Temporal Coherency in Painterly Rendered Computer Animation Using a Cognitive-Based Approach

by

Mozhgan Akhgari

B.Sc. (Computer Science), Amirkabir University of Technology, 2006

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

SIMON FRASER UNIVERSITY

Fall 2013

Approval

Name: **Mozhgan Akhgari**

Degree: **Master of Science**

Title of Thesis: **Temporal Coherency in Painterly Rendered Computer Animation Using a Cognitive-Based Approach**

Examining Committee:

Chair:

_____

Dr. Carman Neustaedter
Assistant Professor

_____

Dr. Steve DiPaola
Senior Supervisor
Associate Professor

_____

Dr. Thecla Schiphorst
Supervisor
Associate Professor

_____

Dr. Brian Fisher
External Examiner
Associate Professor

Date Defended/Approved: December 2, 2013 _____

Partial Copyright Licence

Abstract

This thesis proposes a solution to augment temporal coherency in painterly rendered computer animation sequences, using a computer engineering approach. *Painterly* is a cognitive knowledge-based parameterized Non Photo-realistic toolkit for creating artistically rendered still imagery. Therefore, it is incapable of maintaining temporal coherency of rendered animation frames. Consequently, movies rendered by *Painterly* demonstrate a significant amount of flickering. We proposed and developed CPA - a system to enhance temporal coherency in the sequences rendered by *Painterly*. CPA utilizes *Painterly*'s cognitive and perceptual knowledge space and induces coherency in the outputted results, by controlling and executing the main part of frame synthesis process. We created cognitive-based painterly rendered sequences which showed a good deal of improvement in maintaining temporal cohesiveness. Furthermore, by incorporating the element of 'time' in Painterly's frame synthesis process, we expanded its scope from being a still-oriented and state-less toolkit to a more multipurpose and state-full system.

Keywords: Non Photorealistic Rendering; Painterly Rendering; Stroke-Based Painterly; NPR Animation; Keyframing; *Painterly* Toolkit; Cognitive NPR; Temporal Coherence

To my almost forgotten dear self;

this one is for her after all…

Acknowledgements

I would like to express my deep gratitude and respect to Steve DiPaola, my senior supervisor, for his great support and care in all stages of this thesis, for his incredible personality which was a great inspiration and encouragement throughout the way, and for all that I learned from him.

I'm truly and utterly thankful to Hasti Seifi, whom without a doubt has been the greatest help from A to Z of this work.

My greatest appreciation goes to Brendan Vance, a former undergrad student in SIAT, who assisted in the software development of *Painterly* and never got tired of answering my day and night questions about this *'good old black box toolkit'*.

I sincerely thank my dear friend, Adrin, who has been a great help to me, in starting, continuing and finishing this work. Wasn't it for his helps, this work was not possible.

I would like to thank Nahid and Reza, without whom this long overdue work was merely impossible to finish. They have been, they sure are -and I hope that they will remain- a great support in my life, during all my struggles and frustrations.

Big-time thanks to Graeme McCaig, a member of the iVizLab, for graciously and generously sharing his time and knowledge with me. Thanks to him again for being such a great help to me and this work.

I'm forever thankful to my family, my mom and my loving sister, for supporting me from a 'far-far-away land'.

Thanks to my supportive friends, Setare, Maryam, Golnaz and Farzane; not for being any help to this work, but for their mere existence, somewhere in the world, that warms my heart and keeps me going.

In the end, I want to thank Steve DiPaola and Brendan Vance again, for providing me the software programs that I used for this research and the knowledge of using it as well.

Table of Contents

List of Figures

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
CGI	Computer Graphics Imagery
IB-Frame	In-Between Frame
MSE	Mean Square Error
NPR	Non-Photorealistic Rendering
PSNR	Peak Signal-to-Noise Ratio
SBR	Stroke-Based Rendering
XML	Extensible Markup Language

Glossary

Motion Maps	A motion map represents the position and accordingly the velocity and acceleration of an object in the frame during a certain time span.
Optical Flow	Pattern of objects' motion in a scene
Photorealism	The genre of painting that appears to be photographic, by using photographs to collect visual information.
Stateful / Stateless	In the context of computer science, state is used as a set of conditions at a moment in time. Stateful and stateless describe whether a system is designed to note and remember one or more preceding states in a given sequence of interactions with a user, another computer or program, or other outside element, in a reference time.
Temporal Coherency	Maintaining the aesthetic completeness of the animation by controlling the movement and style of the objects, shapes and pixels in the frames over a certain time span.

Statement of Co-Authorship

Painterly toolkit (conceptualized, designed and developed by Steve DiPaola and past iVizlab team members) was the program used for rendering the keyframes and the in-between frames of the movie sequences. I did not contribute to the software development of *Painterly*, but was a researcher using it on several projects. To leverage *Painterly's* knowledge in making animation sequences, I developed a system/toolkit – CPA – to create the in-between frames and make a painted movie at the end, by stitching the keyframes and all the generated IB-frames together.

1. Introduction

This work proposes a solution to augment temporal coherency[1] in Computer Graphics Imagery (CGI) painterly rendered animation/movie sequences, using a software engineering approach. Our proposed system, which is called *CPA*, incorporates aspects of humans' cognitive and perception knowledge space in the frame synthesis process to create a coherent animated movie. This knowledge space is imported to CPA through using the *Painterly* toolkit as a base system. *Painterly* is a stroke-based parameterized Non-Photorealistic Rendering (NPR) toolkit which produces artistic rendered still images. Here, we explain our approach and provide some background information for the key topics and challenges of painterly animation which lead us to undertake this research work. We proceed with explaining our motivation and finish the Chapter with a short overview of the entire thesis work.

1.1. Background and Challenges

Non-Photorealistic Rendering (NPR) is a computer graphics technique for creating imagery inspired by cartoons, painting, drawing, technical illustration, etc. In this sense, it is different from other computer graphics techniques which focus more on photorealism[2]. An NPR system renders the input image into a non-realistic style such as a drawing, painting or cartoon. Different elements of the scene/image can be emphasized, deemphasized or abstracted to convey different amounts of information, appropriate to the desired narrative or artistic purpose. This fact alone makes NPR images very versatile in style and application. The main focus of this thesis work is on

[1] Temporal coherence means maintaining the aesthetic completeness of the animation by controlling the movement and style of the objects, shapes and pixels in the frames over a certain time span.

[2] The genre of painting that appears to be photographic, by using photographs to collect visual information.

stroke-base NPR systems. These systems typically accept an input image (2D or 3D) and create a list of strokes, which are then rendered onto a virtual digital canvas. Each of these strokes also, has a set of style-properties (e.g. size, color and orientation) determining how that stroke will be rendered onto the reference canvas.

Painterly NPR is a subset of NPR with its basic inspiration coming from different styles of painting such as impressionism, expressionism, pen and ink, watercolor, etc. Painterly rendering holds big promises and potentials, with a wide range of applications from arts and design to gaming and even learning and medicine(DiPaola, 2007).

Past studies have shown a relation between the rendering style (photorealistic or non-photorealistic) and humans' perception and feeling about the rendered object. Even within a single style such as painterly NPR, different emotions and perceptions are evoked in viewers, through changing the settings and properties (e.g. brush size, texture, color palette, etc.) (Colton, Valstar, & Pantic, 2008; Duke, Barnard, Halper, & Mellin, 2003; Halper, Mellin, Herrmann, Linneweber, & Strothotte, 2003; Pelachaud & Bilvi, 2003; Shugrina, Betke, & Collomosse, 2006). Furthermore, recent studies by using our lab's *Painterly* toolkit (DiPaola, 2007; "iVizLab - Simon Fraser University," 2013) have demonstrated that textural detail of a painting, also, affects the viewers' gaze pattern, while looking at the reference painting (DiPaola, Riebe, & Enns, 2010, 2013). As well, the colour and stroke types of painterly rendered 3D faces can affect emotion of the viewer (H. Seifi, DiPaola, & Arya, 2011; Hasti Seifi, 2010). All these studies emphasize the strong connection between a painterly rendering style and humans' perception knowledge, which if can be best understood and utilized via computer modeling, creates a vast research potential to exploit.

Painterly is a parameterized cognitive knowledge-based NPR system which came along as a research toolkit to explore the joint areas of art and design, computational science, and humans' perception and cognition. This toolkit was conceptualized, designed and implemented by Steve DiPaola and former members of iVizLab (DiPaola, 2007, 2009; "iVizLab - Simon Fraser University," 2013). While the majority of the works in painterly NPR have focused more on optimizing the algorithms or creating a wider variety of styles; *Painterly*'s goal and effort has mainly been to provide a research tool to bridge between artistic and scientific approaches in the

process of artistic and cognitive knowledge acquisition. It leverages human artists' cognitive knowledge model, of how they approach (create and/or view) a painting work of art, inside the computational process of painterly rendering (DiPaola, 2007, 2009). By creating a rather 'informed' painting procedure, *Painterly* intends to approach the process of artistic painting, the way a human artist does, using computational modeling of the cognitive process. Note, in this thesis, we will refer to the *Painterly* toolkit software system with italics – that is *Painterly* - and the general NPR subfield without italics – that is "painterly rendering" or "painterly NPR". Despite being part of *Painterly*'s research team, in the context of this thesis we will address this toolkit only by its full name, as *Painterly*. Also, we will refer to its research team as *Painterly team.* This is to avoid any possible confusion about the new proposed solution and developed subsystem of this thesis work – called CPA - and the *Painterly* toolkit.

The computer graphics software field of painterly rendering has been around since 1990 (Haeberli, 1990) and is in a somewhat maturing state. However, painterly animation, that is a moving sequence of NPR images into a full coherent animation, certainly is rather a newly-introduces subject in this field. Regardless, it has been getting a considerable attention from researchers. Many research works have been exploring this paradigm, proposing automatic and semiautomatic systems and algorithms for painterly rendering movies or animation sequences, some being more successful in delivering better quality results than others.

Stroke-based painterly rendered animation, which is also the main focus of this work, still faces two main non-trivial challenges. These two issues can be summarized as 1) '*Shower Door Effect*' or 'drifting apart strokes' (Meier, 1996) and 2) lack of temporal coherency, also known as '*Swimming*' (John P Collomosse, Rowntree, & Hall, 2005), '*Scintillation*', or unwanted flickering (Hays & Essa, 2004; L. Lin et al., 2010). The latter issue has been the main subject of this thesis work as well.

The *Shower Door Effect* occurs when the strokes drift apart from a painted object's surface and appear to be stuck more on the view pane. This issue creates the illusion that the painterly rendered image is seen through a semi-transparent shower door with the strokes fixed on the image plane instead of the object surface (Meier, 1996).

Temporal incoherency appears as an uncontrolled motion or flickering in the animation. It severely damages the aesthetics of the resulted sequence. It also distracts the viewer from the sequence-content by producing unwanted perceptual cues (J. P. Collomosse, Rowntree, & Hall, 2005). Therefore, the temporal coherency issue has been a significant pragmatic challenge, attended to by various researchers through many different approaches and techniques such as: object-space methods, motion maps, optical flow, vector fields, etc. (Hays & Essa, 2004; Hertzmann & Perlin, 2000; Litwinowicz, 1997; Olsen, Maxwell, & Gooch, 2005; Park & Yoon, 2008). It should be noted that some amount of flickering in a computer-based painterly animation is acceptable, since flickering has always existed in the many decades of the traditional hand drawn painterly art animation field. So, human viewers are used to perceiving some amount of flickering as the norm in this type of non-cell based art animation work. Therefore, the main goal of computer based NPR in the subject of temporal coherency issue is getting the flickering to an acceptably low level, without introducing other artifacts in the results.

Apart from the common approaches in creating aesthetically pleasant and more coherent results or wider range of styles; making an 'informed' video painting process is something not many notable researchers have paid attention to (Agarwala, Hertzmann, Salesin, & Seitz, 2004; M. Kagaya et al., 2011; L. Lin et al., 2010). Utilizing image-content information and scene semantics in the process of painterly rendering, with an eye on maintaining coherency, is the general research flow of aforementioned works and also this thesis.

1.1.1. Research Goals: Why Should We Care?

Why do we think using an image-content information approach is important in solving the temporal coherency issue in computer based NPR? Increasing the levels of visual expressiveness and semantic effectiveness, especially in a painterly animation, is what animators are trying to achieve via different techniques and methods, from utilizing various external elements such as lighting to the usage of scene compositions. In this way, they not only enhance the general emotional impact of the scene, but also can specifically convey any desired narrative or evoke an emotion by drawing and directing the viewers' attention to a particular region of the frame/movie sequence. Recent eye

tracking based human vision studies have shown that painters can easily and effectively guide the viewers' gaze through a portrait painting piece, just by using some tacit knowledge about perception and painting techniques to exploit this knowledge (DiPaola et al., 2013). There is also some evidence that painters, as early as Rembrandt, were completely aware of this fact and successfully exploited these techniques to tell a specific narrative by affecting the viewers' perception of the scene or sitter (DiPaola et al., 2010). *Painterly* has also shown in psychological studies conducted at the UBC Vision Lab[3] (H. Seifi et al., 2011; Hasti Seifi, 2010) that variable use of NPR colour and stroke properties used on game like 3D facial characters can influence the viewer's sense of the emotion of that character, and therefor NPR colour and stroke choice can be used as a new emotional authoring tool, much like face expression and music is currently, in avatar and video character design. We believe that by incorporating humans' perception and cognitive knowledge in the procedure of video painting, animators can enhance the visual or semantic expressiveness of the animation and narrative of the sequence. And of course, reducing the annoying and distracting visual artifacts, such as flickering, is an inevitable step in achieving a more aesthetically pleasing and effective result.

Painterly is an NPR toolkit which exerts humans' perception and cognitive knowledge space in the computational painting process, but puts forward an unexplored challenge for video painting which is temporal coherency. Focusing mainly on still images, it renders every single frame separately, without passing through the rendering-information to the next frames. Therefore, generated brush strokes are not necessarily cohesive through successive frames of a sequence, in the sense of orientation, length, size and color. Coupling a system like *Painterly,* which combines this cognitive knowledge space with NPR techniques, with a method to alleviate its coherency problem, would expand the scope of *Painterly*; changing it from a 'time-insensitive' and stateless system, to a stateful and multipurpose one[4]. This expansion in the domain of

[3] http://www2.psych.ubc.ca/~ennslab/Vision_Lab/Home.html

[4] In the context of computer science, state is used as a set of conditions at a moment in time. Stateful and stateless describe whether a system is designed to note and remember one or more preceding states in a given sequence of interactions with a user, another computer or program, or other outside element, in a reference time.

Painterly will certainly be a first step in pushing the boundaries of human perception studies further, by providing a wider space to explore and facilitating the research process with a more powerful toolkit, in entertainment, emotion and gesture research, in cognition research and in health areas such as face to face autism research that our lab is undertaking. This expansion is one of the main goals of this thesis work.

1.2. Motivation

The motivation of this thesis work was reinforced when our *Painterly team* ("iVizLab - Simon Fraser University," 2013) collaborated with the former cinematographer of Pixar, Jerrica Cleland and her pre-visualization company[5] as part of an industry-research partnership grant from NSERC Engage[6]. Cleland was interested in pushing the state-of-the-art in NPR, by providing professional stills and animation sequences of a movie scene with a vast variety of visual styles and aesthetics, in an efficient time scale. The purpose of generating these stills and sequences was to help the movie director decide on the visual style, design and look of the scene they wanted to shoot. At the time, their company could only produce a limited range of painterly rendering styles; therefore, they wanted to expand their business potential by including more painterly styles, as well as emotionally authoring of styles, through collaborating with us – DiPaola's *Painterly team* - and using our system – *Painterly* - as an external component.

In this particular Engage, our research task was to work with them to envision a system and process, which could create a vast variety of artistic painterly styles, (eventually for their clients), while keeping these styles controllable and faithful to their economical production pipeline. The idea was to automatically use the knowledge from a 3D authoring system, like *Autodesk Maya,* in *Painterly*. Moreover, we wanted to incorporate the concerns of their design pipeline, controlled within the authored 3D objects, in the typical and common spline-based keyframe facility, used by designers.

[5] Twenty One Inc. http://www.twentyoneinc.com/

[6] http://www.nserc-crsng.gc.ca/

Figure 1.1 illustrates a number of various styles created with *Painterly*. The reference still images are taken from the movie sequence provided to our *Painterly team*.

In the following, there are 4 examples of painterly rendered videos sequences delivered to the TOI Inc. Company. All the videos are provided in the author's website[7]. These sequences were created by *Painterly* rendering the frames of the provided computer graphic movie. The sequences and stills were delivered to their company as the result of our NSERC Engage Collaboration.

