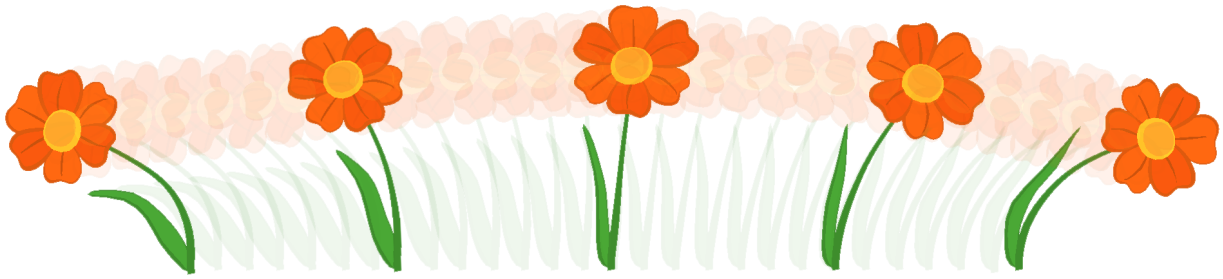
Uniting Cartoon Textures with Computer Assisted Animation

William Van Haevre, Fabian Di Fiore and Frank Van Reeth

Hasselt University
Expertise Centre for Digital Media
transnationale Universiteit Limburg
Wetenschapspark 2, 3590 Diepenbeek, Belgium
e-mail: {william.vanhaevre; fabian.difiore; frank.vanreeth}@uhasselt.be



Abstract

We present a novel method to create perpetual animations from a small set of given keyframes. Existing approaches either are limited to re-sequencing large amounts of existing image/video data, or to interpolating vector based drawings.

Our approach benefits from several ideas and techniques from video textures, computer-assisted animation and motion graphs. It combines the re-sequencing of existing material with the automatic generation of new data. Furthermore, the animator can interfere with the animation process at each arbitrary moment.

First, a given set of keyframes is used to automatically generate a set of in-betweens. The amount of in-betweens required, depends on a distance metric preventing possible visual discontinuities. Next, an optimised cost graph is derived from the generated frames, indicating for all keyframes how many steps are required to travel from one keyframe to another. Finally, by rearranging the generated sets of in-betweens according to the cost graph, new animations can be synthesised from the generated data.

The resulting animations are smooth, broader than the input data and require no postprocessing.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.4.0 [Image Processing and Computer Vision]: General—Image processing software

Keywords: computer animation, structured 2D animation, cartoon textures, automatic in-betweening

1 Introduction

Motivation. Traditionally, 2D animation production has been a labour-intensive artisan process of building up animated sequences by hand. Most work and hence time is spent on drawing, inking and colouring the individual animated characters for each of the frames.

In smaller-scale productions where resources are sparse and animators have to work more independently (due to the absence of dedicated programmers or assistants), this time-consuming work can be reduced by replacing parts of the animated scene with short repetitive animations or endless animations reusing given frames.

Employing existing computer assisted approaches, animators roughly can choose between two extreme options. They either have to create endless looking animations in the traditional way (e.g. frame by frame [Blair 1994; Patterson and Willis 1994]) or in a fully automatic way (e.g. revert to video textures techniques [Schödl et al. 2000; Campbell et al. 2002; de Juan and Bodenheimer 2004]). The latter category which is based on video textures delivers satisfying results but heavily depends on large incremental data sets and cannot cope with visual discontinuities. The first, on the other hand, is based on simple interpolating schemes and requires only a very small set of input data. However, the amount of interaction requested from the user is often too tedious and time-consuming.

Objective. Our objective is to provide a method that facilitates the creation of such endless animations or video loops for smaller-scale productions. From a small amount of input data, animators should be able to easily and interactively create perpetual animations of repetitive motions, and allow the integration of these animations seamlessly into the rest of their artwork. As an exam-

ple, the image of the flower displayed in the inset, shows 5 hand drawn flower positions from which a large amount of new cartoon frames are derived. An interactively created endless animation of this flower can be used to become part of the background scenery in a vast number of cartoon animations.

Approach. Contrary to other approaches [Schödl et al. 2000; de Juan and Bodenheimer 2004], the user only supplies a very small set of extreme poses (i.e. less than 10 frames vs 100s of frames). By re-sequencing input data (which does not need to be preprocessed to remove inadequate material) and synthesising new art from it, a large amount of new animation data is provided to the animator. Based on automatic in-betweening of keyframes, incremental changes are made to the data resulting in smooth transitions without visual discontinuities. No actual preprocessing or postprocessing is required to improve the quality of the resulting animations, preventing possible loss of data.

In order to achieve our goals, our approach combines ideas and techniques from video textures, computer-assisted animation and motion graphs. From video textures creation, we borrow the idea of using a distance metric to estimate the similarity of all pairs of keyframes provided by an animator. Based on computer-assisted traditional animations techniques, in-betweening is employed to extract new data from the input, according to these calculated distances. This data can be represented in a graph similar to a motion graph [Kovar et al. 2002], on which graph walks are constructed using a simple, goal based algorithm. This way, new interactive animations can be synthesised.

