THE UNIVERSITY OF MANCHESTER

COMPUTER SCIENCE AND MATHEMATICS

THIRD YEAR PROJECT REPORT

Video Textures

Author:
Struan MCDONOUGH

Supervisor:
Dr. Aphrodite GALATA

January 29, 2019

The University of Manchester
School of Computer Science

Video Textures

Struan McDonough, supervised by Aphrodite Galata

Abstract
Photos and videos provide a means for capturing the essence of a moment or scene. However, both are vastly different, with videos capturing a finite amount of time. Shödl et al. presented a form of media which displays a static image with dynamic qualities in their 1999 paper ”Video Textures”[1]. By analysing a video sequence, key features can be extracted, and a structure can be formed to determine how we play this video. This project aims to implement a system which can produce these Video Textures, while exploring the Mathematics needed to describe what a visually appealing result is. This project then goes on to experiment with different types of video scenes, to identify those features which produce the most aesthetically pleasing results.

Keywords
Animation, image-based rendering, multimedia, natural phenomena, texture synthesis, video-based rendering, video-based animation.

Contents

1. Introduction

1.1 Motivation

The ability to take a video and have it play indefinitely could have applications in the advertising world. An increasing number of websites are using dynamic video backgrounds to have an immediate visual impact on their customers. These videos tend to be at least a minute long, to avoid the eventual repeat from becoming too obvious, however longer videos can require a significant amount of data to be represented, and these videos aren't very effective towards customers with a slow internet connection. Video Textures[1] would only require the client to download a few seconds worth of footage, and this footage would go on to produce an endless video, which does not appear to repeat.

Other potential applications include usage in video games. Rendering a dynamic three-dimensional area through a portal (such as a door or a window) is a relatively expensive operation. If the portal is not traversable by the player, using video textures would achieve the same visual effect, with a much lower impact on the performance of the game.

The authors of the Video Textures paper also suggest the use in a website for an actor or model, where a series of poses

could be ordered procedurally. The author of the report you now read believes further applications are limited only by imfagination.

1.2 Project Scope

The Video Textures[1] paper introduces a system which takes a video source input, and then produces a series of successive matrices. Each of these matrices are then manipulated by an analysis process to produce the next, before generating a final matrix which can then be used to synthesise a video. The Video Textures paper then suggests several different methods of interpreting this matrix to produce different playback effects, each of which are of varying implementation complexity.

The project presented in the paper you read also aims to create a ready-to-use interface which can be used by an untrained individual, to allow the general public to produce their own video textures.

1.3 Project Goals

The primary focus for this project was to produce a system described in a scientific paper. The Video Textures[1] paper described ideas and some of the mathematical formulae discovered to create the systems, but never described the specifics of implementation. The initial goals were to achieve the following:

1. Learn how to use a new programming language, by working through a project.
2. Successfully implement the analysis steps described in the paper.
3. Successfully implement the synthesis steps described in the paper.
4. Perform some additional rendering steps such as cross fading or morphing.

These goals will be revisited later, as a means of evaluating the success of the project.

1.4 Paper Structure

The rest of this paper is organised as follows. In section 2, concepts and ideas are introduced which are required to understand the following sections of the paper. This section also discusses similar work to Video Textures. Section 3 discusses the design choices, and the overall architecture of the system. Section 4 discusses the technical details of how the results were achieved, and also describes the implementation of the graphical interface. In section 5, the results are discussed and evaluated. Finally, in section 6, the project as a whole is evaluated in the conclusion, and further work and extensions are discussed.

2. Background

Before detailing the project itself, we shall explore the concepts which make the foundations of Video Textures. We will also look at similar work uncovered by Shödl's team.

2.1 Concepts

2.1.1 Textures

Textures are repeating patterns have been in use in art for thousands of years, dating back to woodblock printing from China in 220AD. The key idea for textures is producing a template to ensure that the design on opposite borders "line up", such that when several instances of the template are tiled, they produce a larger seamless result, expressing the characteristics of the template. In the computer graphics world, these templates are represented by bitmaps.

2.1.2 Graphics Interchange Format

The Graphics Interchange Format is a standard for representing raster images developed by CompuServe in 1987[2]. The GIF format is portable and widely supported, leading to widespread adoption on the Internet in the 1990s. GIF also supports very simple frame-by-frame animations, which can loop after the final animation frame has been displayed. Following similar concepts to textures, if the frame sequence at the beginning and the end of the GIF are similar, there is a seamless transition, allowing a small animation to appear to continue forever.

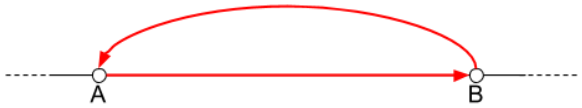


Figure 1. A looping GIF takes the structure of a primitive loop.

One can take a video, and identify points A and B which are visually similar to each other. Using this, one can trim away any of the video outside of this region, and subsample the video into animation frames. While the GIF plays, once it has reached the final frame B, it will then immediately returns to the beginning and plays frame A. We will call this structure a primitive loop.

In a similar fashion to graphics textures, the image seen at both ends of the loop can be taken to be very similar to one another, allowing a seamless "texture" to be developed.

2.1.3 Video Textures

Shödl et al. realised that videos can often have many pairs of points in a video which are visually similar, each of which can give rise to a primitive loop. These video loops can overlap, and gives us an interesting way of traversing the video.

One can start traversing at the frame 0, and continue as normal until the video has reached a frame which has a similar frame earlier in the video. The traverser then randomly chooses whether to continue playback as normal, or to follow the loop to an earlier part of the video. The traverser will

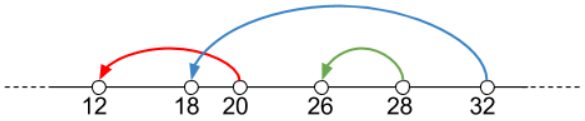


Figure 2. A video can have many similar frames, each of which provides a transition.

never go past the final loop, as this would allow the video to terminate.

In order to be able to traverse a video texture, a data structure describing the links between each pair of frames is required. If there are fewer links available, a list is the most suitable structure, however matrices are better suited to more dense data sets. For this project, matrices are used.

2.1.4 Dead-ends

Under certain situations, a set of transitions can give rise to what we call a dead-end. A dead-end is where the traversal of the video reaches a small subset of transitions, which have no overlap with the rest of the set. This means that the playback is prevented from returning to the rest of the available transitions.

