Extraction of Most Significant Frames from MPEG Data

Jeremy W. Sheaffer

December 11, 2002

Why?

Why do we want to extract significant frames?

What good can it do us?

One obvious use is to produce storyboards, or sequences of frames that tell the story of the whole video, to allow Internet users to preview a movie before committing to downloading it.

Another less obvious application could perhaps be a smart fast–forward feature in digital VCRs.

What do we mean by Most Significant Frames?

How do we determine a frame's significance?

We consider a frame significant if it has bearing on the course of the video. Obviously, the decision of a frames significance is somewhat arbitrary. Typically there is very little change between two frames within the same scene, but over the course of a scene, things could change severely.

We need a method with which we can winnow out a set of frames which will describe the entire story.

Methodologies

Many methods have been explored in the literature for key–framing video sequences. Among the more complex is one involving integration of the 2–space projection of the curve in time of the Euclidean N–space features of the data set. It's rather ugly and quite unintuitive. Occam's Razor suggests we go for a simpler approach.

Last semester I worked with Dr. Latecki and Dr. Vucetic on a similar project to this. The end goal was the same. The means: clustering on the feature space using k–means. Over the feature space as a whole, the results were really quite poor.

Global Clustering on Mr. Bean and Wallace and Gromit

48

361

1708

2056

2092

2332

127

178

588

669

736

757

You can't really tell what is going on here.

Applying some shot detection, and a little bit of filtering to avoid overlaid, ghosted frames causing the appearance of two simultaneous scene changes, I next applied k–means on individual scenes. The results were much better. One might even argue that they were quite good.

Mr. Bean frames with shot detection and k–means clustering

979

1536

2016

2093

2159

2203

2240

2335

2371

Walter and Grommet frames with shot detection and k–means clustering

22

230

308

398

434

482

510

548

600

649

665

712

753

773

780

This data set is from a clip with far more action than Mr. Bean, as well as noise from image ghosting and low frame rate.

But any algorithm that involves k–means *or* shot detection is not truly an algorithm—it is a heuristic, because the shot detection is arbitrary, and the clustering algorithm is randomized. Though the method produces consistently good results, it does not consistently produce identical results, which is a very undesirable trait. Furthermore, it seems to work very poorly on small data sets.

This semester, Dr. Latecki asked me to use his new algorithm, the same which several of you used, and attempt to solve the same problem. Furthermore, it is undesirable to use any form of shot detection, as this introduces an arbitrary factor into the calculation of key–frames.

Running Dr. Latecki's algorithm on a machine with a 933MHz processor and 512MBs of RAM, and an average utilization of 97%, and depending of the parameters passed to the algorithm, on the Mr. Bean data set which contains 2381 63 dimensional points, the algorithm often requires over two hours to complete.

I ran the algorithm in a for loop:

The obligatory MatLab code.

```
files = {'wg.bft'; 'mrbeantu.bft'; 'BladeRunner.bft'; ...
         'HouseTour.bft'; 'Mov00085.bft'; 'Mov1.bft'; ...
         'Mov3.bft'; 'kylie.bft'; 'seciurity1.bft'; ...
         'seciurity7.bft'; 'lion.bft'};

for i = 1 : length(files)
  out = [char(files(i)) '.out'];
  f = fopen(out, 'w');

  for j = 30 : 15 : 105
    for k = .03 : .03 : .3
      [kf] = latecki_global(char(files(i)), j, k);
      fprintf(f, 'for %f and %d mnp, frames are:', k, j);
      fprintf(f, '%d, ', kf);
      fprintf('\n\n');
      clear kf;
    end
  end

  fclose(f);
end
```

It took over a week

And the last data set, lion.bft, I have never managed to complete. It takes quite a bit longer than Mr. Bean, and the system administrators get annoyed.

The sub–routine `latecki_global()` is a front–end to `firstauto1()`, our own Venugopal's implementation of Dr. Latecki's clustering algorithm, which normalizes the data set before passing it on to be clustered, and then sorts the results before returning.

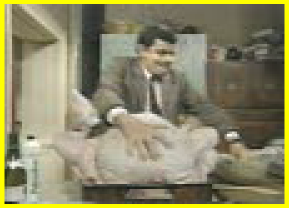
The large range of parameters tested in the nested loops in the previous slide were an attempt to determine some parameters for which the algorithm works consistently well for many data sets.

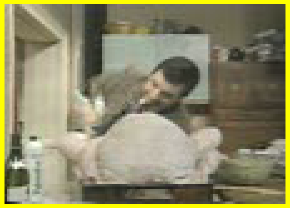
I do not believe I achieved this end.

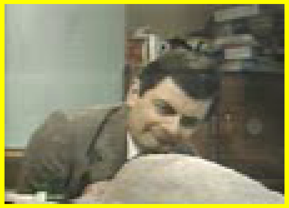
For the larger data sets, like Wallace and Grommet, and Mr. Bean, even small variations in the parameters tend to yield vastly different results. For small data sets, results are consistent, but choosing parameters for which results are good in most or all data sets seems unlikely.