Paper structure. This paper is organised as follows. We start with an overview of related work on the topic of this paper. Then, we give an overview of the animation pipeline used for the creation of our targeted animations. Next, The most important steps from the analysis part of this pipeline are examined in more detail, bringing us to the part where we describe how new animations are synthesised from the available data. We end with some clarifying results, our conclusions and topics for future research.

2 Related Work

Our approach benefits from several ideas and techniques from computer-assisted traditional animation, video textures and motion graphs. In this section we briefly elaborate on related work in these fields.

2.1 Computer-Assisted Animation

One of the most obvious methods for generating new animations is creating in-between frames by interpolating between two or more given frames. In Computer-Assisted Animation (CAA), there are essentially two types of interpolating systems: shape-based [Burtnyk and Wein 1971; Reeves 1981; Sederberg and Greenwood 1992; Kort 2002] and skeleton-based [Burtnyk and Wein 1976; Shapira and Rappoport 1995; Sederberg et al. 1993]. However, in-betweening in traditional animation, is not just interpolating between key drawings. When drawing the in-betweens, animators utilise their background knowledge of the physical rules of the world, their expert knowledge of when to bend or ignore these rules, and the emotions they intend to evoke by the animation.

Bregler et al. use capturing and re-targetting techniques to track the motion from traditionally animated cartoons and re-target it onto new 2D drawings [Bregler et al. 2002]. By using animation as the source, similar new animations can be generated. This approach leads to very impressive results, but unfortunately some drawbacks prevent it from being used extensively. The re-targetting process is very dependent on a good choice of the source and target key-shapes which one has to select and draw manually. The animator has to watch carefully that the chosen key-shapes cover the entire cartoon space (the entire range of possible poses). Furthermore, the creation of the target key-shapes — these shapes replace the source key-shapes — is a very tedious task since each source key-shape requires a target key-shape to be drawn manually.

Rose et al. presented an inverse-kinematics methodology exploiting the interpolation of example-based motions and positions [Rose III et al. 2001]. The key issue of their system is to allow an artist's influence to play a major role in ensuring that the system always generates plausible results. Starting from a small number of example motions and positions, an infinite number of interpolated motions between and around these examples are generated. This methodology is highly focused on positioning articulated figures and therefore does not lend itself to traditional 2D animation.

2.2 Video Textures

Recently, Shödl et al. presented a technique that allows the creation of repetitive and endless video given a short video sequence [Schödl et al. 2000]. To achieve this, a sufficient number of pairs of similar frames needs to be present. This amount depends on a distance metric calculated on the image data of the individual frames. These similar frames then act as smooth transitions from which normal video playback can deviate, and thus rearrange the original frame sequence into a new synthesised video. A drawback of this method is revealed when no good smooth transitions are available. In this case, the creation of a video texture is impossible.

More recently, de Juan et al. introduced cartoon textures [de Juan and Bodenheimer 2004]. Borrowing the idea from video textures to use a distance metric between all pairs of frames, they also rearranged a given set of data. To reach a sufficient number of smooth rearranges of cartoon data, a large amount of input data is required which is mapped to a lower dimensional manifold (a similar approach was taken by Campbell et al. [Campbell et al. 2002] on video data). Within this image space, the shortest path can be created between any two frames. However, several issues can be identified. First, a very large data set of input frames (sometimes more than 1000 frames) is required to ensure a decent variation of the animation. Next, a large amount of preprocessing is needed to reposition the image data and apply background removal. Moreover, the generated cartoon animations also demand a postprocessing step, in which gaps (due to insufficient cartoon data) need to be filled with some extra user input, and visual discontinuities are removed.

Our approach is based on data creation and starts with just a few keyframes (less than 10) drawn by an animator. These frames will serve as the required transitions to rearrange the generated animations as automatically desired or interactively commanded. By creating in-between frames for all pairs of keyframes all transitions will remain smooth. This will be elucidated in the next sections.

2.3 Motion Graphs

Motion Graphs [Kovar et al. 2002] provide a method to control realistic motion through a database of motion capture data. On the

constructed graph, that encapsulates connections among different pieces of motion, graph walks are constructed that satisfy specific constraints.

We applied a similar approach on image data and vector based drawings, by constructing a graph that connects individual keyframes by a path containing the required in-betweens that bridge them. From this data structure, graph walks are extracted in a goal based manner, resulting in interactive animations that incorporate user specified parameters.

3 Overview

Creating new animations from a small set of input data requires several steps to be executed one after the other. These can be divided into two large phases: the first part analyses the input data and generates new data from it. The second one employs this new data to create new animations by re-sequencing the generated material according to specific rules and animator interactions.

The complete pipeline to create a cartoon animation from scratch consists of the following steps (see Figure 1):

PHASE 1: analysis + data generation

1. The animator draws a few basic keyframes (Section 4.1).
2. The similarity between all pairs of keyframes is evaluated using a distance metric (Section 4.2).
3. From this measurement the number of required in-betweens is derived after which the actual interpolation steps can be performed (Section 4.3).
4. Steps 2 and 3 can be repeated iteratively until a sufficient amount of frames are generated — 1 or 2 passes are sufficient most of the times. The animator also can limit the inbetweening to only a few frames within the first passes. This way new keyframes are created, which can be altered to satisfy specific conditions and constraints the animator had in mind for the resulting animation. During the last pass, all final in-betweening frames are generated.

