DELFT UNIVERSITY OF TECHNOLOGY
- COMPUTER GRAPHICS -

TI3800 BSc. Project Report

DEPTH AND MOTION INFORMATION FOR 2D IMAGE SEQUENCES

Authors:
G.A. Kolpa 4009886
M.Y. Santokhi 4093968
J.R.J. Vincendon 1312774

Exam committee:
Project Coördinators: M. A. Larson, H. G. Gross
Supervisor: Prof. Dr. E. Eisemann
Client: M. Holländer

July 18, 2013

Contents

Preface

In order to complete the Bachelors programme of Computer Science and Engineering at TU Delft, students are required to do a Bachelors project in the end of their third academic year. The authors of this paper, Alex Kolpa, Maniek Santokhi and Jérôme Vincendon, chose to research and implement a diffusion curves creation system.

This report describes the result of the endeavor to create the system. It discusses theory and application with argumentation.

We want to thank our supervisor Prof. Dr. E. Eisemann and the client M. Holländer for giving us the chance to work on this project and for their input.

Delft, July 2013

Summary

The problem presented to the authors was to build an application which could be used for constructing diffusion curves to augment 2D image sequences. To achieve this, the application needed to contain a system to construct and show the diffusion curves, some system to reuse these curves to prevent repetition, and a friendly user interface to make creation an easy process.

Diffusion curves are defined as a “vector-based primitive for creating smooth-shaded images”. They are Bézier curves defined by a set of control points, each with an assigned 'left' and 'right' color. This creates a colored division of the screen space through which the Bézier curves run.

Several different implementation techniques can be used, namely those of: Orzan et al., Jeschke et al. and Bezerra et al. Orzan et al. describe diffusion curves as cubic Bézier splines defined by a set of anchor points, each augmented with two color control points, and a blur value. The diffusion is achieved by iteratively sampling the colors along the line and averaging them. Jeschke et al. show that this process can be sped up by finding the closest anchor point for each pixel, allowing fast calculation of the diffusion process. Bezerra et al. focuses on controllability of the diffusion curves, and achieves this by introducing soft constraints which limit flow directions for each pixel.

The technique described by Jeschke et al. was chosen to implement. The DiffusionCurves system centers on usability, which favors speed and large datasets to work with, something that is provided by their implementation.

How the user interacts with the system has been a focus point of the project. Two extremes of possible users have been defined to design the system for: the computer scientist and the artist. To create an intuitive user interface, existing design principles have been used as much as possible.

The layout of the applications centers on a canvas area to accommodate for curves creation possibilities. Every function that frequently gets used is placed around this canvas. Other less used features are hidden away in the menu bar.

Colors have been used to communicate to the user in what state one is. So called flat colors, which are brighter more saturated versions of primary and secondary colors, in combination with a black tinted background accommodate that.

Minimalistic design tries to create a distraction less environment for the user. This includes: diminishing the number of options to the bare minimum, the use of lines and spacing for grouping and text as clarification.

Keyboard shortcuts reduce the need to travel long distances with the mouse to activate certain states or functions. It also removes the ambiguity of using the mouse as diffusion curves creation method and function activator. The natural position of the hands for someone who types with ten fingers has been taken into account.

The editor represents the canvas area. It provides a context sensitive tool to create diffusion curves and to edit them when in the Create state. Assigning color to each control point is done in conjunction with the Color state.

Interpolation in the system is necessary to prevent the user from doing excessive work. This means having a good way to transform diffusion curves from one frame to the next. In the application, this is achieved by using a pyramidal implementation of the Lucas-Kanade optical flow algorithm. To allow automatic interpolation over many frames, some kind of error estimate is necessary. This is achieved by finding the Mean Squared Error of all the separate tracking errors.

A good system design contributes to a stable and extensible system. The chosen design pattern was a combination of the Model-View-View-Model (MVVM) and the Model-View-Controller (MVC) principle. A separation is created by using events and direct object references as communication mean. Extensibility is reached by unused space in GUI areas and the code modularity, which allows for new subsystem inclusions.

Via continuous unit testing, integration testing and regression testing a certain kind of code quality guarantee has been made.

Improvements for this system can be made by adding cyclic paths creation to the editor, by adding a diffusion process to each layer, by using different algorithms for diffusion curves and by using a separate feature detection algorithm for interpolation. Several requirements were not met that could also be implemented as improvements.

1 Introduction

The assignment given by the Computer Graphics & Visualization group of the TU Delft was to develop a tool that would enable users to augment sequences of 2D images with depth and motion vectors, using a unified framework.

In order to do this, the user should be given the possibility to define constraints on images, which would then spread diffusion of user-defined colors over every image and in this way spread depth and motion information [22]. Interpolation would play a big part in this as well, in order to automate the process of defining constraints over a large number of consecutive images.

In this paper the process of engaging in this task will be discussed, what kind of problems were encountered and how the final goal was achieved.

The structure of the paper will be as follows: Section 2 will kick off by analyzing and defining the problem as was done in the first weeks of the project.

Section 3 will talk about the Diffusion Curves used for the diffusion. In this section primary focus will be on the analysis of different techniques, the chosen algorithm by Jeschke et al. and the implementation of this algorithm.

In Section 4 it will be explained how the authors tried to make the user interaction as smooth as possible. The main topics in this section will be the layout, color coding, minimalism of the application and program usability.

Section 5 will follow with a discussion of the context based Editor that was created, and the tools it contains to let user work on creating diffusion curves.

In Section 6 the interpolation will be discussed, explaining the research that was done on different interpolation techniques, the chosen Pyramidal Lucas-Kanade Optical Flow implementation and the overall implementation and integration.

Section 7 will elaborate on the software architecture developed during the course of this project as well as the easy nature of extensibility built right into the tool.

Section 8 will explore the testing that was done right from the start of this project, including the chosen testing strategy, the frameworks that were used and the different testing techniques that were used.

To wrap up the discussion about the tool itself, Section 9 will look back at the requirements that were decided upon at the beginning of the project and discuss what made it, what did not, and especially the reasons why things were moved forward, postponed or eventually rejected.

In conclusion of this paper, Section 10 will summarize everything that was discussed, followed by suggested future work in Section 11. In this last section, recommendations will be made about possible improvements as well as any possibly unmet requirements

2 Problem definition

In an earlier stage of the project the problem posed to the authors was to construct an application for creating Diffusion Curves. More specifically, the application would be able to construct depth and motion information over image sequences (Appendix A). In practice, this would result in the use of diffusion curves to add information to already existing pieces of film, represented as a sequence of images. This information could then be used to produce interesting effects such as high-refresh frame rates, stereoscopy, or artistic emphasis.

During the process, the focus shifted from the depth and motion information to the actual construction of diffusion curves as an augmentation for the image sequences. At first, the focus lay with the implementation of an interesting effect such as described above, but after several meetings with the client, this was changed.

Currently, the project focuses on the construction of a tool for creating Diffusion curves without the extra depth and motion information. So the problem statement is defined as follows:

> The client requires a single, unified application to construct diffusion curves over image sequences, through an intuitive and fast implementation.

No current system exits that provide diffusion curves creation to accommodate these kinds of applications. So this project is centered on building a system that provides such a technique. It is called DiffusionCurves and its focus points are to create:

- An intuitive interface for users to work with;
- A stable and extensible system;
- An efficient diffusion calculation.

This document describes the result, the methods used and the research that has been done.

3 Diffusion Curves

Diffusion curves are defined as a “vector-based primitive for creating smooth-shaded images”.[36] The primitive is defined as follows: the actual curve is a Bézier curve, defined by a set of control points, with each assigned two colors determining the ’left’ and ’right’ color of the control point. This creates a colored division of the screen space through which the Bézier curves run. Both the control points as well as the colors are defined by the user, giving them almost complete control of the resulting image. The actual image can then be constructed by solving a Poisson equation with constraints as given by the colors of the Bézier curve. Such a result can be seen in figure 1. Designed by Orzan et al. in 2008, several adaptations of its implementation have been made since then.

In the next section, the approaches will be analyzed, after which the preferred approach will be selected, and a description will be given of the implementation into the system.

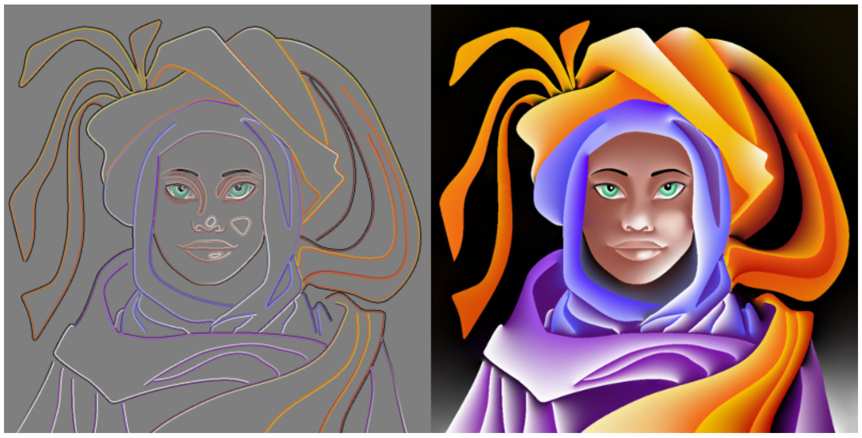


Figure 1: Diffusion primitives (left) Image after diffusion of primitives (right)

3.1 Analysis of different techniques

In this section, several different implementations are analyzed and discussed. First, the original implementation by Orzan et al. will be examined, then several implementations which improve in some way upon the original implementation.

Orzan et al. As explained in section 5, Orzan et al. described the diffusion curve as a cubic Bézier spline defined by a set of control points, each augmented with two color control points, and a blur value. The resulting image is achieved by the diffusion of colors on each side, and the smoothness of the transition across the boundary is defined by the blur value.

Their implementation is phased in three steps: first, the rasterization of the control points to a color source image, then the diffusion of this color source image, which is similar to heat diffusion as it is an iterative process that slowly spreads the colors of the source across the resulting image. Finally, the result is blurred using the blur map constructed from the blur values in a similar fashion to the diffusion of the colors.

This intuitive approach proves to be relatively slow, requiring approximately 20,000 iterations for a 512x512 pixel image.[29] Even though this process runs almost entirely on the GPU, significantly speeding up the fully parallel process, it can still take up to several seconds to generate a satisfying result.

Jeschke et al. To improve upon this performance issue, several new approaches were taken. One of which is introduced by Jeschke et al.[29] They observed that the Jacobi iterations as used by Orzan et al. proved to be slow because of their low convergence rate. As they described it themselves: "... color cannot quickly travel from one part of the image to another."[29] To solve this problem, they presented their solution in the form of a varying stencil size. This solution works because the Jacobi iterations converge, no matter the stencil size. For optimal size, they have shown that it should be set to the distance to the closest fixed boundary point.

To construct this solution, they applied a GPU-based implementation. To find the distance to the closest fixed boundary point for each pixel, some form of space division was required. This is easily achieved using a Voronoi diagram. Using this diagram, the stencil size can be determined. By reducing the stencil size each iteration, convergence of the diffusion can be achieved relatively fast. Jeschke et al. give two strategies for shrinking, namely "shrink always" (SA) and "shrink half" (SH). With SA, the stencil size is reduced linearly each iteration. For SH, the stencil size is halved each iteration. Compared, SH converges faster, but artifacts remain visible for more iterations.

Bezerra et al. One of the limitations of diffusion curves as defined by Orzan et al. is that they do not allow occlusion of the curves, meaning that only one side of the curve should diffuse its values. In many images, this results in unwanted artifacts, or they require extra curves to be added that were otherwise unneeded.

Bezerra et al. [15] describe a method of constructing diffusion curves with this occlusion, as well as giving control over the strength of the diffusion, and giving control over the diffusion orientation.

They achieve the first effect by giving each pixel a soft constraint, limiting diffusion from certain pixels into others. These constraints can then be used to give the user control over the orientation of the diffusion, by constructing the soft constraints similar to a flow field, giving some control over the orientation.

The color strength is achieved through different means. Bezerra et al. describe the strength control as an interpolation process of all sampled colors involved in a diffusion iteration. They achieve the result by using homogenous colors, where the r, g, and b channels are multiplied with the color strength a, which is stored in the alpha channel. This way, the colors can be interpolated, through which the strength of the diffusion can be controlled.

3.2 Choice of algorithm

Before the implementation, the direction of the diffusion system needed to be decided. Either the system would get a large amount of control, by using the implementation by Bezerra et al., or it would gain a decent amount of speed, by using the implementation by Jeschke et al. Unfortunately, due to time constraints, constructing a novel method which would possibly combine both approaches was impossible.

Going with increased control meant that the user would get a better experience whilst designing and drawing the curves. Giving them full control over almost every aspect of the diffusion would also mean the images would most likely become more visually pleasing.

Going for increased speed meant that the user would get a better experience whilst drawing as well as exporting the images. Because of the increased speed, alterations to the image would be visible faster, and constructing many images for export would be significantly faster than the slower approach.

In the end, it was decided to go for the implementation by Jeschke et al., since the DiffusionCurves system will be designed for large image sets, meaning that exporting the diffusion curves for every image in such a set would require several minutes or more. Jeschke's approach allows for real-time rendering, meaning each image should take less than 100ms to construct. Extrapolating this over a 1000 images (from a movie, this would be about 41 seconds of material), it would take less than two minutes to export. Besides this, the implementation by Jeschke et al. allows for some flexibility, meaning that in future adaptations of the system, other approaches can be integrated as well.

3.3 Implementation

Though the theory behind the approach of Jeschke et al. sounds complex at first, it is actually a pretty straightforward technique. As described in section 3.1, it first requires a Voronoi diagram giving the closest points. This diagram is constructed using the GPU. First, the geometric primitive needs to be written to buffers for GPU usage. Since the system only stores the anchor points as well as their control points, the Bézier spline needs to be generated every time the buffers are invalidated. Though this takes a couple of milliseconds, not every buffer is invalidated every time, and invalidation only occurs when the primitive gets altered in some way. These anchor points are constructed into the following buffers:

```
Buffers: | $P[n_{pos}]$: array of (x, y, z)
         | $C_l[n_l]$: array of (r, g, b, a)
         | $C_r[n_r]$: array of (r, g, b, a)
```

These buffer are then drawn with a geometry shader. Inside the geometry shader, the Voronoi diagram gets constructed. For each 2 points in the buffer, a line is drawn, and the tangent perpendicular to the screen is constructed. This tangent is then used to construct two colored planes, colored with the left or the right color depending on the orientation of the tangent. These planes are slanted, which lets the GPU's Z-Buffer generate the Voronoi diagram, which looks as follows:

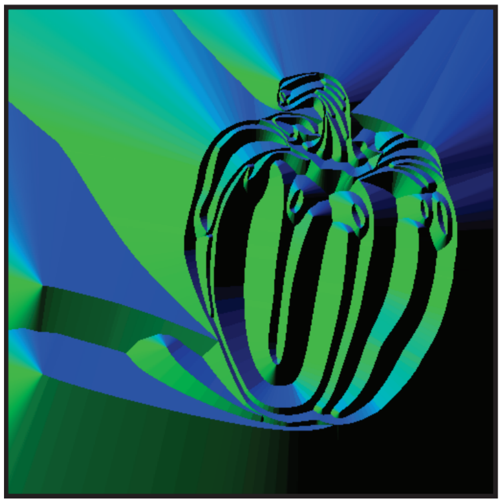


Figure 2: Closest point map (constructed by Jeschke et al.)