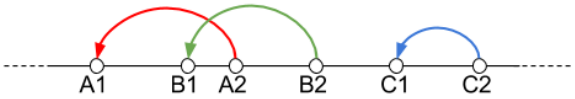


Figure 3. C2 → C1 exist in their own subset.

Figure 3 shows us an example of a dead-end. Should the video playback traverse past B2, then the playback is isolated to only play the C2 to C1 loop. Since the playback will choose not to terminate at the end of the video, we are left in the situation where C2 to C1 loop is the only available environment of execution, removing the random element from the traversal. People are very good at spotting patterns and will quickly realise that this footage is contrived.

The series of transitions available to us can be represented as a di-graph. From a graph theoretic standpoint, a dead-end is any subgraph which is not strongly connected to the rest of the graph.

2.2 Similar Work

Shödl et al. identify that the closest work to their own is ”Video Rewrite”[3]. In this paper Bregler et al. look at image synthesis ”to create automatically new video of a person mouthing words that she did not speak in the original footage”. Training data is provided to the system and then reordered to match an audio track of phonemes.

A similar technique to Video Rewrite was showcased in ”Forrest Gump”, where footage of Nixon is edited to say new words. However, this editing was done by hand using cinematic special effects, whereas the Video Rewrite paper provides an automatic method.

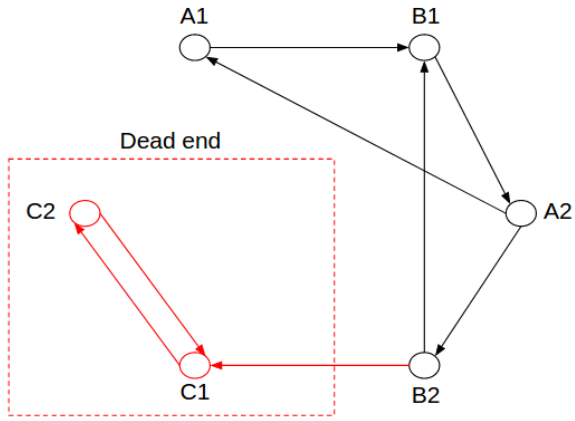


Figure 4. Frame transitions can be represented as a di-graph. It is not possible to traverse from C1 to B2.

Figure 5. Nixon’s synthesised words, as seen in Forrest Gump (1994).

3. System Overview & Methodology

In this section, decisions made while planning the project are discussed, and justifications for each decision made. The structure of the project is also outlined.

3.1 Overview

There are three main components of the system. The first, which shall be called the “Video Analysis” stage, takes the video, and produces data structures which can be read by other parts of the system. The second component is the “Video Synthesis” stage, which uses different techniques to read the data structures produced by the analysis, and output a video to be viewed by the user. The third and final component is the Graphical User Interface, which allows the user to interact with the system in an intuitive way.

The following section details these identified stages at a high level, describing the pipeline from input video to the output Video Texture. In section 4, each of these stages are described in greater detail.

3.1.1 Video Analysis

The role of the analysis stage is to identify the transitions which will be used by the video synthesis stage. We begin with a video input, and we first compare every frame with

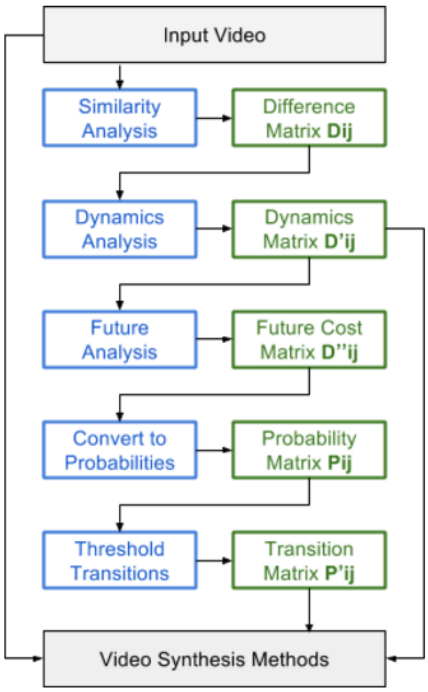


Figure 6. An overview of the Video Analysis stages.

every other frame in the video, in order to build a "Difference Matrix", which in shorthand we will refer to as (D_{ij}). Each cell in the matrix represents how similar the two frames are, with lower values representing higher similarity.

We then take this distance matrix, and extract the dynamics of the scene. Frames can be similar, but failing to take into account the motion of the video can lead to visible discontinuities. The result of this is the "Dynamics Matrix" (D'_{ij}).

After this, we perform future-cost analysis, which takes a given transition and finds if it will cause the video execution to either terminate, or get trapped in a dead-end. This produces the "Future Cost Matrix" (D''_{ij}). We then turn these modified transition costs into the "Probability Matrix" (P_{ij}), and threshold this to produce a matrix of possible transitions - the "Transition Matrix" (P'_{ij}). For each video input, the preprocessing steps only need to be performed once.

We also ensure that each of the analysis steps have an independent output. This gives us the flexibility of removing certain analysis steps, and evaluating their effects on the final result.

3.1.2 Video Synthesis

This component synthesises the video clip, and generates the new video based on a set of rules, each of which determine the order to play the transitions of the original clip. This uses the matrices generated in the analysis step, and produces a video output. For this project, the following four video synthesis methods have been implemented:

Random Play: At any given frame, either play the next natural frame, or choose a random one.

Repeat Play: Identify the largest primitive loop found in the video clip, and traverse using this only.

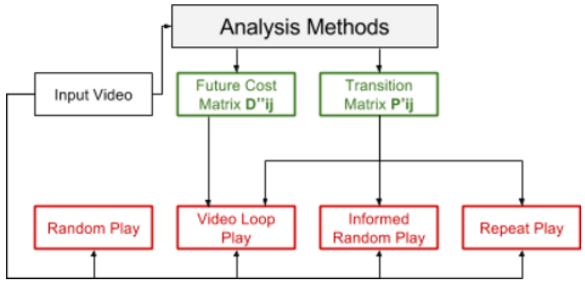


Figure 7. An overview of the Video Synthesis Methods. The initial state is the Input Video.

Informed Random Play: Use the transitions found in the video clip, and traverse them randomly.

Video Loop Play: Build up a table of compound loops, using a dynamic programming technique, then determine the playback order using this table.

3.2 Configuration

Each of the analysis steps contains formulae, which describe the mapping from one matrix to the next. These formulae contain variables, which provide trade-offs between different results. Due to the different characteristics of scenes, changing these variables can potentially lead to improved results.

Configuration for this project is a Python file, which is formatted in such a way where knowledge of Python is not needed in order to make meaningful changes.