Sequence #1.1. FirstScene-OriginalPalette

Sequence #1.2. ThirdScene-PurplePalette

Sequence #1.3. FirstScene-OriginalPalette_BiggerStrokes

Sequence #1.4. FirstScene-OriginalPalette_Blurry

In these example images and videos, the content region maps also contain some non-portrait elements like a 'bottle' (Figure 1.2). These region maps were automatically created and labelled by the company's 3D authoring system, using the industry leading software Autodesk Maya. For painterly rendering the reference animations, first, we selected three different scenes of the provided movie, and then we selected a set of keyframes from each selected scene to be 'painted' by *Painterly*. These generated still images were then interpolated by an uninformed and low-level pixel-based algorithm to create the final painterly animation sequences.

[7] http://ivizlab.sfu.ca/research/PainterlyAnimThesis/

Figure 1.1. ***Stills, taken from the animation sequence, provided by TOI Inc. The stills were rendered by Painterly toolkit with various painterly styles.***

Figure 1.2. An example still image, with the provided region map that contains a non-portrait element [a bottle].

The final results were only semi-coherent - as much as what was achievable at the time. However the pixel-based interpolation process – which is discussed in further detail in the next sub-section - was not anything close to 'smart' in sense of incorporating Painterly's knowledge model and using labeled cognitive regions (See Section 4.3.1 for the term definition) in the procedure of making in-between frames. Therefore, the whole process was not so faithful to the basic concerns of Painterly, plus the fact that the results were aesthetically poor. This fact, reinforced the need for a system, which exerts Painterly's cognitive knowledge space in the frame synthesis process to create a cohesive and cognitive-based animation and also enhance visual and semantic expressiveness of the animation sequence through doing so.

This thesis work sets out to alleviate temporal coherency issues in *Painterly* rendered animation/movie sequences. We propose a solution for making temporally coherent sequences through a more cognitive-based process, using *Painterly*; a process

which knows about the contents of a scene/frame, and uses this knowledge in painting process accordingly – more close in process to how a human painter would attempt to paint a movie sequence.

1.2.1. Pixel-Based Interpolating

We used a low-level pixel-based interpolating method to generate the results of TOI Collaboration task. This method contained a cross-dissolving algorithm to morph between successive keyframes of the *Painterly* rendered reference keyframe-set. The morphing process generated a number of in-between frames – based on the corresponding parameter value provided to the algorithm. These generated in-between frames were then stitched together with the keyframes, at the end of the process, to create a QuickTime movie sequence. The generated in-between frames had poor quality, due to the nature of morphing process. This process worked with the pixels of the source and target keyframes. For every pixel from the source keyframe, the corresponding pixel from the target keyframe was used to generate an *in-between pixel* which combined aspects of both source and target pixels' color value, in a linear manner. The rate of this linear warping also was provided to the cross-dissolving algorithm. The resulting in-between frames had poor object contours and were visually blurry. Besides, none of the generated in-between frames were an actual *painted frame* since they contained no real brush stroke. Moreover, the cross-dissolving technique was unable to perceive any knowledge about the keyframes, as *Painterly* does. Accordingly, the in-between frames were generated without any knowledge. As a direct result, this method failed to satisfy *Painterly*'s important concern which is leveraging aspects of humans' cognitive knowledge in the process of painterly rendering images and generating frames. Still this cross fade of interpolated frames using a pixel based approach is one often employed by the animation industry and is considered a viable approach. Our interest was to improve on this standard technique.

1.3. Synopsis of This Work

Painterly is focused on still imagery. Therefore, any *Painterly* rendered animation sequence lacks temporal coherency by a great deal. This is because every single frame

is analyzed, painted and rendered individually. Further, this individual rendering occurs without passing or considering previous frames' rendering-information or their passes' and strokes' properties such as density, orientations, sizes, and positions. Altogether, as mentioned before, *Painterly* conveys a human's cognitive knowledge space in painting images but does not support temporal coherency.

As a follow up to the aforementioned NSERC Engage (Section 1.2), this thesis work focuses on creating temporally coherent and cognitive-based painterly animation from CGI movie sequences. We create a whole new temporally coherent animation sub-system to *Painterly*. This wholly new subsystem is called CPA, standing for Cognitive-based Painterly Animator. CPA uses *Painterly* and its NPR single image calculations as a base. This means that *Painterly* provides CPA with inputs and a plan of how to generate all the new coherent frames. Moreover, *Painterly* is used to re-render CPA's outputs as well; this way, CPA leverages *Painterly*'s cognitive knowledge model in its own frame-synthesis process. Our goal in designing and implementing CPA was to not only make temporally coherent cognitive-based painterly animation, but also automate the process so that CPA could be applicable and adaptable in an industry production pipeline, bringing coherent and improved animation sequences to the open source *Painterly* toolkit and its many research and production uses in entertainment, health, arts and sciences.

Looking at existing literature in painterly animation field, our work is mostly motivated by Lin et. al. and Kagaya et.al. and to a lesser extent by Hertzmann (M. Kagaya et al., 2011; L. Lin et al., 2010; O'Donovan & Hertzmann, 2012), in the sense of using frame semantics, optical flow and objective functions. We extract the information of meaningful cognitive regions, encapsulated in our inputs, provided by *Painterly* toolkit, and set up a meaningful connection between these cognitive regions throughout the successive keyframes. Accordingly, we maintain this connection during the frame-synthesis, painting and rendering phases. We use this connection as an anchor-point in generating cohesive and yet cognitive-based frames, based on the scene's optical flow and through using objective functions. Therefore, we believe our approach has benefits to the general field besides its use within Painterly.

As an attempt to exploit Painterly's potentials to the utmost and attend to its temporal coherency problem, we proposed a new solution to cover for Painterly's lack of coherency through a software engineering approach. This new approach which is the main work of this thesis, takes a systematic and quantifiable method to design and develop a new subsystem – introduced as CPA - to take care of temporal coherency issue. In the context of this thesis works, the title 'CPA' is used interchangeably for the proposed model and the corresponding developed subsystem. CPA is a parameterized system that works with Painterly as an external subsystem/component and/or can be added to Painterly's main body and used as an internal component later on. The working scenario of CPA, in each round of painterly rendering a reference computer animation/movie sequence starts with a set of keyframes. The reference animation/movie is manually or automatically keyframed. This keyframe-set is then rendered with Painterly to produce CPA's input. To induce coherency in the generated frames, CPA intervenes in Painterly's frame-synthesis procedure. By out-sourcing a part of this process and taking over the execution of that part by itself, CPA induces cohesiveness in the outputted results. To accomplish that, *Painterly* provide CPA with a scripted painting plan which encloses some information of its cognitive knowledge-based approach in painterly rendering the reference frame. This painting process-plan is in the form of an XML[8] scripted stroke log files (See Section 5.6.3 for an example of this XML files). This XML file is a hierarchical tree representation of passes and their corresponding strokes, by the order they are rendered onto the digital canvas, plus the information about semantic regions of the frame. CPA uses this scripted plan and its encapsulated information as a resource and guideline to generate new coherent frames, between each two successive keyframes.This way, CPA takes advantage of *Painterly*'s well-developed, cognitively inspired algorithms for semantic parsing and hierarchical, blob-based stroke filling algorithm, and yet controls and induces coherency in the final results. Figure 1.3 better demonstrates the general flow of CPA and its interaction with *Painterly*.

[8] A markup language for encoding documents in a format that is both human-readable and machine-readable.

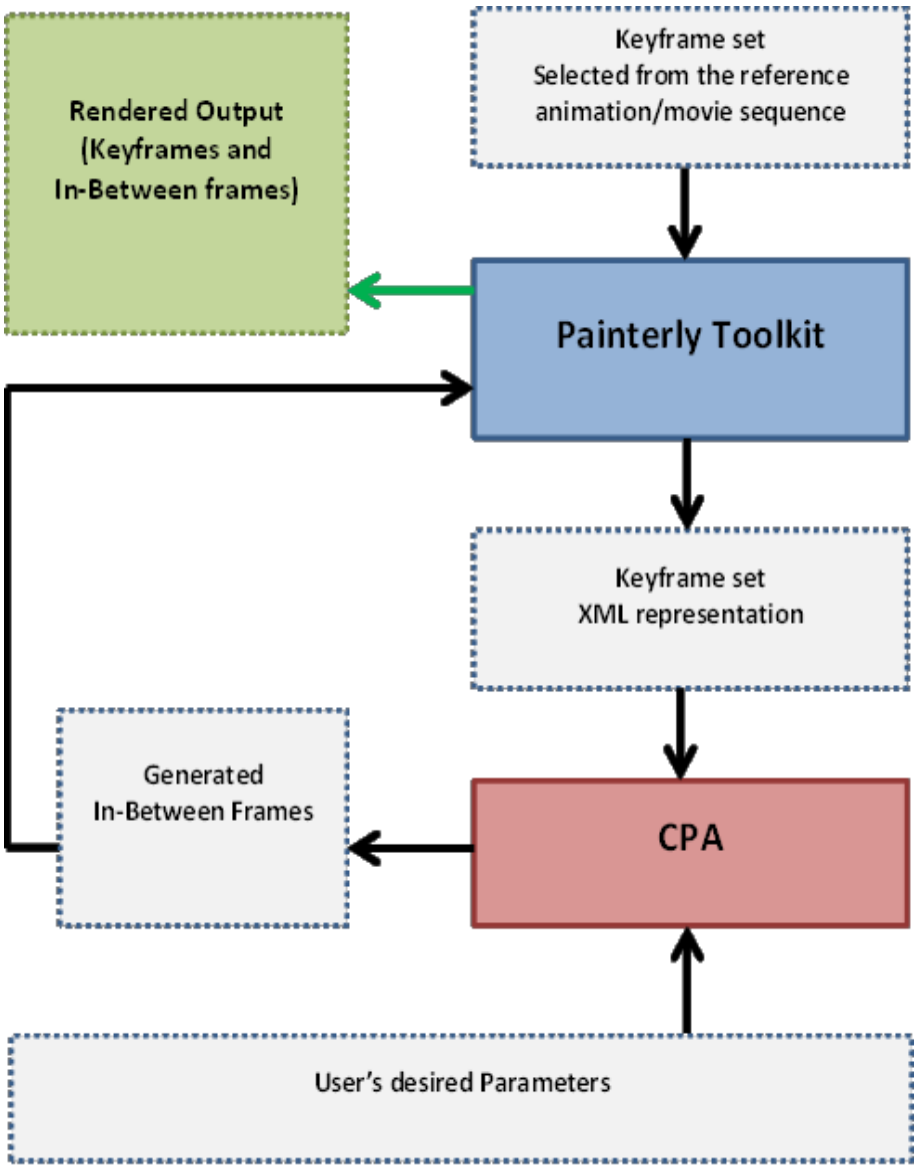


Figure 1.3. The main ongoing scenario of CPA and its interaction with Painterly toolkit.

The process of generating in-between frames consists of 1) mapping the strokes of the first/source keyframe to the strokes from the second/target keyframe based on their position and color; and 2) transforming and propagating every reference mapped stroke from its source configuration to the target configuration. Stroke's configuration is best described as the values of its style-properties such as size, brush type, color, order, etc. This process is done with an eye on corresponding cognitive regions in both source and target frames and their semantic connection. By transforming mapped strokes, CPA

generates a much smoother and more coherent interpolation between two reference frames. Produced in-between frames are then fed back into Painterly to be rendered onto the digital canvas. All keyframes and their corresponding synthetized in-between frames are eventually stitched together with a simple low-level movie making algorithm (See Section 5.7) to create a QuickTime movie.

1.3.1. Contributions

This thesis work proposed a solution for generating coherent and cognitive-based animated NPR. By designing a two-way communication between the open source toolkit *Painterly* and our system – CPA – we could maintain and also extend the key capabilities of *Painterly*. Accordingly, we generated movie sequences based on a cognitively-informed model of human artistic practice. We incorporated high-level semantic knowledge such as semantic/cognitive regions, in CPA's frame synthesis process; allowing for further elaboration of computational intelligence in the system in the future. We provide the users with the chance and possibility to creatively explore and generate different styles of outputs, through a parametric design.

We presented an algorithm based on keyframing and mapping/interpolation of strokes, to extend *Painterly*, which is a still-based, time-insensitive and stateless NPR system, to a stateful and time-sensitive one, capable of processing animation. We reduced the amount of undesired flickering and increased temporal coherency in the resulted sequences, compared to a 'un-informed' approach for generating animation - which renders the frames independently, without any knowledge of previously rendered frames' configuration. We maintained the identity of brush strokes and generated genuine painted frames which look plausible and similar to Painterly's output images, in comparison to a low-level pixel-based interpolation technique. By employing the strokes which can move, flow and transform smoothly over time, we created an inserting interpretation of an "animated painting".

We expand the research domain of Painterly in the interdisciplinary area of human perception, computational art and science. Accordingly, our work makes a potential contribution to the body of researches in psychology and cognitive science regarding human perception of art and visual stimuli.

We developed a flexible and standalone system architecture which can work with alternative frame-rendering modules other than Painterly.

CPA is open source. For validation we have provided our source code and early comparative results for researchers in NPR computer science field – as well as other communities and fields. This is to facilitate any future reproduction of our system or replication of the model; as well as further studies and evaluations of this thesis work or extensions made to it.

Ultimately, this thesis work is a proof of concept for our proposed solution to alleviate the temporal coherency issue in *Painterly* rendered sequences. It is just the beginning point of a deeper study of the cognitive and perceptive aspects of painterly movies. There is still a long way to go for a totally automatic and informed interpolation system which incorporates human's cognitive knowledge in the frame synthesis process and creates coherent and high quality output with a wide range of styles.

1.4. Thesis Outline

Chapter 2 is a literature review on both painterly NPR and painterly NPR animation. In this Chapter we first review NPR in general, its origins and roots and early stages. We narrow down this large research field to the scope of our work which is stroke-based NPR, give an overview of state-of-the-art issues and proposed solutions. We continue the Chapter by reviewing animation NPR, major streams of this field, different techniques and approaches, most influential practices, and issues and open-ended problems. We also explain how our work is related to some of the ongoing research streams in this field and what issues we are aiming to address.

Since *Painterly* NPR toolkit is the backbone of CPA, in Chapter 3 we describe this toolkit; first giving a short overview of the system and how it works, plus its advantages over similar existing systems. Then we delve into more detail, reviewing its goals, design model, data structures, components, process flow and results. In the end, we present a number of fine painterly rendered still images, generated with *Painterly*, to better show the real power of this toolkit.

Chapter 4 describes our proposed solution; the model and the system we developed based on that model. We start this Chapter by providing our design goals and discussing our higher-level approach, giving a brief problem definition, describing the challenging issues we aim to address and specifying the scope of our work. We then proceed with our proposed solution and give a big picture of CPA system. We break this model down to its main phases and explain how the scenario goes on in each phase and what the key steps and points are in each one.

In Chapter 5 we review the implementation detail of CPA. We describe its data structure, component model, process flow and control structures in more detail, followed by providing some examples of the input/output scripts of the system. We finish this Chapter with explaining the limitations of our implementation, stating the framework and libraries used in developing the system and a quick review of the system's open-parameters.

Having reviewed CPA's parameters, we start Chapter 6 with an overview of CPA's parameter calibration, discussing the impacts of each of the parameters on the outputted results. We proceed by presenting more fine-tuned results to better illustrate the capabilities of CPA followed by a short evaluation of the presented sequences.

In Chapter 7 we discuss the contributions and limitations of current thesis work. We continue with suggesting directions for future and further research and studies and then finish this thesis by briefly summarizing the entire work.

The Appendix includes the URLs to all of the videos included in this thesis.

2. NPR, Painterly NPR, Painterly Animation: State of the Art

Obviously, a painterly NPR animation/movie is just an extension of painterly NPR for still images; and similarly, painterly NPR is just a variation of NPR in general. Therefore, to understand the dimensions of either one of these subfields, we should start with the roots. We start this Chapter with reviewing NPR literature in general, from the early works, and move on to the state-of-the-art in painterly NPR in particular. We go through its roots, main streams and challenges.