This image is stored in a RenderTexture, for use as an initial guess for the Jacobi solver. Besides the color information, the depth information of the Voronoi diagram is stored in a RenderTexture as well. These two RenderTextures are then passed along to the Jacobi solver. This solver takes as input the two textures, as well as the current iteration. It outputs its color-result to a third texture, which is used to iterate on as well. This way, a Ping-Pong model between the two color textures is constructed, swapping the output for the input every time an iteration is completed.

The actual solver itself samples the depth texture to find the sampling offset for the color image. It then samples the color image at four points that are in axis-aligned directions using the sampling offset to determine the sample position. These values are then averaged, producing the color output of the solver. This solver uses the shrinking strategy SA as described in section 3.1, though it could be easily adapted to support the SH strategy as well. After the final iteration, the resulting RenderTexture is then overlaid on the screen and drawn as a full screen texture inside the viewport of the application to present to the user.

4 User interaction design

How the user interacts with the created system has been a major focus point of this project. Two extremes of possible users have been introduced in the Requirements Analysis Document (RAD) –which can be found in Appendix C. Namely the computer scientist, who wants control over the system and extensibility, and the artist, who likes artistic freedom and ease of use.

Keeping these two in mind while creating and refining the user interface captures the needs of our target audience and everything within that spectrum, as argued in the RAD.

The difficulty of creating a graphical user interface lies within creating an effective, efficient and satisfying environment for the user.[46] These broad terms allow for many opinions, which makes designing the interface hard. It is not as quantifiable as computer science where one looks for objective truth.[45] This also makes it difficult to test.[33] The DiffusionCurves system therefore tries to build on existing design concepts as much as possible and to introduce new concept only where good argumentation allows for it. Strong claims as to why it will work will still be difficult to make.

Discussed will be the overall layout choice of the graphical user interface (from now on abbreviated as GUI), the choice of color in combination with its use as state indicator, the minimalism that served as philosophy and the program usability.

4.1 Application layout

In figure 3 one can see the main window of the system in an inactive state. This grouping of GUI components creates a certain kind of workflow and simplicity. Each grouping has been labeled as reference points for clarification later on.

Quite early in the design of the system it became clear that the creation of the diffusion curves should be the main function of the application. Drawing the user's attention to this function therefore became the primary focus. So the idea came to center a large canvas area in the main window with curves creation possibilities, and every GUI element around this area would then be in service of the canvas. The latter is embodied by the Toolbar area, the Layers area, the Control area, the Timeline area and the Menu bar.

This choice of drawing attention to the canvas by centering sounds trivial, but some automatic tooling like automatic acquisition of diffusion curves, interesting for computer scientists and described in the RAD, would eliminate the need to make the canvas the center of attention. The artist however, who want to have artistic freedom in the curves creation, could be drawn away by this.

Breaking with some conventional layout principles that artists as well as computer scientists have already been accustomed to would only frustrate the use of the system and steepen the learning curve.[35] So inspiration for the layout has been drawn from commercial software that possible users of this application are already familiar with (e.g. Adobe Illustrator and Visual Studio).[1, 9] This can be noticed by the use of dialogs for extra functionality, the use of a Menu

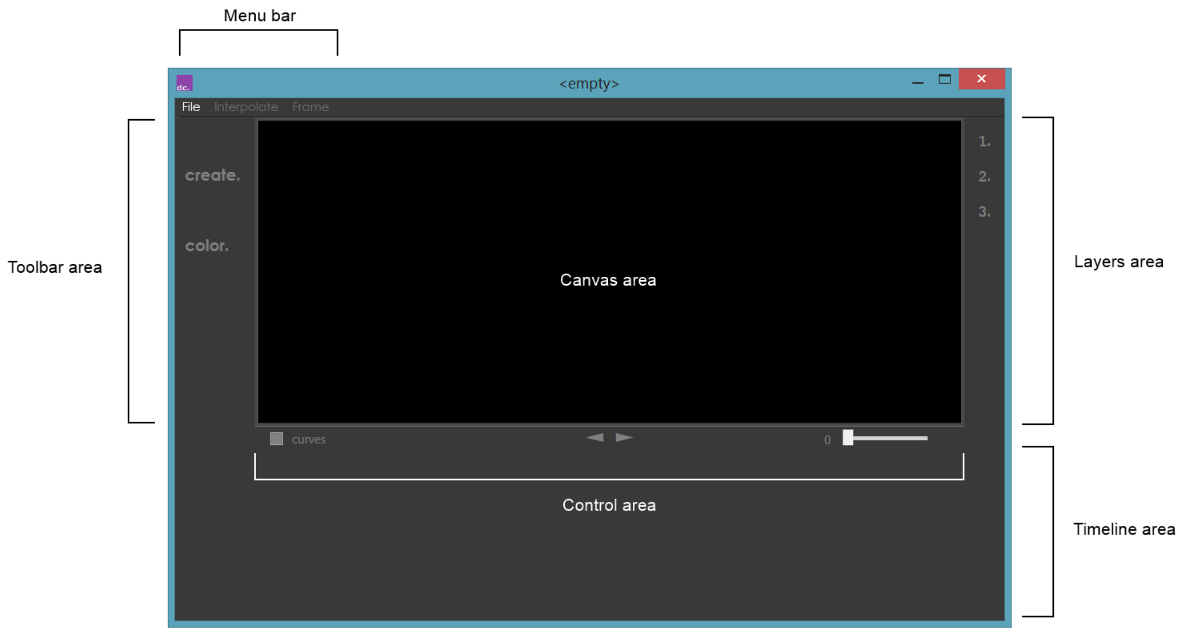


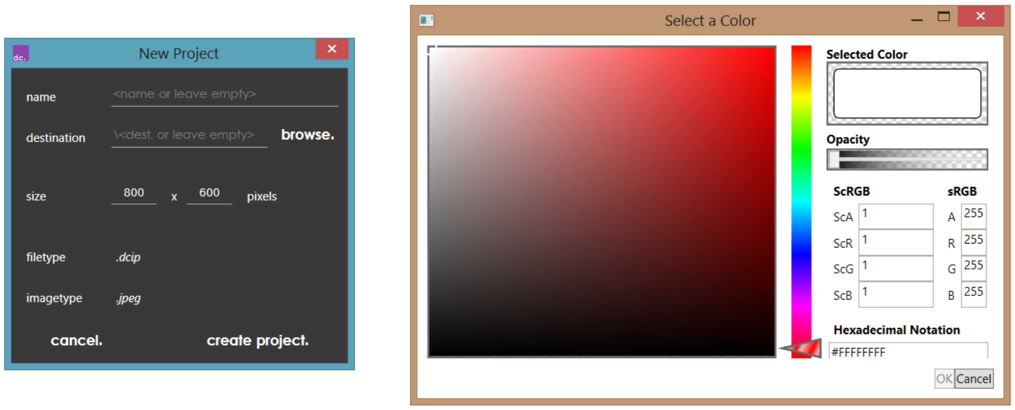
Figure 3: The main window

bar with familiar naming conventions to hide infrequently used features and the grouping of functions that will frequently be used in designated areas around the canvas such as the Toolbar.

The overall workflow of the system is centered on the diffusion curves creation, which is fairly straightforward. However, at times it is necessary to break from this workflow to enable features that are not always necessary to be shown on screen all the time. One such break is the creation of a new project (figure 4**a**) or to add color to a point in a Bézier curve (figure 4**b**). For both a dialog is introduced on the center of the screen that draws the users attention and disallows any other action apart from what's in the dialog. Letting dialogs appear is oftentimes in conjunction with clicking on buttons hidden away in the Menu bar.

One of the designated areas is the Toolbar (figure 3) which contains two buttons: Create and Color. Both describe a state that directly affects what the user can do on the canvas. With Create enabled one can create curves and due to its context sensitivity the user can also select individual points and manipulate them further. The latter eliminates the use of an extra button that represents an edit state. The other button is Color. Activating this button will let the user assign colors to points in the curve aided by the color dialog (figure 4**b**).

Another designated area is the Timeline area (figure 3). If frames are loaded into the project, the timeline area (which consists of a slider) allows for easy



name <name or leave empty>
destination \<dest. or leave empty>
size 800 x 600 pixels
filetype .dcip
imagetype .jpeg

ScRGB | sRGB
ScA 1 | A 255
ScR 1 | R 255
ScG 1 | G 255
ScB 1 | B 255
Hexadecimal Notation #FFFFFFFF

Figure 4: **a** New Project dialog **b** Color dialog

navigation. An earlier implemenation of this area showed individual frames as thumbnails –equivalent to a video editor– opposed to a slider showing a relative position, as is the case in the current implementation (figure 5). A project scenario with many frames would give bad frame management because it is hard to see in what part of the sequence one is. The user cannot deduce that from just looking at the thumbnails, as their small size would make it hard to distinguish frames that all quite look te same.

Figure 5: The Timeline area with frames loaded

The last designated area is the layers bar (figure 3). It allows for curve management for each frame. If a certain layer is selected then it lightens the curves in inactive layers so the artist can focus on the current layer without losing context.

So the doctrine behind the layout of this implementation is to create a straightforward workflow around the canvas to accommodate diffusion curves creation. Anything directly related to this curves creation is around the canvas and anything that is frequented sporadically is hidden away until needed.

4.2 Color coding

Some commercial graphics applications, like Adobe Photoshop, use monochrome looking buttons which slightly change color tone to indicate a state.[2] What users might find troubling is not directly knowing what button or layer is active. It can therefore be hard to keep track what can be done at a particular moment.

The system described in this document uses colors to overcome these problems.

In figure 6 below one can see the main window, a project loaded into the memory that shows a certain state and all GUI elements enabled so that the user can interact with them.

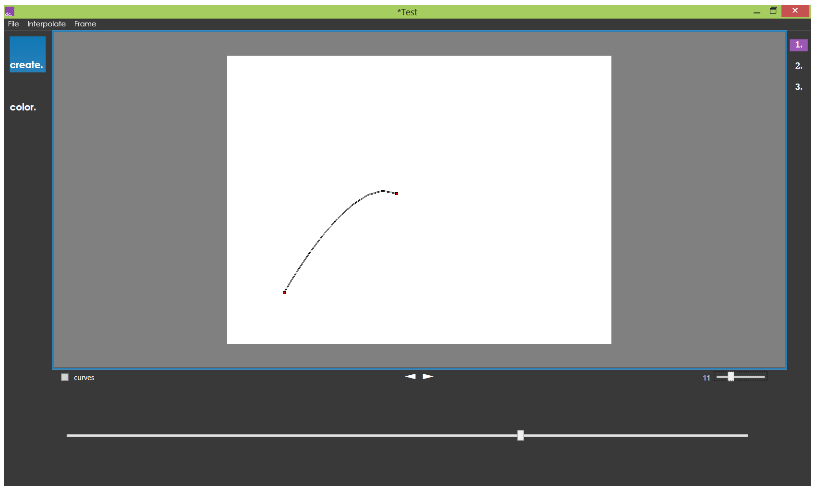


Figure 6: Main window with a project loaded

The use of colors should be apparent in figure 6. An impression the user needs to have upon seeing this is directly knowing that he can create curves on the canvas in the first layer and that color diffusion is not showing (the box on the left side under the canvas is otherwise light orange).

The toolbar is perhaps the pinnacle of the color coding implementation. Figure 7 **a** shows the active and inactive states of both the Create and Color button. Activating any of them also gives the border around the canvas the same color (figure 7 **b**). Not only does the user directly know what he can do in the context sensitive GUI, but also what this will affect, namely the diffusion curves in the canvas. So instead of the monochrome use for state indication as in Adobe Photoshop, the DiffusionCurves system shows a color coded version of this.

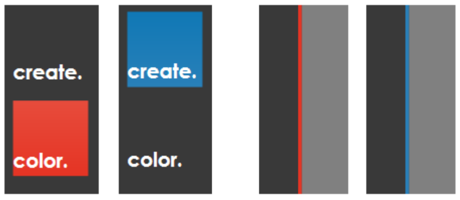


Figure 7: **a** Activation of buttons **b** Canvas border changed on activation

The colors chosen are what is called ’flat’. These are brighter, very saturated versions of primary and secondary colors which stand out on the black tinted background and inactive GUI elements.[19] A choice has been made to only use colors that have a good contrast between each other to avoid confusion. Some minor variations to certain flat colors are also used in conjuctions with its primary flat color for some added subtlety. An example of this is the hovering over buttons in dialogs seen in figure 8 which shows a lighter tone than when clicked. This should create constant feedback to the user input.

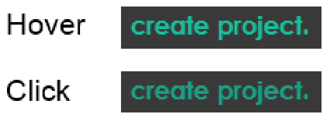


Figure 8: Different tone use per flat color

Color, or lack thereof, is also used to show whether a GUI element can be interacted with. For example when no project is loaded, the user should not be able to create curves or set another layer. So these state buttons should be disabled as well as look disabled. Figure 9 below shows this inactivity of the Layers area.

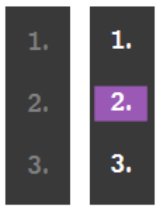


Figure 9: Different tone use per flat color

When disabled the background of the buttons matches the overall background and the foreground of the buttons is less bright. It should immediately show that one cannot interact with it nor should it distract.

In conclusion, this color coded model tries to let the user know at any time in what state the application is and what it can do/affect, without the user being distracted by other elements.

4.3 Minimalism as design philosophy

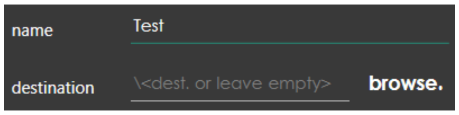
Much inspiration has been drawn from existing software to use as reference for what could work and what could not work for the system described in this document.

It is common in creative commercial applications like Adobe Photoshop and Apple Final Cut Pro X to use black tones as GUI background.[3] The idea behind this is not to distract from the content that is being created. The application described in this document also centers on content creation and would therefore also benefit from the idea to use black tones as distraction deterrent. Theses tones also contrast quite heavily with the flat colors used for color coding, making them stand out to let the user know at all times in what that he is in.

In for example Adobe Illustrator, with its many features and possibilities, a window management has been introduced to accommodate it. Since the DiffusionCurves system doesn't have the same prowess as Adobe Illustrator, it would only add possibilities that might distract the user. Instead, a fixed layout has been chosen to give the idea of steadiness, meaning: always knowing where what is once learned. The system described in this document also uses context sensitivity to diminish the number of buttons and dedicated overlain dialogs for some more complex functions. The latter keeps the workflow simple and the overall GUI minimalistic.

Minimalism is more than diminishing the number of buttons and keeping the workflow simple. It can also be expressed graphically with spacing, lines and text.

In general, spacing and lines introduce grouping of elements.[17] In the main window only one line is used to separate the menu bar from the rest as is usual with many applications. Grouping for this system is therefore mainly done by spacing, because it better fits the minimalistic approach compared to lines. Lines can make the GUI look crowded. Exception to this are the input fields. In a classic environment –e.g. filling out an application form with pen– people are already acquainted with writing on lines. This is reflected in figure 10, to create a familiar surrounding. This further lessens the learning curve.



name Test

destination \<dest. or leave empty> browse.