There are alternative formats available which could be used for configuration, such as XML or JSON, however given the simplicity of Python, it wasn't believed to be necessary to use such formats, as XML and JSON would have required the use of further libraries to manipulate.

```
# ====================================
# INTERESTING PARAMETERS
# ====================================

# Input {cars, clock-hands, clouds,
    manchester, mothecombe, row}
inputName = "manchester"

# Similarities
skipFrames = 1

# Future Costs
qualityExponent = 2

# Thresholding
thresholdValue = 0.9 # Lower will allow
    more transitions
useLocalMaxima = True
useThreshold = True

# Preserving dynamics
```

```
adjacentFrames = 2

# GUI
colouringType = "Rainbow"  # Island,
    Rainbow or Gray

# Video Loops
buildTable = True

... further options are available ...
```

4. Implementation

So far, an overview of the system has been presented. Now each component will be presented detail, explaining the Mathematics and algorithms required to describe the system.

4.1 Choice of Technologies

There's always more than one way to solve a problem, however some approaches are better than others. In the current software landscape, there are several programming languages and libraries which can be used to produce a system. Each of these have their own strengths and weaknesses, and are better suited for certain projects. In this section, the choices made for the project will be described, and justified.

4.1.1 Programming Language

The analysis component of the project involves extensive use of matrix manipulation. Each of the analysis steps produces a matrix, which can then be read by the next analysis step. Certain matrices are also made available to the synthesis stages. Hence, it was believed to be appropriate to choose a programming language equipped with comprehensive matrix libraries.

Execution Speed: The speed the language processes at isn't critical for the analysis stage, since the output isn't expected to be in real time. However, the synthesis stage would need to be able to read and manipulate video in real time, hence the language chosen would need to be relatively fast.

Development Speed: The third consideration is speed of development, and how easily it would be to experiment with different ideas. Languages such as MATLAB and Python have very large built-in libraries which perform a lot of common tasks. This means that less code needs to be written, and debugged. Another consideration is that the C languages also require the programmer to take into account memory management, which is an additional factor which adds to development time.

Evaluation: Given these requirements, the use of MATLAB, Java, C++ and Python was investiaged. Python was chosen due to its extensive matrix library: NumPy, and also that it is very quick to develop in. This would allow the project's development energy to be focussed on experimenting quickly iterating through different ideas, as opposed to spending considerable time on the intricacies of the implementation. Python was also chosen due to it being the language this report's author was least confident in, who also wanted to learn a new skill.

4.1.2 Libraries

The project has two main tasks which require the use of libraries:

1. Reading the video file, and presenting frames to the user.

2. Manipulating the matrices in the preprocessing steps.

The library for reading the video file would need to be able to extract frames from a video and be able to provide meta-data about the video, such as frames-per-second and total frames in the video. This information is needed to map the current execution time to a frame. It would also need to be relatively fast, especially while rendering the video texture's output.

Two contenders for video manipulation were identified, GStreamer and OpenCV. GStreamer describes itself as a "library for constructing graphs of media-handling components"[5]. GStreamer is written in C, but provides bindings for C++, Python and Ruby. OpenCV provides the same functionality, and provides bindings for C, C++, Java, Python and MATLAB/Octave. Under recommendation of this project's supervisor, the author opted to use OpenCV, since there are experts within this project's department familiar with the library, who would be able to offer support, should it be needed.

In terms of manipulating matrices, Python provides NumPy, which provides all the tools needed for accessing and modifying matrices, while also providing tools for reading/writing them to and from CSV files.

4.2 Video Analysis

4.2.1 Determining Similarities Between Frames

We begin by determining a measure of similarity between every pair of frames in the video. To achieve this, we read every frame in the video, and convert it to grayscale. This grayscale image is then stored as a $N \times M$ matrix, $\mathcal{I}_i$ where N is the height, M is the width of the video frame, and i is the frame number. Each value of the cell represents the brightness at that pixel, with 0 being black, and 255 being white.

We go through each pair of frames, and determine the matrices for each. We then determine the Euclidean norm for each of the matrices, and then determine the difference between these two norms.

Each of these results is placed in a $N \times N$ matrix, where N is the number of frames in the video (D_{ij}). This matrix is then saved, to be used in later processing steps.

$$D_{ij} = ||\mathcal{I}_i - \mathcal{I}_j||_2 \tag{1}$$

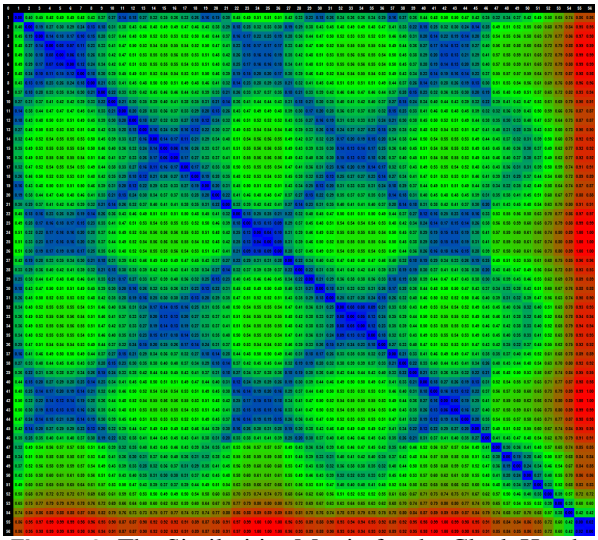


Figure 8. The Similarities Matrix for the Clock Hands example. This matrix was produced using this project's matrix visualisation tool, described later in this report.

4.2.2 Preserving the Dynamics of the Scene

Scenes have a natural flow of motion. If we take the example of a pendulum, we expect it to perform motion periodically, and hence we have expectations of how it should move. It will always swing from left to right, and back again. If the video contains several such periods, then there will be several frames where the pendulum is in the same position. However, at this position, the pendulum could be traveling from either the left or from the right. Jumping to a random previous frame would mean that the transition could send the pendulum off in the wrong direction, which is very unnatural. Many different scenes have similar motion, which we need to preserve, to give a convincing result.

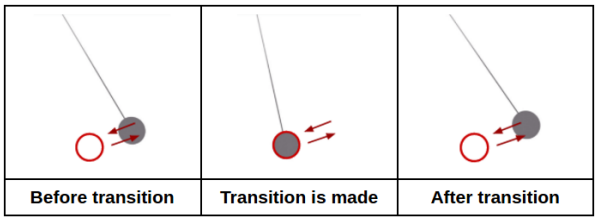


Figure 9. The need to preserve dynamics.