2.1. NPR Systems

NPR, standing for Non-Photorealistic Rendering is a computer graphics technique for creating images in a non-realistic way. As obvious from the title, its approach is contrasting to other traditional computer graphics techniques which focus mainly on photorealistic outputs. The inspiration for the styles of non-photorealistic rendering comes from paintings (e.g. different styles of painting, such as pen and ink or impressionism), drawings, cartoons and illustrations. Its applications also range from movies, cartoons and video gaming to scientific visualization, experimental animations and technical or architectural illustrations; plus other rising fields such as computational photography, learning and medicine (e.g. communication systems for autistic children where filtering out some unnecessary details and information is crucial (DiPaola, 2007)). NPR also is referred as painterly rendering, artistic rendering, expressive rendering, art-based rendering and many other similar titles.

The focus of this work is on painterly or stroke-based rendering, which typically uses a 2D source such as a photograph and creates a list of strokes to be rendered onto a new digital canvas (Figure 2.1).

Figure 2.1. Painterly rendered stills, created by 'Painterly toolkit'

Basic elements of painterly rendering can be listed as a) a virtual digital canvas, which has different attributes and properties such as size and occasionally texture for some styles (Hays & Essa, 2004) and b) a list of brush strokes. Each brush stroke by itself carries a set of properties such as length, color, width, size, etc. Brush strokes are placed on the virtual digital canvas in a predetermined order, generating a painting from the input image source. Painterly NPR styles are diverse as painting styles, painted by artists of different times and eras. However, the majority of the painterly NPR works are

based on impressionism (Litwinowicz, 1997; Meier, 1996) and to a lesser extent on other styles like expressionism and pointillism (Klein, Sloan, Finkelstein, & Cohen, 2002).

The following sub-section briefly reviews existing painterly NPR literature on still imagery (Section 2.1.1) followed by painterly NPR videos and animation (Section 2.2).

2.1.1. NPR Related Works

Painterly style NPR is a type of NPR concentrating on making painting style imagery which was first explored by Haeberli (Haeberli, 1990). He described the notion of painting as a set of ordered brush strokes with color, orientation, size and shape, on a canvas. Adopting Haeberli's work as a base, Litwinowicz (Litwinowicz, 1997) proposed a fully automated algorithm for generating paintings from an input image, using short and linear strokes. Many other approaches, inspired by Heaberli's algorithm, used local image processing techniques to gain a better control over the placement and color of the strokes. Later, on this thread, Hertzmann (Hertzmann, Jacobs, Oliver, Curless, & Salesin, 2001; Hertzmann, 1998) developed an algorithm that used several passes for creating the brush strokes. These passes were ordered from much coarser strokes to finer ones, using an image difference grid layered with the same coarse-to-fine order aligned with those passes. His work became a turning point in the field since he was the first one to leverage the actual process of painting – as done buy a human artist- in the painterly rendering procedure. A painter artist starts the painting with broad and large strokes and then refines them progressively. They create all the desired details in the art work, by applying smaller strokes over previously painted areas. Hertzmann also categorized many of the Stroke-Based Rendering (SBR) techniques. Later on for creating different painterly styles, Hertzmann - alongside many others like Hays and Essa (Hays & Essa, 2004) - utilized low-level stroke parameters, like stroke length throughout the whole rendering phase.

'*Painterly* toolkit' (DiPaola, 2007, 2009) was initially inspired by Hertzmann's work. This toolkit uses human artists' tacit knowledge and soft rules of painting in the form of parameters and exerts the values of these parameters in the system. *Painterly* also has been going through many improvements, moving away from using a coarse-to-fine multi-pass system to the current use of tonal masses based on lightness/darkness

and drawing type (DiPaola, 2008). Instead of color sampling with perturbations, it uses luminance (i.e. source tone) to remap the image into any color mapping. Such mappings may include any one of the existing cognitive color temperature mapping models or some other custom-created one. The latter change is different from any typical NPR color choice algorithm and has come from artistic practice investigations (See Section 3.3.1).

To automate the whole NPR process, especially the higher-level of modeling scene semantics, and also to enhance the aesthetics of the results, many computer vision techniques have made attempts. Most of them had a more global oriented vision of the process in mind. Gooch (B. Gooch & Gooch, 2001) proposed segmenting the image into homogeneous grayscale regions. This approach led to a great reduction in the number of brush strokes.

The segmentation was also used by Santella and Decarlo (Santella & DeCarlo, 2002, 2004). They added a rather important extension to the process which was guiding the painting process by eye movements. In this approach, users' eye movements were the key element in specifying more important areas of painting and adding more details to them. This attempt was faithful to the actual cognitive process of painting by a human artist since a human painter emphasizes and/or de-emphasizes the different regions of the scene/ sitter based on their personal goal of how to present the scene/ sitter. They extracted this information from tracking the users' eye movements while they were looking at the source image and then utilized that data to render the reference scene.

A notable and important adaptive automatic system for painting was also presented by Collomosse and Hall (J. P. Collomosse, 2004; J.P. Collomosse & Hall, 2002; John P Collomosse et al., 2005), since they used machine learning techniques in their process. Also JoaoCarvalho (Du Buf & Rodrigues, 2006) used human vision techniques in an endeavor to make a better tie between humans' visual perception theories and painterly NPR procedures.

There are also some common drawbacks and shortcomings in most NPR systems. Gooch, as well as some others such as Collomosse (J.P. Collomosse & Hall, 2002; B. Gooch & Gooch, 2001) have described one of these shortcomings as 'limiting

attention to local image analysis by utilizing more local techniques for examining local and/or small pixel neighbourhoods'. These kinds of blind local analyzing techniques result in losing the rather important global view of the entire image. Several researchers including Collomosse et al, Gooch et al, and Kagaya et al, looked for efficient global solutions (J.P. Collomosse & Hall, 2002; B. Gooch & Gooch, 2001; Mizuki Kagaya et al., 2011).

Considering the localization issue, *Painterly* toolkit, which will be introduced in a later Chapter, uses a hierarchical parametric system by implementing a perception blob tree structure. Using a general semantic map for the portraits (background, skin, hair, clothes, etc.), the "blob thinker" – a subcomponent of Thinker (See Section 3.3.1), generates their corresponding blobs as "parent blobs". As the hierarchically iterative process continues, blobs become progressively smaller and turn into child-blobs. These child-blobs can communicate up and down in the tree, providing global and local comparative information towards the decision making process for position and color of the next brush strokes. This process both benefits from and simulates humans' cognitive painting process. This simulated process perceives a small cognitive region to work on rather than a single brush stroke on its own (e.g. working on eyes of the portrait, rather than working on a brush stroke on the canvas). Fully benefitting from this local and global communication system and exploiting it in the system depends on a deeper understanding of humans' creative and cognitive painting process. This process is still the source of many ongoing researches in this field.

Apart from all the works which are breaking the boundaries of painterly NPR, and taking it one step further, there are also some other works in the area which have deeply evaluated NPR's current state, techniques, and possible future directions in a systematic way. Gooch et. al. (A. A. Gooch, Long, Ji, Estey, & Gooch, 2010) gave a state of the art of the field. Salesin (Salesin, 2002) counted seven major challenges in the NPR field and discussed them. Collomosse et al, Hedge, Isenberg and Vanderhaeghe (J. Collomosse, Kyprianidis, Wang, & Isenberg, 2012; Hegde, Gatzidis, & Tian, 2013; Tobias Isenberg, 2013; Vanderhaeghe & Collomosse, 2013) discussed possible future directions that can be taken. There is also an ongoing discussion on evaluation methods used in NPR field about whether we should use science-based evaluation methods

(Tobias Isenberg, 2013) to assess the results or accept it as a form of art which should be appreciated on its own rather than measured (Hall & Lehmann, 2013).

DiPaola (DiPaola, 2007, 2009) proposed a cognitive-based parameterized painterly NPR system for creating portrait paintings from input imagery – photographs in particular. This toolkit, shortly referred to as *Painterly* throughout this thesis work (See Chapter 3), was developed based on extracting a qualitative tacit knowledge space from art books and human artists – particularly painters. This knowledge space was then combined with human cognitive and vision knowledge and used to build a parameterized painterly NPR toolkit. The set of painting parameters includes information about brush strokes (i.e. brush stroke properties), number of passes and the rendering techniques. These parameters are provided to the system as an XML script file. *Painterly* toolkit is built up on Hertzmann's multi-pass technique but also benefits from the cognitive studies on the humans' vision system. Therefore, it is capable of producing more expressive portrait pictures, compared to a general painterly system. *Painterly* toolkit is used as the backbone of our system - CPA - providing us with our input set, plus rendering our input/output frames (See Chapter 4). More information on stroke-based rendering, state-of-the-art, possible future directions and evaluation methods can be found in Hertzmann's survey as well as some other evaluating works (A. A. Gooch et al., 2010; Hegde et al., 2013; T. Isenberg, 2013; Kyprianidis, Collomosse, Wang, & Isenberg, 2013).

2.2. NPR and Animation

NPR originally started focusing on still images to create different non-photorealistic looks and styles. One of these styles is painterly NPR, inspired by various painting styles, the most widely explored of which is impressionismm. With painterly rendering animation emerging, soon computerized painterly rendering became more than a tool for still imagery. The exhausting, expensive, time consuming and extremely difficult traditional paint-on-glass technique or frame-by-frame method of creating painterly animation made the extreme potential of computerized painterly rendering systems seem even more welcomed. Not to forget that in the traditional procedure, the degree of temporal coherency was much less than what was normally expected from a

computer-generated animation (O'Donovan & Hertzmann, 2012). New animation tools promised – and in many cases delivered - not only more temporal coherency but also greater speed and wider variety of animation styles.

According to Hertzmann (O'Donovan & Hertzmann, 2012) a painterly animation system should combine at least two main goals of 1) capturing all the fine details and 2) following the moving objects and optical flow[9] throughout the whole movie sequence. What that adds up to these two main goals is also producing an artistic style and avoiding unwanted flickering. All of these, also, should be done in a timely efficient manner. Giving the animator the benefit of having a total control over the production process counts as a big advantage most of the times. This way the animator artists can also control the style, quality and visual aesthetics of the final results. However this control might be a downfall of any system, when automation is needed more than interactivity. These goals are sometimes conflicting with each other, while each of them are challenging on their own. The result of them adding up together is a rather elaborate and arduous problem to deal with.

In the following sub-section of this Chapter we give a compact overview of the existing literature in the field of NPR video/animation.

2.2.1. NPR Animation Related Works

There are two general approaches for creating NPR videos in the stream of Stroke-Based Rendering, according to Hertzmann and Perlin (Hertzmann & Perlin, 2000). The first approach depicts video as a painted world. Brush strokes, in this approach, are attached to geometric objects (Daniels, 1999; Meier, 1996). Managing the density of strokes, as the objects gets closer to or farther from the camera, is the main challenge of the algorithms in this approach. The second approach creates painted representations of the world, therefore brush strokes are detached from geometric objects, spatially and temporally (Hertzmann & Perlin, 2000; Litwinowicz, 1997). The common challenge in this approach is *Scintillation*, also known as *Swimming* or unwanted flickering. Videos made based on this approach can flicker due to lack of

[9] Pattern of objects' motion in a scene.

temporal/spatial coherency in style-properties or orientation of the strokes. For instance, the directions or sizes of brush strokes change drastically in subsequent frames, without any knowledge passed from one frame to the next.

In the first group of aforementioned approaches, a set of particles are attached to the scene objects. Each particle represents a brush stroke. Meier (Meier, 1996) - from Walt Disney Feature Animation - used vectors of geometric objects to determine brush stroke orientation. Style-parameters such as colour and size were assigned to a number of strokes which were propagated to other strokes and used afterwards in the rendering process. Otherwise, a reference image was used to obtain values of those parameters. The common issue with these approaches is handling particles' proper density throughout the movie/animation. The reason is moving objects cause the particles to move along and either get too close to each other or too far apart.

Cornish et al (Cornish, Rowan, & Luebke, 2001) tried to address this issue by using a hierarchical view-dependant approach, based on algorithms for view-dependent polygonal simplification. They used a densely sampled polygonal model for each object. Each vertex of the model represented a particle. So during the movement of the object, the number of vertices of the polygon was adjusted (increased or decreased accordingly) based on the distance and other view-properties. This approach also allowed for having various amount of detail in different areas of the scene.

The most common issue in the second aforementioned approaches is flickering. This undesired visual artifact occurs, when successive frames of a movie/animation sequence are painted separately, without any knowledge bridging between previously rendered frames and current or future ones. Therefore, the number, size, length and orientation of brush strokes in the corresponding reference areas change within those frames.

To mitigate this problem and avoid visually cluttered frames, many approaches and techniques have been proposed and used. Any SBR painterly animation method basically consists of two main steps which are 1) stroke-based rendering of the input image and 2) propagating the strokes over video frames. Apart from the rendering

phase, the main difference of various approaches is in their take on propagation phase and how they have handled it to maintain coherency.

Daniels (Daniels, 1999) used optical flow for controlling the orientation of strokes; His method thereafter became a common approach in the field. In addition to optical flow, motion maps[10] have been used quite a lot in an attempt to regulate stroke orientations and make the object movement smoother throughout successive frames. Motion information is extracted from the content of movie/animation. This information is used to determine the position of strokes in the next frame. Also a fairly good number of the works done in the field have used vector fields to control the orientation of strokes. These fields were either defined manually by the user or automatically, using normal vectors and curvatures of the geometric objects (Hays & Essa, 2004; Hertzmann & Perlin, 2000; Litwinowicz, 1997; Park & Yoon, 2008; Snavely, Zitnick, Kang, & Cohen, 2006).

Litwinowicz (Litwinowicz, 1997) used linear brush strokes and textures to create painterly animation. The orientation of brush strokes were determined using gradient interpolated by a thin-plate spline. By adding random values to parameters of each brush stroke a variety of brush strokes were created. These strokes were clipped to the edges of the scene objects and followed the motion of corresponding objects between successive frames. Object motion was determined using optical flow. Because of a high possibility of changes in the neighbouring gradients, their approach was prone to create significant flickering.

According to Hays (Hays & Essa, 2004), using optical flow might push strokes far apart and consequently, create a gap between them. To care for that, they used motion vectors to change the position of brush strokes from frame to frame. This resulted in regular spacing between strokes. However relocated brush strokes were not consistent throughout consecutive frames which consequently produced flickering.

Hayes and Essa (Hays & Essa, 2004) added temporal constraint as another property (e.g. size and opacity) to brush strokes and created a dynamic set of brush

[10] A motion map represents the position and accordingly the velocity and acceleration of an object in the frame during a certain time span.

stroke properties. This set, then, changed gradually based on reference image and motion properties. For orienting brush strokes over time and space, Radial Basis Functions (RBFS) were used. Additionally, they used edge detection techniques for decoupling output resolution from the input dimension. Besides, they utilized brush stroke textures and a simple lighting model to enhance not only the temporal coherency, but also the aesthetic of output video.

Park and Yoon (Park & Yoon, 2008) replicated the manual technique of paint on glass, using motion maps to create a hand painted style video. They used two types of motion maps, 'strong' and 'weak', to represent the regions that had changed between successive frames. In order to create these maps, they used block matching method to estimate motion, through displacing the object edges by motion vectors. The maps where then, used to generate the necessary brush strokes for converting a frame to the next one. They also used some systematic methods like MSE (Mean Square Error) and PSNR (Peak Signal-to-Noise-Ratio) to evaluate their results. According to Park and Yoon, generally, evaluation methods for estimating the degree of maintenance of temporal coherency between two frames can be divided into two following groups:

- Segmenting each frame and creating a spatio-temporal volume by connecting each segment to a corresponding spatio-temporal space. Fitting continuous surfaces to voxel objects in the frames can give a temporal smoothing estimation, which can be measured using MSE and PSNR.
- Calculating the magnitude of frame flicker. Flickering occurs due to irregular visual changes in painted local areas and it highly depends on shape and color of the brush stroke, therefore using brush strokes' color can be a reasonable way of estimating the magnitude of flickering. Due to the overlapping of strokes in the painted frames, using the stroke-shapes is very dubious.

Nonetheless, majority of the works in this area do not perform ay systematic evaluation, considering the outputs as forms of art which should be appreciated on their own (Hall & Lehmann, 2013). A common evaluation method, used by many notable researchers such as Hertzmann, Lin et.al. and Kagaya et. al, is done based on the traditional software engineering approach and by providing the source code and the results to the public communities for further reproduction and evaluations (Hertzmann & Perlin, 2000; Mizuki Kagaya et al., 2011; L. Lin et al., 2010; O'Donovan & Hertzmann, 2012).

Klein et al. (Klein et al., 2002) defined a set of functions, called rendering solids, which span over time. They could successfully create more painterly styles such as cubism and abstract styles – unlike common approaches that mainly aimed for an impressionistic look.