Figure 10: The use of lines to create familiarity to the real world

The application uses text for multiple purposes. It is used within buttons with a descriptive text and as description to clarify certain input fields. The first is viewable in figure **7a**. The latter is viewable in figure 4**a** where labels are used to let the user know what to do. To distinguish text from state defining buttons a dot is introduced at the end (e.g. 'create.').

This dot is also nicely reflected in the logo of the application as seen below in figure 11. Notice also the flat purple color, a rectangular shape comparable to the state buttons and the use of the same font.

dc.

Figure 11: The application logo which reflects the GUI design

With minimalism as design philosophy a clear and distraction less environment should be created. With the use of spacing, lines, text and color this has tried to be met.

4.4 Program usability

Mouse movement can be used for two things: creating diffusion curves and going to a clickable space like a button to activate a certain state. This idea introduces two problems. The first is added time and effort by moving the mouse to these designated clickable places, especially on large screens. The second problem is the ambiguity as to what a mouse movement can be used for: diffusion curves creation and movement to certain areas on the screen.

Both problems can be eliminated by introducing shortkeys. These are reserved keyboard keys or combinations of keyboard keys that activate a state or let the user jump to another section on the screen.

A few examples are: 'd' for activating the create state, 'f' for activating the color state and pressing the Tab-button in dialog will let you jump to another input field.

There is a motivation behind the choice of the buttons. For instance, the 'd' and 'f' promote a familiar hand position for users that type with ten fingers. The Tab-button to jump to other input fields is a design choice used in many applications like Visual Studio. A one-on-one principle is also used with the keys that correspond to the layers. For instance pressing the '1' key will activate the first layer.

The program is made more usable by keyboard shortkeys to eliminate added time and effort and to remove certain ambiguity.

5 Editor

The editor of the system is the actual part through which the user expresses control over the resulting diffusion curves. For that reason, it needs to be well-designed, providing the users with a pleasant experience. Creating this experience is described in this section.

5.1 Tools

As described, this editor is used to construct diffusion curves. Diffusion curves are described by Orzan et al. [36] as a Bézier spline defined by a set of control points, each augmented with two color control points, and a blur value, as further described in section 3.1. Because of time constraints, it was decided to leave the blurring out of system. This meant that only three values were necessary per point on the spline: The position of the point, and the two assigned color values.

5.1.1 Input

To define a curve, some input by the user is required. It became clear that several ways of curve construction were possible, which will be discussed below.

Freehand The first and most basic input to be considered was freehand drawing. Freehand drawing means the user can simply click on any part of the screen, drag the mouse around, and a curve will be constructed. This provides with ease of control since it becomes relatively easy to trace parts of an image. However, one of the main problems freehand drawing posed was that the lines that are drawn are not resolution-independent. This meant that when the user would zoom in, the drawn line would no longer resemble a smooth curve, but instead become a jagged line.

Cubic Bézier Curve The cubic Bézier curve is frequently used for vector graphics in many commercial applications, such as Adobe Photoshop, and GIMP.[4, 7] It provides control over the shape of the Bézier curve through two control points at each endpoint, which determine the curvature across a line section. Because this type of curve can be defined as a continuous mathematical formula, it is resolution independent. Besides this, only two endpoints and control points are necessary to draw visually complex lines.

Catmull-Rom spline The Catmull-Rom spline is a vector based graphics structure as well, but unlike the Bézier curve, it doesn't require the input of control points, as these are deduced from the input points. One problem it has is that it requires at least four input points to construct the spline for the line segment between the center two.

Result It was decided early on that the user should get as much freedom as possible with the editor, without making usage cumbersome. Though the freehand drawing does provide that freedom, the lack of resolution-independence curves was undesired. Cubic Bézier curves have been used for many years now in many professional applications which support vector graphics. For this reason, it was decided that it should at least be included in the editor.

After experimenting with an editor with just the cubic Bézier curves, it was found that constantly defining the control points was rather cumbersome. This meant some addition or altercation needed to be made to the editor. This addition was found in a combination of cubic Bézier curves and Catmull-Rom splines. The points are still stored as cubic Bézier points, with two control points each, but the calculation of these control points is done automatically, based on the two neighbors of the cubic Bézier point.[5] This meant that only three points were necessary to provide a smooth curve, which proved to be adequate for the system. By allowing the user to decide whether the control points should be calculated automatically, he or she is given full control over the system, whilst also providing some feature that prevents the process from becoming too cumbersome.

5.1.2 Storage

The input generated by the user needs to be stored in some fashion to be drawn onto the screen. Storing this data takes memory space, which needed to be taken into consideration. Therefore, the system only stores the bare minimum of what is required to construct the smooth curves, and rebuilds this every time the curves are altered.

To do so, a container is required to store all these points. As described in the previous section, cubic Bézier points are used to construct the curves. This means that every cubic Bézier points in a path needs the following information stored with it to construct a diffusion curve:

Diffusion point:	Position	a vector with (x, y)
	LeftControl	a vector with (x_l, y_l)
	RightControl	a vector with (x_r, y_r)
	LeftColor	a vector with (r_l, g_l, b_l, a_l)
	RightColor	a vector with (r_r, g_r, b_r, a_r)

5.1.3 Coloring

Every diffusion curve takes as input two colors, for the left and right side of the line. Defining this color is also part of the editor. The user can set the mode of the system to “color selection”, so that when points are selected, a color picker menu pops up, giving the user full control over what color each side should be.

To provide the user with some extra assistance, the color for the next added point is automatically copied from the previous point in the path. This is because it can safely be assumed that in a single line, very little color changes occurs. Even when the user does decide to change the color, it would need to be done in either case, so it can only be an improvement from the original situation with no copying.

5.2 Context based

The entire editor is built around a context-sensitive setup. This means that the user has one tool at his or her disposal, which changes behavior depending on the context. This context can be manipulated through various means.

The main manipulation is through state modification by using the keyboard. The editor keeps track of its current state, and depending on the key currently pressed on the keyboard, that state gets changed. This means that, depending on the current state, clicking a point on the screen can either mean that the system will drag any underlying point or its control points, it will add a new point to the currently active path, or it will construct a new path with this newly created point.

Currently, the coloring system is considered a separate state, because the process of coloring the curves is usually one that happens separately from placing the curves. Switching between coloring and placing curves is handled like a state machine as well, meaning that both cannot be done at the same time without a change of internal state.

6 Interpolation

One of the features that separates the DiffusionCurves system from others should be its ability to operate on a sequence of images, and provide an easy way to apply the diffusion curves. Having the user manually move around the curves from one frame to the next would be cumbersome and unnecessary. For this reason, some form of interpolation is needed.

In this next chapter, the different types of interpolation will be analyzed and compared, the final algorithm will be described in detail as well as the reason it was chosen, and finally how this algorithm was implemented in the existing parts of the system to provide the user with a friendly experience when working on large sequences.

6.1 Analysis of different techniques

For interpolation, it is necessary to do some type of motion tracking or motion estimation on the frames in order to analyze the images' movements for several key points, and to transform these points based on this information. However, in many situations it proves quite difficult to correctly track motion; rapid movement, occlusion, image noise, a cluttered background, and all sorts of other difficulties turn motion tracking into a non-trivial problem. To address these problems, a multitude of approaches have been formulated, each with its own strengths and weaknesses depending on the type of sequence it is applied to. In this next section, several possible motion tracking approaches and algorithms will be examined.

6.1.1 Contour tracking

Contour tracking algorithms track motion through a sequence of images by finding the contours in the image. Many contour tracking algorithms use some adaptation of a Kalman filter [20], which is used to filter out noise and provide an estimated guess of the movement in the image. Several improvements have been made on this approach, one of which is the *Contracting Curve Density* algorithm.[37] The algorithm takes a sequences of images and a parametric curve as an input, and tries to find the approximation that best matches the image data through curve-fitting.

6.1.2 Blob tracking

Blob tracking focuses on finding regions known as blobs in image data that differ from the surrounding regions through information such as color and intensity. These regions can then be used to track certain features, by finding similar regions in following images. One such implementation is Block-based motion estimation, where spatial coherence of small parts of the image are used to determine the motion.[30] These techniques are commonly applied in video encoding to reduce file size by storing motion information, instead of the entire image.

6.1.3 Optical Flow

The concept for optical flow is to construct a vector field with the same dimensions as the images that represents estimated motion in the image. Introduced by Horn and Schunck in 1981, many adaptations have been created which use some form of this approach.[28],[12] Though many algorithms take a differential approach to the problem, several algorithms have been constructed which use a phase based approach.[24] The algorithm by Horn and Schunk constructs the velocity field by minimizing a differential equation whilst taking into account a smoothness factor, to allow for some image noise. In a discretized image, this comes down to iterative equations for the velocity $(u,\ v)$, by solving the equation $f_x u + f_y v + f_t = 0$. This provides for a global solver of optical flow between two images.

Lucas-Kanade Over the years, several improvements have been made to the concept of Optical Flow, one of the earliest and most successful methods being the Lucas-Kanade algorithm.[31] Unlike the Horn-Schunck algorithm, which is a global solver, the Lucas-Kanade algorithm is a local solver. It assumes that the optical flow in a small local region is nearly constant, and tries a least squares approach on the local neighborhood to find an approximation of the optical flow. Usually some weighted window is applied to favor the pixels closest to the current pixel, further strengthening the concept of local consistency for the optical flow.

Pyramidal Lucas-Kanade One very successful adaptation of the Lucas-Kanade algorithm is the pyramidal Lucas-Kanade algorithm.[16] This algorithm iterates the Lucas-Kanade algorithm over increasingly downsampled images. By down sampling the images, it removes high-frequency noise thereby increasing robustness of the algorithm, while preserving local accuracy at higher sampling rates.

6.2 Algorithm selection

One of the main concerns for the interpolation algorithm selection for the application was the time it would take to implement and integrate it in the existing system. As described in the previous section, many feasible solutions to this problem exist, with many different strengths and weaknesses based on the sequence it is applied on. This meant that creating a novel algorithm was out of the question, especially since it is not the goal of this system to demonstrate a novel technique, but merely to provide a good interpolation to improve user experience.

The authors of this paper have considered programming an existing algorithm, but it eventually was decided that this would be unattainable. Again, many implementations already exist, and are available through third party libraries. By not building upon existing code, it would also be more error prone, since there is a lack of experience with these kinds of algorithms in the team.

Using third party libraries means the code has been written and used by others, giving some guarantee of stability and quality, something which is nearly impossible to achieve by using own code.

Several libraries exist for image processing, but only a few provide interpolation capabilities. Two of the most interesting libraries for this subject are OpenCV [10] and VTK [11], which are even being recommended as two of the most interesting libraries available.[43] Seeing as how both are described as being designed for Computer Vision as well as Computer Graphics and Visualization, these were high on the list of considerations. Of course, many other image processing libraries are available, however not all as complete as the two aforementioned. Several others have been examined, but were not picked in the end: CxImage, ImageMagick, and IM.[6, 8, 43] The main reason they were not chosen is because they did not contain an interpolation module which satisfied our system requirements.

Given these two libraries, OpenCV and VTK, which both have a wrapper for C# -therefore meeting the system requirements- a comparison could be made between the two for usage in the system. Unfortunately, little information is available comparing the two libraries, both in usability as well as performance, two of the main factors determining the final decision. Due to time constraints, it was impossible to experiment with both and pick the best, since that would mean integrating the interpolation twice, requiring too much time. In the end, it was decided to use OpenCV, because of the large amount of documentation and community support available.

OpenCV provides several interpolation algorithms, most interestingly the Pyramidal Lucas-Kanade approach, which has been described in section 6.1.3. Since this approach has a very good performance as well as providing a good accuracy [44], it was decided that this algorithm would be used for interpolation in the system.

6.3 Integration

For the integration into the DiffusionCurves system, a separate module was designed for the interpolation to make the connection of the actual algorithm and underlying logic easier. This could then be attached using an interface to the actual system. This way an early version of the user interaction could be constructed in order to do some use case testing, while the actual module was still being constructed. Due to this modular implementations, as well as ease of use from the side of the actual algorithm, integration proved easier than expected.

The pyramidal Lucas-Kanade algorithm can be used with a sparse feature set. This means the algorithm already does the actual interpolation, instead of having to pull the data from the output flow field. However, this posed a problem, because all the control points were stored in world space, while the input was expected to be in image space. This meant that some form of transformation on the control points needed to be done, as well as an inverse on the result. This transformation was achieved through the RenderState, which kept track of the current state of the rendering module, and could be used to transform world space coordinates to image-space coordinates and back.

Error estimation Next was the problem of error estimation. Because the module can run automatically – meaning it will interpolate until the interpolation error becomes too high – some way to determine this error was necessary. The algorithm already provided the system with some information, most importantly whether it was able to track a control point at all, and if so, what its estimated pixel error was. This error was determined by comparing the local neighborhood of pixels of each control point and its interpolated version. Though this data was very valuable, it could not be used directly for error estimation. For example, by simply summing the errors, one big outlier could strongly affect the outcome of the set.

One estimator proved to be very useful in this case, namely the Mean Squared Error.[49] The mean squared error estimator has been used for many years in statistical analysis and can be used for comparing statistical models.[42] By implementing this estimator, the user could be asked for the maximum mean squared error he or she allowed during the interpolation, and then the algorithm could interpolate until it hit that level.

Using these techniques, a robust interpolation module has been constructed for the DiffusionCurves system, providing the user with a useful tool to construct diffusion curves over many consecutive frames. Without it, the user would have had to draw all lines by hand, making the entire system needlessly impractical.

7 System design

One of the objectives of the project was to create a stable and extensible system. A system that users can rely on and third parties can easily extend from the same code base. How the system is designed contributes greatly to this.

This chapter goes into detail how the system is blueprinted. First, the chosen design pattern will be discussed. After that the structure will be detailed. Lastly, the code base extensibility will be explained.

7.1 Chosen design pattern

In software engineering literature different definitions are given as to what software design precisely is.[40, 18] Freeman has broadened this as follows: ".. the activity following requirements specification and before programming."[26] According to the Waterfall Model this would be the conceptual phase.[14] However, due to the Agile approach of this project, which promotes iterative development, this phase is spread out more over time.[13] Software requirements (and therefore the software itself) can change quite often into different directions. This could negate earlier efforts if too much time has been spent on designing. Architecting systems in great detail (like in the Waterfall Model) is therefore avoided. But how does one lay a good foundation to build upon when using an Agile approach?

The answer to this is to use a Model-View-Controller principle (from now one abbreviated as MVC) throughout each change. The MVC principle states that there must be a clear separation between the View (GUI) and Model (the logic that can be saved) which is bridged by a Controller with a user in between. The Controller handles the communication between the View and the Model.[41] In figure 12 MVC is represented graphically.