The paper suggests two approaches, the first is to match velocities, where the optical flow is computed at each frame. The authors of the paper did not attempt this approach, since the performance of optical flow is dependent upon a visible, repeating texture being available. They also found this approach to be "brittle".

The second option is to use a weighted window. In order for a frame to be considered similar to another frame, we also take into account frames adjacent to each, weighted with a Binomial Distribution such that those closer to the desired transition have a higher weight. We match subsequences as opposed to just frames. This subsequence matching is found by filtering the similarity matrix with a diagonal kernel, of weights $[w_{-m}, \cdots, w_{m-1}]$:

$$D'_{ij} = \sum_{k=-m}^{m-1} w_k D_{i+k,j+k}. \tag{2}$$

The value of m is usually chosen to be 1 or 2. m requires frames to be available both before, and after the given frame to be evaluated. Frames close to the beginning of the video may not have the required previous frames, hence will not be evaluated. This produces a "border" m cells thick which will not be evaluated. Hence D'_{ij} will be of size $(N-2m) \times (N-2m)$. Higher values of m lead to less available video to form transitions.

After filtering the matrix using this kernel, the transitions which relate to a change in direction are assigned a higher distance, which we shall now refer to as cost. A higher cost means that the transition is of lower quality, and less likely to be chosen during playback.

We can see the periodic motion of the pendulum in D_{ij}. The bottom left to top right diagonal bright patches represent transitions where the pendulum would suddenly change direction. As m increases, we see that these transitions are filtered out, resulting in D'_{ij}.

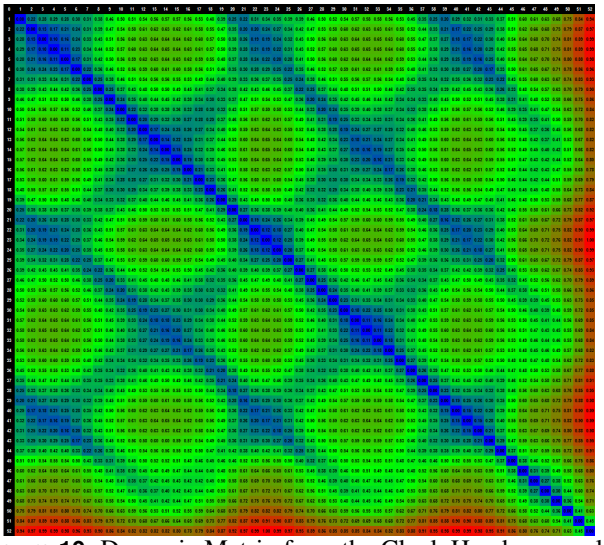


Figure 10. Dynamic Matrix from the Clock Hands example, generated with $m = 1$. Compare this to D_{ij} and notice a 1 cell border has been removed.

4.2.3 Calculating Future Costs and Avoiding Dead Ends

Earlier in this paper, we introduced the concept of a dead-end. Identifying the entry points to these dead ends and removing them is an important step in the analysis, to allow the creation of a convincing scene.

One measure we can take to remove dead-ends is by introducing a future cost between any pair of frames. The future cost takes into consideration the transitions which will

be available beyond the current one, and will provide a less favourable score if the found future choices are limited.

We define D''_{ij} to be the anticipated future cost of a transition between frame $i-1$ to frame j. This represents the average cost of future transitions. We define D''_{ij} by summing over all future anticipated costs:

$$D''_{ij} = (D'_{ij})^p + \alpha \sum_k P''_{jk} D''_{jk}, \tag{3}$$

where

$$P''_{ij} \propto \exp(-D''_{i+1,j}/\sigma). \tag{4}$$

p is a constant which controls the trade-off between few, high quality transitions and multiple, low quality transitions. a is a constant which represents the relative weights of future transitions, and must be $0 < a < 1$, however the system converges quickly if $0.99 < a < 0.999$.

This is a pair of simultaneous equations, which can be solved iteratively, by evaluating D''_{ij} and P''_{ij}. Each evaluation brings the system closer to convergence, and we terminate evaluation when the system reaches a point where further evaluations no longer affect the matrix. However, this technique is slow to converge.

We can improve this algorithm by noticing that in (4), that as $\sigma \to 0, P''_{ij}$ will tend to 1 for the best transition, and 0 for the worst. This allows us to rewrite (4) as

$$D''_{ij} = (D'_{ij})^p + \alpha \min_k D''_{jk}. \tag{5}$$

This equation is known as Q-Learning[4], and is used frequently within the reinforcement learning community. It treats the matrix as a graph, with transitions being weighted edges, to find the best traversal. We can also be selective about which rows in D''_{ij} are updated at each step. The lowest cost path often involves a transition from a frame near the end of the sequence, this cost can be propagated forward. We initialize with $D''_{ij} = (D''_{ij})^p$ and define

$$m_j = \min_k D''_{jk}. \tag{6}$$

Working from the last row, to the first, and vice versa, we compute

$$D''_{ij} = (D'_{ij})^p + \alpha m_j \tag{7}$$

and update the mj entries using equation (6).

4.2.4 Translating Costs to Probabilities

When we traverse the video, at any point during playback we must know the next frame to play. Since the traversal has a random element, we present the transitions in terms of probabilities. To map the costs to probabilities, we use the exponential function

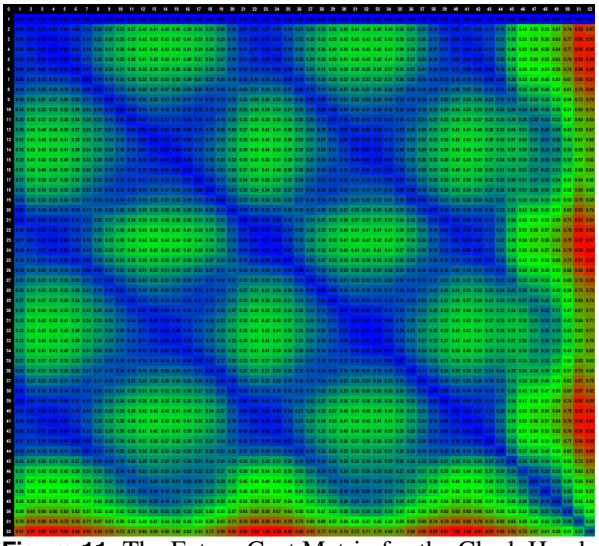


Figure 11. The Future Cost Matrix for the Clock Hands Example. Notice near the bottom right, the main diagonal becomes less intense.

$$P_{ij} = \exp(-D_{i+1,j}/\sigma). \tag{8}$$

The effect of this function is that the higher the cost of the transition, the lower the probability generated. σ controls the emphasis of the mapping: a lower value of σ emphasises the best transitions, whereas a higher value allows a greater range of different transitions. This maps D''_{ij} found in the previous stage to P_{ij}.