For achieving temporal coherency in the painted output sequences, Snavely et. al. (Snavely et al., 2006) used depth information from a special camera together with data, from 2D sequences (which they call 2.5D video). To decrease the noise in the captured data, they filtered the video to the depth information of the input frames. They proposed a technique for calculating normal and surface direction at each pixel in order to use it for determining the orientations of brush strokes. Their output videos were showing two different NPR styles of painterly rendering and cross-hatching.

Most of the painterly rendering techniques for animation use gradient to decide on the orientation of the brush strokes. This method is good to express the shape of the object but not as much informative and expressive in showing the flow and movement of the objects. According to Lee et. al (Lee, Lee, & Yoon, 2009), aligning strokes with the corresponding flow and movement of the objects, amplifies the viewers' experience and perception of the dynamic of the objects' motion. They proposed an algorithm to express objects based on their motion information (e.g. magnitude, direction, standard deviation). They segmented consecutive frames of the movie sequence to dynamic and static segments. They, then, determined the orientation of strokes in each region by extracting motion information from moving objects in the scene.

Interactivity has been another rather important motif that divides approaches based on their take on this concept. The approaches that have focused on fully automating brush stroke synthesis for the entire video sequence have showed promising results. Users of these systems are not required to have any major skill or knowledge which can be considered an advantage for any technical system. On the contrary, there are some issues which make these systems unsuitable for all types of applications. Some of these issues can be listed as: low amount of control over the artistic side, limited style variety of the final result, poor edge definition and undesirable stroke movements. The other approach, however, rotoscoping, allows the user to keyframe the strokes using spline interpretation in a spacetime domain. This feature provides a full

control over the artistic side of the result. Nonetheless, keyframing, when acting as a major requirement for these systems, is a laborious procedure and requires a lot of time and effort. All in all, incorporating more interactivity to fully automated systems and enhancing the temporal coherency has been the ultimate state in the field of video painting, which many of the works have set themselves up to.

Hertzmann and Perlin (Hertzmann & Perlin, 2000) used their still image algorithm for making painterly videos. They painted the new frame over the previous frame, only when there was a significant change between the two frames. They warped the previous brush strokes towards the new output to keep them attached to the objects. Besides, they used optical flow in determining the orientations of strokes. They successfully created results with paint-on-glass style. According to Hertzmann, in a 30 Hz painterly video, even a slight amount of flickering is noticeable. Moreover, 30Hz frame rate, rather than a moving painting, looks like a usual video with bad artefacts. Hence he suggested using 10-15 frames per second.

Later on Hertzmann (Hertzmann, 2002) added a new feature to his system by adding texture properties to the brush strokes. This feature required the addition of a height dimension to the set of previous style-properties of brush strokes. The height was then used in calculating the light, leading to highlight and shades across the painting. This was rather a good step in achieving more realistic results like oil paintings.

Agarwala (Agarwala et al., 2004) created a notable system which combined the features of rotoscoping (i.e. tracking contours in a video) with automatic video processing and interactive editing. In their proposed system, users specified curves in a number of keyframes. Using that information, the system interactively tracked contours of objects, through computer vision techniques. The curves could interactively be modified and optimized by the users throughout the process. The basis of their work has been vastly used after that, most efficiently by Hertzmann (O'Donovan & Hertzmann, 2012). Agarwala's work had another breakthrough, in sense of showing painted strokes and tracking curves can be paired together.

Hertzmann (O'Donovan & Hertzmann, 2012) proposed an interactive system in which the user can control stroke synthesis – by specifying or modifying stroke

placement, orientation, movement and color, and manipulating them directly. They also benefited from an automatic stroke-generating system, which traced the strokes in the given video sequence using an objective function instead of a vector field, to determine the best placement. Their method increased temporal coherency as well as producing cleaner region boundaries and finer details. Their system however, was aimed for more professional users with a minimum knowledge of the procedure; since they have made more emphasis on users' control over the painterly processing of frames. Due to the greedy nature of their objective function, previously drawn strokes were limitedly considered. This means that the strokes in the static regions slowly appeared and disappeared. *Painterly* was inspired by Hertzmann's NPR generating algorithm, therefore, CPA is benefitting from this algorithm indirectly, through using *Painterly* as its base system. But despite benefitting a lot from Hertzmann's algorithm, this thesis work moves away from their approach, regarding interactivity. We are aiming for a more automated system with the minimal human supervision or manipulation so that the procedure can be injected into an industrial movie production pipeline. As a result, CPA does not handle users' realtime manipulation on the brush stroke synthesis process. Also, as mentioned before, the focus of this thesis work is on computer animation/movie sequences with automatically generated regioning inputs rather than movie sequences. Nonetheless, CPA is able to work with both 2D and 3D images and frames, if the proper region map files and ultimately the required XML scripts are provided to it.

Lin et. al. (L. Lin et al., 2010) have proposed a two phase system for creating painterly animation, using video segmentation. The video parsing phase extracts and labels semantic objects from the target video assigned with different properties based on their material (hair, skin, clothes, etc.). For that, the user labels multiple regions by drawing scribbles on them. The annotated regions, then, are categorized into twelve semantic material classes. The segmentation is propagated over other frames, using a video cut-out algorithm. Hence, each object is segmented as a space-time volume. A number of discriminative features are extracted from each volume. These features are used to calculate dense correspondences over frame, for propagating brush strokes. The painterly rendering phase stylizes the video based on the correspondent features and assigned semantics of the objects. They also use example-based brush strokes, which are created by the artists for each different material/object category. An automatic

process chooses the style of the brush based on the target object's class. As a result, different styles were automatically applied to different regions.

Their system performs a two-pass rendering process, which is different from *Painterly*. This is because *Painterly* is a multi-pass system and the number of passes can vary to the users' liking. As a result and to be compatible with *Painterly*, CPA is a multi-pass-based system as well, to have the flexibility of working with any number of passes provided in its input frames. Placement of the strokes in both of the passes in Lin et. al.'s system is determined by the orientation-property which is calculated from the region contents. After rendering each reference keyframe, the brush strokes are propagated over the rest of the frames, before the next keyframe. Keeping track of moving objects throughout consecutive frames and synthesizing strokes based on the global object deformation, makes the corresponding strokes remain more coherent with movement of the objects. It also creates cleaner edges. The system, however, has problems in rendering sudden and drastic movements as it is based on a presumption that all movements are continuous and smooth. They have achieved good, solid and less-flickering results, due to their vast use of a large database of paintings which leads to their system learning different forms of strokes used in different regions. But all this has been achieved at the price of creating just one style of painting.

Kagaya et al (Mizuki Kagaya et al., 2011) proposed an object-based painterly rendering system, which they called it a 'multi-style' rendering system. This system extracted spatial and layout information of objects in the image/frame and used it to determine style-parameters and orientation of strokes for individual regions in each keyframe. Their goal was to more realistically and truthfully mimic the way a human artist paints. They used different types of stylization to control the focus of viewers on different objects in the scene they were painting. In this sense, they are close to the main goal of *Painterly* itself. Similar to Lin at al, Kagaya et al used a semiautomatic video segmentation, dividing the scene into temporally coherent moving or deforming regions (objects or background). Style-properties were assigned to each region of marked keyframes by the users and then propagated to the target object through all the other frames. Stroke-orientation parameter also could be assigned to each object, in every keyframe and propagated likewise. However, the general and underlying

movement direction of the corresponding object was considered in transporting the strokes. Orientation could also be incorporated with color gradient, applied in the region.

There is a great deal of similarity between our proposed solution in this thesis work and Kagaya's system, in the sense of keyframing and propagating style-parameters from keyframes to all the other frames generated in between those keyframes. The difference is that, our system - CPA - do not rely on users to assign the style-parameters directly, as *Painterly* determines them after rendering the input keyframes. Nonetheless, users get to determine their output style, by tweaking *Painterly*'s and CPA's parameters directly. Another difference between Kagaya's' work and this thesis work is that we do not account for the underlying movement of the cognitive regions; therefore the orientation of strokes in the generated in-between frames is determined by morphing source strokes' orientation to the target. Another feature which Kagaya shares with Lin et. al. was 'batch computation' of the strokes. It means they didn't provide any user interface for the artist to tune and refine the results in no other way than re-running the program after changing the parameters. So, intermediate strokes could not be modified and changed after creation. This feature is another similarity between our approach in the current thesis work and both Kagaya's and Lin's works. Kagaya et. al. achieved much smoother results due to using a blending feature, but still showed a noticeable amount of flickering, especially in the boundary areas of objects in the images.

This thesis work, as mentioned before, has a lot in common with Lin et al and Kagaya et al, in the sense of using video segmentation information and propagating style-parameters from keyframes to the produced in-between frames. Moreover, like Hertzmann, our system - CPA - uses an objective function in determining the placement of synthesized strokes in generated in-between frames. CPA uses the information about cognitive regions, extracted from the frames' contents, by *Painterly* rendering the reference keyframes. CPA creates a strong and coherent connection between these region blobs throughout the keyframes and uses this connection as an anchor point in propagating the synthesized strokes more cohesively throughout the generated in-between frames. As such, the style-parameters are specified for each deforming or moving region per keyframe. These parameters are then propagated to the next frames coming in-between each pair of consecutive keyframes. Correspondingly, CPA does

batch computation as well. However, this thesis is focusing on computer animations/movies (i.e. CGI) with fully automated regioning and labeling process, using a third authoring tool (e.g. Autodesk Maya, Adobe Photoshop). Whereas Lin et al depends on the user for this step of the process. CPA, also, differs with their system in another sense which is the number of rendering passes. They do have a predefined number of passes (two in total) while CPA is a multi-pass-based system. The reason is that, as a subsystem to *Painterly, CPA needs to be compatible with it. Painterly*, as mentioned before, is a multi-pass toolkit with which the users can render as many passes as serves them best.

Similar to Kagaya et. al, CPA does not have a global stroke orientation throughout the entire image/frame. Stroke orientation and curvature is determined completely by *Painterly* and based on many other factors like size of the semantic blob, stroke -density of the blob and previously synthesised strokes in the same region. As an external subsystem which does not interfere with *Painterly*'s internal algorithms, CPA does not deal with harmonizing or globalizing orientations in either a region or the entire image. In fact, CPA transforms the strokes' orientation and shape from their source state to the target. Also, similar to Kagaya et. al. CPA is a multi-style painting system. *Painterly* provides CPA with a wide range of painterly style stills to begin with. Moreover, by incorporating interpolation-style parameter in the frame synthesis process, CPA is able to create a wide range of in-between frames and interpolation styles in the resulted sequence. Another difference between CPA and their system is in their take on interactivity. Kagaya et al. opted for an interactive system whereas CPA is aiming for a more automated procedure, with the least amount of user modification and supervision.

Some other approaches in the paradigm of painterly animation have used different techniques and proposed avant-garde algorithms, some of which have achieved impressive results. Haro and Essa (Haro & Essa, 2002) based their work on Hertzmann's (Hertzmann et al., 2001), which used machine learning algorithms to learn NPR image transformations from pairs of source images and their painted versions. They, then, extended the approach to painterly movies. Their approach was successful in maintaining coherency through the movie sequence, but couldn't capture a wide range of painterly styles.

According to T.Lin et. al. using optical flow to propagate brush strokes over the frames may cause some critical downfalls and severely damage the visual aesthetic of the output, if calculated incorrectly (T. Lin, Lin, & Wang, 2012). To get a way around this issue, they proposed a system which combined motion segmentation and occlusion handling to calculate more accurate dense-feature correspondences. They automatically divided the frame to non-overlapping motion layers, to eliminate ambiguous motions happening near the juncture of motion areas. Dense correspondences were then extracted inside each motion layer with an occlusion handling procedure. This made brush propagation more robust against complex motions or occlusions.

There have been many other successful innovative approaches in the video painting field using machine learning, computer vision, vector fields, rotoscoping and many other methods, some of which have achieved fairly good results. Reviewing all this literature is beyond the scope of this thesis work.

3. The *Painterly* Toolkit

The *Painterly* toolkit works as the backbone of CPA. As a result, our work in this thesis is tightly interrelated to this toolkit's domain, internal processes, inputs and outputs. Its strengths will count as our strengths and its limitations will affect the scope and results of our work. Therefore, it is important to have a general idea about this toolkit and how it works. In this Chapter, we provide a brief overview of *Painterly*, going through its goals, design model, data structure, main components and internal data flow. We then proceed with its internal sub-components and modules, inputs, outputs and finally its limitations (DiPaola, 2013). It should be noted, however, that *Painterly* is not a contribution of this thesis. Our contribution – which is extending and enhancing *Painterly* in the area of NPR animation with our whole new CPA system - is provided and discussed in detail in Chapter 4 and above.

3.1. Introduction to *Painterly*

Painterly NPR Toolkit – briefly referred to as *Painterly* in the context of this thesis work - is a cognitive-based parameterized painterly NPR system for creating portrait paintings from still imagery inputs. The toolkit is conceptualized, designed and implemented as part of iVizLab main research streams (DiPaola, 2007, 2009; "iVizLab - Simon Fraser University," 2013) with two main goals in mind: First, to provide a tool for delving into possible combinations of painting parameters; thereby creating a wider range of painterly styles; Second, to provide a tool for exploring the interdisciplinary space of art and human perception. A large number of existing NPR software/toolkits are utterly built up on computational image processing techniques and algorithms such as image segmentation, edge detection, etc.. In this regard, *Painterly* is different from most other peer NPR toolkits/softwares. It can distinguish between different semantic areas of a painting - portraiture or scenery - such as background, skin, eyes, etc. Therefore, it

can treat each cognitive area differently; based on the desired parameter setting defined by the user.

Painterly accepts an input image, a number of map files and a set of painting parameters (e.g. brush size, stroke length, colour palette, etc.) specifying the style of output result, in an XML script format. It then renders or 'paints' the input image in the specified style using the 3D OpenGL standard. The input XML file contains the initial conditions, painting parameters, number of passes and the algorithms to use. Cognitive regions of input images are specified and labeled by the user or through other authoring tools (e.g. Autodesk Maya or Adobe Photoshop) in the form of a region-map file. Each region can be painted with different colour palettes, brush sizes, types or styles. *Painterly* also accepts matte files, used to denote the areas of input image which need to be treated differently (for instance painted with more or less detail). All of the style parameters and the attributes are pixel resolution independent. This allows for global and local re-scaling through the corresponding parameters.

Aside from the rendered image, *Painterly also* outputs an XML translated form of the rendered image known as '*XML stroke log*' file (See Section 5.5 for a sample of this XML script). This XML file, which can act like a rendering plan, contains a hierarchical tree representation of passes, semantic blobs, strokes and strokes' style-parameters. These style-parameters control the behaviours of strokes and eventually, the aesthetic look of the final painted result. Furthermore, by feeding such XML file back to *Painterly,* it can re-render it and output the image representation of it – same as the first original rendered one. This feature makes *Painterly* a very robust system.

Painterly's internal NPR algorithms are high-level cognitive painterly techniques to enhance emulate a human artist's painting process. Similar to Haeberli and Hertzmann (Haeberli, 1990; Hertzmann, 1998), *Painterly* is a multi-pass, coarse-to-fine system. It starts with coarser strokes in first passes. Brush strokes get finer progressively in the latter passes to achieve a more detailed result. The coarse-to-fine method best captures the way a human artist paints; starting with large tonal masses and progressively adding more details over the previously painted layers. *Painterly* provides the user with different color palette options such as using reference image's original color palette or a specific one provided by the user (Figure 2.1).

This thesis work has its roots in *Painterly*'s latter goal, as well as spreading its leaves in the first one. As mentioned before, *Painterly* is originally aimed at creating still painterly rendered images – with an emphasis on portraiture - and this thesis is an endeavor for exploiting its capacities for creating temporally coherent painterly animation sequences. *Painterly* toolkit is used as the base system of CPA, in the sense of providing CPA's input in the form of some XML stroke log files, as well as painting the input keyframes and CPA's generated outputs. These XML files are scripted representations of the input keyframes, rendered by *Painterly*. Therefore, they encapsulate the cognitive-based aspects of the rendered keyframes.

In the following Sections we describe *Painterly* more in details, its design model, components, data structures, process flow and results.

3.2. System Overview

Collecting qualitative and tacit painterly knowledge from art books and interviewing human artists (painters in particular), converting it to a quantitative parameterized model and then implementing it as a toolkit, is *Painterly*'s main building structure (Figure 3.1) (DiPaola, 2007).