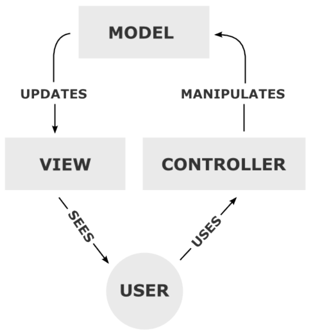


Figure 12: A graphical representation of the MVC pattern[21]

However, the structure of the system described in this document is not entirely based on this principle. The use of the .NET framework in combination with Windows Presentation Foundation allows for an extension to the

MVC principle called Model-View-View-Model. With MVVM (abbreviation of Model-View-View-Model) one layer is introduced which contains the Controller and adds capabilities. This layer is called the View Model which is responsible for exposing the data objects from the Model in such a way that those objects itself are easily managed, converted and consumed.[25] In figure 13 a graphical representation shows the MVVM principle.

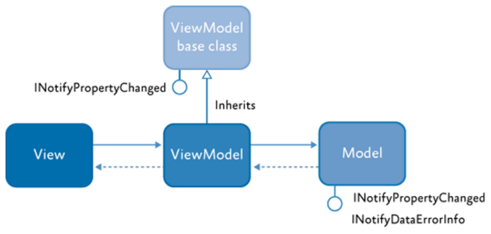


Figure 13: A graphical representation of the MVVM pattern[32]

The MVVM model is an interesting concept for applications that rely heavily on user input. To put it differently, MVVM is an event-based design principle that can handle the fast pace of change in the model (due to heavy user input) and updates the GUI accordingly. With the possibility of creating diffusion curves, which fires a lot of events, the DiffusionCurves system would benefit of the MVVM design pattern.

The DiffusionCurves system actually takes a middle route between MVC and MVVM. Any change in the Model by user input will be notified. Then the right objects will be exposed by the View Model and converted if necessary. At the end the GUI gets updated accordingly. As to why the DiffusionCurves system does not entirely make use of the MVVM principle lies with the View Model. The View in the DiffusionCurves system adds the role of the View Model to its own arsenal. This is done to keep complete control over what should change in the GUI. Updating the Graphical User Interface would otherwise be abstracted away from the View by data binding possibilities in Windows Presentation Foundation on standard defined elements. So, with the strict separation of View and Model and the use of a View Model principle within the View class, this system still fits the MVC pattern as well as the MVVM pattern, making the DiffusionCurves system a compromise between MVC and MVVM.

The added control over what should change in the GUI might also be a point of critique. Extra code that needs to be written, tested and maintained makes for overhead on the project and system. But that was not a deterrent for using this compromised approach of MVC and MVVM.

7.2 Structure of the system

In figures 12 and 13 in the previous part an abstraction of the MVC and MVVM concepts have been given. It is interesting to compare those with how the implementation of the DiffusionCurves system came about, seen in figure 14. This figure only shows a fraction of the whole system.

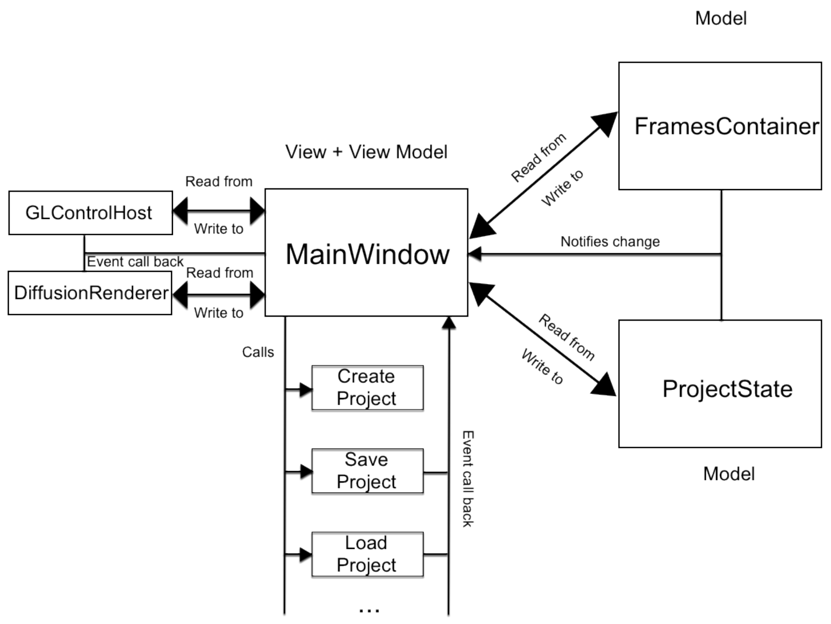


Figure 14: A graphical representation of a part of the system

The figure above shows a graphical abstraction of how the system is designed. As can be seen, there is a separation between the Model classes (FramesContainer and ProjectState) and the View + View Model class (MainWindow). There is a direct reference from the MainWindow class to either Models for communication. The communication consists of reading and writing to either Model classes and a notification of change from FramesContainer and StateProject to the MainWindow.

For most other classes that get called, an event system is used to communicate. For instance, the LoadProject class is static and gets called when the user wants to open an existing project. Right before the call a function that is within the scope of MainWindow gets injected into LoadProject. Anything that gets generated can be send back to MainWindow by a call of the injected function within the LoadProject class. This construction keeps the design separated.

The GLControlHost and DiffusionRenderer both have a direct reference within MainWindow and they make use of the event system described earlier. The direct references are to give them a place in the memory while the events are used as a communication mechanism to update the GUI.

The MVVM principle can be seen in the notification change events that get called once a Model object has been changed. This in turn gets updated in the

MainWindow which functions as View Model as well as View definer. The MVC principles can be seen in the strict separation of Model defining classes and the View.

In conclusion, the DiffusionCurves system makes use of a compromised MVVM and MVC model. This has been achieved by a clear separation between Model classes and the View merged with a View Model, as well as an event system to connect separate parts of the system.

7.3 Extensibility of the system

A design goal for this project was to make an extensible system. What is meant by this is that third parties can easily extend the application with the given code base. To achieve this, certain decisions have been made.

One of these decisions can be seen in the GUI (figure 6). The placing of the areas doesn't allow for much change, but within the areas much can be extended. An example of this is with the state buttons, where more can be defined and displayed. This also holds for the Menu bar as well as the Control Area.

Another decision that has been made is within the code itself. Due to the separation of Model classes, the View with View Model and dedicated subsystems, any new features and subsystems can be added. This separation also makes the code portable to other platforms, allowing for an easily extensible system.

8 System testing

The only testing experience any of the team members had is what was taught in the Software and Quality course given in the second year.[47] This was a great introduction into testing and introduced concepts like unit testing, integration testing and mocking.

During the brainstorm and research phase in the first two weeks of the project, it was very quickly decided that testing would play an important role throughout the whole project. This was in order to guarantee that the application would function exactly as it was intended to, and as the users expected it to. Even though delivering 100% bug free software would almost certainly be an impossible task, the main goal was to get as close to this target as possible.

In this phase at the beginning of the project, even before the implementation started, it was very important to get a good idea of what a good testing strategy would be. It was soon decided that it would be a good thing to make sure that there would be a fully functional test environment ready when development started, in order to ensure that testing would be a profound and fully integrated part of the process. So the first questions that came to mind when trying to figure out what the approach was going to be were “What to test?” and “How to test it?”

It turns out to be easier to answer the second question than the first one. When nothing has been built yet, it is hard to figure out what to try to break. In order to achieve this, a Test Plan was written in which all the research that had been done was explained, and which choices were made and why. Interested readers can find this document in Appendix D.

8.1 Testing strategy

Thanks to the research done for the test plan, it was soon decided that a combination of *unit testing*, *integration testing* and *regression testing* would be used.

Unit tests would ensure that every basic, fundamental parts of the developed software would function as expected and needed. These would mostly be automated tests, which could be run over and over again without a hassle. This way every time something was changed or added, it would be very easy to check that nothing else was accidentally broken in the process.

After this, integration tests would test components built out of small units and thus containing more functionality. Because the units these components are made of had been checked using unit tests, integration tests would ensure that interfaces between these units function correctly. This showed to be less trivial than the unit tests, as not everything that needed to be tested in the integration tests can easily be automated. For example, creating a new project or saving a project is easier tested by hand than automatically, as it greatly depends upon GUI manipulation. Besides this, it was deemed that the whole process of checking that a project got properly saved was easier to check by hand than automatically, as it involves multiple actions and variations.

Finally, regression testing pretty much came down to a combination of unit testing and integration testing. Every time a team member was ready to submit his newly written code to the version control, he first had to make sure his new – or revised – code did not break any old tests, as well as passed any new tests. This made sure that there was a correctly functioning product online at all times.

8.2 NUnit framework

The next step was to actually set up the testing environment before the start of development, in order to start testing as soon as the first code was written. As it was decided that testing would be closely integrated with implementation during this project, having a fully functional testing environment right from start could only encourage everyone to make use of tests, and thus keeping the software as robust and bug free as possible.

It was thus very important to find a testing framework that would best fit these requirements. As the only experience any of the team members had had this far involving testing revolved around Java oriented solutions, quite some research had to be done. It soon came down to two options: the NUnit framework or Microsofts own MSTest, which was built right into Visual Studio 2012, the IDE that was used during this project. Upon digging further into this it quickly became clear that NUnit was the better candidate for a wide variety of reasons; ranging from stability, to user experiences.[23, 34, 48]

8.3 Rhino Mocks

As some classes or methods in a project are dependent upon other classes – or methods in other classes – to function, it is important to have a way to decouple this when it comes to testing. Tests should only test the functionality of one piece of code, and not depend upon other functionality that this specific piece of code might depend upon itself. In order to achieve this, a mocking framework was necessary.

An easy choice here would have been to go with the NUnit Mocking project, however it is no longer being developed, and its author even discourages its use in production work.[38, 39]

After some discussion with several external sources about the process mocking and stubbing, Rhino Mocks seemed like a good candidate for the job at hand. And as expected, the few times mocking or stubbing dependencies was needed; this framework was very helpful and did the job correctly, effectively lowering dependencies in the written tests.

8.4 Unit testing

When all the above was finally decided upon and development had begun, it was time to start writing unit tests. From the start, functionality was grouped into packages, which made it very easy to write appropriate tests for different kinds of functionality. It was also decided soon that there would not be unit tests for every piece of code, as not everything is easily tested -or needs to be tested- by means of unit testing. It was deemed that views for example, as well as several parts of the code that are graphics related and lean a lot on EmguCV, and therefore OpenCV, are more easily tested by working with the application and trying to find bugs this way, than by writing unit tests.

However all the logic that is fundamental to the correct functioning of the application has been thoroughly unit tested. 100% code coverage through testing for these parts of functionality seemed like a good goal, in order to find bugs in early stages and not be struggling with unexplainable errors in a later stage.

Some parts of the software development heavily depended upon *Test Driven Development* (TDD). For example, the implementation of the save and load mechanism was completely realized using TDD. Let's look at the *Create* method, which gets called when a new project needs to be created. This method itself calls four other methods (in the order stated here), each having its own functionality: *Pathstring*, *CreateFolder*, *ZipFolderRenameToDCIP* and *DeleteOriginalFolder*. It is not hard to see that the input to every called method depends heavily upon the output of the calling method, so it had to be made sure that methods worked correctly before starting implementation of the next called method. This is just one of the situations where the TDD methodology was very welcome.

8.5 Integration testing

For the integration testing, a combination of unit tests and user tests has been used. As stated earlier, not everything is easily tested by automation, and if it was felt that a human would perform better at testing one specific functionality, testing would be done over and over again by hand to make sure that everything worked as expected.

Let's take the save and load functionality again as an example. While some separate units that are used to write projects to disk – and load them back into memory – can easily be tested through automated unit tests, a human is way better at determining if the project he just made was correctly saved and reloaded. This includes the curves he drew, the images he imported, the diffusion he applied, and the list goes on. So that's what happened. For the basic building blocks, automated unit tests were used, and for the bigger picture, the save/load functionality as a whole, humans were used (the team members).

On the other hand however, there are also big chunks of logic that are easily tested by automated unit tests. This time, let's look at the create project functionality for example. The way a project is created (outside of the call from the View) is split up in several methods. First, these separate methods are all unit tested to ensure correct functioning. After this it is very easy to check the functionality as a whole. A call from the view is faked, and after that all that has to be checked is that a project was correctly created in a predefined location. Same goes for locations where there was no write permission, or locations that could not exist due to invalid characters in the path or filename. The smart usage and combination of these techniques made sure that efficient tests were run through the whole development process, keeping the code as clean and bug free as possible.

8.6 Regression testing

In order to check that newly added code – or perhaps changed code in a later stage of development – didn't break any functionality, it was agreed upon a few things in the beginning of the project. First of all there would be two branches in the version control, one master branch and one development branch. Everything that was uploaded to version control and wasn't a new release of the application would be sent to the development branch. The master branch would only contain a version of the application that was fully functional, though be it with a minimum of functionality. If it is not there, it cannot be broken, meaning that the uploading of small commits of code would always be done to the development branch.

Besides this, every team member was responsible for checking and testing his code before commit. Making sure that it would correctly build, and not break any tests. As code was committed on regular basis, tests were run on regular basis as well. It was then also necessary to check the manual tests and make sure that nothing had changed in that area. As the manual testing took place mostly in the last 2 weeks of implementation, because it had not been necessary up to before that point, luckily this wasn't too much of a hassle.

In the end, this testing strategy has made sure that a stable application was present at all times during development. And even though once in a while something would break, it never took too long to find out where things were going wrong, and patching up was always relatively quickly done.

9 Requirements Evaluation

In this section, an analysis will be made of the delivered system with regard to the functional requirements that were decided upon in the Requirements and Analysis Document (RAD). As is common in an Agile development process, requirements are likely to change over time in comparison to those described in the RAD. This can be due to time constraints, client wishes or even budget cuts. This project is no exception to this principle, so this section will elaborate on these changes and the motivation behind them. The interested reader can find the RAD in Appendix C for references.

One thing that immediately stands out when reading the RAD is that the main focus of the application has completely changed. While it still serves as *"a platform for the Computer Graphics and Visualization Group to use for motion interpolation research"*, the ability *"to use interpolation in order to augment a 24p movie sequence to a 48p sequence"* has been removed in its entirety. This is mostly due to a meeting early on in the development process. During this meeting the client and the coordinator of the project made it clear that their primary interest was a tool that created the possibility to construct diffusion curves, interpolate these curves over several images and preferably export them.

The requirements of the application have been grouped in a so-called MoSCoW model.[27] This model differentiates between *must haves*, *should haves*, *could haves*, and *would haves*, must haves being the most important features, would haves being the least important. In the remainder of this chapter, an overview of the requirements will be given, explaining why it was chosen not implement certain things if necessary, or why requirements were slightly changed.

Must haves

1. **Easy to use GUI:** Everything except the meaningful pictograms and feedback about slow processes has been implemented. This is due to the fact that the theme of the application relies on text instead of pictograms, and that slow processes do not take up enough time to justify giving feedback about them.
2. **Import of single JPEG images:** Done.
3. **Import of multiple JPEG images:** Done.
4. **Drawing diffusion curves:** Done.
5. **Edit diffusion curves:** Done.
6. **Erase diffusion curves:** Done.
7. **Snap tool:** As this relies heavily upon image processing, it was decided that due to time constraints the primary focus would be on diffusion, which is also heavily dependent on image processing.

8. **Covering parts of an image for exclusion for automatic acquisition:** As automatic acquisition has been removed from the requirements, it follows logically that this feature was removed as well.