PHASE 2: animation synthesis

5. Random animations can be synthesised from the generated cartoon data, by rearranging the generated sets of in-betweens (Section 5.1).
6. To allow for fast animator interactions, a cost graph is derived from the generated frames indicating for all keyframes how many steps there are required to travel from one keyframe to another (Section 4.4).
7. This cost graph is optimised to represent the cost that is minimally required to travel between any pair of keyframes (Section 4.4.1).
8. Using the interpolated frames and the optimised cost graph, interactive cartoon animations can be synthesised (Section 5.2).

Steps 1 to 4 (also numbered in Figure 1) allow the user to create random animations. When steps 5 to 8 are added to this process, interactions with the generated animations can be incorporated. In the following sections each step of the presented pipeline is explained in more detail.

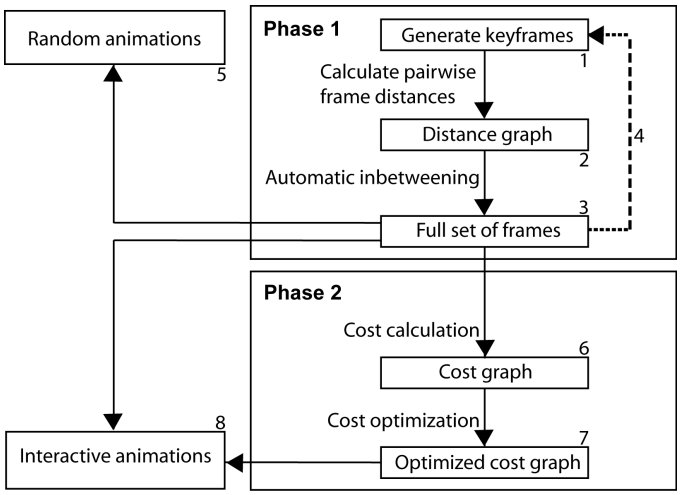


Figure 1: The pipeline of our process to create endless random or interactive animations. Phase 1 consists of data analysis from a few input keyframes and generates the required in-betweens to bridge them. Phase 2 allows for the creation of random animations which can be extended to comply with animator interactions.

4 Creating Cartoon Data

To relief the animator from the very time-consuming work that is involved in generating sufficient frames for a new animation, a computer aided alternative is required. Starting from a few keyframes, drawn by the artist, a set of new frames should be generated automatically.

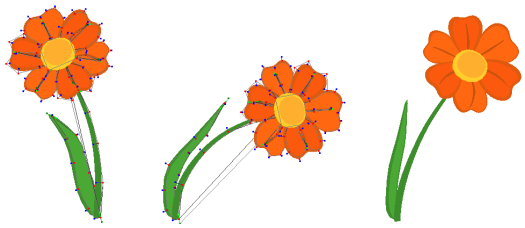
4.1 Keyframes

The first step in our animation pipeline consists of the creation of a few basic keyframes. To this end we employ user-controlled structured 2D modelling and animation techniques [Di Fiore et al. 2001] in combination with automatic in-betweening. This methodology clearly distinguishes between a separate modelling and an animation phase. This is similar to the 3D animation process and has been proven to be very useful for the purpose of creating convincing 3D-like animations, starting from pure 2D drawings, while preserving the artist's personal style.

Considering 2D animation from a technical viewpoint, two different categories can be distinguished: (i) transformations in a plane parallel to the drawing canvas (the XY plane), and (ii) transformations outside the drawing plane, especially all rotations around an axis different from the Z-axis.

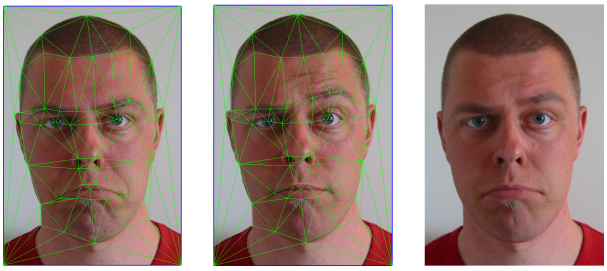
The former category of transformations is relatively easy to deal with, whereas the latter is the main cause of all the trouble in automating the in-betweening process (i.e. the underlying subproblems of silhouette changes as well as self-occlusion). It is in the latter type of animation where the 3D structure comes into play that is underlying the objects and characters in traditional animation (and which is present in the animator's — and viewer's — mind), but which is not present in the 2D drawings.

To tackle this without introducing too much 3D information, Di Fiore et al. developed a solution based on structured 2D modelling and animation techniques [Di Fiore et al. 2001]. This is implemented as a multi-layered system. At level 0, objects are modelled



(a) (b) (c)

Figure 2: a–b) Two extreme poses of a drawn flower using subdivision curves (see depicted control points) as drawing primitives. c) Generated in-between frame.