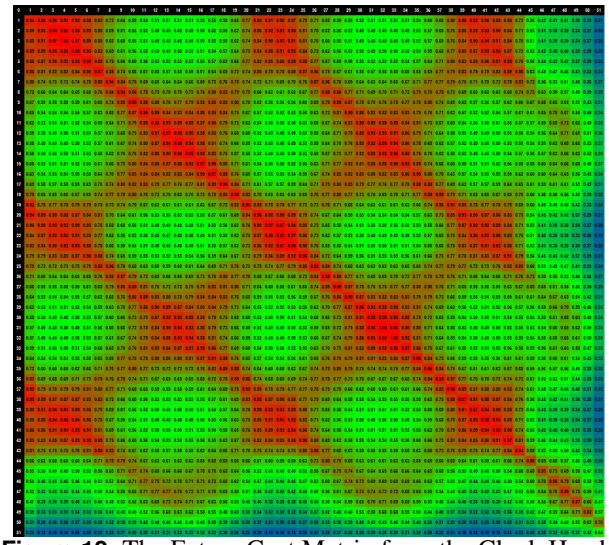


Figure 12. The Future Cost Matrix from the Clock Hands example is mapped to probabilities.

4.2.5 Choosing the Best Transitions

The previous steps will produce a reasonable set of transitions, however we can control the trade-off between the number of transitions, and the quality of these transitions. Hence we prune the set of transitions to a certain extent, which can be controlled by a parameter by the end user.

There are two main methods which allow us to reduce and improve the set of transitions available:

1. Select only the local maxima in the given probability matrix.
2. Set the probability of any transition below a certain threshold to 0.

Although it is possible to use these paradigms independently of one another, the best results were yielded when they were used together.

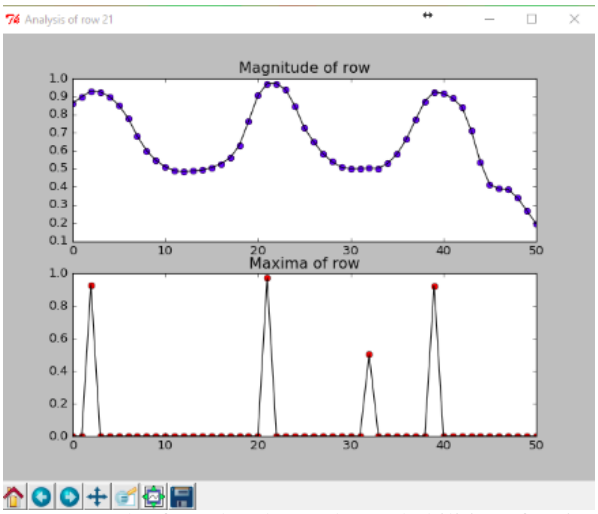


Figure 13. The first plot shows the probabilities of a given row, the second identifies the local maxima.

4.3 Synthesis

4.3.1 Random Play

Random Play is a very simple procedure, and does not make use of any of the data prepared in the analysis steps. At any given frame i, choose $i+1$ as j, such that $j < i$. The output is very choppy and doesn't capture any essence from the scene.

Random Play is not interesting in itself, however provides a useful contrast to the Informed Random Play, to illustrate the visual performance gained when taking into account desirable transitions.

4.3.2 Loop Play

Loop Play takes the transition matrix and identifies the longest loop available. It then discards every other transition, and traverses based on this primitive loop. This method can then be used to produce an effect which is similar to that exhibited by GIFs, however it performs the seamless effect automatically, without any manual editing.

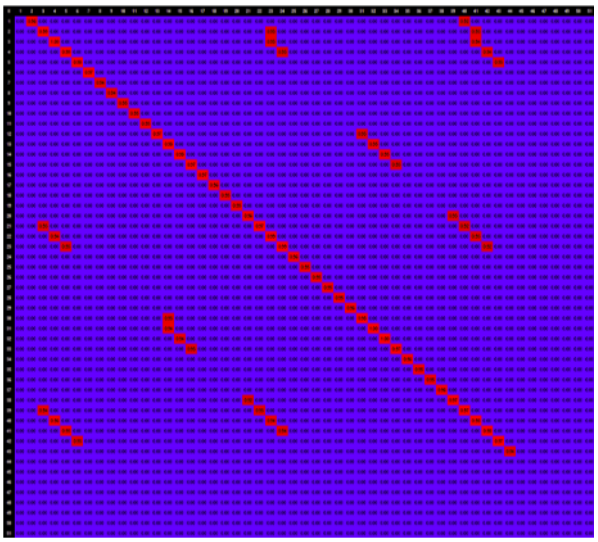


Figure 14. The Probability Matrix after thresholding.

4.3.3 Informed Random Play

The Informed Random Play makes use of the transition matrix, P_{ij}. We define i to be the source frame, and j to be the destination frame. After displaying frame i, the next frame j is chosen using P_{ij}. In most cases, $P_{i,i+1}$ is the largest probability, since the distance between i and $i+1$ is close to 0. However, this is not always the case, since during our future cost analysis, we remove transitions which lead to a dead end. If there are no backwards transitions after $i+1$, then this transition would have been filtered out.

4.3.4 Video Loops Play

The compound video loop method is the most intricate to describe. We reintroduce the idea of a primitive loop, which connects our source frame i to our destination frame j. Since our loops take the video backwards, $j < i$, and we say that the primitive loop has a range of $[j,i]$. The cost of this loop is defined as the filtered distance D'_{ij}.

We can combine primitive loops to create compound loops. Two primitive loops must have an overlapping range for a compound loop to exist. If there is no overlap, then it is not possible to play the second loop after the first. This compound loop has a range which is the union of the ranges of the loops which make it. The length and cost of the compound loop is the sum of the original lengths and costs. Compound loops can have any number of primitive loops, and these primitive loops may be repeated.

The authors of the Video Textures paper suggest a dynamic programming method for building up these compound loops. This algorithm however doesn't permit forward loops, such that $j > i$, hence we may only consider backward loops, such that $j < i$.

There are two stages to producing video loops, the first involves choosing the set of optimal loops, and the second schedules this set of found primitive loops.