9. **Automatic acquiring of curves:** not implemented due to time constraints and primary focus on other features –such as diffusion.

10. **Edit automated curves:** Curves can be edited, however automatic curves are no longer generated as mentioned earlier.

11. **Interpolation between frames:** Done.

12. **Edited interpolated curves:** Done.

13. **Optimization of existing image sequence interpolation algorithms:** Due to the shift of focus to diffusion, this was not implemented.

14. **Labels:** Done, however it was chosen to use 3 labels (or layers) instead of 9, as this showed to be a bit of overkill.

15. **Zooming capabilities:** Done, from 0.5x to 5x zoom.

16. **Save/load functionality:** Done.

17. **Output frames to JPEG:** Done.

Should haves

1. **Drawing diffusion curves:** Done.

2. **Artistical stylization:** Done, however this is not the main focus, but it is possible to do this by drawing curves and coloring them.

3. **Coloring of diffusion curves:** Done, diffusion curves can be given any color in the visible color spectrum.

Could haves

1. **Import of movie formats:** Not implemented due to complexity and not within the focus of this tool.

2. **Output of movie formats:** Not implemented for the same reason as importing movie formats.

3. **Switching between interpolation algorithms:** Not implemented, the chosen interpolation algorithm functions correctly. More choices of interpolation techniques might become too confusing for the user.

Would haves

1. **Definition of distribution curve of every interpolation:** Not implemented as the focus changed from interpolation to diffusion.

2. **Parallel processing of curves:** OpenGL takes care of this.

3. **Parallel processing of interpolation of diffusion curves:** Depends upon implementation in the used libraries. The tool described here does not support this in itself.

4. **Multithreaded application structure:** Done through WPF, .NET and Windows 8. The tool described here does not support this in itself.

Taking into account the complete change of focus that occurred during development, some of the requirements have been changed accordingly and some have been removed completely. All in all the authors believe that the application meets most of the goals that were set by the client and the coordinator of this project.

10 Conclusion

During this project, the authors of this paper have created a unified framework that facilitates the creation, editing and exporting of diffusion curves, in order to augment two dimensional images.

The assignment that was given in the beginning was broad, so the team had to do sufficient research in the field of diffusion curves and interpolation, in order to gain a clear understanding of the possibilities in this area.

During one of the first meetings with the supervisor and the client, it became clear that their intentions were to focus on a tool that would let users draw diffusion curves on a sequence of images, interpolate the drawn curves over multiple images and give users the possibility to export these curves.

After some intensive research, the algorithm for diffusion that was chosen was decided to be the fast implementation by Jeschke et al. As designing an own algorithm would be too intensive and probably not as accurate as existing ones, it was decided to use one that already existed and had largely proved itself. The speed and efficiency of the Jeschke et al. solution was found to best fit the needs this application required.

Besides the diffusion, there was also the need for an interpolation algorithm. This would allow the user to draw diffusion curves only once, after which the application would infer these curves over the rest of the image sequence. Here again, it was decided to make use of existing solutions, as these have proven themselves to be adequate and tested well enough to give a result that can be trusted. Through the use of OpenCV, a version of the Pyramidal Lucas-Kanade was implemented which showed good results in use tests.

In order to make the user experience of the application as smooth as possible, it was decided that curves should be defined using a combination of cubic Bézier curves and Catmull-Rom splines. This enables full control over the system, while making sure the application is easy to use and never gets too confusing. Besides this, the context based implementation of the editor makes sure that the editor does not get bloated with too many buttons, possibly confusing users. In order to keep this confusion as minimal of possible, the application relies heavily on the use of colors to indicate in what state the application is, clearly showing the user what he is able to do at any given moment.

In order to keep the application easily extensible and keeping dependencies low, the system design is based on a compromise between MVC en MVVM. This ensures an easily extensible system with a clean structure which also proves a good basis for testing. Running a continuous combination of unit tests, integration tests and regression testing from the start of development guarantees that errors were found in early stages, making them easy to fix. This results in a well tested system to which new features can easily be added in the future.

11 Recommendations

In this section, recommendations for possible improvements of the system will be listed.

11.1 Improvements

Cyclic paths The current editor only allows for non-cyclic paths to be drawn. Allowing the user to draw cyclic paths allows for more complex shapes that do not interfere with themselves like what would happen with a non-cyclic path. Since the current underlying data structure is a linked list, only the logic managing the adding and removal of points to the path would need to be adjusted for a cyclic path.

Layers Though the current system has some layer support, at the moment it is only used for interpolation. Implementing a proper layer system that would separate the diffusion process between layers could be used to improve the visual quality of images. We believe the layered system, combined with transparent colors for the diffusion could result in an effect similar to the occluding boundaries as described by Bezerra et al.

Diffusion Curves The diffusion curves as they currently stand are based on the implementation of Jeschke et al. Several other implementations are available that provide improved control over the current implementation. Though no current approach is known to the authors at the time of writing, finding a combination of the controlability of Bezerra et al. and the speed of Jeschke et al. would greatly improve the current system as a tool for constructing diffusion curves.

Interpolation The interpolation algorithm as it currently stands uses the points of the path as sparse feature set for the interpolation. However, the underlying features of these points might not always be the most useful to track. Imagine for example a uniform field that gets translated, with a path point in its center. The algorithm would produce little offset for the point, though in reality a translation is taking place. Using a separate feature-detection algorithm would produce much more interesting points to be tracked, which can then in turn be used to determine the transformation of the path points -for example through bi-linear interpolation of the closest four points. The authors expect that this would produce fewer visual errors with the interpolation.

11.2 Unmet requirements

One feature that was originally in the requirements but didn't make it into the final product was the snapping tool. Since the system is mainly designed for image sequences, and thus as an augmentation tool, the authors believe such a functionality would provide the user with an extremely useful automation tool.

Besides this, the ability to automatically generate curves based on edge detection could prove to be of great value. This way, large sequences of images can automatically be augmented with diffusion curves, allowing the user to later on edit eventual errors that occurred during this process. This would relieve the user of doing a lot of work and could greatly enhance the workflow.

Another feature that is still missing from the application is the ability to blur parts of the diffusion curve. Though technically not a requirement itself, it is part of the definition of a diffusion curve. At the time, it was not added due to time constraints and because the added value was too low compared to the complexity of its implementation. However, since it is part of the diffusion curve definition, it should definitely be included.

References

[1] Adobe illustrator. http://www.adobe.com/en/products/illustrator.html, july 2013.

[2] Adobe photoshop. http://www.adobe.com/en/products/photoshop.html, july 2013.

[3] Apple final cut pro x. https://www.apple.com/finalcutpro/, july 2013.

[4] Bezier pen tool — flash professional cs5. http://helpx.adobe.com/flash/kb/bezier-pen-tool-flash-professional.html, july 2013.

[5] C# - xna catmullrom curves. http://stackoverflow.com/questions/14915849/xna-catmullrom-curves, july 2013.

[6] Cximage. http://www.xdp.it/cximage.htm, july 2013.

[7] Gimp - bezier selections. http://www.gimp.org/tutorials/Bezier_Selections/, july 2013.

[8] Imagemagick: Convert, edit, and compose images. http://www.imagemagick.org/script/index.php, july 2013.

[9] Microsoft visual studio. https://www.microsoft.com/visualstudio/eng, july 2013.

[10] Opencv — open source computer vision. http://opencv.org/, july 2013.

[11] Vtk - the visualization toolkit. http://www.vtk.org/, july 2013.

[12] John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

[13] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[14] Herbert D Benington. Production of large computer programs. *Annals of the History of Computing*, 5(4):350–361, 1983.

[15] Hedlena Bezerra, Elmar Eisemann, Doug DeCarlo, and Joëlle Thollot. Diffusion constraints for vector graphics. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 35–42. ACM, 2010.

[16] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5, 2001.

[17] Dempsey Chang, Keith V Nesbitt, and Kevin Wilkins. The gestalt principle of continuation applies to both the haptic and visual grouping of elements. In *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint*, pages 15–20. IEEE, 2007.

[18] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting software architectures: views and beyond*. Pearson Education, 2010.

[19] Carrie Cousins. Making it work: Flat design and color trends. `http://designmodo.com/flat-design-colors/`, july 2013.

[20] Klaus Dorfmüller-Ulhaas. Robust optical user motion tracking using a kalman filter. In *10th ACM Symposium on Virtual Reality Software and Technology*. Citeseer, 2003.

[21] Wikipedia editors: Wdror-wsu ap and Regis Frey. Mvc-process.png, july 2013. [Online; accessed 13-July-2013].

[22] Prof. Dr.Elmar Eisemann. Test and motion information for 2d image sequences. `http://blackboard.tudelft.nl/`, 2013.

[23] Aaron Evans. Thoughts on nunit and mstest. `http://fijiaaron.wordpress.com/2013/02/05/thoughts-on-nunit-and-mstest/`, july 2013.

[24] David J Fleet and Allan D Jepson. Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1):77–104, 1990.

[25] Martin Fowler. Presentation model. `http://martinfowler.com/eaaDev/PresentationModel.html`, july 2013.

[26] Peter Freeman and David Hart. A science of design for software-intensive systems. *Communications of the ACM*, 47(8):19–21, 2004.

[27] Duncan Haughey. Moscow method, july 2013.

[28] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1):185–203, 1981.

[29] Stefan Jeschke, David Cline, and Peter Wonka. A gpu laplacian solver for diffusion curves and poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 28, page 116. ACM, 2009.

[30] Lurng-Kuo Liu and Ephraim Feig. A block-based gradient descent search algorithm for block motion estimation in video coding. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(4):419–422, 1996.

[31] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[32] Microsoft Developers Network (MSDN). Implementing the mvvm pattern. `http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx`, july 2013.

[33] Brad A Myers. Why are human-computer interfaces difficult to design and implement. Technical report, DTIC Document, 1993.

[34] Editor of Nexussharp.com. Showdown: Mstest vs nunit. `http://nexussharp.wordpress.com/2012/04/16/showdown-mstest-vs-nunit/`, july 2013.

[35] Reinhard Oppermann. User-interface design. In *Handbook on information technologies for education and training*, pages 233–248. Springer, 2002.

[36] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *ACM Transactions on Graphics (TOG)*, volume 27, page 92. ACM, 2008.

[37] Giorgio Panin, Alexander Ladikos, and Alois Knoll. An efficient and robust real-time contour tracking system. In *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference on*, pages 44–44. IEEE, 2006.

[38] Charlie Poole. Nunit.mocks 2.6.2, july 2013.

[39] Charlie Poole. Posted 05-08-09. `https://groups.google.com/forum/#!msg/nunit-discuss/n37XzCE2Sss/3DbllJDl9VUJ`, july 2013.

[40] Paul Ralph and Yair Wand. A proposal for a formal definition of the design concept. In *Design requirements engineering: A ten-year perspective*, pages 103–136. Springer, 2009.

[41] Trygve Reenskaug and James O Coplien. The dci architecture: A new vision of object-oriented programming. *An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html*, 2009.

[42] Louis L Scharf. *Statistical signal processing*, volume 98. Addison-Wesley Reading, 1991.

[43] Antonio Escao Scuri. Image representation, storage, capture and processing. `http://www.tecgraf.puc-rio.br/im/`, july 2013.

[44] Tobias Senst, Brigitte Unger, Ivo Keller, and Thomas Sikora. Performance evaluation of feature detection for local optical flow tracking. In *ICPRAM (2)*, pages 303–309, 2012.

[45] Scott Stevenson. Implementing the mvvm pattern. http://theocacao.com/document.page/513, july 2013.

[46] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User interface design and evaluation*. Morgan Kaufmann, 2005.

[47] Prof. dr. A. van Deursen. Softwarekwaliteit & testen. http://studiegids.tudelft.nl/a101_displayCourse.do?course_id=26879, july 2013.

[48] Jerome Vincendon. Unit test, nunit or visual studio? http://stackoverflow.com/questions/1554018/unit-test-nunit-or-visual-studio, july 2013.

[49] D Wallach and B Goffinet. Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological Modelling*, 44(3):299–306, 1989.

Technische Informatica

Bachelor Project

Faculty of Electrical Engineering, Mathematics and Computer Science, Mekelweg 4, 2628 CD Delft, The Netherlands

bachelorprojectTI-EWI@tudelft.nl
TI3800 (15 ECTS)

Contact and Coordination

Martha A. Larson
M.A.Larson@tudelft.nl
http://homepage.tudelft.nl/q22t4/

Hans-Gerhard Gross
H.G.Gross@tudelft.nl
http://swerl.tudelft.nl/bin/view/GerdGross/WebHome

Project Proposal
BSc Technische Informatica

offered by

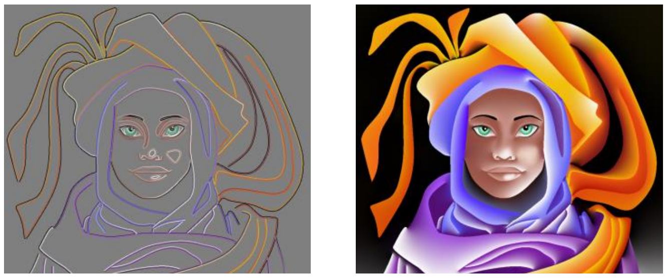
CGV Group

Mekelweg 4

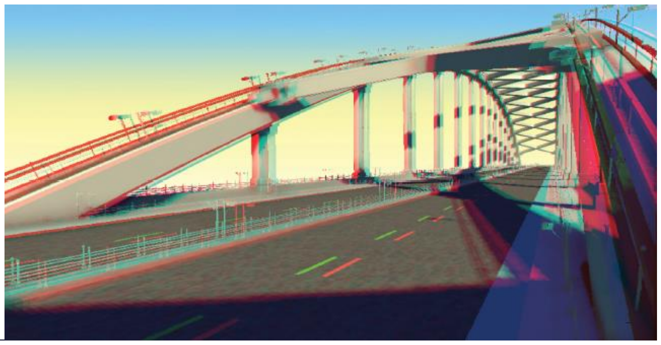
graphics.tudelft.nl
+31 (0)15 27 82528
e.eisemann@tudelft.nl

Project description

Depth and Motion Information for 2D Image Sequences

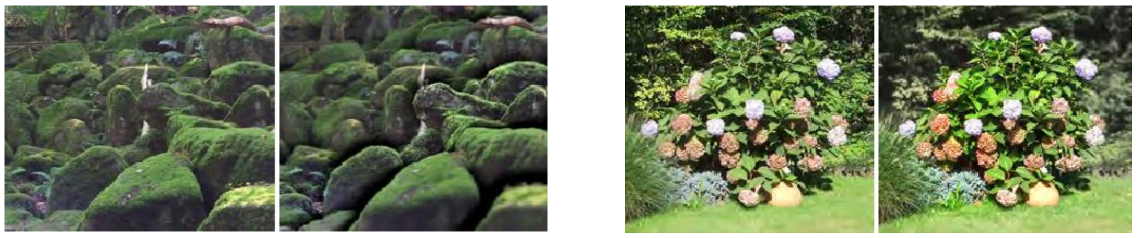

A few diffusion curves can define a complete image. Information is diffused out of the curves into the surrounding space

This project aims at the augmentation of a 2D image sequence by depth and motion vectors in a unified framework. Hereby, it will be possible to interpolate between images to produce high-refresh rate movies, to compute stereo images, or to perform artistic emphasis (adding motion blur, depth-of-field, stylized stereo etc.).
Initially the idea is to rely on a diffusion process leaking out a few user-defined constraints (curves, points etc.), which will spread depth information over the image. As a starting point, diffusion curves will be used, whose correspondences will allow us to define motion. This additional information could be easy to compress and store and users might be able to download this additional information to integrate it in their standard video codecs to benefit from high-quality 3D and frame interpolation.