(a) (b) (c)

Figure 3: a–b) Two extreme poses of a face using subdivision meshes over real images. c) Generated in-between frame.

as sets of depth-ordered 2D drawing primitives (e.g. subdivision curves or subdivision surfaces). Level 1 manages and processes explicit 2D modelling information and is fundamental in realising transformations outside the drawing plane: for each set of 'important' XY-rotations of the object relative to the virtual camera, the animator creates a set of ordered 2D primitives. This is functionally comparable to the extreme frames in traditional animation [Blair 1994; Patterson and Willis 1994]. Level 2 incorporates 3D information by means of 3D skeletons or approximate 3D objects, while level 3 offers the opportunity to include high-level tools.

Multi-level 2D strokes, interpolation techniques and on-the-fly resorting are used to create convincing 3D-like animations starting from pure 2D information. A rigid 3D look is avoided through varying line thickness and the ability to have subtle outline changes that are either impossible or tricky to achieve utilising 3D models (see Figures 2(a–b)).

Besides *drawing* keyframes, our animation system also includes the possibility to create keyframes by incorporating real images depicting extreme poses. To this end we provided a tool which allows the animator to define a layered mesh structure over certain image parts that contain interesting information. This is shown in Figures 3(a–b) which depict two extreme poses of a human face. During the animation, in-between images of these 'real' keyframes are constructed by warping the meshes imposed on the extreme frames to each other in the same order as defined by the layered structure.

This structured 2D approach (i.e. explicit 2D modelling and automatic in-betweening) is the computerised version of the animator's work place. Unlike purely 3D-based approaches, the resulting animations still have many lively aspects akin to 2D animation. Depending on the purpose of the resulting animation and the quality that is required, the amount of keyframes can reach from just a few to several dozens. In general less than 10 keyframes suffice to cover the usual viewpoints which occur in cartoon space.

4.2 Distance Graph

Given a sequence of user specified keyframes, in-betweens should be generated automatically. An important aspect of this process is the calculation of the amount of in-betweens needed. To determine this number, a distance value d between all pairs of frames is calculated. This distance is defined by a function of the displacement of the control points from which the frames are constructed. For each pair of extreme frames the following rules need to be considered:

1. The amount of in-betweens needed between two keyframes depends on the properties of the deformation of the displayed object. If the deformation is global (the magnitude of the largest displacement of any control point remains below a factor x times the average movement of all control points), a sufficient amount of in-betweens should be introduced to minimally interpolate the average deformation. If the deformation is local (only specific parts of the object are changed or the largest displacement is minimally x times bigger than the average control point transformation), the largest deformation defines the required amount of in-betweens.

2. This amount is also relative to the magnitude of the deformation of the displayed object. Small changes to the displayed object require only a few interpolation steps, large deformations on the other hand, need a larger number of in-betweens. When the extreme frames are too different from each other, the interpolation should be prevented.

For each pair of keyframes the distance between the data is measured using the actual positional information of the control points from which the image was built. By averaging all control point deformations between a pair of keyframes i and j and keeping track of the largest control point displacement, the previously mentioned rules can be taken into consideration. A user specified threshold t prevents unwanted frame interpolations. The distance value $d(i,j)$ is a function of these individual components:

$$d(i,j) = f\ (\ maxd(i,j),\ avgd(i,j),\ t\ )$$

These values can be stored in a nxn matrix D with n the number of keyframes. Each matrix entry $d_{i,j}$ represents the distance from keyframe i to keyframe j ($d_{i,i} = 0, \forall i$). The following matrix depicts the distance graph corresponding to the 5 keyframes shown in Figure 4:

$$D = \begin{bmatrix} 0 & 102 & 189 & max & max \\ 102 & 0 & 97 & 180 & max \\ 189 & 97 & 0 & 89 & 216 \\ max & 180 & 89 & 0 & 130 \\ max & max & 216 & 130 & 0 \end{bmatrix}$$

The absence of a value (denoted by *max*), indicates that no in-betweens should be generated between the corresponding keyframes because there was not enough similarity detected (a large d value), making automatic interpolation difficult.

4.3 Automatic In-betweening

Provided with the user specified keyframes and the distance matrix D, the actual interpolations of the keyframes can be performed resulting in the required amount of in-betweens to accomplish a smooth transition from one frame to another. As a global property

$$F = \begin{bmatrix} 0 & 5,...,23,1 & 24,...,60,2 & - & - \\ 23,...,5,0 & 1 & 61,...,78,2 & 79,...,113,3 & - \\ 60,...,24,0 & 78,...,61,1 & 2 & 114,...,130,3 & 131,...,172,4 \\ - & 113,...,79,1 & 130,...,114,2 & 3 & 173,...,197,4 \\ - & - & 172,...,131,2 & 197,...,173,3 & 4 \end{bmatrix}$$

Matrix 1: Frame matrix F for the flower example (see Figure 4): each element $f_{i,j}$ contains the path to follow within the frame space, when moving from keyframe i towards keyframe j. For example, to direct the animation from keyframe 1 to keyframe 3 (zero based), the in-between frames 79 till 113 need to be displayed, ending with frame 3 itself.

of the interpolation process, the animator can decide on the amount of interpolation steps used to bridge a specific distance.

The actual interpolation process is part of the employed structured 2D modelling and animation methodology (see also Section 4.1). Figures 2(c) and 3(c) illustrate the in-betweening process.

