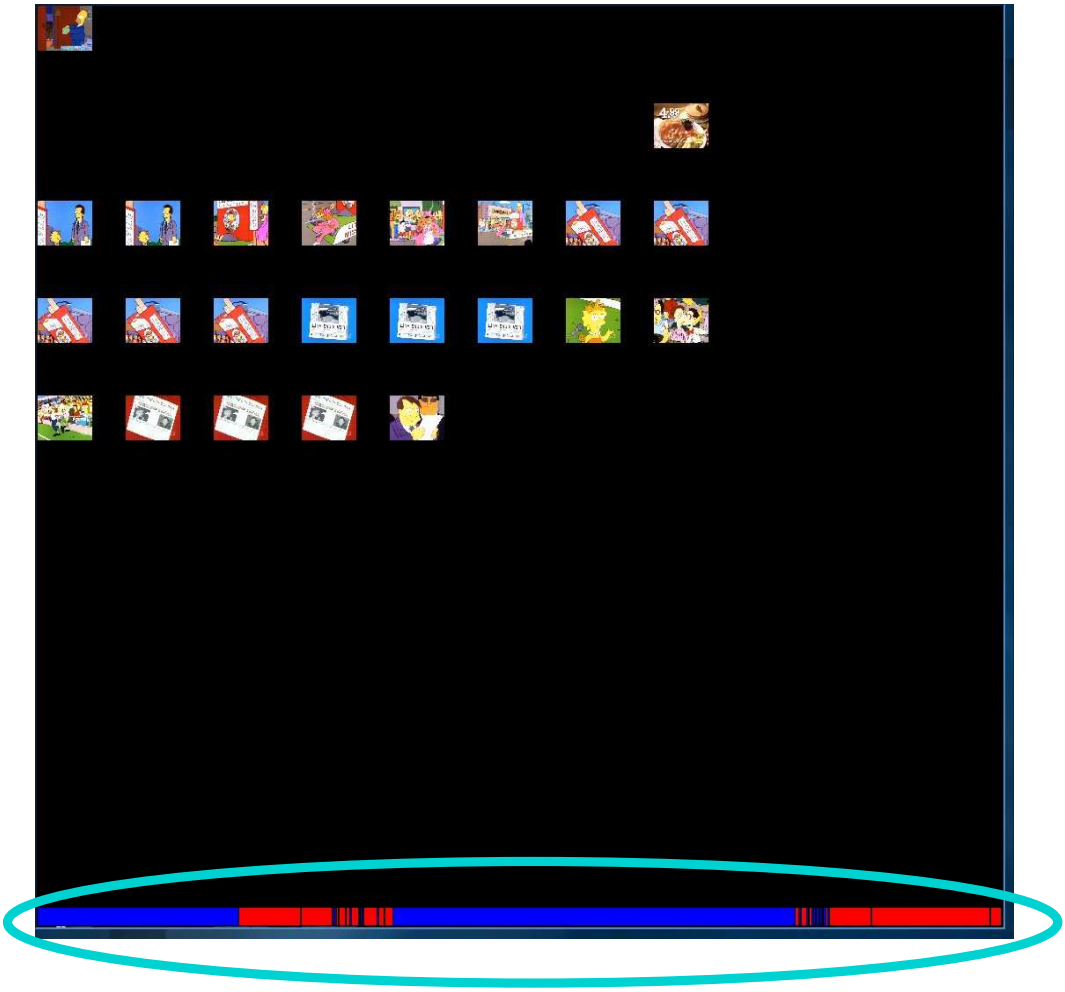

Figure 4: After Editing

References

[1] Wen, X. *et al.*, "Wavelet-Based Video Indexing and Querying for a Smart VCR", Princeton University, http://www.cs.princeton.edu/∼wxd/draft.html

[2] D. Harel and Y. Koren, "On Clustering using Random Walks", Proceedings of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01), Lecture Notes in Computer Science, Vol. 2245, Springer Verlag, pp. 18–41, 2001.

[3] D. Harel and Y. Koren, "Drawing Graphs with Non-uniform Vertices". Proceedings of Working Conference on Advanced Visual Interfaces (AVI'02), ACM Press, pp. 157-166, 2002.

[4] Foley *et al.*, *Computer Graphics: Priciples and Practice*, Addison-Wesley, 1996.

[5] Richards, I., Mythtv, http://www.mythtv.org