An anglyph stereo image produced from a single 2D image. The second view was created in a warping process derived from the associated depth information.
Although not perfect, the illusion of depth is quite convincing because the human visual system is relatively tolerant with respect to depth artifacts.

There are many research questions involved in this project:

- Primitives - What are the right primitives to allow a quick augmentation of a standard 2D movie?
- Interface - What is an intuitive and easy-to-use interface? How to help the user?
- Diffusion - How to (efficiently) compute the diffused information based on the constraints?
- Sequences – How to transfer information from one frame to the next?
- Compression - How to compress information and approximate the user input to reduce file size ?
- Applications – What applications are possible? (Artistic) Stereo, emphasis, segmentation…


Depth-based artistic modifcations: Depth-of-field and sharpening, saturation and color hue.

Project team

Supervision

Elmar Eisemann, e.eisemann@tudelft.nl, +31 (0)15 27 82528

Student team

- Jerome Vincendon
- Maniek Santokhi
- Alex Kolpa

DELFT UNIVERSITY OF TECHNOLOGY

Plan of Action

DEPTH AND MOTION INFORMATION FOR 2D IMAGE SEQUENCES

Authors:
G.A. Kolpa
M.Y. Santokhi
J.R.J Vincendon

Abstract

This document will describe how the Bsc. project Depth and Motion Information for 2D Image Sequences for the Computer Graphics and Visualization Group will be approached by the authors. It will contain any agreements between the authors and the client and will state how all the requirements will fit into the available time for this project.

May 17, 2013

1 Introduction

This project was set up by Elmar Eisemann to address the lack of a unified framework to augment 2D image sequences with depth and motion vectors. This information can then be used to apply further processing on the image sequence like image enhancement, convert it to a stereographic sequence or performing artistic emphasis.

1.1 Motivation

The motivation from this project came from discussions with the Computer Graphics and Visualization Group about possible Bsc. projects. The project was formulated as such because there is no unified framework for creating depth and motion vectors on image sequences, which could prove to be a useful tool for the Computer Graphics and Visualization Group and possible future research.

1.2 Approval and adjustments

This plan of action will be signed by the authors as well as the client. Any changes that need to be made to this document will be discussed, and a new version will be released.

1.3 Document structure

This document is split up into four sections. Firstly, the project assignment is described. Secondly, a schedule is given where a weekly plan will be laid out for the project. Thirdly, the project plan is used to structure and organise this project. Finally, the quality assessment of our product and process will be detailed.

2 Project assignment

2.1 Project situation

The standard frame rate in the film industry is 24 frames per second, for a variety of reasons [1]. However, several studios have recently started experimenting with recording films at 48 frames per second, most notably Peter Jackson The Hobbit [2]. Though the increased frame rate has some drawbacks at this point of time, many believe the standard has a future in film [3][4]. Many older films have been recorded in 24 frames per second, and there has been an interest in converting these to 48 frames per second. Several techniques exist to make this transformation, such as Twixtor and Optical Flow, but there are cases where they provide less than optimal quality [5].

2.2 Project goals

This project aims to improve existing techniques such as the ones described in the previous section with the input from a user. This user input is provided through diffusion curves, which can then be used to improve the visual quality of the interpolated frames.

2.3 Assignment description

The assignment description is specified in the document Depth and Motion Information for 2D Image Sequences as created by Elmar Eisemann. The main focus of this project will be on creating an application to apply interpolation between multiple frames of an image sequence to create a frame rate that is higher than that of the original content.

2.4 Final product

The final product will be a single, stand-alone application, where the user can overlay curves on an image sequence. These curves can be manipulated in the image itself as well as transformed over time. This information can then be used to interpolate the image as defined by the curves to create an artificially increased frame rate. The output could either be an edited sequence with the new information directly applied to it, or a special encoding of the sequence, perhaps added as part of a special container for the sequence.

2.5 Demands and restrictions

The main demand for the application is to provide an intuitive interface for the user to create augmented image sequences at a high speed. This means the process should run real-time or at least at an interactive rate, to make it possible for the user to quickly provide feedback to the process. This also means that any feedback provided by the user can be made relatively fast, to keep the workflow going. Because part of the project is a code evaluation by the Software Improvement Group, there is a large focus on quality of code and testing. This means both unit tests as well as user tests should be done regularly, to guarantee a product of high quality. The concept of this project is to incorporate diffusion curves into an existing technique, so no completely new techniques will be created.

2.6 Conditions

Because the product uses a technique that is relatively new [6], the authors will need to get a deep understanding of the underlying techniques as specified in the assignment description, as well as a clear understanding of the existing techniques used for interpolation.

3 Approach and schedule

To come to a proper conclusion of the project, several steps need to be taken, which have partially been described by E. Eisemann in the proposal. Based on these steps, a division has been made to allow for a partial parallel approach of the problem. This has been planned as followed:

Alex	:	Primitives, Diffusion, Compression
Jerome	:	Techniques
Maniek	:	Interface, Sequences, Storage

Possible further subdividing of these problems might prove to be necessary in a later stage. This document should be updated accordingly. For the implementation planning of the final product an agile approach will be maintained. This allows there to be a working prototype at all times, as well as giving ease of testing. The approach will start off with the construction of an emergent architecture design. Based on this design several spike solutions will be made by the project member to test possible implementations of the different components. These spike solutions are then combined into a single application which will be the prototype of the final product. After the creation of the prototype, it will be iterated upon through several sprints

4 Project plan

This section will detail the plans that will lead to the successful conclusion of this project, as well as any agreements that need to be made.

4.1 Organisation

The responsibilities of each group member have been partially described in section 3. At this stage of the project, each group member is responsible for the implementation of his own research field. He is also the main focal point, meaning that he is the one to address issues to concerning certain responsibilities. An extension to the list described in section 3 is provided below for each group member. This way, a team of experts is created, each with knowledge about a specific section. This should provide for a fast implementation, a clear group structure as well as an in-depth project report. The responsibilities as defined at this point are distributed based on the different research questions as well as the SCRUM roles:

Alex	:	Primitives, Diffusion, Compression, Product owner
Jerome	:	Applications, Quality Engineer
Maniek	:	Interface, Sequences, Storage, Scrum master

At the beginning of each week a specific task distribution will be created, based on the other tasks and responsibilities each project member has.

4.2 Participants

Each member of the project is expected to be present according to verbal agreements made by the project group. Due to the lack of a fixed workplace, work-sessions will be planned based on the availability of project rooms in the TU Delft library. The project members will get together as often as possible to provide a productive environment, as well as allowing for quick feedback. Each project member is expected to have extensive knowledge in the field he has been assigned to, as well as a proper understanding of the fields of the other project members. This way, every member has knowledge of the entire project and can supply feedback where needed.

4.3 Administrative procedures

A daily meeting will be held where each member will be asked what he did, what he is going to do and what problems did he face. This will take 15 minutes maximum. A biweekly meeting will be held to make sure the project is up to date, as well as plan out the coming iteration. This plan will detail the tasks at hand and the person responsible for these tasks. Every two weeks a progress report will be filed and sent to the client. This provides them with an update of the project, and the project members with a better overview of the progress so far.

4.4 Financing

For this project, financing is not deemed necessary to result in a finished product.

4.5 Reporting

As described in section 4.3, both minutes as well as progress reports are made available to the client. Frequent meetings with the client will be held as well to update them on the progress and get feedback about the current situation of the project.

4.6 Resources

The resources available to the project members at this point in time consist of non-fixed room which is reserved weekly at the TU Delft Library. Every group member is expected to bring his own laptop to work on the project, and that this laptop is adequately powerful enough to develop the product.

5 Quality assurance

For this project, several measures to assure the quality of both the product as well as the project have been taken. Firstly, there will be the frequent meetings with the client to assure the product is functioning as desired and that the project is not falling behind. Secondly, biweekly progress reports are sent to the client to give a progress update of the project. Thirdly, the product will be submitted for code evaluation to the Software Improvement Group to guarantee a good code base. Lastly, the product will be thoroughly user tested as well as unit tested, to provide a proper experience for the user and to make sure the product is stable.

References

[1] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. A tutorial on magic numbers for high definition electronic production. In *132nd SMPTE Technical Conference, Preprint No. 132-119*, page 1. SMPTE, 1996.

[2] Warner Bros. The hobbit: Unexpected journey, 2013.

[3] Wesley Fenlon. 48 fps and beyond: How high frame rate films affect perception, 2013.

[4] Brendan Bettinger. 2013, 2013.

[5] Mitch Payne. Twixtor vs optical flow fcpx, 2012.

[6] Alexandrina Orzan, Adrien Bousseau, Holger Winnemoller, Pascal Barla, Joelle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *Proceedings of ACM SIGGRAPH 2008*, volume Volume 27 Issue 3. SIGGRAPH, 2008.

A Planning

The following weekly schedule shows a broad planning.

Table 1: Weekly planning

Project week	Date	Tasks	Notes
1	22-04	Picking platform Setting up platform Reading on subject Working on Plan of Action	
2	29-04	Working on Requirements Analysis Document Reading on subject	Queensday
3	06-05	Turn in Requirements Analysis Document Turn in Plan of Action Working on Preliminary Report Setting up testing system Setting up Sprints	Hemelvaartsdag
3	13-05	Turn in Preliminary Report Start of Sprint 1	
5	20-05	Sprint 1	
6	27-05	Start of Sprint 2	
7	03-06	Sprint 2	
8	10-06	Start of Sprint 3	
9	17-06	Sprint 3	
10	24-06	Start of Sprint 4 End of Sprint 4	Exams
11	01-07	Turn in finished product Turn in finished report	Exams

Note that this schedule is subject to change due to the agile approach of the project. Also note the broadly defined 'Sprints'. For each Sprint a document is created which details what needs to be done, what went right the previous time and what can be done better.

DELFT UNIVERSITY OF TECHNOLOGY

Requirements Analysis

DEPTH AND MOTION INFORMATION FOR 2D IMAGE SEQUENCES

Authors:
G.A. Kolpa
M.Y. Santokhi
J.R.J. Vincendon

Abstract

This document describes the system requirements for the Bsc. project Depth and Motion Information for 2D Image Sequences for the Computer Graphics and Visualization Group. It covers the proposed system through functional and non-functional requirements. The first details verifiable functions on an importance scale. The latter encompasses requirements regarding quality, platform and process. Also the system model will be discussed which describes the system components and architecture as well as the graphical user interface, use cases and testing of the system.

May 9, 2013

Contents

1 Introduction

This document provides insight in the requirements of the product to be made. This product is a piece of software that acts as a complete system to interpolate images via diffusion curves. It contains all the functional and non-functional requirements of the product as well as the system model and serves as a contract between the project members and the client.

This chapter describes the purpose of the system, the scope of the system and the project objectives to give an overview of what needs to be done product and project wise.

1.1 Purpose of the system

The purpose of this system is mainly to create a platform for the Computer Graphics and Visualization Group to use for motion interpolation research, as well as to provide an answer to the following questions: "In what way can diffusion curves be used to improve the visual quality of existing image sequence interpolation algorithms?" and "In what way can the interface be made as intuitive as possible for the user with this new system?"

1.2 Scope of the system

The main focus of the application is to use interpolation in order to augment a 24p movie sequence to a 48p sequence, while we want keep the possibility open of using other frame speeds as well.

This will be done by either letting the user draw diffusion curves on single frames after which the application will interpolate a new frame between 2 existing (key)frames, or by letting the system automatically draw diffusion curves over frames, as discussed by Orzan et al [1]. A combination of these 2 approaches is a possibility as well, e.g. a user could be able to edit the automatically generated diffusion curves should the application have inferred wrong edges.

The other focus is on building an intuitive interface for the user. Since this system will be used by the Computer Graphics and Visualization Group as a tool for future use, providing them with an interface that is easy to use is an important part of this system. How this will be achieved, will be explained in the function requirements.

If there is enough time, the application could be extended to support artistic stylisation of movie sequences such as basic color manipulations (hue/saturation) and rotoscoping.

1.3 Project objectives

The objectives for this project can be ordered into three distinct goals: personal, team-wise and system-wise.

Personal objectives include more experience in designing and implementing a software system that pushes the boundary in its domain. The team objectives are making sure that the dynamic and cooperation in the group for such an undertaking are good. The system objectives are to create a system that not only is reliable and intuitively to use but also pushes the boundary with it's newness.

The overall objective is to create a final product that is better and easier to use than existing systems. This coincides with our previously stated research question. This requires a thorough analysis of existing systems, as well as a proper analysis of the requirements and the implementation to give the user a high quality product. How we want to achieve this is detailed in this document.

1.4 Document structure

This document will focus on the proposed system. Firstly, the currently available systems will be detailed, which will be used for reference in our system. After that, the proposed system will be described.

The proposed system consists of several parts. The first part will be the functional requirements, which will also detail the stakeholders for this system. After that, the non-functional requirements will be elaborated. Thirdly, the way these requirements will be verified will be determined. Finally, several system models will be constructed to describe the proposed system.

2 Current system

Though no current unified framework for the interpolation of image sequences with diffusion curves exists at this point, several techniques which interpolate image sequences do already.

Of those, most focus on the automatic interpolation in television, such as Philips Natural Motion, Sony Motion Flow and Samsung Auto Motion Plus [**?**] due to the upped framerate in modern television models (120Hz). Other techniques that are widely used as well are Optical Flow [**?**] and Twixtor [**?**].

Though these techniques are continuously improved, they still contain artifacts, meaning there is room for improvement. This is where our project steps in and it will be discussed at length in the following chapter.

3 Proposed system

This chapter documents the requirements for the proposed system that is going to be created for the BSc. project Depth and Motion Information for 2D Image Sequences for the Computer Graphics and Visualization Group.

It will describe the functional aspects that need to be met for the software to be marked as completed. These are verifiable claims that can be used as a guide to focus while preserving the flexibility of the project. Also the non-functional requirements will be taken into account, which detail what needs to be met in terms of quality, but also cover the platform details and the process requirements. Lastly the System Model will be described. It illustrates the inner workings of the system in an abstract way as well as the interface users will interact with.

3.1 Functional requirements

The functional requirements for the system are constructed based on the MoSCoW Method.[2] The MoSCoW method helps describe what must be met, what should be met, what could be met and what would be met. It gives focus and flexibility which fits the agile approach.

The requirements provide the stakeholders (3.1.1) a clear overview of what needs to be met function-wise and it will meet the needs of the target audience (3.1.2) and the stakeholders for the final product in this project.

3.1.1 Stakeholders

The stakeholders for this project can be listed as follows:

- the client;
- the advisor;
- the developers.

Each stakeholder has input to and agrees on the functional requirements as stated in this document.

3.1.2 Target audience

The target audience for our product are those who will benefit the most by using this system. We can categorize the audience into two sections: the artists and the computer graphics researchers. It is important to keep their interest central while thinking of the software requirements.