Finding the Optimal Loops

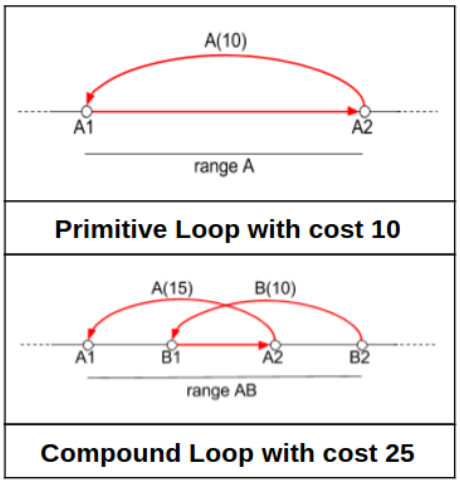
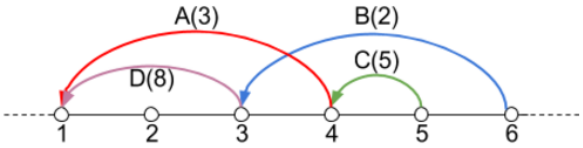


Figure 15. Examples of a Primitive and Compound loop.

The most straightforward way to find the available optimal loops of a length L is to enumerate through every multiset which has length L, filter out the combinations which are not compound loops, and then choose the best ones. However, this approach is exponential for the number of transitions which need to be checked.

An alternative approach is to use a dynamic programming approach. We build up a table, which has L rows and N columns. Each column represents a primitive loop (transition) identified in the analysis stage, and L is the maximum length of a compound loop we will consider.



Length	A(3)	B(2)	C(5)	D(8)
1			C(5)	
2			C^2(10)	D(8)
3	A(3)	B(2)	C^3(15)	
4	AC(8)	BC(7)	BC(7)	D^2(16)
5	AD(11)	BD(10)	BC^2(12)	

Figure 16. A transition structure with its associated optimal compound loop table.

The algorithm finds the best compound loop of the given length containing the primitive loop identified at the top of the column. The cells contain a list of the primitive loops which constitute the compound loop, as well as the total cost.

The algorithm works by traversing the table, updating each row in turn. For each cell, we have the following process:

1. Examine every compound loop of shorter length in the same column,
2. Combine these found loops with compound loops from columns whose primitive loops have ranges that overlap that of the column being considered,
3. The combination with the lowest total cost becomes the new entry.

For each of the $L \times N$ cells examined, the algorithm needs to check all smaller lengths in the same column $(L-1)$, and for each of these smaller lengths, check against potential overlaps in the other columns $(N-1)$. Hence we have an average complexity of $O(L^2N^2)$.

This stage is technically a preprocessing (or analysis) step, since it doesn't have to be processed each time the video is run. Hence, it is sensible to store this generated table to a file, so it can be read later.

Choosing an order in which to play the loops

After determining which primitive loops are in the lowest cost compound loops, we need to order them in such a way which leads to forming a valid compound loop. Shödl et al. present the following algorithm to achieve this:

1. Schedule the transition at the end of the sequence as the first to be taken.
2. When we remove this transition, we can be left with disjoint, overlapping ranges. In our example Frame j is always contained in the first such set and we schedule next any transition from this set whose source frame occurs after j.
3. Repeat step 2, removing the transitions $i \rightarrow j$ until there are no more primitive loops left in the first range.
4. Schedule any primitive loop in each of the following disjoint ranges, repeating from step 2.
5. Continue from step 2 until every primitive loop has been removed.

The complexity of this algorithm is $O(n^2)$, where n is defined to be the number of transitions in the given compound loop.

4.4 Graphical User Interface

During the early development of the project, the only way to execute the algorithms was to use the command line, and remember the specific commands for both the analysis steps, and synthesis steps. The aim was to have the system to be intuitive to use, such that the user wouldn't have to be taught how to create a video texture. Hence it was decided that there was the need to create a launcher, a window which appears when the program is started, offering options to the user.

Further motivation for developing a GUI is to be able to visualise the output of each of the analysis steps. Every matrix visualisation presented so far in this report has been taken from this GUI tool. The raw matrix files don't offer an intuitive way of interpreting the output, and makes it difficult to judge

whether the preprocessing algorithm had been implemented correctly.

The GUI attempted to solve these two problems. Firstly, by providing a launcher, the system allows the different processes to be started in a familiar way. Secondly, by providing several visualisation windows, the system provides a better means of evaluating output.

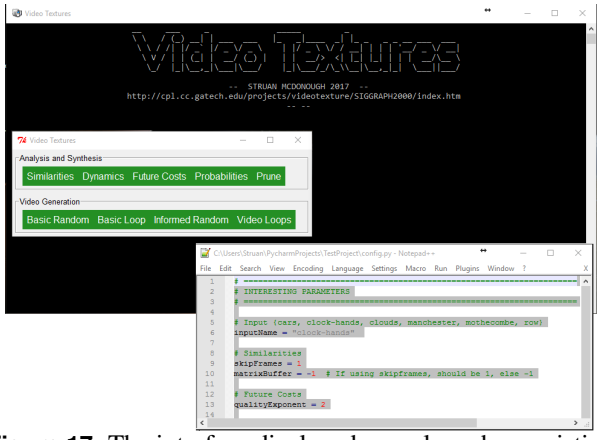


Figure 17. The interface displayed upon launch, consisting of the output window, launcher, and configuration window.

4.4.1 The Launcher

The Launcher contains simple controls, which allows each of the scripts to be executed. Since some of the analysis steps are dependent upon the output of previous steps, the button for each step is disabled until the previous step has been executed.

The Launcher also provides buttons for the four synthesis methods, which when clicked, display the appropriate video output on the user's screen.

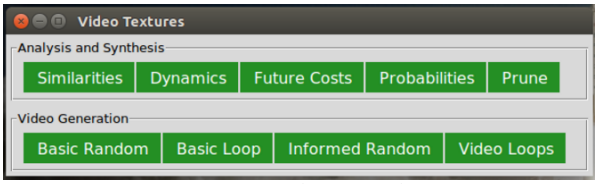


Figure 18. The Launcher.

4.4.2 The Matrix Viewer

The Matrix Viewer reads the matrix output from each of the processing stages, and displays the variables at each cell. It also colours each of the cells in a relative heatmap, such that the highest value is red, while the smallest value is blue.

Clicking on any of these cells will show the two frames which are being compared, which is a useful tool to evaluate if the computed distance is consistent with our expectations.