The first step in the design model is collecting soft and cognitive rules of painting (obtained from interviews and reference data) and quantifying them in the form of a parameterized computer model. This step is the key to make not only the base low-level NPR components, but also more complicated and higher-level ones which are built up from them correspondingly. This methodology provides the user with control over the painting process as well as guaranteeing the customization that is promised by the toolkit. One big promise of the toolkit – like of many other similar software/toolkits - is creating outputs with a wide variety range of styles. However – unlike many other systems - the main concern of this toolkit is to build a scientific and artistic inquiery system from scratch. To attain this goal, it uses a knowledge based domain, together with hierarchical and mutidimentional parameter spaces (DiPaola, 2009). These spaces have been used rather successfully in existing reserches in computer science filed, such as facial animation (DiPaola & Gabora, 2009; DiPaola et al., 2013). Using low-level

parameter-based and object-oriented language to build up a more complicated and higher-level components, is the main structure of this approach.

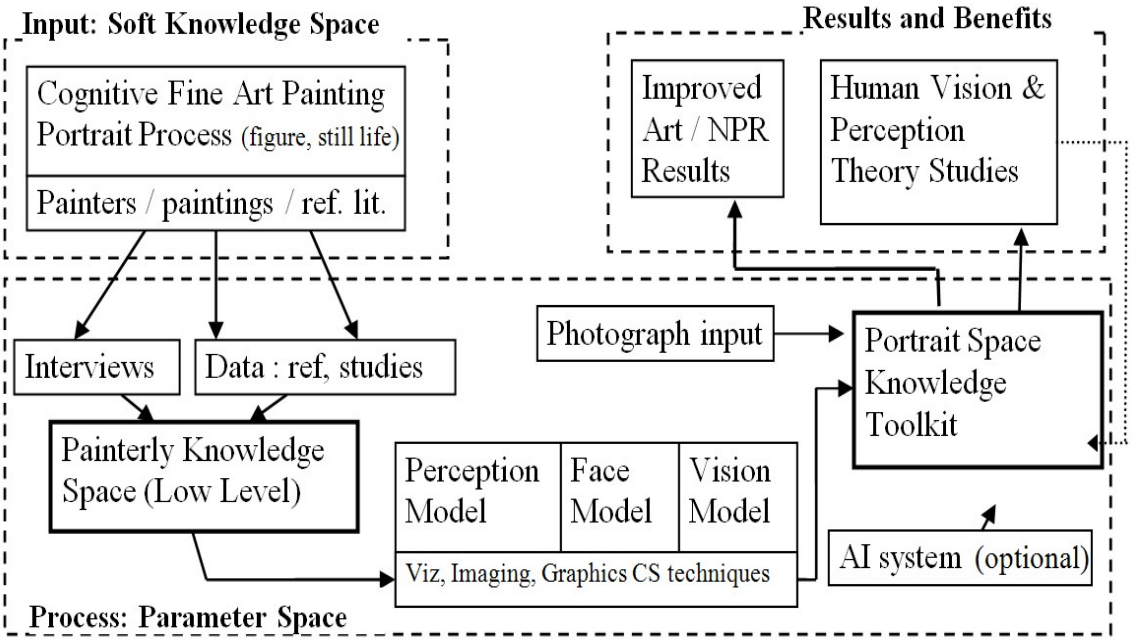


Figure 3.1. Painterly's Process Chart (Image taken from (DiPaola, 2007))

Rigorousness and universality are two important notes that had to be taken into account in selecting *Painterly*'s low-level language. To achieve that, XML scripting language is used as the low-level dimension in the knowledge space. These dimentions can be accessed via higher-level constructs such as 2D maps or simple low-level equations. Also higher-level constructs are completely build up on lower-level parameters with temporal, spatial and logical attributes and properties. For instance, in a painting, a high-level construct like a 'brush stroke' is buid up on low-level parameters such as opacity, scale, color and length. Low-level parameters of *Painterly* fall into three main functional groups as followes:

- Constant parameters: the basic and low-level parametes like brush scale, color or opacity. These parameters can be floats or can be remapped into other knowledge buffers like depth maps.
- Method parameters: basically a method that is used as a parameter for a higher-level method and provides a refinement over the lower-level parameters, preparing them for the next level class/function which is going to use them.

- Process method parameters: these parameters more or less control and guide the flow of processes and their parameters.

3.3. System Inner-View

Painterly's framework, referred to as '*Thinker-Painter*' in the context of this work, consists of two main components of Thinker and Painter which are explained in details later in this Section. Instead of considering the individual perception blobs, or regions or brush strokes, Thinker-Painter 'Thinks' in terms of '*Paint Actions*'. PaintAction refers to the hypothetical 'painting action' of a human painter in every state, such as specifying an area of canvas to paint on, picking a colour, deciding on the region to paint and determining the desired attributes (e.g. density of brush strokes, size and length of them and their physical place on that working area of canvas).

The Thinker-Painter component analyzes the current state of painting together with the source image. This component considers the collection of 'high-level intentions' of the artist and creates a PaintAction. The high-level intentions are in fact, a set of parameters defined by the user which are fed to the system. The PaintAction is then passed to another component – Painter - to be 'painted' on the digital canvas, starting by perception blobs and getting down to strokes inside the blobs. Besides Thinker and Painter, there also is another component coming between these two main components, known as '*Concerns*'. This component might tweak the PaintAction, whenever necessary, based on *Painterly*'s considerations and rules (e.g. making some regions more detailed based on their surrounding regions). This way the PaintAction actually is separated from the painterly techniques. By separating the cognitive process of painting from the final result, a wider variety of output styles will be supported. Besides, it takes the system one step closer to its main goal of exploring humans' cognitive process of painting. Not to mention that using an object-orientated design model in implementing the system makes it easier for any further modifications, in terms of adding more features, changing or reusing any existing one.

The Thinker component is like the brain of *Painterly*'s framework. It identifies and analyzes various areas of painting, as well as giving instructions on how to paint on them. On the contrary, Painter is like the hand. It renders/paints the PaintAction onto

the digital canvas. The PaintAction is in fact, the communication package passed between these two components; A package that contains high-level parameters and information (Figure 3.2) (DiPaola, 2009). The Thinker component, in fact, analyzes and re-examines the painting area after synthesizing and laying down each stroke, as opposed to doing it after the entire pass. This sway, it is guaranteed that the number of redundant strokes are reduced, accuracy of modeling the painting process will increase and the entire painting process becomes more efficient, as well as the fact that this method enables system to use different '*End Conditions*'.

3.3.1. Main Components: Thinker, Painter, Palettes and Concerns

In this sub-section we discuss *Painterly*'s main components and their role in the whole scenario of *Painterly* rendering in more detail.

Blob Thinker

Blob thinker takes on the job of making a hierarchical knowledge structure from the source image. A human painter sees the scenery or sitter in regions of light and dark and bases the painting process on that. Blob thinker, in its attempt to mimic this process, is set to capture this lightness/darkness regioning. The resulted hierarchical tree structure divides the image into regions of "light" and "dark", which get finer progressively.

This method incorporates a rather cognitive and semantic level, higher than just strokes/pixels, in the process of painting. Thus, it gives an upper hand to the system to actually see the big picture. This, consequently, enables the system to a) "think" about different areas of the source image and reference canvas, b) to "decide" on "which" area is going to be treated next and c) "how" the act is going to be executed. This rather semantic area is called a semantic blob or sub blob. Our system – CPA – uses these semantic regions in its frame synthesis process to keep the propagation of strokes coherent throughout the generated in-between frames (See Section 4.3.2).

Thinker - Analyses the current canvas together with source image. It then creates PaintAction objects representing high-level plan for painting on a certain region of the canvas in a specific way. Any number of Concern objects suggesting certain possible modifications to the PaintActions can be exerted in the process.

Painter - Receives a PaintAction object as the input and decides how to render it onto the digital canvas.

Concern - A lightweight, modular object that can be injected to Thinker to suggest modifications to the PaintActions it produces (e.g. choice of Palettes is a type of concern).

PaintAction - An interface consist of high-level parameters specifying the region which should be painted and how it should be painted.

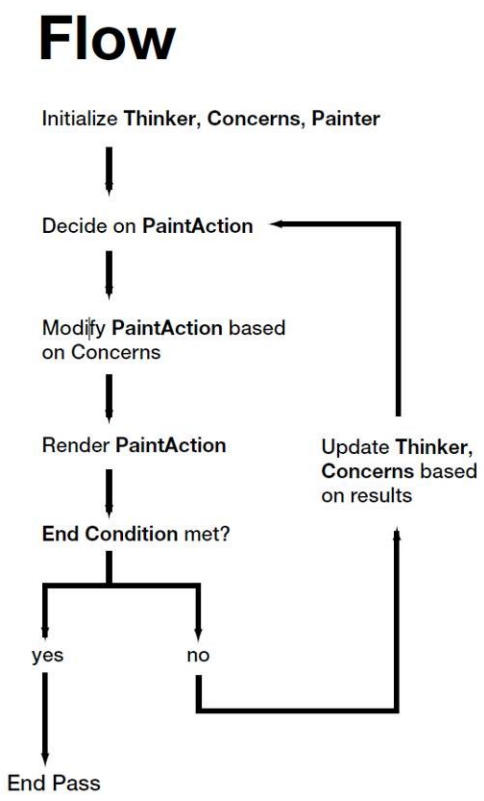


Figure 3.2. Process flow and main modules of ThinkerPainter (Image taken from (DiPaola, 2013))

The thinking process is recursive and is repeated after rendering the synthesized strokes on the digital canvas. However, there are some higher-level constructs which can and will affect the blob selection, as well as the rendering process. These high-level constructs, called "*Concerns*", may, for instance, dictate the Thinker that a cluster of leaf blobs should be treated and rendered as a rather bigger 'parent blob'.

Action Painter

This component is responsible for painting/rendering the PaintAction onto the digital canvas by making splines on the reference image's gradient to direct strokes. Painter literally tries to be faithful to the true shape of each stroke as appeared in PaintAction. Moreover it incorporates '*detail*' and '*density*' parameters in the rendering process. These two parameters determine how gross/fine and dense/sparse the final result will be. Painter's process can be interrupted, intervened and modified by Thinker,

most specifically by the PaintAction, determining how the strokes should be painted onto the reference canvas.

Concerns

The interception that happens between Thinker and Painter components and on the PaintAction is via '*Concerns*'. These *Concerns* are based on *Painterly*'s concerns and considerations. For example, it is a concern of the system to make the regions around a center of interest more detailed with a specific stroke density. *Concerns* can communicate upwards in the perception blob tree to modify the PaintAction. Therefore, by performing a more global analysis over the state of the painting at each point of time and exerting the set of *Concerns* in the process, it is guaranteed that the resulted PaintAction is always updated and optimized. *Concerns* are the ideal place to leverage more high-level considerations and soft-rules into the system and put them into test.

Meta Palettes

PSDPortraitPalette is actually a 'meta Palette'. It is used for applying and/or combining different color palettes for different regions of the source (hair, skin, clothes, etc.). *BlendPalette* acts true to its name which is blending colors from different palettes. These two palette components belong to the same class. In a higher-level work, the regions of reference source image can be semantically assigned a color. This color can be taken form another image (e.g. a painting masterpiece that the user happens to like for its color palette) by using a tone-to-color matching technique. *RelativeJCHPalette* assigns a color to every single stroke. It selects the color from the palette, belonging to the same semantic area of the corresponding stroke (Figure 3.3).

Figure 3.3. The Maps affecting brush parameters across the canvas. SmartPalette is extracted from the original photo's color palette.

Relative JCH Palette

CIECAM02 colour space is the final result of an experimental project. The project started with FullRangeValuePalette, which lead to SmartHSVPalette and then RelativeHSVPalette. Mapping and converting tonal values to color values is rather helpful in mimicking the actual cognitive process of a human artist. A human painter works with a higher-level value system consisting of values for light distribution across the canvas, rather than the color or tonal values themselves. CIECAM02 color space has been incorporated in current version of *Painterly*, since observing and learning from

art practices had shown that the usual software color conversion system were nothing close to the cognitive process that a human painter undergoes for choosing color palettes (DiPaola, 2013). Using this color space is a great advantage for CPA in synthesis of in-between frames (See Sections *4.3.2 and 4.3.3*). RelativeJCHPalette maps color values and lightness/darkness values together. It gets a color value system as an input and applies it to the target image in the same way a human painter does; by remapping and rescaling tonal/color values to lightness/darkness values.

A major drawback of using CIECAM02 colour space in *Painterly* is the time overhead it has added to painting process. Converting color values to lightness/darkness values slows down the system by a great deal. However, being more close to the actual cognitive process of humans' painting, incorporating this color space in the toolkit has been a step forward towards the ultimate goal of *Painterly*.

3.3.2. Sub-Components: Blobs, Strokes and Concerns

The main elements a human painter combines in their work are light, volume and also the content of the art piece being created. These are also the elements incorporated in *Painterly* by using relative scaling, a higher-level smart Palette decision, a set of comparisons and an iterative decision making process. *Painterly* can communicate with any 2D buffer/map, reference data or any other knowledge equation however it also has the potential of leveraging 3D knowledge into the process, but for that the 3D volume or the plane of the scene or reference object/image should be available.

The hierarchical blob tree, together with the semantic content maps and labels, enables the user to connect the tone-color map with the semantic content of the reference image (e.g. larger brush with constrained color will be used for 'background'). But also, there are other features allowing the user to leverage the knowledge about the 'space' or 'volume' of the objects – as they appear in the source image - in the final result of painting process.

In reality, a human artist uses not only the luminance but also the depth of field as well; and in fact, a combination of these two factors. Keeping this in mind, *Painterly*

has provided a set of complicated semantic *Concerns,* enabling the system to go back and forth between these two approaches, based on the reference content and the other knowledge collection. For instance, if calculated 'depth planes' is over/under a specific error angle value, system changes to using gradient.

3.4. Limitations

Since *Painterly* is the backbone of CPA, knowing its strengths and weaknesses is a crucial matter as they will affect our work, directly or indirectly, and work in our favor and/or against us. Accordingly, there are some limitations in *Painterly*'s design and performance which are explained in the following.

Painterly can be used in many different research areas, from art to vision, for experimenting new NPR techniques. However, the research oriented nature of the toolkit (DiPaola, 2007) has compensated speed, which means delaying the research process itself, waiting for the results of an experimental render to come out. The rendering process, thus far is between 1 to 15 minutes per image, on an average computer, PC or laptop with an average graphics card. This number can vary depending on the number of passes specified in the XML script input. The number of passes in its own turn is directly related to the desired amount of details and refinement in the final result. So creating a more elaborate output might take even more than the previously mentioned time. *Painterly* mainly uses OpenGL as the 3D language for the graphic techniques which is hardware accelerated. But the challenge here, which slows everything down, is the conversion of RGB to JCH color space. Since there still is not a GPU (Graphics Processing Unit) implementation of these colour conversions available.

As a research tool, current *Painterly* system does not create a good end user experience, due to not having a proper user interface. One can definitely say it is not implemented for ordinary end-users because of its multiple language based scripting choices. Part of a currently ongoing research of *Painterly*'s team is to create a higher-level authoring system, capable of writing and executing scripts automatically. Using AI (genetic algorithms in particular) in creating a better and smarter front-end is one of the

considered options. Having a suitable user interface will ease out the working experience for the users by a great deal.

Communicating with different knowledge buffers, such as color palettes, mattes and depth buffers through scripting language, makes the system highly customizable and knowledge-based. However, a lot of these components and data files, the matte files for instance, are still somewhat 'out-sourced', meaning that they are created separately with other software and applications such as Adobe Photoshop or Autodesk Maya. This disconnection regress the aspired 'ease of use'. Painterly is not fully automatic as the preparations from tweaking style parameters to generating matte files are done manually. However, the step of generating matte files can be automated, mainly when the input image[s] are imported from a CGI 3D source. Wielding more automation in the process is a desired goal and part of planned future works for *Painterly's* team. Nonetheless, it should not be forgotten that some steps of the process such as regioning and labeling the semantic perception blobs are highly content-driven and completely different for various types of images (such as portraiture images and landscape scenes). *Painterly team* has begun discussions with Zhao (Zeng, Zhao, Xiong, & Zhu, 2009; Zhao & Zhu, 2011) about collaborating on a shared research in semantic labelling area. The research might include using semantic parse trees onto and above perception blob trees.

Currently, *Painterly* is focused on portraiture, not in the internal JAVA code but more on the assumptions, knowledge buffers and scripting; although it has been tested and used it on reference images containing other focal objects besides faces (Section 1.2). Extending the content knowledge semantics to include other genres of imagery such as landscape, figurative, still life, etc. is also another goal worth pursuing for *Painterly team*. This expansion would ultimately affect several aspects of our CPA system as well. In order to achieve that, knowing other knowledge approaches towards these painterly genres is a must. Implementation and testing are the steps coming after that. Building up a system which is capable of working with different styles of imagery is the ultimate state for *Painterly team*.