At this stage in the pipeline, for each pair of keyframes with a valid frame distance, a short animation sequence is created that connects them. Storing these sets of interpolating frames into a matrix F (Matrix 1), allows rapid retrieval of the correct frame sequences while generating a new animation. At each matrix position $f_{i,j}$, the frames created between keyframe i and keyframe j are stored. The content of matrix F can be interpreted as a simplified motion graph [Kovar et al. 2002], built from the keyframe indices i and j, and connected through the generated sequences that bridge them.

Matrix 1 shows the frame matrix corresponding to the flower depicted in Figure 4.

4.3.1 Iterative Process

Using the method of automatic in-betweening, an iterative process can be established to increase the amount of initial keyframes. By forcing the interpolation of one or a few extra keyframes between frames with a large relative distance, large displacements of control points can be broken into smaller steps. Furthermore, the animator is always in control and hence allowed to modify these generated in-betweens to his/her specific desires.

After the creation of the frame matrix F, the first stage in the creation of a new animation is finished. Section 5.1 explains how at this point in the pipeline random animations can be generated using a straightforward algorithm.

However, transitions between two keyframes are often not possible because the required in-betweens are absent. If the distance matrix D contains insufficient entries, the animator should consider drawing more frames (possibly aided by a forced calculation of an in-between). Otherwise, an alternative path to travel between these frames, making use of other reachable keyframes, is required. This alternative route, should connect the keyframes in a straightforward manner without deviating too much from the planned animation sequence. To accomplish this, a cost graph is added to the animation pipeline.

4.4 Cost Graph

At this point in the pipeline, a computer generated number of interpolated frames have been created between several pairs of original keyframes. The cost to travel from frame i to frame j can be defined as the length of the frame sequence that connects these frames. These costs can be stored in a nxn matrix C with n the number of keyframes. Each matrix entry $c_{i,j}$ represents the cost to travel from keyframe i to keyframe j ($c_{i,i} = 0, \forall i$). The following matrix depicts the cost graph corresponding to the 5 keyframes shown in Figure 4:

```
OptimiseCostGraph( )
Label undefined positions with maximum cost;
while a cost value changed do
  for each matrix position (i, j) do
    search an index t for which: c_{i,t} + c_{t,j} ≤ c_{i,j};
    when found: update c_{i,j} and store t;
  end for
next
```

Listing 1: This algorithm optimises the initialised cost matrix C so that it contains the lowest cost between all pairs of keyframes.

$$C = \begin{bmatrix} 0 & 20 & 38 & max & max \\ 20 & 0 & 19 & 36 & max \\ 38 & 19 & 0 & 18 & 43 \\ max & 36 & 18 & 0 & 26 \\ max & max & 43 & 26 & 0 \end{bmatrix}$$

This cost matrix C can be interpreted as a cost graph G, containing 1 node for each keyframe and edges between all pairs of keyframes for which in-betweens are generated. The edge between frame i and frame j is labelled with cost value $c_{i,j}$.

4.4.1 Cost graph Optimisation

At this point the cost graph is not fully connected. Several edges between keyframes are still undefined due to frame distance calculations that result in a value laying above the earlier mentioned user specified threshold. These connections, however, can be created iteratively. By means of successive traversals of already existing connections, a path can be found between any two original keyframes (if the cost graph was closed). In addition, several connections can be improved by means of alternative routes in the existing cost graph. To accomplish this, the algorithm of Listing 1 is used:

1. label positions with an undefined cost value with a maximum cost;
2. for each pair of keyframe indices (i, j), search for an index t for which: $c_{i,t} + c_{t,j} \leq c_{i,j}$; if such a value t is found, store $c_{i,t} + c_{t,j}$ as the new cost to travel from keyframe i towards keyframe j and mark keyframe t as a new possible frame to go to from frame i when travelling towards frame j;
3. repeat step 2 until no more improvements are encountered.

The improved cost matrix C indicates for each pair of frames i and j the minimal cost to connect them:

$$C_{optimised} = \begin{bmatrix} 0 & 20 & 38 & 56 & 81 \\ 20 & 0 & 19 & 36 & 62 \\ 38 & 19 & 0 & 18 & 43 \\ 56 & 36 & 18 & 0 & 26 \\ 81 & 62 & 43 & 26 & 0 \end{bmatrix}$$

The encountered deviations from the original path, improving the cost to travel between two frames, can be stored in a matrix P. At each matrix position $p_{i,j}$ a set of frame numbers is constructed, indicating which keyframes to aim for, when travelling from keyframe i to keyframe j. Whenever a better path is found, the current list is cleared and a new set of better deviations is built up. When no further optimisations are encountered, an element from the set at matrix position $p_{i,j}$ represents the next node in a shortest path from keyframe i to frame j. This information can now be used to travel between keyframes using a minimal number of frames.

$$P = \begin{bmatrix} - & 1 & 2 & 1,2 & 2 \\ 0 & - & 2 & 3 & 2,3 \\ 0 & 1 & - & 3 & 4 \\ 1,2 & 1 & 2 & - & 4 \\ 2 & 2,3 & 2 & 3 & - \end{bmatrix}$$

5 Video Synthesis

In the following sections two methods are proposed for generating a new cartoon animation from the generated data. Both of them are highly related to the concept of a graph walk, introduced by Kovar et al. [2002].