Clicking on the first entry in any row will display a chart plotting the row. Beneath this row, the result of a local maxima calculation is also presented.

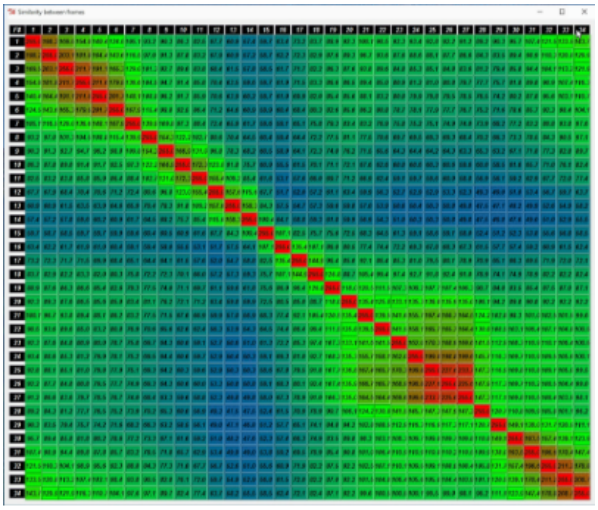


Figure 19. The Matrix Viewer.

Figure 20. The Frame Comparison Tool.

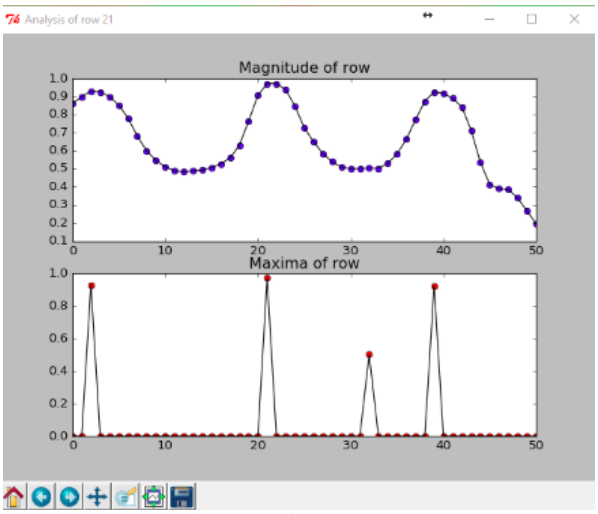


Figure 21. Visualisation of local maxima thresholding.

5. Results & Evaluation

To evaluate the Video Texture system, different scenes were specifically chosen to have different qualities. Some were chosen due to their periodic motion (Clock and Rowers), others were chosen to see the distinction between a static and

variable camera position (Rowers vs. Clouds), and some were chosen due to their apparently random motion (Beach and Candle).

5.1 Clock

For this 2 second clip, no subsampling was performed. This example is useful for both evaluating the future cost and dynamics analysis stages. Since the pendulum moves periodically, it is very obvious if the natural dynamics of motion are preserved or not. For this example, there were no discontinuities in the direction of the pendulum.

The future cost analysis stage also appeared to be functioning correctly. In the source clip, a pair of hands appear which would naturally lead to a dead end. This section of footage however is isolated, and does not appear in the output.

This example is featured in the Video Textures paper, along with the matrices from each of the analysis steps. This is useful as it allowed the comparison of this project's results to that of Shödl's team.

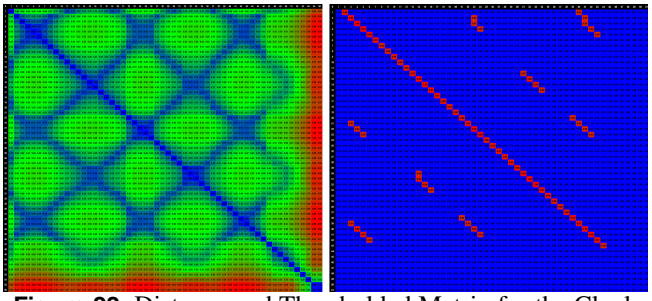


Figure 22. Distance and Thresholded Matrix for the Clock Example.

5.2 Cars

This 4 second clip came from a timelapse published to Vimeo, a video sharing website. A timelapse is a video of a scene which has been sped up dramatically. Due to the high speed of the footage, it is very difficult to identify the jumps made. Since the camera is static, it was found that the threshold had to be set far higher, to 0.8 account for the high similarity between frames. This example produced a very convincing result.

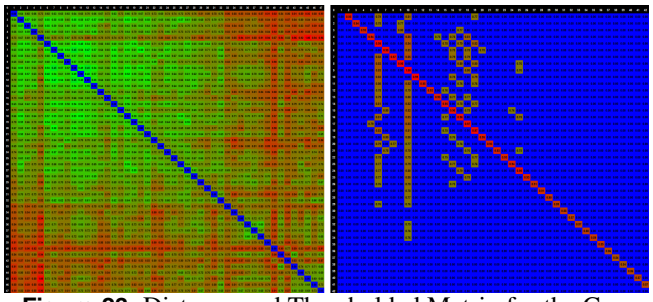


Figure 23. Distance and Thresholded Matrix for the Cars Example.

5.3 Rowers

A 91 second video of a pair rowing was taken, and subsampled at every second to produce 91 sample frames. The rowing motion is periodic, so it was hoped this would produce a successful texture. However, the future cost analysis isolated the first five seconds from the remainder of the film, hence not producing a believable result.

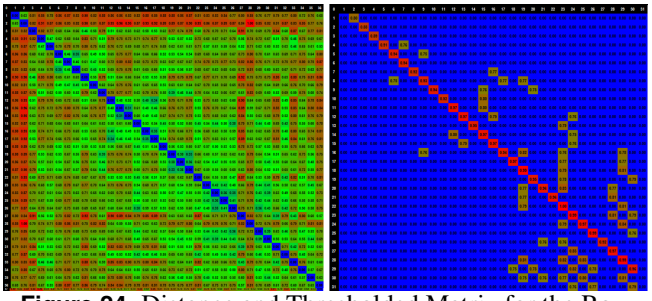


Figure 24. Distance and Thresholded Matrix for the Row Example.

5.4 Billboard with Clouds

This clip was taken from the same time-lapse video as the cars clip. We have different light illuminations of the billboard, depending on if the sun penetrates through the clouds. This example had far more noticeable discontinuities as opposed to the car clip, since the clouds in the background would flick around in an unnatural pattern. The author believes with the application of crossfading, the clouds would become more blurred, but produce a much more convincing result.