3.5. Fine-Tuned *Painterly* rendered stills

In this Section we present a number of fine-tuned painterly rendered stills, created with *Painterly*, to show the real power of this toolkit and the wide spectrum of styles it can create.

Figure 3.4 shows several styles generated from one reference still image. These different styles demonstrate just a small part of the wide range of styles that *Painterly* can generate. All different aesthetics and looks have been achieved through modifying *Painterly* toolkit's painterly parameters.

Figure 3.5 shows more fine-tuned examples that can be generated with Painterly. These examples were used in four eye tracking studies which gave empirical evidence that the artists, in general, and Rembrandt in particular, can influence the viewer's appreciation of art by selective use of painted detail. The aforementioned works showed that Rembrandt clearly anticipated the scientific knowledge (i.e. vision science), which was also verified by these works for the first time, through the collection of objective data of viewers gaze patterns (DiPaola et al., 2010, 2013). It should be noted that *Painterly* does not merely mimic the surface properties from the Rembrandt source but uses an elaborate cognitive model of the human painterly process to recreate Rembrandt's style in the previous and following images. The findings from this work are being applied to emotional authorship in rendered characters (H. Seifi et al., 2011; Hasti Seifi, 2010) and within face to face communication for autism studies to better filter an autistic viewer to the goal of the communication. All these research, by iVizLab and other researchers which use open source *Painterly* will presumably benefit from the strong temporally coherent NPR animation sequences that CPA can provide.

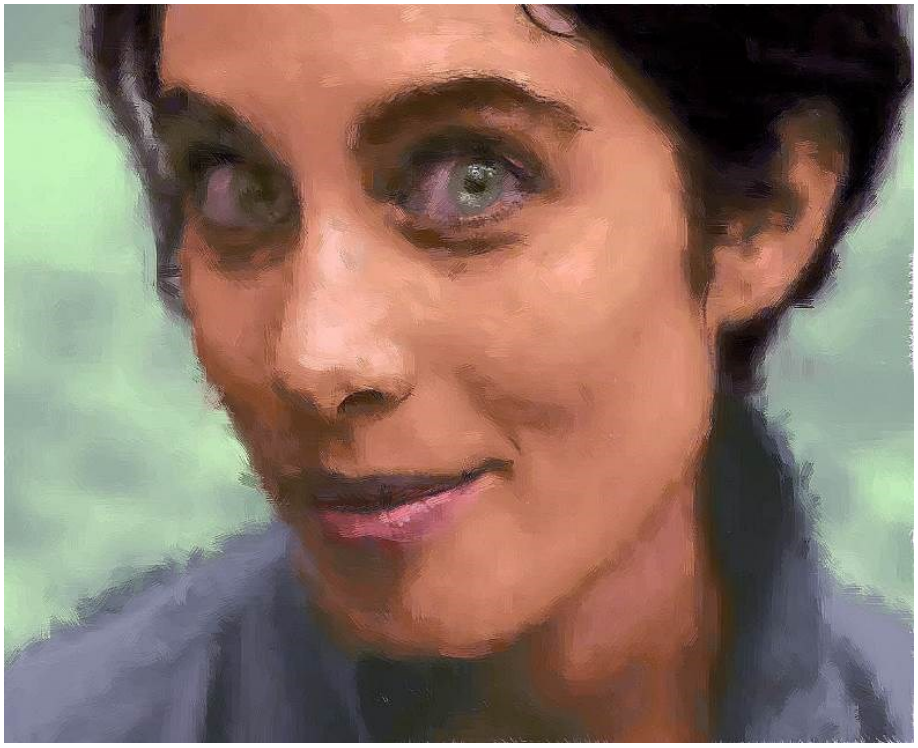


Figure 3.4. Different styles of painterly rendered stills from one reference image (the very last image), generated by Painterly

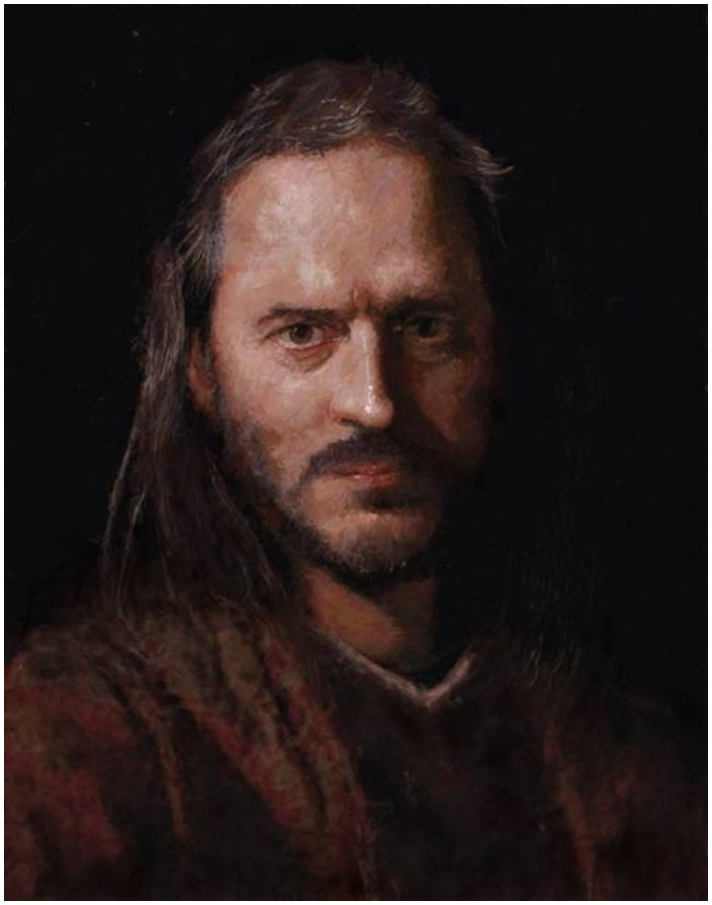


Figure 3.5. ***Different painterly rendered stills, using Painterly toolkit, specifically showing that empirical research can both support the arts (e.g. art critics on Rembrandt's contributions) and learn from the arts (scientists can mine innovations intuited by artists) supplying an early vision/perception toolkit and process.***

4. Our Solution for extending *Painterly* to Create Time-coherent Animations: Architectural and Algorithmic Design

We start this Chapter by discussing the challenges to which we are proposing a solution and articulating our high-level design goals. We then describe the essential aspects of our proposed solution, discussing first the overall system architecture (and its integration of the existing *Painterly* system), and outlining the key algorithmic aspects of our solution including key-framing, stroke-to-stroke matching, and stroke interpolation. The system implemented based on this solution model is called CPA, standing for Cognitive-based Painterly Animator. In the context of this thesis and this Chapter accordingly, this name is used for the designed model and the developed system, interchangeably.

4.1. Design Goals and High-Level Approach

Painterly is a knowledge-based parameterized NPR toolkit with the goal of creating still 2D and 3D painterly NPR images, mainly from portraiture (See Chapter 3). Time is not a salient factor in rendering still imagery. As a result, this toolkit did not previously include mechanisms for ensuring temporal coherency if used to produce multiple frames for a movie/animation sequence. The aim of this work was to extend *Painterly* with a means for creating such coherency, thereby obtaining a suitable system for creating animation in a painterly style.

Part of our motivation in utilizing *Painterly* was to take advantage of its well-developed, cognitively inspired algorithms for semantic parsing and hierarchical, blob-based stroke filling (based on research into the practice of skilled human painters). Therefore, one specific design goal was to extend *Painterly* for animation in a way that preserves and leverages a large degree of *Painterly's* cognitively-based processing.

To summarize, the following basic goals informed our system design:

- Extend *Painterly* to the task of creating animations
- Maintain and utilize *Painterly's* cognitively-inspired rendering capabilities
- Provide a solution that ensures a high degree of temporal coherency across time (i.e. across the sequence of image frames) which performs better than previously used pixel-based interpolation algorithm discussed in Section 1.2.1.

Considering the challenge of creating painterly animation, many different fundamental approaches could be considered, having different mechanisms as well as different aesthetic outcomes. A simple approach is to treat each frame as a separate painting; however, this approach generally leads to a large amount of flickering and fails to meet the goal of temporal coherence. As an example of a practical and modestly successful alternate approach from previous literature, the system of Hertzmann (Hertzmann & Perlin, 2000), painted over the previously rendered frames, only on the areas that had been changed. In the current work, we explored a different approach, which focuses on achieving coherency and the sense of 'flow' by treating the animation as *a collection of strokes which gradually move, deform and change color in a smooth way, over time.*

4.2. System Architecture Overview: Extending *Painterly*

Maintaining cohesiveness in a painted video requires the system to be stateful. The internal components of such a system should be able to keep detailed information about the rendering process of each reference image and utilize this information in processing and rendering the next images/frames. *Painterly* is currently incapable of passing on and utilizing this information in its Thinker component's internal process for generating blob trees and corresponding strokes of the ongoing procedure. Every image created and rendered by *Painterly* is unique, and therefore it differs from the previous and next images. The change in color, orientation, length and position of each single brush stroke happens – significantly or very subtly - in all areas of the image - regardless of the semantic region in which the stroke falls in.

4.2.1. Extending Painterly with External vs. Internal Modifications

For addressing this issue, there can be two different approaches. The first approach is to implement the required information flow inside the body of *Painterly* itself, within the Thinker-Painter framework. This way, all the rendered frames would initially be made in coordination with each other and eventually will have a cohesive flow. This approach might sound ideal, since it gets to the root of the issue and addresses it from the bottom up. However, this approach has the drawback of modifying *Painterly's* algorithms somewhat arbitrarily and therefore moving away from *Painterly's* well-researched foundation in the cognitive practice of human painters (and away from the potential to build on this research with further cognitive and perceptual modeling elements). If a human painter had to constrain their perceptive and creative thinking and techniques to procedures that only work correctly (or the same) with 100s of frames in an animation rather than just the one painting in front of them, that work would suffer. Artists do not work with that severe constraint. This internal animation thinking approach might also limit *Painterly's* ability in creating various styles of painterly imagery by limiting the behavior of painting elements (e.g. brush strokes, Concerns). As a drawback on the pragmatic level, this approach would require a complete re-designing of the structural and process model of the system. A redesign that would move away from artist and cognitive knowledge centered approach.

These drawbacks lead us to the other possible approach, which we elected to follow: addressing the information flow problem from outside the body of *Painterly* itself and through an extended system which uses *Painterly* as one element. Such an external solution guarantees that no internal process or component of *Painterly* will be changed or modified. The result of this work will practically expand the scope of *Painterly,* from being a stateless and still-oriented system to a stateful and multipurpose one, without changing its original ‘time-insensitive’ nature. Since *Painterly* sets itself apart from other similar system by being more of a research tool, this expansion will also affect the scope of its target research domain to include new dimensions, to study and explore.

4.2.2. Keyframing, Interpolation and Data Flow to/from Painterly

Our solution to induce coherency in *Painterly* rendered animation/movie sequences involves intervening in *Painterly*'s frame synthesis process, and doing a part of this job with our system - CPA. We use *Painterly* as our base system, being the main planning and rendering engine. This means that we use it to initiate a planning process on how to execute the frame-synthesis procedure, incorporating its cognitive knowledge space into the process. We take on the responsibility of doing the frame synthesis procedure based on this planning process. Then we have *Painterly* render our generated frames (Figure 4.1). We are able to do this backward processing with Painterly, due to the robustness of this toolkit (See Section 3.1). Our solution depends on 2 types of data which flow between *Painterly* and the rest of the system: *keyframes* (used as input to *Painterly*) and XML *stroke log files* (a data format, both output from *Painterly* to acquire a stroke-plan for the keyframes and input to *Painterly* to render additional frames) (See Section 5.6 for an example of this XML files).

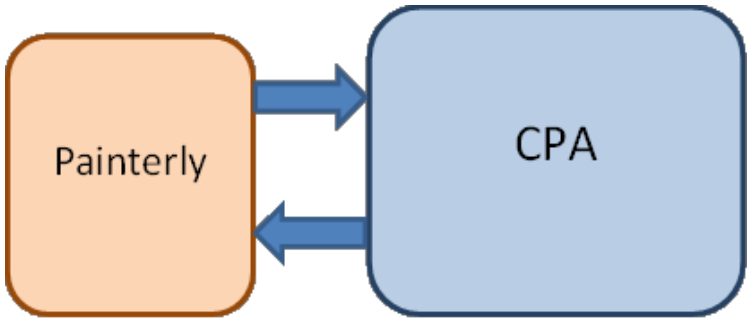


Figure 4.1. CPA as a black box, in relation to Painterly. CPA uses Painterly to initiate the painting process and also to finish it.

Keyframing has widely been used in many different approaches in the video painting field. Keyframes are frames which contain some content that define the starting and ending points of a smooth transition. So, keyframing is to assign some specific parameter values to any of the content of keyframes, as a specific point in time. In our solution, the use of keyframing provides several benefits. It allows us to render certain frames using *Painterly's* full Thinker/Painter apparatus, thereby leveraging *Painterly's* cognitively-based algorithms. At the same time, it allows us to avoid painting every frame separately with *Painterly,* thus avoiding the temporal incoherency discussed in Section 4.2 and also mitigating the large time-cost of *Painterly*'s time-consuming

rendering process. Keyframing allows our extended system to intervene in the frame synthesis process by intelligently interpolating the non-keyframes (we shall refer to these as in-between or 'IB-frames') in a manner which encourages temporal coherence.

Our solution starts with a set of keyframes, provided by the user or any other system. In this sense we have taken a similar approach as the three most influential research works on this thesis which are Lin et. al., Kagaya et. al. and Hertzmann (Mizuki Kagaya et al., 2011; T. Lin et al., 2012; O'Donovan & Hertzmann, 2012). These keyframes could be selected and extracted either manually or automatically, from the target movie/animation sequence. The keyframe set is processed by *Painterly* to get to use the result of its thinking, painting and rendering algorithms in the form of outputted XML stroke log files. This XML file contains a hierarchical tree representation of passes, semantic blobs/regions, strokes and strokes' style-parameters (See Section 5.6). The information in these XML files is used as reference points in propagating and generating the necessary IB-frames between the reference keyframes. For producing IB-frames, each two consecutive keyframes, referred to as *source* and *target* keyframes, are used as beginning and ending points of a transformation procedure. This transformation happens to the contents of source keyframe in order to metamorphose them towards their corresponding ones in the target keyframe. The IB-frames are passed as XML stroke log files, back to *Painterly* for rendering. All the keyframes and the generated IB-frames are then stitched together to create a movie/animation sequence (See Section 5.7). The result will hopefully be both cohesive and yet cognitive-based, due to actively incorporating *Painterly*'s cognitive knowledge space in the course of generating IB-frames. Figure 4.2 demonstrates the whole process, from beginning to the end, more clearly.