5.1 Random Animations

The first method simply creates a new animation by randomly picking a new keyframe, j, as a goal to purchase from the current displayed keyframe i. That way, the $c_{i,j}$ frames of the short sequence $f_{i,j}$ that bridges the distance between the current keyframe i and the target j are played. If matrix F doesn't provide the required in-betweens to connect keyframe i with keyframe j, another random goal can be selected.

The resulting animations are completely random and can best be compared with the usability of cartoon textures [de Juan and Bodenheimer 2004]. The keyframes from which they are built serve as the transitions required to rearrange the generated data and to combine the different short animations that exist between the individual keyframe pairs. In most cases, the action represented by the generated cartoon is very simple. More meaningful animations demand a different approach as explained in the next section.

5.2 Interactive Animations

Contrary to random animation creation, this section describes an algorithm to generate new animations containing a *well defined* and a *meaningful* context.

Exploiting the cost matrix C in combination with the shortest path info from matrix P animations can be generated interactively. While the algorithm is generating random goals for the animation to aim for, the user can interfere with this process and provide the next keyframe to reach for (see Listing 2). For example, while the animation system is displaying one of the in-betweens that connects keyframe i with keyframe j, the user can propose a new goal g. Several candidates might exist to reach g starting from j, all of them through different paths of minimal length (e.g., $p_{4,1}$ for the flower proposes 2 possible shortest paths: passing through keyframe 2 or through keyframe 3). As soon as keyframe j is reached, one of the elements of matrix position $p_{j,g}$ is randomly chosen as the next intermediate goal to aim for. This new goal is the next keyframe within the shortest path from keyframe j towards keyframe g. As long as keyframe g is not reached, new intermediate goals are introduced, always bringing the playback of the animation closer towards keyframe g. Finally, when g is reached and no further interactions from the user are recorded, the algorithm continues to pick new random goals (if an endless animation is desired). At any moment the user can make a new request for a specific keyframe, deviating the current graph walk to a new destination.

```
GenerateInteractiveAnimation( )
while generating animation do
  if displaying a keyframe j then
    if user interaction was recorded: g is latest request then
      select new goal from p_{j,g};
    else
      pick new random goal;
    end if
  end if
next
```

Listing 2: This algorithm extends the random animation generation of cartoon textures to an interactive approach, creating more meaningful, user directed animations.

6 Results

In Figure 5 a few snapshots are displayed from an animation of a flower, based on the keyframes displayed in Figure 4. It took an unskilled animator less than 30 minutes to draw all extreme frames and almost no time (less than a minute) to generate the animation. From the 5 initial keyframes a set of 193 in-betweens was derived, according to the pairwise inter-frame distances. The resulting animation is smooth and the appearance of the flower can be controlled interactively by the animator (e.g. go far left, left, middle, ...). By applying a user defined threshold on the allowed inter-frame distances, and using the optimised cost graph, bad transitions are prevented, and a shortest path can be used to reach the requested flower position. In this example, the in-betweening process was restricted to movements that only bridge keyframes that are located 2 positions (in terms of keyframes) from each other (e.g. from the far left position to the middle one).

Another example of curved based drawings is shown in Figure 6. To create this animation (from which a few snapshots are shown in Figure 7), a few snapshots from a moving figure were modelled. In addition, several intermediate frames were created and altered in a extra iterative pass of phase 1 of the presented animation pipeline to control the direction of the animation. Consequently, our optimised cost graph not only allows smooth interactive transitions between the letters but also guides the animation between the actual positions through the specifically intended intermediate keyframes. This is accomplished by applying a well chosen threshold on the pairwise frame distances (in this case restricting the automatic in-betweening process to frames with a high similarity). In total, 13 frames were drawn and 473 were added automatically.

Figure 9 demonstrates how a video-like result can be achieved by applying our algorithm to image data instead of vector based draw-

ings. The displayed snapshots of this animation are part of a short animation created from a set of 26 keyframes from which a few are displayed in Figure 8. This first approach, based on the mouth positions of the individual letters of the alphabet already proves the usefulness of our technique. Improving this by actually modelling the phonemes of a language should give an even more convincing and more usable result.

A similar example (Figure 10) was created from facial expressions. In this case, only the 5 keyframes displayed in Figure 11 were used. As an example of the applicability of the resulting interactive animation, this (and the previous) example could be used to extend a standard chat application with smooth facial expression transitions enriching communication with 'realistic' emotions.

Discussion In all the given examples, the actual creation of the basic keyframes requires most of the animators' time. However, this artistic part of the animation creation process will always remain a task to be performed by the animator himself/herself. The creation of the in-betweens takes only a few minutes while the creation of the animation itself happens in real-time (including possible interactions of the animator). Calculating the pairwise interframe distances and the resulting optimised cost graph takes less than a second and is therefore negligible from the rest of the calculations.

All examples were tested on a commodity personal computer (Pentium IV 3.06 GHz, ATI MOBILITY RADEON 9000).

7 Conclusion and Future Work

We presented a novel method that facilitates the creation of endless animations or video loops for smaller-scale productions. To this end, a hybrid approach was applied benefitting from several ideas and techniques from video textures, computer-assisted animation and motion graphs. By combining the re-sequencing of existing material (phase 1) with the automatic generation of new data (phase 2) into a two-phase animation pipeline, new animations can be synthesised. Furthermore, the animator can interfere with the animation process at each arbitrary moment.