These two types of audiences differ quite a bit and so profiling these types gives direction:

1. The artist prefers a certain amount of *artistic freedom*, good *quality output*, *new capabilities* that traditionally wouldn't be possible and *productivity gain* due to time constraints (deadlines).
2. A computer graphics researcher is purely interested in the *general capabilities* of the software, the added *newness* (new to the discipline) and the *reusability* of the code or system to further build upon.

We consider these audience types as extremes on the scale of possible users for our system. By upholding their interest we believe that we can capture anything in between the scale as well, e.g. a film hobbyist or computer science student.

3.1.3 System requirements

The requirements will follow the MoSCoW method. It categorizes the requirements in must haves (must be implemented for this release), should haves (must be implemented for later releases), could haves (are nice to have) and would haves (least critical) respectively. Each category down means that it is of lesser importance to this software release.

The needs listed in the profiling of the target audience will be used as a guideline and will be added to every requirement if it fills the need. The functional requirements are:

Must have

1. Easy to use and intuitive Graphical User Interface (*productivity gain*, *reusability*):
   - Only the necessary things visible or active at that moment on the screen: Context-sensitive User Interface.
   - Shortkeys for Tools (defined as: Must have 4, 5, 6, 8, 7, 9, 11 and Should have 1).
   - Shortkeys for general capabilities (Must have 14, 15).
   - Meaningful pictograms on Tools and buttons.
   - Mark active UI elements.
   - Give feedback to the user on slower processes of the system.
2. Ability to import single images: JPEG Part 5 (*general capabilities*).
3. Ability to import JPEG Part 5 images in a folder at once (*general capabilities*).
4. Ability to draw diffusion curves (in absolute pixels) on every image from the sequence that gets converted to bezier curves (*artistic freedom*, *quality output*, *newness*, *reusability*).
5. Ability to edit the diffusion curves acquired by 4 with a tool (*artistic freedom*, *quality output*).
6. Ability to erase the diffusion curves acquired by 4 with a tool (*artistic freedom*, *quality output*).
7. Snap Tool. Ability to draw diffusion curves (absolute pixels) and change their position in real-time to the closest border (defined as edges of the image): Active Contour Model (*new capabilities*, *newness*, *artistic freedom*).
8. Ability to cover parts of the image with a Tool for the user to exclude that part of the image for automatic diffusion curve acquiring described in 9 (*quality output*).

9. Ability to automate the process of acquiring the diffusion curves of frames in vector form (*productivity gain*, *newness*).

10. Ability to edit the automated diffusion curves from 9 with Tools from 5 and 6 (*artistic freedom*, *quality output*).

11. Interpolate diffusion curves on frames/images between the keyframes. A keyframe is a chosen image/frame of the sequence that already holds diffusion curves (*quality output*, *new capabilities*, *newness*, *reusability*).

12. Let the user edit the automatically generated diffusion curves caused by the interpolation with Tools 5 and 6 (*artistic freedom, quality output*, *newness*).

13. Using the newly defined, interpolated curves from 11 to optimise existing image sequence interpolation algorithms (*newness*).

14. Ability to use up to 9 labels over each frame/image on the image sequences to store information (e.g. diffusion curves) (*artistic freedom*).

15. Ability to zoom in on the canvas that displays the current active frame of the sequence with a range from 40% to 500% (*general capabilities*).

16. Ability to save the working project to the disk for later use as a project file (*general capabilities*, *reusability*).

17. Ability to output frames to a series of JPEG Part 5 images in a folder (*general capabilities*).

Should have

1. Ability to draw bezier constrained diffusion curves on every image from the sequence or still from the movie (*artistic freedom*, *quality output*, *reusability*).

2. Options to artistically stylize the input sequence using diffusion curves: rotoscoping (*artistic freedom*).

3. Ability to give up to 10 colors to diffusion curves (*artistic freedom*).

Could have

1. Ability to import movie formats:

   (a) Raw Video;

   (b) RGBA uncompressed 32 bit Alpha, Red, Green, Blue;

   (c) MPEG-4 with H.264 codec,

   as an image sequence up to 24 frames per second (*general capabilities*).

2. Ability to output the result to formats:
   (a) Raw Video;
   (b) RGBA uncompressed 32 bit Alpha, Red, Green, Blue;
   (c) MPEG-4 format (.mp4, .m4v) with H.264 codec,

   for up to 48 frames per second (*general capabilities*).
3. Switching between available image sequence interpolation algorithms (*artistic freedom*).

Would have

1. Ability to define the distribution curve of every interpolation between keyframes that contain diffusion curves per label (*new capabilities*, *newness*).
2. Parallel processing of the diffusion curves (*newness*).
3. Parallel processing of the interpolation of diffusion curves between keyframes (*newness*).
4. Multithreaded application structure (*newness*, *productivity gain*).

During the creation process of the software, requirements can be added, removed, changed and moved up or down on the importance list. Any changes will be documented with a motivation.

3.2 Non-functional requirements

This section focuses on requirements that are not directly related to functionality. These requirements are sectioned into quality requirements, platform requirements, and process requirements and will be discussed below. **??**

These non-functional requirements are very generally stated to preserve the agile approach.

3.2.1 Quality requirements

Because the entirety of the system is based on the improvement of existing algorithms, the quality requirements will be based upon the amount of improvement over the original system. This includes the items described below

Response time Because of the use of heavy processing (interpolation, generating diffusion curves etc.) one cannot guarantee a system that takes less than half a second to complete a task in order to be categorised as responsive. Therefor no claims will be made by us, not in absolute terms nor relative to existing systems.

Must have 1 states that there must be user feedback with heavy processing task and in general too to compromise on this.

Throughput We cannot do any claims on the computation throughput performance other than the system needs to work on the developing machines.

Resource usage The system shouldn't need to use network bandwidth, since the entire process can be managed locally.

No claims can be made on the memory usage of the system other than that it needs to be usable on the development computers.

Reliability The reliability can only be guaranteed by regression testing and endless tryouts of existing image sequences and comparing the results. This will be discussed in the Testing Plan.

Availability Since this system purely runs locally, it should have no problem with availability as long as the reliability is good.

Recovery from failure Even with all the testing, the system might crash occasionally. When this happens the user can restart the application and open the project the user was working on. Nothing gets saved automatically, the user is responsible for this.

Maintainability and enhancement The system will be highly modular due to it's object oriented approach. Meaning that by the theory of OO, the system will be highly maintainable and enhanceable. [**?**]

Reusability As specified in the previous paragraph, the entire system will be highly modular. Therefor by the OO design, it will be highly reusable for third parties. This satisfies also one item of the target audience.

3.2.2 Platform requirements

This section specifies all requirements and contraints related to the environment and the technology used.

Computing platform The software system will be created on and created for Windows 7 and 8.

Concerning application size, it should not be bigger than 700 Mb so that it can be burned on a CD.

Technology to be used The focus of this system is to demonstrate a new use of an existing technique. This means that many of the techniques used in this system will be from third party programming libraries and systems, to prevent unnecessary work from being done as well as building on stable and tested code.

The entire system will be built on the .NET framework in C#, though the use of libraries and existing code from other languages for certain components is permitted.

The use of Windows Presentation Foundation was chosen for the GUI and graphics because of its flexibility with third party libraries, as well as a relatively low learning curve.

3.2.3 Process requirements

This section provides constraints for the project plan, as well as the development process and methods used.

Development process The entire project will be done with an agile approach. This means that there will be a bi-weekly sprint, resulting in a new prototype.

For quality assurance, regression testing will be used, assuring that new components don't break older ones. More about these approaches can be found in the Plan of Action as well as the Test Plan.

Cost and delivery date These constraints can be found in the Plan of Action.

3.3 Verifying the requirements

This section covers the general approach to verify the functional and non-functional requirements.

During the analysis phase it is important to have the stakeholders check the functional and non-functional requirements, and make sure that they agree with what is stated. In order to achieve this, a meeting with the client will be held before the start of the implementation. During this meeting, the team members will go through this Requirements Analysis document to make sure all requirements are validated by the client.

After every sprint, we check which features have been implemented, if they have been implemented correctly and if they have been tested correctly. It will also be the moment to check if our previously stated requirements need fine tuning in any way. Are we still on schedule with the must haves? Do we have to remove should haves in order to focus on more important must haves? Are we ahead on schedule and can we move could haves to should haves?

Also must every quantified claim in the non-functional requirement (neatly grouped under subsections) be verified in the same manner as the functional requirements are verified. Are they met already? Are they still relevant?

As this will be checked after every sprint, we will have a constant overview of the direction in which we are going, making sure that requirements are verified on a regular basis.

3.4 System model

This section describes the technical aspect of the system to be created. From the most inner workings of the system, to the view the users will see. It builds on the functional requirements from the previous chapter.

Firstly the component design will be discussed. From there on the system architecture can be detailed. To make the translation to actual use of the system by our target audience, use cases are constructed. From that point the a general GUI can be inferred. Lastly the testing of the system will be discussed.

3.4.1 Component design

By examining the functional requirements one can deduce a natural linear flow of what the user can do with the software. This flow is visualized in Figure 3. The figure details 6 distinct component. Each component is responsible for their own set of requirements to fulfil, yet still dependent on each other in a sequential, linear way by design of the system.

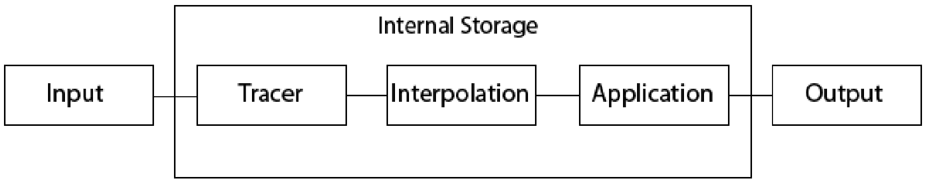


Figure 1: Software components

An exception to this flow is the Internal Storage, which will be detailed below as well as all other components.

Input The input is the video the user wants to enhance. In order to work with the video it needs to be converted into an image sequence.

The functional requirements states in the Must Haves that it needs to be able to handle single JPEG Part 5 images (Must have 2), as well as the JPEG Part 5 images series in a folder (Must have 3). In the Could haves it states that there must be movie input (movies can be converted into image sequences), although it is not urgent for this release of the system (Could have 1).

Tracer The Tracer component encompasses the tools the user can use to draw curves as well as the transformation into diffusion curves.

The functional requirements state that the curves can be drawn freely, with bezier curves, with snapping, they can be edited, erased, covered and automated (Must have 4, 5, 6, 8, 7, 9 and Should have 1).

After acquiring artistic input, the lines can be be transformed into diffusion curves or can aid in the automatic acquisition of the diffusion curves (Must have 8, 9).

This system component solely focuses on the artistic abilities to draw curves and the transformation to diffusion curves, including the automated processes around it.

Interpolation After acquiring the diffusion curves on the keyframes, an interpolation process will be done. This is controlled by the interpolation component.

Must have 11 defines this interpolation of the diffusion curves and Must have 13 defines the interpolation of the pixels within the curves. The editing of these automatically generated curves belongs to the Tracer component.

Applications A field in which a lot can be gained is artistic stylization.

Thanks to the use of diffusion curves, its pretty easy to add tools to visually augment user given inputs to our application. This, for example, could be of great help in the world of animation, as we can easily define color points on diffusion curves as stated in Could have 3.

Further development of this technique could give artists a great way – besides the interpolation – to create visually attractive effects, e.g. rotoscoping, changing hue/saturation or adding depth.

These added applications don't have a high priority during this release of the system.

Output The output is the result of how the user wants it to be. There are several features described for this component in the functional requirements.

In the Must Haves it states that the output can be in images sequences (at a higher frame rate) in a folder, 17). It would also be nice if the result can be exported to various video formats as well (Could have 2).

Internal storage The Internal Storage component encompasses the tracer, the interpolation and the application, because all these system components need a way to get information from each other and store it.

The Internal Storage consist of two parts: the in-memory storage and the disk storage. The in-memory storage describes all the elements of the project that can be edited in an efficient way in the RAM. The disk storage describes all the elements of the project that can be edited in an efficient way on the disk. The latter one is described in (Must have 16), the former is implied when developing the system.

Neither Input nor Output components are part of the internal project storage. They are just the serialization to and from our internal storage respectively.

3.4.2 System architecture

The use of .NET in combination with C# and WPF laid out the system architecture for us. It makes use of the Model View Controller model. Also it already has a lot of interfacing components, for instance to do Graphics, to do GUIs, to do networking etc. We will follow the natural flow of .NET, every other path we take will be documented and motivated.

3.4.3 Use cases

Several use cases will be constructed based on the natural flow of the component design as seen in 3.4.1. These use cases aid in visualizing a GUI (3.4.4).

It is important to note the generality of these use cases and the incompleteness compared to the functional requirements. We kept it general to preserve our flexibility while at the same time give an overall idea how the software can be used.

Use case 1:

Use case	`Import image sequence`
Actor actions	*System responses*
1. Open file browser.	2. Displays native file browser.
3. Select and open image sequence file.	4. Loads image sequence into application.

Use case 2:

Use case	`Trace image to create curves`
Actor actions	*System responses*
1. Select pen tool (Must have 4).	2. Updates GUI to make button 'active'.
2. Use pen tool and draw the bounds on the frame.	2. The GUI renders the lines as absolute position over the frame in real-time.

Use case 3:

Use case	`Add diffusion`
Actor actions	*System responses*
1. Click button to diffuse the drawn lines.	2. Checks if there are any line on the active frame. If there are lines, then it diffusifies it.

Use case 4:

Use case	`Interpolate keyframes`
Actor actions	*System responses*
1. Select the first keyframe.	2. Checks if it's a valid keyframe.
3. Select the second keyframe.	4. Checks if it is a valid keyframe.
5. Click a button to interpolate.	6. Interpolates the diffusion curves and frames to the set number of extra frames.

Use case 5:

Use case	`Application: Artistic Stylization`
Actor actions	*System responses*
1. Select the type of stylization and click to apply it.	2. Add the stylization to the image sequence.

Use case 6:

Use case	`Output image sequence`
Actor actions	*System responses*
1. Click the export button.	2. Opens native file browser.
3. Type in a name	4. Update GUI.
5. Select output format.	6. Update GUI.
7. Click the save button.	8. Serialize from internal storage structure to the specified format.

3.4.4 Graphical User Interface

From the requirements and the use cases (3.4.3) one can infer a Graphical User Interface (GUI).

In order to keep our flexibility during the software creation process, two extremes of interfaces are described in this section. These extremes take the role of the two different types described in section 3.1.2 Target audience, that is: computer graphics researcher and artist. This idea introduces a scale. A scale of possible users we try to satisfy.

Below these types of GUIs will be detailed in a general manner. It provides a designers impression on the functional requirements and system model.

Computer graphics researcher

As described in the Target audience (3.1.2) a computer graphics researcher cares about: *general capabilities*, *newness* and *reusability*. In Figure 2 an impression is created that kept these keywords in mind while trying to satisfy the functional requirements.

The general capabilities are the abilities to save, to create a new project and to export, as well as to view the active frame and to scroll through all the frames via Next and Previous.