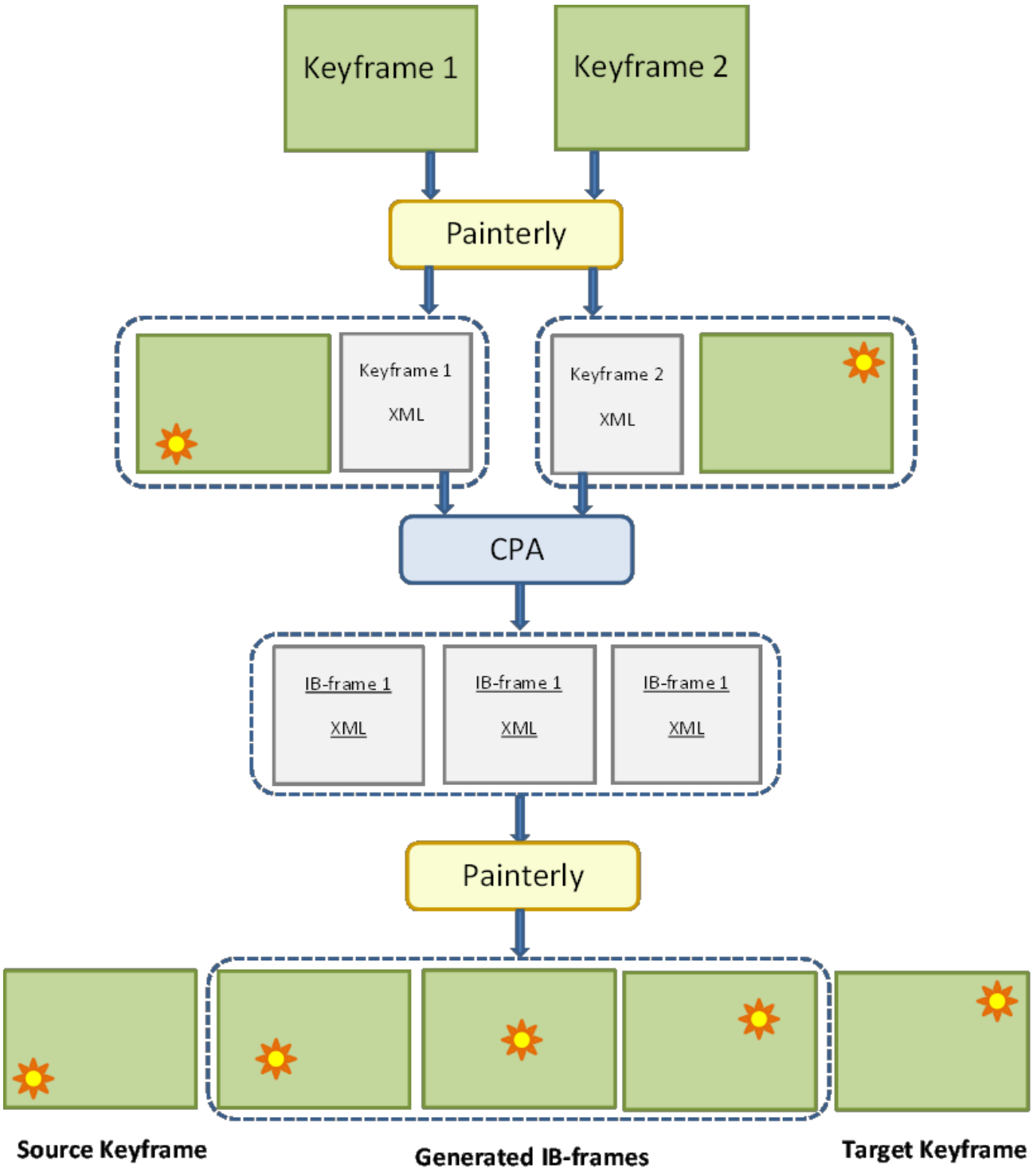


Figure 4.2. ***Generating IB-frames with CPA. Keyframes are fed in Painterly. The XML output of this phase is then used by CPA. The XML representations of IB-frames, outputted by CPA, are then fed back to painterly to get rendered.***

Examining our proposed technique for generating IB-frames, the algorithm comprises two main phases: **1) Analyzing/Mapping** which analyzes source and target XML files and maps together their corresponding content and **2) Transforming/Generating** which transforms the source's mapped content towards their corresponding target states and generates the output IB-frames. These two phases are discussed in more details in Section 4.3.

4.3. Proposed Technique for Stroke-based Keyframe Interpolation

In this Section we describe the technique we have designed for generating stroke data for IB-frames based on intelligent interpolation of stroke data from keyframes. Our proposed technique consists of two main phases which we will call Analyzing/Mapping and Transforming/Generating. The Analyzing and Mapping sub-phases work together to accomplish the task of creating an optimal stroke-to-stroke mapping between the collection of strokes in one keyframe and the collection of strokes in the following keyframe. The Transforming and Generating sub-phases utilize that mapping, and work together to generate interpolated strokes which smoothly transform in position, shape and color from a starting state in one keyframe to an ending state in the next. These two phases and the functions they perform are illustrated in Figure 4.3 and discussed in more detail in the proceeding sub-sections.

4.3.1. Review of Terms: Pass, Region, Stroke

In this sub-section we briefly review some terminology and concepts used heavily in the following Sections and sub-sections.

Semantic Regions/Cognitive Blobs

These similar concepts (used here interchangeably) refer to sub-regions within an image that are associated with semantic label data. This data construct is used by *Painterly* and our proposed extended system – CPA - to incorporate cognitive/semantic information in the process of painting, rather than just strokes and/or pixels. Using the semantic region data, *Painterly* and CPA are able to a) "see" the big picture of the

painting process; b) “think” about different areas of the source image and reference canvas and c) “decide” on *which* area is going to be treated next as well as *how* the act is going to be executed.

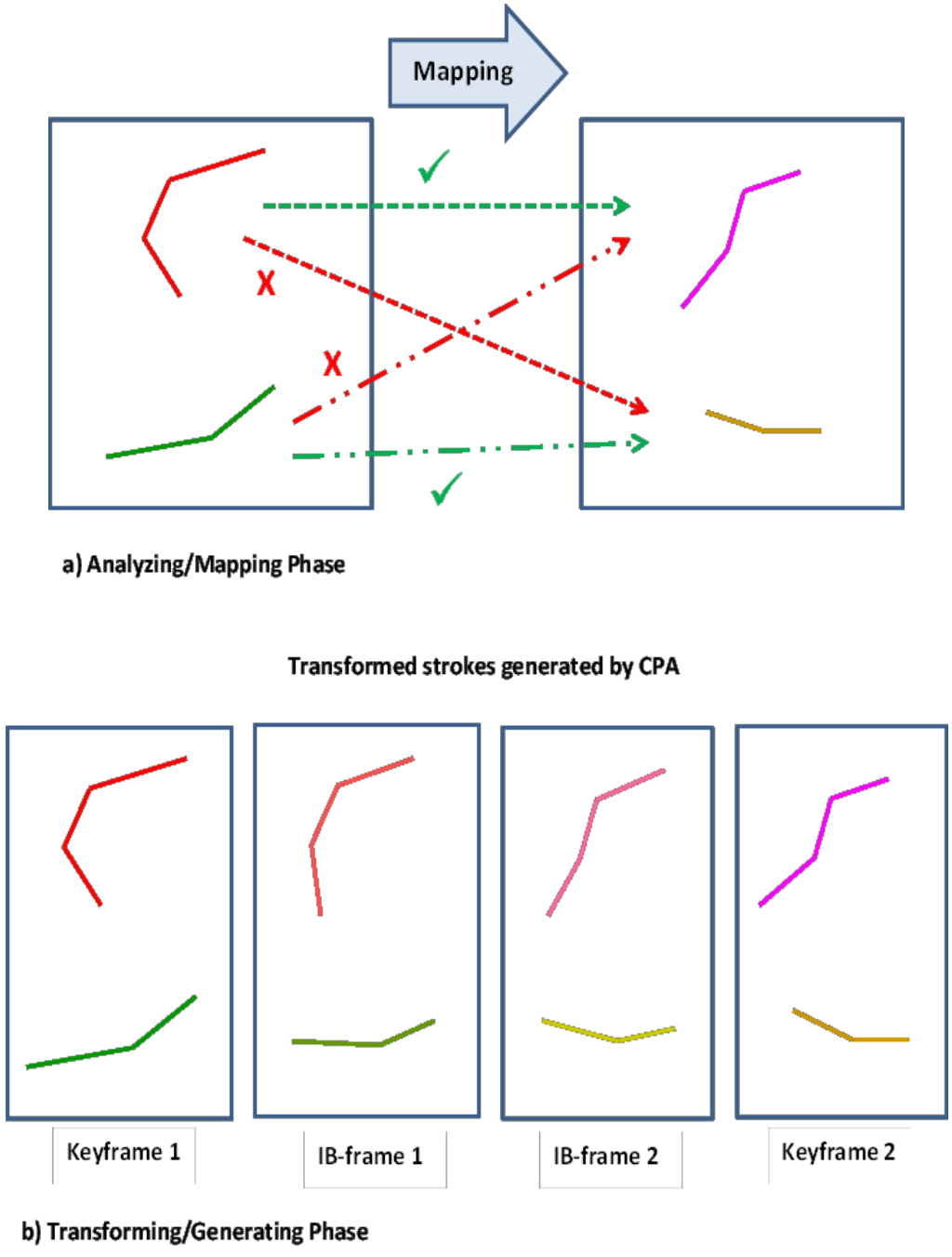


Figure 4.3. CPA’s main phases. (a) Analyzing/Mapping and (b) Transforming/Generating phase

Pass

Pass refers to a round of stroke-placement on the virtual digital canvas. Also on the script level, a *Pass* consists of a number of *Regions* together with a *Pass Header* carrying its attributes and properties (See Section 5.5).

Stroke

In this context, stroke refers to a high-level construct, built of low-level parameters such as opacity, scale, color and points. A stroke models several parameters of an actual brush stroke in a real painting.

4.3.2. Analyzing/Mapping Phase

The main goal of the Analyzing/Mapping phase is to analyze, compare and map between the contents of source and target reference keyframes. Recall that, the main information blocks which are passed back and forth between *Painterly* and our extended system – CPA - are XML stroke log files. These XML scripts, as explained before, are output by *Painterly* and serve as a [painting] process-plan, containing all the information about passes, semantic regions and strokes with their hierarchical orders and their specific property values. The Analyzing/Mapping phase must compare between two given XML stroke log files which represent two successive keyframes from the input frame-set.

To transform the XML elements which are passes, regions and strokes, from the source keyframe script towards the target, we first need to find an optimal mapping between these elements. Since every pass and its corresponding semantic regions are built out of simple strokes, we need to map between the strokes of source and target XML files. By finding the most optimal mapping, we obtain a smoother transformation and eventually enhance the coherency of generated IB-frames.

A painted frame is made up of multiple passes of strokes. The order of these passes affects the final aesthetic look of the resulted image. To maintain this order, we need to find several mappings, one per each pass of strokes. Additionally, each pass contains a number of cognitive regions. It is not only crucial to keep these regions intact, but also we need to form a strong connection between them and coherently maintain it

throughout successive keyframes. Each region consists of a number of brush strokes. Each brush stroke has an order in the XML's hierarchical tree, in addition to other style-properties. Ultimately, in order to maintain the orders' priority and regions connection, we should map between strokes of each two corresponding semantic regions from their corresponding passes in source and target XML scripts.

Finding the most optimal one-to-one (stroke-to-stroke) mapping is not a straightforward procedure. Although the two Source and Target XMLs typically have the same number of passes, there is no guarantee that they also have the same number of strokes. Moreover, no two strokes from the source and target XML files are necessarily the same in shape, size, orientation or color. Consequently, we need to compare various pairs of source and target strokes, to find the ones which are the closest to each other based on some criteria. This criteria and the stroke distinction are explained in more detail in proceeding sub-sections.

Region Equality as the Eligibility Criteria

In the sense of using scene semantics, this thesis work's approach is highly similar to Lin et. al.'s work (L. Lin et al., 2010). CPA extracts the semantic region information from the provided XML inputs and utilizes it to generate a strong connection between semantic regions in the given set of keyframes, similar to Lin et.al. However, unlike Lin et. al. which extracted the scene semantic information inside their segmentation and video parsing algorithm, this semantic information is provided to CPA from an external source – which is *Painterly*. This connection is used as an anchor point in synthesizing the IB-frames. In other words, the first level of comparison made on the source and target strokes is on their semantic regions. Any target stroke is *Eligible* for further analysis only if it belongs to the same semantic region as the reference source stroke. This first step filters out all the strokes from other semantic regions and guarantees to keep the mapping in the same region. The resulted set of all the eligible target strokes is called the 'Eligibility-Set'. All further analysis is done only on the members of this Eligibility-Set.

Position-Color 'Similarity' as the Mapping Criteria:

CPA maps each two corresponding eligible strokes together, based on their degree of '*similarity*' in color and position, or more precisely, a combination of these two. In this context, we define 'similarity' as the degree of fulfilling all the necessary criteria that makes a stroke a proper candidate for mapping among the rest of the eligible strokes. In this case, similarity in position simply means that the actual position of source and target strokes, across the X and Y axis (with pixel being the measuring unit), is the most similar to each other. Just like position, similarity in color also means that source and target strokes have the minimum difference in color. These criteria inform a set of rules and constraints which direct the process of mapping and affect the result of this phase which consequently determines the aesthetics of the resulting IB-frames. Figure 4.4 illustrates the position- color-based mapping strategy. The degree of similarity, either in position or in color, is the objective function which we are minimizing and optimizing, since it directly determines the fitness of the mapping and therefore configuration of the generated stroke, and consequently the aesthetics of the results. In this sense, we have taken the same road as Hertzmann (O'Donovan & Hertzmann, 2012), as we also based our objective function on utilizing optical flow of the scene objects.

Position Similarity; Minimum Movement

The key to achieve and maintain cohesiveness is to minimize the amount of changes happening between source and target keyframes. Since the two keyframes are different in content, their XML representations also vary, in the sense of strokes' positions, orientation and color. We need to make these changes morph smoothly from the source keyframe to the target one.

According to our experience and observations, the number one enemy of cohesiveness is the movement of strokes, which naturally happens when source and target keyframes are rendered separately. Most commonly, reducing the amount of physical movement of a stroke produces a greater improvement in perceived coherence compared to other changes (e.g. color). We speculate that the reason lies in how humans perceive movement and the effect of this perception on directing their gaze and

attention. Investigating the psychological aspect of this phenomenon further would be interesting but is beyond the scope of the present work.

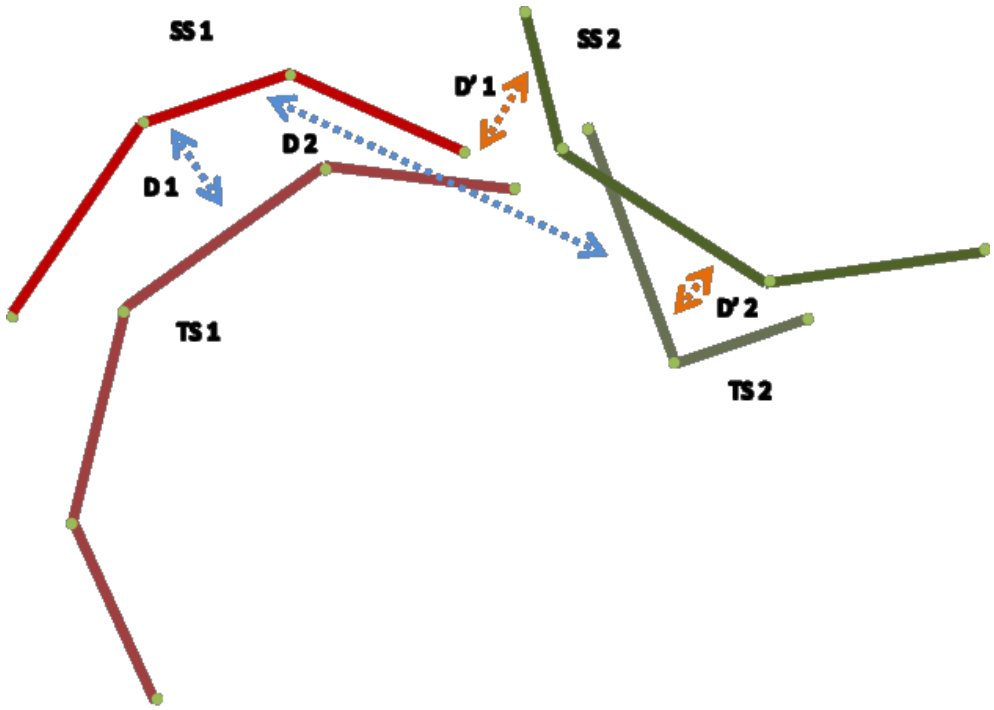


Figure 4.4. ***Mapping source and target strokes, based on position-color similarity. Source Stroke 1 (ss1) is compared to target stroke 1 and 2(TS 1 and TS 2) in terms of position and color. D1 and D2 are the corresponding distances between SS 1 and both TS1 and TS 2. The combination of these position distances and the color distances will determine the best mapping for the SS1.***

Thus, to minimize the amount of undesired and annoying visual artifacts and enhance the sense of coherency, we need to keep the stroke-movements to a minimum possible amount. Of course the rendered keyframes do not necessarily contain identical strokes. (In fact, practically they have nearly none.) Therefore, our best option to minimize the amount of stroke-movement is to map each source stroke to one from the target keyframe which is the most 'similar' one, in terms of position. This means that the two source and target strokes should have the least distance among all other strokes. This will guarantee that each mapped stroke will eventually undergo the minimum amount of movement form the source keyframe to the target one.

Each stroke possesses more than one control points, and also the strokes do not have any direction – unlike the regular vectors. For these reasons, we needed to find the minimum distance between source and target strokes' control points and ultimately average the calculated distance to come up with an overall position-distance score for two given source and target strokes. Figure 4.5 better illustrates the method of calculating this distance score for every given source and target stroke.