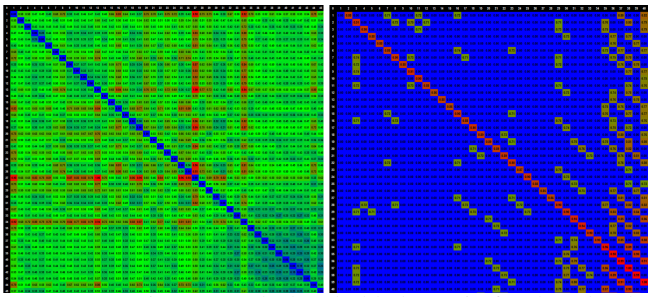


Figure 25. Distance and Thresholded Matrix for the Clouds Example.

5.5 Beach

The author took this 4 second clip with their phone. After subsampling at every twentieth of a second, it gives quite an interesting effect. The whole output was quite choppy, but consistently so. It also preserved the wave dynamics, meaning that the waves would crash before new ones were introduced. Overall, it gave a very pleasing effect.

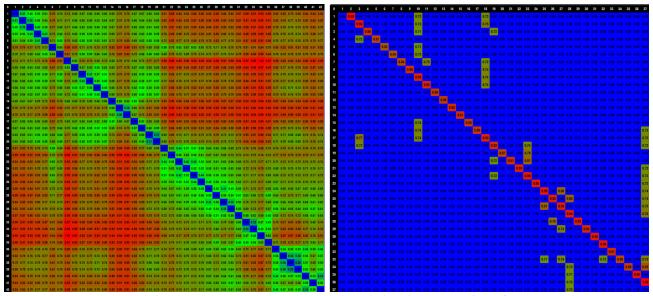


Figure 26. Distance and Thresholded Matrix for the Beach Example.

5.6 Manchester

This 7 second clip was taken from Vimeo. The results are very similar to both the clouds and cars example, with very few visible discontinuities. It was found that if the threshold was low, the higher number of available transitions caused a frame-rate drop.

Increasing the level of thresholding leaves less available transitions, and hence a higher frame-rate during playback.

The system performed the best for input which had a static frame of reference i.e. the camera being held still. Those clips which had predictable motion such as the clock, performed very well, since there existed pair which are close to identical, meaning less visible transitions. The scenes with faster

playback (Road, Clouds, Manchester), implicitly masked the transitions and produced convincing results.

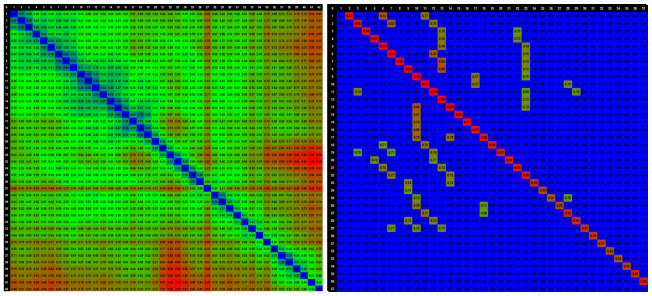


Figure 27. Distance and Thresholded Matrix for the Manchester Example.

6. Conclusion

This project lended itself to be a great union between Computer Science and Mathematics. Having had little experience bringing Mathematical ideas into the domain of Computer Science, the author found it was satisfying to see both sides of their degree come together. The project itself was a success (when evaluated against the stated goals), and it produced meaningful output which can easily be extended to provide an online interface, to allow the general public to produce their own video textures.

6.1 Evaluation against the Project Goals

As stated earlier, the project goals were identified to be the following:

1. Learn how to use a new programming language, by working through a project.
2. Successfully implement the analysis steps described in the paper.
3. Successfully implement the synthesis steps described in the paper.
4. Perform some additional rendering steps such as cross fading or morphing.

In terms of the first goal, this report's author had very little experience with Python before beginning this project, and had no experience using OpenCV or programming to solve Mathematical problems. Now, the author feels confident using Python for larger projects, debugging programs, and structuring a larger project in the language. The remainder of the goals were also met, with the exception of the additional rendering steps. As the project developed, different synthesis methods were experimented with, which not described in the Video Textures paper. The project also saw the building up an easy to use GUI, giving far more insight to the results than what otherwise would have been possible.

6.2 Further Work

The results from this project can certainly be improved. In the Video Textures paper, Shödl, et al. used morphing and cross-fading to reduce the visible discontinuities during the transitions, and it is believed that the testing clips which do not have a static camera would benefit the most from this approach.

Shödl's team also introduced the concept of motion factorisation. This is where the original video is split into regions, where the perceived motion is independent in each region. Each of these regions are analysed independently, allowing for far more variation in the scene, while decreasing the number of frames needed. These regions can either be determined automatically or manually, and can be of any shape.

The author of this report also believes that this technology has the scope to be made available online, hosted in a website. By allowing users to upload their own clips and manage the parameters, the website could provide a player, which could then be embedded in the user's own works.

Figure 28. Examples of Motion Factorisation. The first example splits the scene into two regions, the second has five defined regions.

Video Sources

All videos used in this project are available online for the reader's interest. All clips were accessed on the 9th of April, 2017.

1. **Clock:** http://b.gatech.edu/2pD0bCH
2. **Cars:** https://vimeo.com/90275165
3. **Rowers:** http://bit.ly/2oBJ2eV
4. **Candle Flame:** http://b.gatech.edu/2nP4dvp
5. **Billboard:** https://vimeo.com/90275165
6. **Beach:** http://bit.ly/2pmoCEU
7. **Manchester:** https://vimeo.com/23877263

References

[1] Shödl, Arno et al. (2000). Video Textures. Available from: http://www.cc.gatech.edu/gvu/perception//projects/videotexture/SIGGRAPH2000/videotex.pdf. [Accessed 9 April 2017].

[2] CompuServe Incorporated. (1987). Graphics Interchange Format: A standard defining a mechanism for the storage and transmission of raster-based graphics information. Available from: https://www.w3.org/Graphics/GIF/spec-gif87.txt. [Accessed 9 April 2017].

[3] C. Bregler, M. Covell, and M. Slaney. (1997). Video rewrite: Driving visual speech with audio. Computer Graphics (SIGGRAPH'97), pages 353–360.

[4] GStreamer. (2017). GStreamer: open source multimedia framework. Available from: https://gstreamer.freedesktop.org/. [Accessed 9 April 2017].

[5] OpenCV. (2017). OpenCV library. Available from: http://opencv.org/. [Accessed 9 April 2017].

[6] L. Kaelbling, M. Littman, and A. Moore. (1996) Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4.