The newness is all in the Controls part. The artistic way to trace parts as well as the automatic way to do that. The researcher can select the keyframes, the curves and the amount of added frames.

The reusable part lies in the simplistic separated UI components that can easily be changed and extended.

The overall look and feel is to support the researcher to just focus on the algorithms and the control over them (Control part) as well as the result (Canvas part).

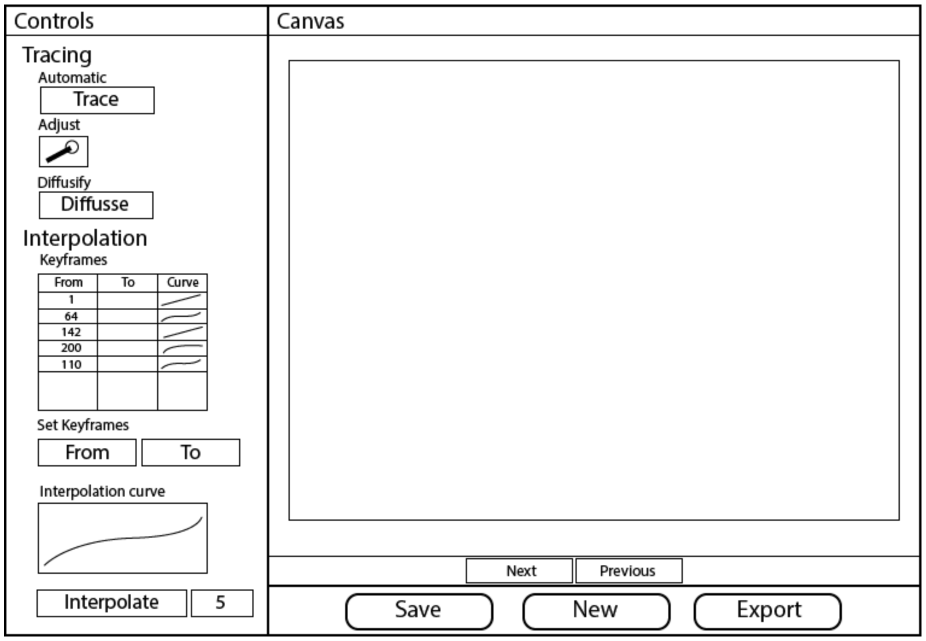


Figure 2: Computer graphics researcher

All these features are described in the functional requirements.

Artist
Also described in the Target audience (3.1.2) is what the artist prefers: *artistic freedom*, *quality output*, *new capabilities* and *productivity gain.* In Figure 3 an impression is created that kept these keywords in mind while also trying to satisfy the functional requirements.

The artistic freedom lies within the Tools part. The artist can draw absolute lines as well as bezier curves but can also choose to automatically outline bounds.

The quality output lies with the composition of capabilities this GUI has (Tools, Interpolate, Frames and the Canvas) to create quality output.

New capabilities are created by the freedom to choose what frames are key and what kind of interpolation can be used and how many new frames must be added. The artist can also choose to keep or delete any intermediate frames that are already part of the image sequence (the checkbox).

The productivity gain is within the automated possibilities of the software. This includes the interpolation and the automatic selection of boundaries.

The overall look and feel is to support absolute artistic freedom and capabilities and this GUI gives this extreme.

All these features are described in the functional requirements.

Our GUI for the system will lie between the two extremes mentioned above

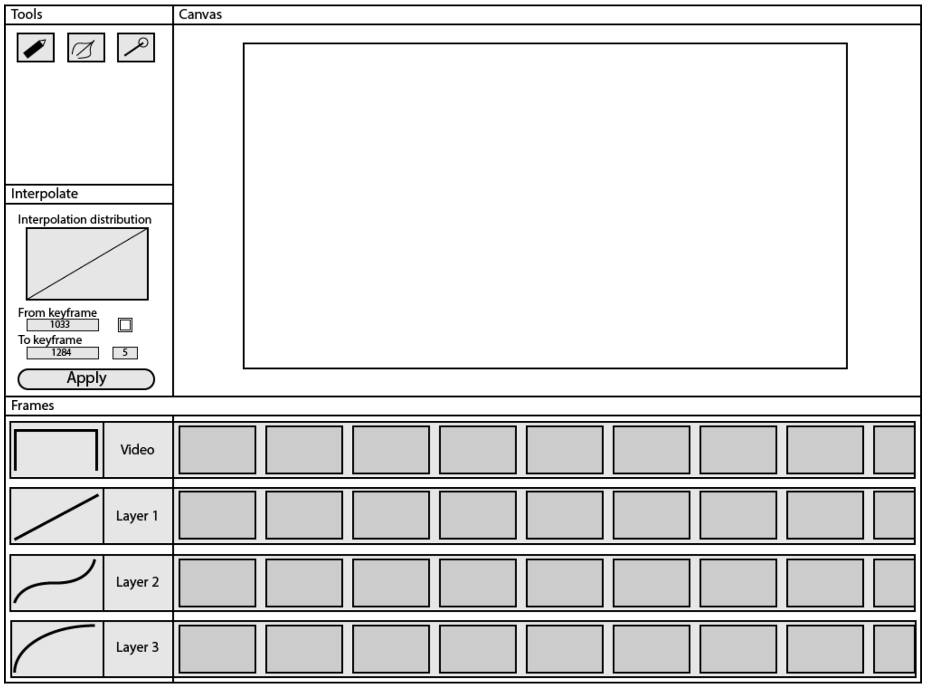


Figure 3: Artist

(Computer graphics researcher and the artist).

3.4.5 Testing the system

As mentioned in the Test Plan, our testing will depend heavily upon unit testing, integration testing and regression testing. This will make sure that our application runs as it is supposed to without producing unexpected results.

First of all, every smallest component of the application will be tested through unit testing to make sure that the most low level building blocks work exactly as they should. Once this is accomplished, integration testing will take place. This means that we group components together, resulting in the functional requirements which we stated above. We can then test these groups of previously tested components, making sure that the interfaces between them work correctly and thus our functional requirements perform as they should. During the development of the application, regression testing will take place after every sprint. This ensures that newly added functionality didn't corrupt already existing functionality, and that bug fixes didn't accidentally break something else.

A full description of the testing methods will be provided in the Test Plan.

References

[1] Alexandrina Orzan, Adrien Bousseau, Holger Winnemoller, Pascal Barla, Joelle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *Proceedings of ACM SIGGRAPH 2008*, volume Volume 27 Issue 3. SIGGRAPH, 2008.

[2] Kevin Brennan et al. *A guide to the Business Analysis Body of Knowledge (BABOK guide), version 2.0.* Iiba, 2009.

DELFT UNIVERSITY OF TECHNOLOGY

Test Plan

DEPTH AND MOTION INFORMATION FOR 2D IMAGE SEQUENCES

Authors:
G.A. Kolpa
M.Y. Santokhi
J.R.J Vincendon

Abstract

To make sure the application we develop matches the standards we envision, we present our test plan. The purpose of this plan is to make sure that we will always have a working, bug free prototype of our application and that new releases can quickly be tested to see if newly developed features dont cause problems, or interfere with already implemented features that ran bug free in the previous iteration.

May 20, 2013

1 Introduction

We want to make sure that errors in the system surface early in development, as common estimates show that a problem that goes undetected and unfixed during early development can turn out to be $40 - 100$ times more expensive to fix when the application is actually released [1].

Besides this, we want our application to behave exactly as we have described in our Research & Analysis Document (RAD), as we don't want our application to behave unpredictably. In order to achieve this, planning of the testing is crucial, ensuring tests are not forgotten or duplicated unnecessarily.

In our Requirements Analysis Document, we have already made sure that the requirements are written with the following guidelines in mind:

- All requirements should be unambiguous and interpretable only one way.
- All requirements should be testable in a way that assures the applications functionality.
- All requirements should be binding because the customer demands them.

This test plan will show an overview of our testing process including: the objectives of testing and tasks for the team (section 2), and our testing strategy (section 3) consisting of Unit testing (section 3.1), Integration testing (section 3.2) and Regression testing (section 3.3).

The Quality Engineer, a team role specifically designed for our project, will make sure that coverage of the code through testing is thorough, while tests are not being repeated unnecessarily.

2 Objectives & tasks

2.1 Objectives

This document will serve as a guideline for testing during development of the application. In order to ensure continuous integration during this process, one of the team members is assigned the role of Quality Engineer (QE). He will be responsible for the overall quality of the project and testing of the application. His task is described in more detail in section 2.3.

Testing throughout development will consist of a combination of unit testing, integration testing and regression testing.

2.2 Tasks

While writing code, every team member will be responsible for writing tests for his code. The Quality Engineer will check these tests in order to make sure that everything is covered and works as it should. If this is not the case, the Quality

Engineer will notify the responsible team member about this in order to resolve issues together as soon as possible.

Bugs can be reported in TFS by any team member, and be assigned to the appropriate team member. If the bug is specific to a piece of code developed by one person, the bug fixing will be assigned to that person, else the Quality Engineer will look into it and take appropriate measures. This can range from fixing the bug himself to working together on it in order to repair it.

2.3 Quality Engineer

One of the key parts during development will be to always have a working version of the application in the master branch. Development will be done in the development branch that will be copied to the master branch after every sprint, when it has been made sure through regression testing that no new features have broken already implemented functionality.

The QE will be responsible for writing the Unit tests and Integration tests, as well as for the regression testing. A part of the testing will be done through user testing, by letting users go through special developed use cases. This is especially necessary for testing of the GUI of the application.

In order to ensure that everything has properly been tested, code coverage will be an important measure to find parts of the application that could be tested more thoroughly.

3 Testing Strategy

To make sure that the application is fully functional at all times, we will be making use of Evolutionary testing [1]. This means that every unit of the application is tested until it is deemed functional, after which new features can be added to the application. The two units can then be tested as an integrated component, adding to the overall complexity as we proceed.

This way of work ensures that we will always have a functional application that could be released at any time if requirements are added in priority order. The modular approach ensures that we dont have to start with a huge test plan right away, but can compose our final test plan from the units that are tested piece by piece. This approach fits our agile way of development as well.

In order to test the GUI, use tests will be written. These will consist of scenario's that thoroughly test every part of the GUI. and will look a lot like what a user would actually do when using our application.

3.1 Unit Testing

With unit testing we take the smallest testable piece of code, test it and make sure it behaves exactly as we want it to [2]. Each unit of the application is tested this way, after which they can be merged into modules and we can test the interfaces between these modules.

As testing attention is given to each unit of the application, this will greatly boost the code coverage and help discovering errors in even the more complex parts of our system.

3.1.1 Drivers & stubs

In order to efficiently use unit tests, drivers and stubs will have to be written. A driver simulates a calling unit while the stub simulates the called unit [3]. The investment of developer time to write these cannot demote the usage of unit testing to a lower level of priority, as this testing provides undeniable advantages, as mentioned above.

In order to reduce the cost of writing drivers and stubs, we will reuse them so that constant changes to our application can easily and frequently be tested without writing large amounts of test code.

3.2 Integration Testing

Integration testing follows logically from unit testing. During this phase, components consisting of two or more tested units are tested to make sure that the interfaces between are working correctly. After this, these components can even be grouped into larger parts of the program, while making sure the interfaces between are doing what they are supposed to.

By using integration testing, we want to make sure that we can easily identify problems that can occur when units are combined. By testing each unit, making sure that it behaves exactly as we want it, we know that errors that occur when we combine these are likely to come from the interfaces connecting them.

The strategy we will adopt for integration testing is the bottom-up approach [4]. This means we test the lowest-level modules first through unit tests, followed by a more integrated test of combined modules resulting in components with actual functionality as described in the RAD.

3.3 Regression Testing

Every time a new iteration of our application will be released, we will test it using regression testing. This means we will be running existing tests to check if implemented changes might have broken anything that was working before [5]. New tests will also be added to make sure that the new functionality works as required. An important thing we have to keep in mind while doing regression testing is to spend as little time as possible doing it, without reducing the probability of finding new failures in existing, already tested code.

In order to be as efficient as possible, we will be using a library of tests that can be run every time we build a new version of our application. This library will mostly consist of automated tests, test cases involving boundary conditions and timing. Besides this, the library will be reviewed after each sprint, in order to remove unnecessary or redundant tests.

Finally, here are some strategies to consider during regression testing:

- Fixed bugs should be tested fast, as the symptoms might have been handled while the underlying cause is still present.
- Look out for side effects of fixes, as these fixes might introduce other bugs in other parts of the application.
- A regression test should be written for each fixed.
- Remove similar tests, keeping the most effective.
- Trace the effects of changes on program memory.

References

[1] msdn.com. Testing overview. [Online; accessed 17-May-2013].

[2] Tom McFarlin. The beginners guide to unit testing: What is unit testing? [Online; accessed 17-May-2013].

[3] Amit Badola. Drivers and stubs. [Online; accessed 17-May-2013].

[4] msdn.com. Integration testing. [Online; accessed 20-May-2013].

[5] webopedia.com. Regression testing. [Online; accessed 17-May-2013].

E SIG Feedback

E.1 First Feedback

[Aanbevelingen] De code van het systeem scoort bijna 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Component Independence.

De code bevat een duidelijke component-indeling, die in zowel de productie- als testcode is doorgevoerd. Dit maakt het makkelijk om na een aanpassing aan de productiecode de bijbehorende tests te vinden. Daarnaast geeft het enig inzicht in de mate waarin de verschillende componenten getest worden. Dat er berhaupt unit tests aanwezig zijn is bijzonder positief, hopelijk lukt het jullie om de hoeveelheid testcode verder te laten stijgen op het moment dat er functionaliteit wordt toegevoegd.

Voor Component Independence wordt er gekeken naar de hoeveelheid code die alleen intern binnen een component wordt gebruikt, oftewel de hoeveelheid code die niet aangeroepen wordt vanuit andere componenten. Hoe hoger het percentage code welke vanuit andere componenten wordt aangeroepen, des te groter de kans dat aanpassingen in een component propageren naar andere componenten, wat invloed kan hebben op toekomstige productiviteit. Deze meting zegt dus iets over de grootte van de interface versus de grootte van de implementatie.

Bij jullie project is er met de component-indeling zelf niet zoveel mis, de lage score voor Component Independence komt doordat te veel bestanden binnen een component door andere componenten worden aangeroepen. Bij "model" zou je dat ook verwachten, maar het is niet direct logisch dat de helft van "diffusion" vanuit de views wordt gebruikt. Het diffusion-component lijkt ook voor de helft uit een datastructuur te bestaan (Path.cs, BezierPoint.cs) en voor de andere helft uit een OpenGL view. Het is aan te raden deze indeling nogmaals kritisch te bekijken en waar mogelijk helderder te maken.

Maar over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

E.2 Second Feedback

[Hermeting] In de tweede upload zien we dat het codevolume is gegroeid terwijl de score van onderhoudbaarheid ongeveer gelijk is gebleven.

Wat betreft de component-indeling valt het op dat jullie, zoals in de vorige analyse werd voorgesteld, het model-gedeelte en het view-gedeelte van Diffusion gesplitst hebben. Dit zorgt voor een lichte toename in de deelscore voor Component Independence, maar de totaalscore blijft 4 sterren.

Het is goed om te zien dat het volume van de testcode net als het volume van de productiecode is gestegen.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.