The resulting animations are smooth, broader than the input data and require no postprocessing.

We would like to address the issue of controlling the speed of the animation which in general is related to the content of the animation itself. In the current system, the speed of the animation is defined by the inter-keyframe distances and the timing enforced by the animator. In the future we would like to incorporate dynamic changes of velocity into an earlier stage of the animation pipeline.

Acknowledgements

We gratefully express our gratitude to the European Fund for Regional Development (ERDF), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT), which are kindly funding part of the research reported in this paper. Many thanks go also to Bjorn Geuns for his artistic contribution.

References

BLAIR, P. 1994. *Cartoon Animation*. Walter Foster Publishing Inc., ISBN: 1-56010-084-2.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, vol. 21(3), ACM, 399–407.

BURTNYK, N., AND WEIN, M. 1971. Computer-generated keyframe animation. *Journal of the Society Motion Picture and Television Engineers 8*, 3, 149–153.

BURTNYK, N., AND WEIN, M. 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM 19*, 10, 564–569.

CAMPBELL, N. W., DALTON, C., GIBSON, D., AND THOMAS., B. 2002. Practical generation of video textures using the autoregressive process. *British Machive Vision Conference* (sep), 434–443.

DE JUAN, C., AND BODENHEIMER, B. 2004. Cartoon textures. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 267–276.

DI FIORE, F., SCHAEKEN, P., ELENS, K., AND VAN REETH, F. 2001. Automatic in-betweening in computer assisted animation by exploiting 2.5D modelling techniques. In *Proceedings of Computer Animation (CA2001)*, 192–200.

KORT, A. 2002. Computer aided inbetweening. *NPAR2002: Symposium on Non-Photorealistic Animation and Rendering* (June), 125–132.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 473–482.

PATTERSON, J. W., AND WILLIS, P. J. 1994. Computer assisted animation: 2D or not 2D? *The Computer Journal 37*, 10, 829–839.

REEVES, W. 1981. Inbetweening for computer animation utilizing moving point constraints. *Computer Graphics 15*, 3, 263–269.

ROSE III, C. F., SLOAN, P.-P. J., AND COHEN, M. F. 2001. Artist-directed inverse-kinematics using radial basis function interpolation. In *Proceedings of Eurographics symposium (EG2001)*, vol. 20(3), 239–250.

SCHÖDL, A., SZELISKI, R., SALESIN, D., AND ESSA, I. 2000. Video textures. In *SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press, New Orleans, Louisiana, USA, K. Akeley, Ed., 489–498.

SEDERBERG, T. W., AND GREENWOOD, E. 1992. A physically based approach to 2D shape blending. *Computer Graphics 26*, 25–34.

SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 1993. 2D shape blending: an intrinsic solution to the vertex path problem. *Computer Graphics 27*, 15–18.

SHAPIRA, M., AND RAPPOPORT, A. 1995. Shape blending using a star-skeleton representation. *IEEE Computer Graphics and Applications 15*, 44–51.

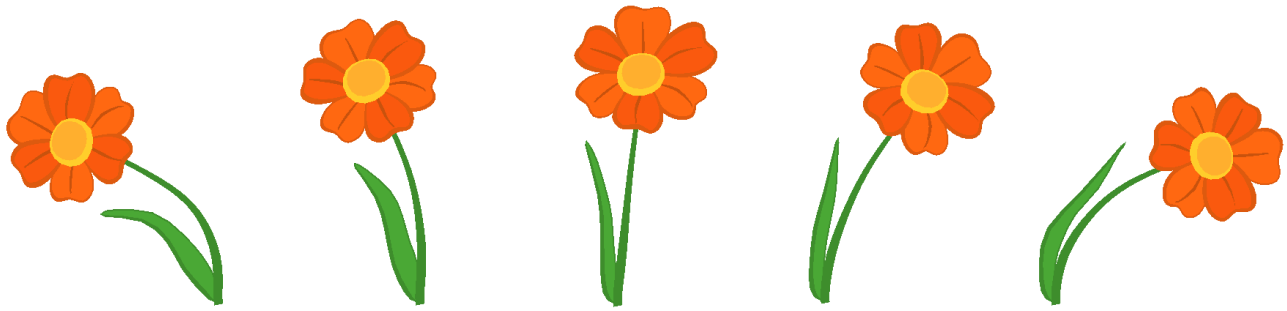


Figure 4: Five curve-based keyframes depicting extreme poses of a flower.