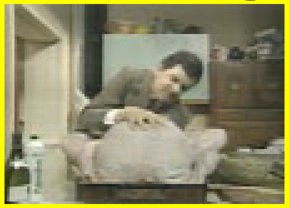
The quality of the results with the Mr. Bean data are similar to those shown above, as are those achieved for Wallace and Grommet:

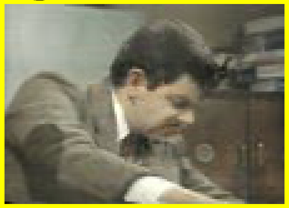
Mr. Bean with Dr. Latecki's clustering algorithm

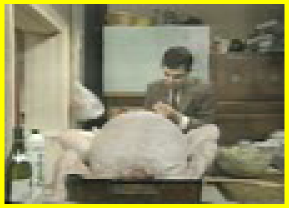

809

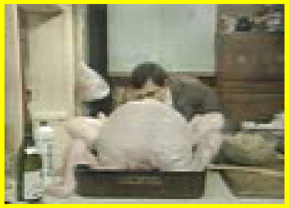

988

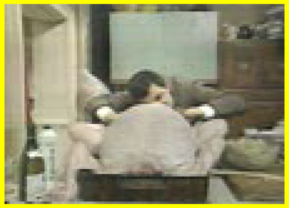

1004

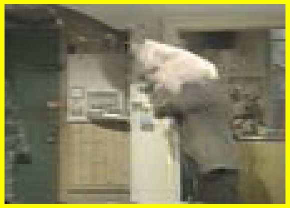

1267


1330


1608


1853


1928


2296

This set is the result of a 75 frames per cluster minimum and 12% delay parameter. It seems to miss some important points, like the woman at the door, but what is not obvious from these images is that it did very well with respect to choosing frames from different shots.

Wallace and Gromit with Dr. Latecki's clustering algorithm

EVENING POST
GROMIT GETS LIFE
12

33

233

243

345

461

531

608

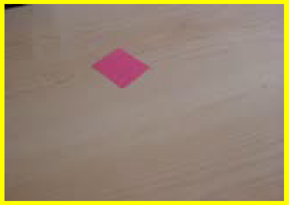
693

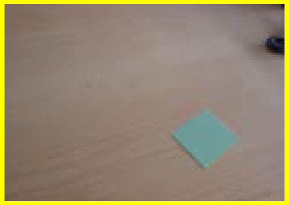
752

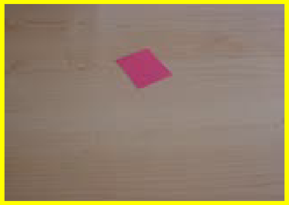
I feel that this set is actually quite good. It has all the major points of the story. This clip is less than $\frac{1}{3}$ the size of Mr. Bean. This set used the parameters of 30 frames per cluster minimum and 12% delay.

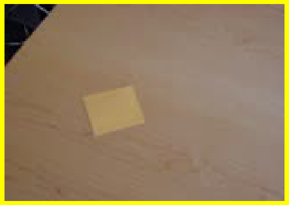
On small data sets, with few or no scene changes, this algorithm works very well. The k–means based algorithm faired very poorly on such data. Dr. Latecki's algorithm consistently returns results similar to these:

Mov1: 30 minimum frames per cluster and 21% delay

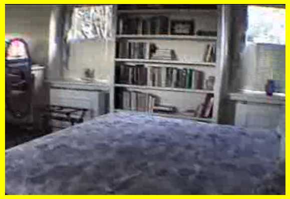

107

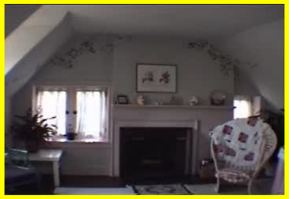

221

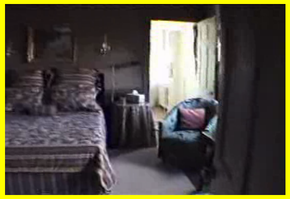

287

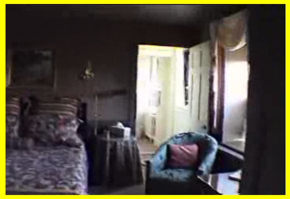

359

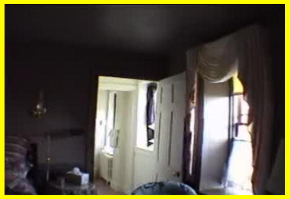
HouseTour: 30 minimum frames per cluster and 12% delay


174


318


458


487


538

Security 1: Many different parameter settings yielded this result

32

128

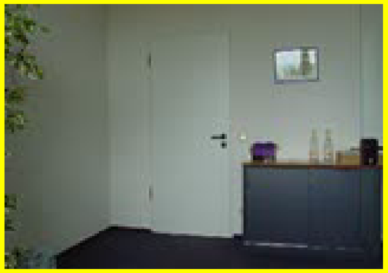
347

Security 7: 30 minimum frames per cluster and 15% delay factor

1

122

206


329

Conclusions:

We have developed a method to extract significant frames from MPEG video streams.

The method seems to work well in instances of small movies, but in instances of large movies, the values of the parameters to the clustering algorithm far too greatly affect the result of the computation—The chosen frames give a feel for the movie.

However, the method is very computationally intensive and slow, thus rendering it, at least in the near future, impractical for real–time applications such as smart fast–forward.