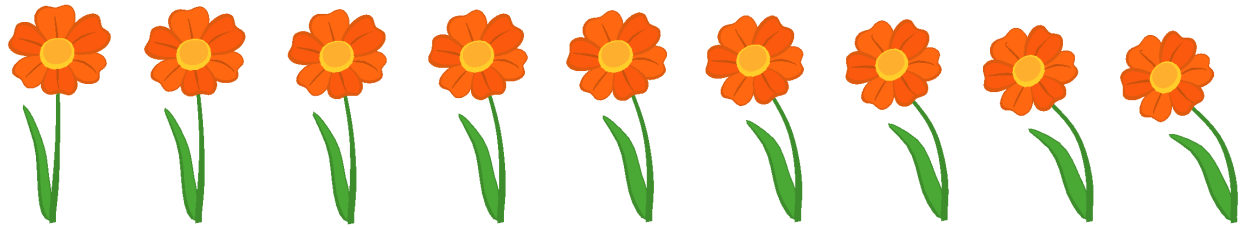


Figure 5: Several snapshots from an animation created from the keyframes in Figure 4. The animator can decide whether the flower bends towards the left or the right side (and also how far), simply by pointing out the required new flower shape. Using the optimised cost graph bad transitions are prevented, and the shortest path can be used to reach the requested flower position.

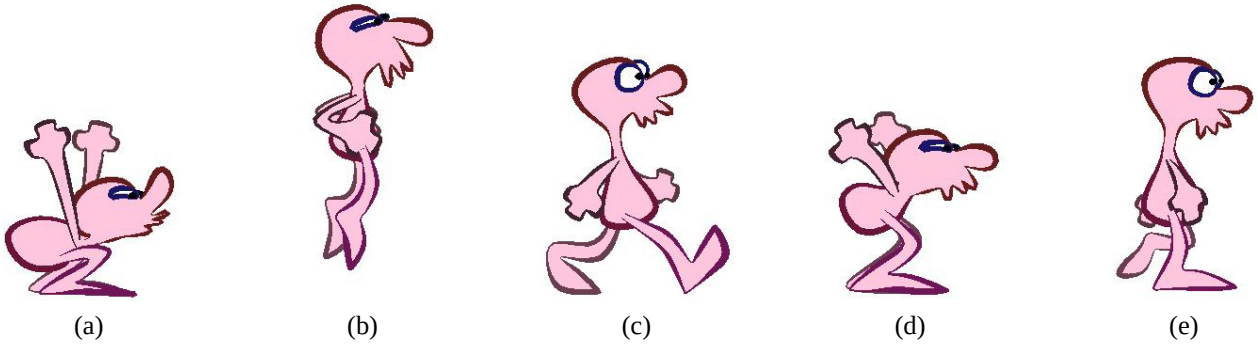


Figure 6: Five of thirteen keyframes used to create a moving figure animation. To create a smooth animation, several poses of this figures body were modelled (a - c). In addition, several intermediate frames were added to control the direction of the animation (d - e). This way, our optimised cost graph not only allows smooth transitions between the different position but also will guide the animation through the 'correct' keyframes, which were specifically created by the animator to uphold his/her original intentions.

Figure 7: Several stills from an animation created from the keyframes in Figure 6. By applying a good threshold to the calculated distance matrix *D*, only those in-between sequences are generated that connect sufficiently similar keyframes. In total, 13 frames were drawn and 473 were generated automatically. In combination with the corresponding optimised cost matrix *C*, a path between two poses always passes through the intermediate keyframes, specifically intended by the animator.

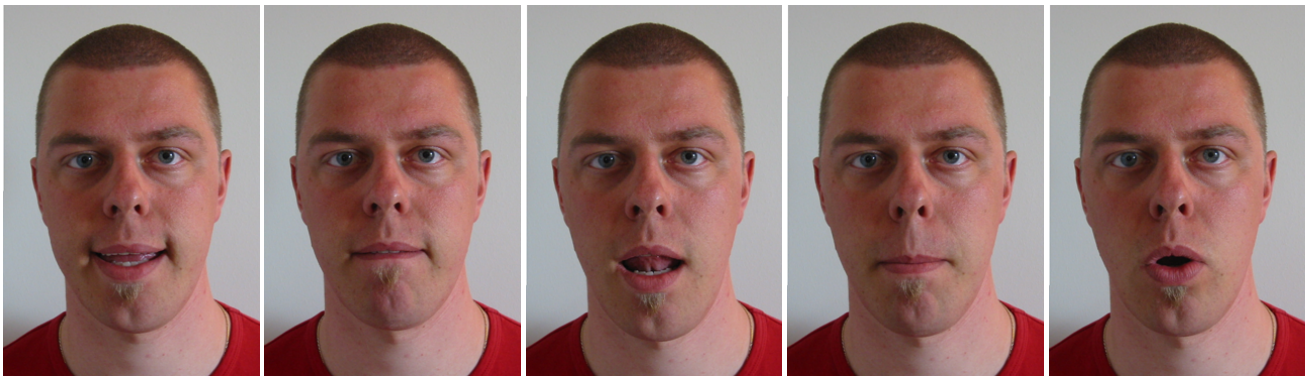


Figure 8: Five image examples out of a set of 26, displaying a few of the most important placements of the mouth while speaking. Using our interactive animation pipeline on this mesh-based example, animations can be created in which the displayed person simulates speaking actual phrases.

Figure 9: Several stills of an animation created from the keyframes in Figure 8. The successive smooth changes of the mouth position resemble footage of a speaking man in a very convincing way.

Figure 10: Keyframes used to generate a facial expression animation. Only these 5 pictures had to be taken to allow for all possible transitions between these emotions based on in-betweening. For each keyframe, a mesh is placed on the face to create the required interpolations afterwards.

Figure 11: Several generated frames of an animation created from the keyframes in Figure 10.