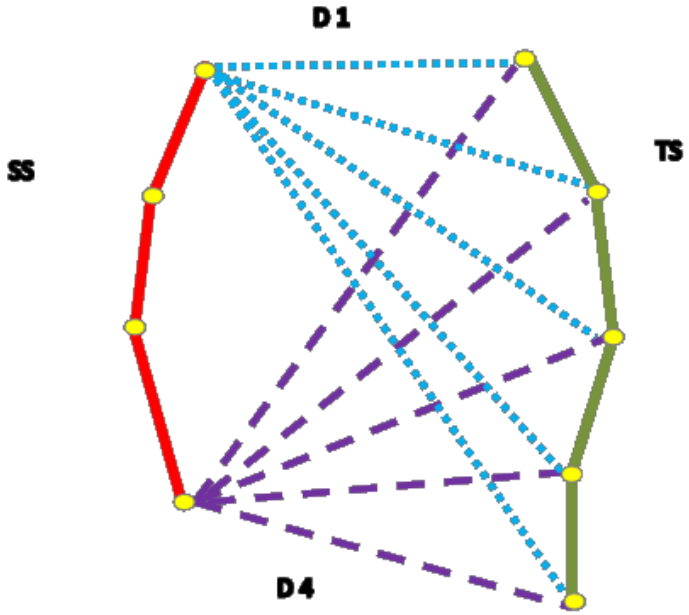


Figure 4.5. ***Calculating distance between source (SS) and target (TS) strokes. D_i is the summation of all Euclidian distances between the reference control point of the SS and all points of TS. The overall distance is the summation of all D1 to D4.***

Color Similarity as another Determining Factor

Current version of CPA is, in fact, the second version of this system, after applying a round of revisions to its former design model. The revision was based on some observations on the performance of the former system. Through these observations we detected a number of issues which addressing them was the base for the first round of revisions. There have been a number of modifications and additions to the first version of CPA, to get this system to its current configuration.

Two rather important additions were made to the former version of CPA after the first round of revisions. Firstly, we incorporated 'color similarity' in the course of mapping

strokes and secondly we focused on ‘how’ we can incorporate this similarity. As mentioned above, spatial position – shortly referred to as position in the context of this thesis - is the number one factor to watch for, when we want to minimize the amount of changes between source and target keyframes. Nonetheless, when talking about the ‘similarity’ of source and target strokes, we cannot disregard ‘similarity’ in terms of the strokes’ color values. Since it is almost impossible to find two strokes with the exact same color value number, our best option was to consider ‘color similarity’ instead of ‘color equality’ in finding the best match. We also had to transform the source stroke’s color into the target stroke’s color, later on, in the Transformation phase (See Section 4.3.3 for more detail on color transformation). ‘Closeness’ in the source’s and target’s color values, leads to more smoothness and subtlety in transformation and therefore more cohesiveness in the resulting animation sequence.

Using CIECAM02 color space was a huge advantage and imposed a big challenge at the same time, towards tackling the ‘color similarity’. In this 3-dimensional color space, a perceptually relevant difference between two color values can simply be calculated as the Euclidian distance between the representative points of these color values. As a result, to find the two strokes with the most ‘similar’ color, we just needed to find the target stroke whose color point had the shortest distance from the source’s color point, among the eligible candidate strokes. The calculated value of this distance is assigned to each candidate stroke, as their color distance score. (See Section 4.3.3 for more details on the implementation of color metrics.)

But how this color space is structured to allow this conversion? The Jab color space guarantees that any pair of points within it would seem exactly as similar as any other pair of equal Euclidean distance from one another to the average human (someone who is not visually impaired or have faulted sight). This means that ‘J’, ’a’, and ’b’ correlates are based on humans’ actual visual perception. Their limits, therefore, correspond to the biological limits of humans’ vision and visual perception (which are complex and out of the scope of this thesis work). The shape of this color space is also somewhat quirky; the 'red' end tends to have larger limits than the 'blue' end because humans’ eyes are far better at distinguishing between reds than blues. This is because humans evolved the ability to see yellows/blues fairly recently in their genetic history; long after they became able to see reds/greens

In this color space, ‘J’ is somewhat normalized arbitrarily to a 0-100 scale; this means that ‘a’ and ‘b’ are scaled relative to J’s value, such that 1 unit of distance in any direction represents an equal change in perceived color distance. The extreme values of ‘a’ and ‘b’ are determined mostly by the biological and perceptual limits of humans’ eyes. In other words, they depend on the range and type of the light which humans’ eyes can detect; there are values of 'a' and 'b' that theoretically exist but are ignored, since most humans cannot perceive them as being any different from some other less extreme values.

To be able to make comparisons between color values of different frames, the color space and the correlates need to be normalized. This ‘Normalization’ was the actual curveball in using this color space, since it is not properly and fully normalized. The value of ‘J’ ranges from 0 to100 and can therefore be normalized with some success, but the two Chroma correlates - 'a' and 'b' – are not as straightforward. A value of '0' for ‘a’ or ‘b’ means 'without Chroma’, that is gray. However, that is the only real assumption we can make about ‘a’ and ‘b’ correlates; since each one of them can get positive or negative values and their ranges are not conveniently defined.

Whit all that been said, there was no convenient way to normalize ‘a’ and ‘b’ without being arbitrary or clipping off valid parts of the color space. In fact, any selected ratio would be essentially arbitrary.

Position-Color Ratio

Rather than being separate factors which can be calculated and measured individually, color and position have a nontrivial correlation in determining the degree of fitness for a possible mapping, for any given stroke. So the similarity score for any two given stroke, is in fact a combination of position similarity and color similarity scores. The important question in this state of the method is how to find a balance point between ‘position’ and ‘color’ in this combination. In fact, this ratio may depend on the content of the images.

There was no standard solution for determining the best ratio in various conditions. Moreover, normalizing the color space was a nontrivial task which we had to deal with. Therefore, we needed to come up with some heuristic method to find the best

ratio range, eventually obtaining a suitable value to use in our implementation. Regarding the color distance, we were just looking for finding the minimum difference between the source and target strokes' color values, and therefore the maximum possible differences were not a matter of salience. As a result, to find the best combination of position similarity and color similarity, our best option was to find different scale factors between color and distance and observe the results. Nonetheless, no one ratio will be 'better' than the other; they will merely represent different styles. Therefore, the best thing we could do was to parameterize these ratios, so that users can modify them as they please, to get the desired look in the final result. The details of our method and findings are discussed further in Section 5.5 and Section 6.1.1.

Stroke Mapping: Situations Where 1-to-1 Mapping is Not Possible

Three possible scenarios might happen in the course of mapping corresponding strokes from source and target XML files. The first, ideal, scenario is that for each given stroke from each pass and cognitive region of the reference source XML, a fit match is found in the corresponding pass and region of the target XML. The second possible scenario happens when there remains at least one stroke in the source XML script which is not matched with any eligible stroke form the target XML file: we refer to this source stroke as a 'Not-Matched Source'. The third scenario is when there remains at least one stroke among the target XML script that is left unmatched. We refer to this target stroke as a 'Not-Matched Target'. Such Not-Matched strokes are dealt with in the Transformation/Generation phase by applying *Fade-in* and *Fade-out* processes (See Section 4.3.3).

Stroke Mapping: Analysis and Processing of Sub-Strokes

CPA encodes the strokes by a series of control points, making it possible to consider each stroke as a list of sub-strokes. Each sub-stroke possesses at least 2 control points. All of these sub-strokes share the same style-properties, as the original stroke. When rendering each stroke onto the reference canvas, these control points are used to generate a spline, to represent the reference stroke. In the course of designing CPA, we explored 2 different techniques for taking the multi-segmented nature of strokes into account.

The earlier - the former – technique was to break each stroke to its constructive sub-strokes. This means that each stroke was broken down to a set of smaller strokes, each having just 2 control points but sharing the same style-properties, as the original non-broken stroke. For instance, a stroke with 5 control point would be broken down to 4 smaller sub-strokes. Figure 4.6 shows a better view of this process.

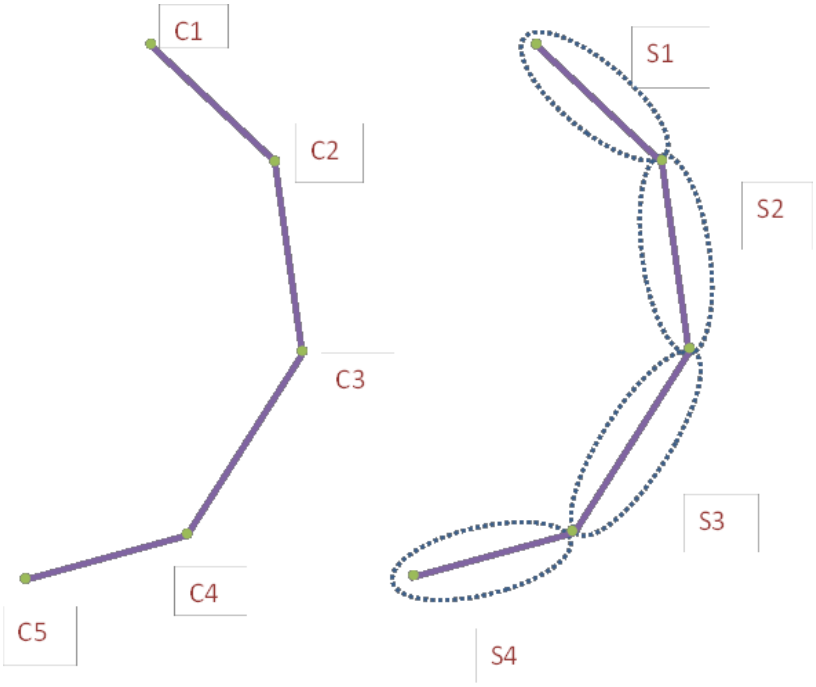


Figure 4.6. A Stroke with 5 control points broken down to its 4 constructing sub-strokes, marked by blue dashed circles.

By choosing this approach, we could increase the chances of each sub-stroke for getting mapped to the most 'similar' sub-stroke. Theoretically this would have created much subtle changes in position and color of the mapped strokes throughout successive key-frames. As a result, we would have created a more cohesive animation/movie sequence

There were a number of issues with our first technique which eventually lead us to make some changes to get to the current technique used in CPA. Through generating our very first batches and test-runs using the first technique, we realized that this method was not increasing the chances of a sub-stroke for getting matched and was, in fact, worsening the incoherency of the final sequence. The reason was that this approach would end up matching different sub-stroke particles from a source stroke, with sub-

strokes of a number of different target strokes and transforming them accordingly towards the target particles. Therefore, each unified stroke would end up disintegrating and blending with other strokes, by the end of interpolation process (Figure 4.7).

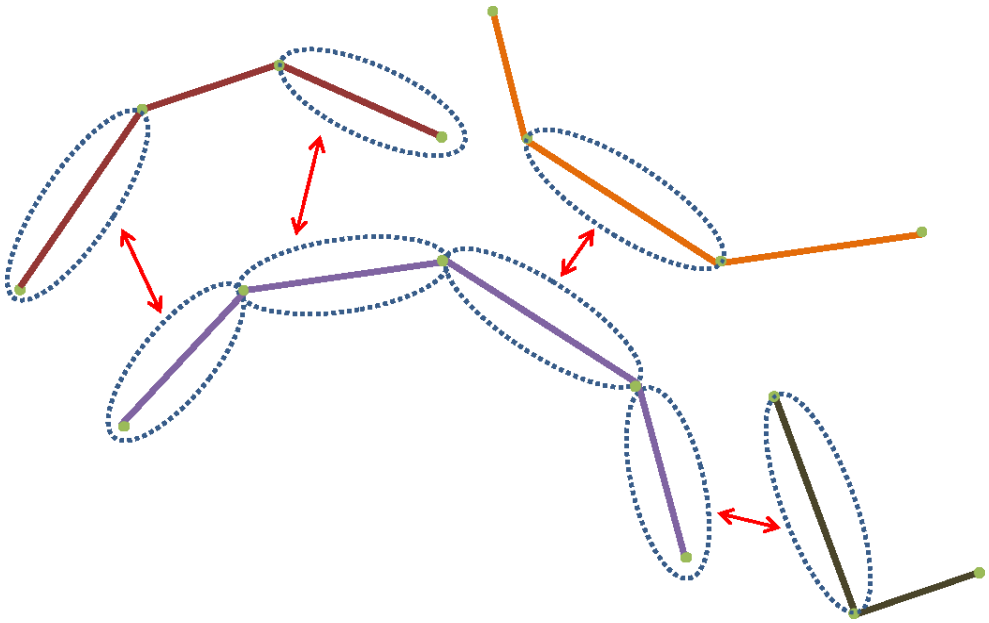
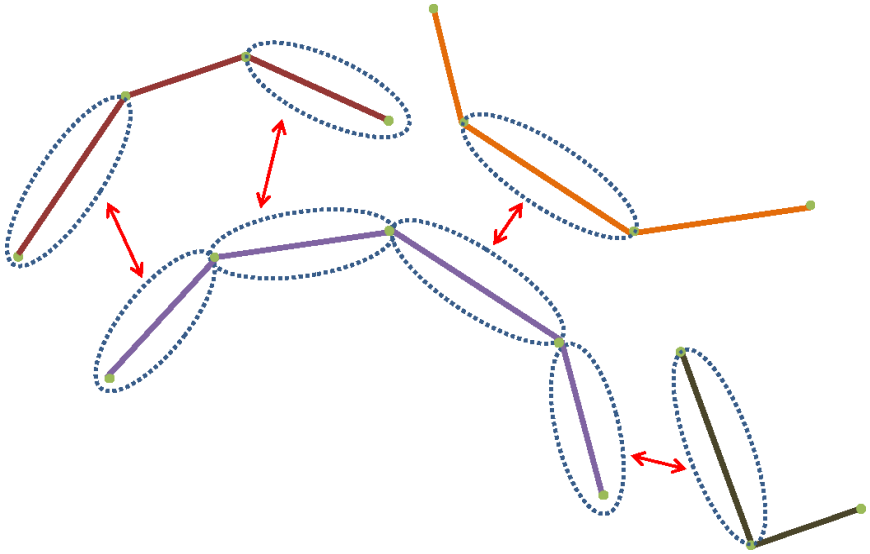


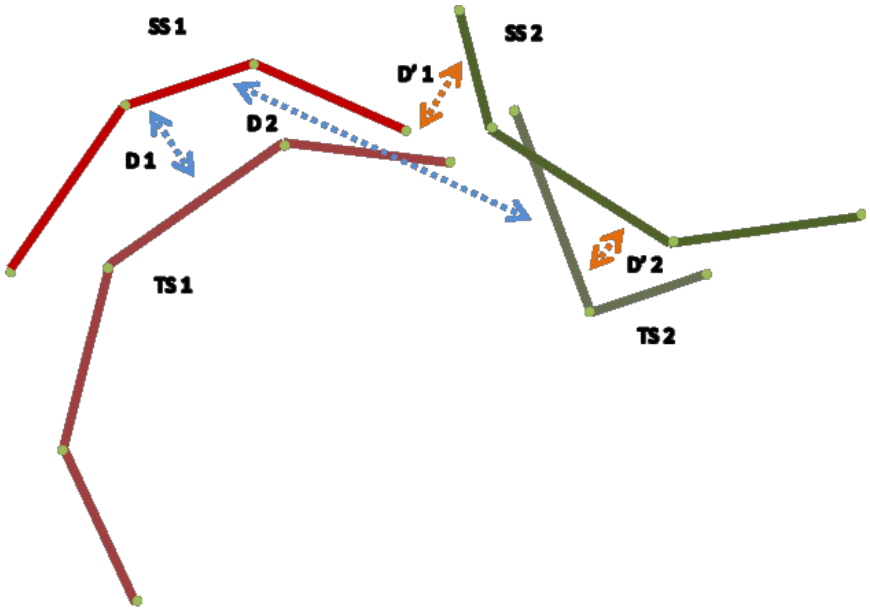
Figure 4.7. Sub-stroke particles get mapped to multiple target strokes' particles.

Nevertheless, these sub-stroke particles were never put back together to form a whole complete stroke as it appeared in the next key-frame. This was because by the end of morphing process each sub-stroke had drifted far apart from the body of the original stroke, so putting them back together was practically impossible. Therefore this method led to a stream of steady small changes and movements happening across the frame, and failed to yield the desired cohesive outcome. The second issue was the high memory complexity of this approach. Increasing the number of strokes by breaking down each stroke was doubling, tripling or in some cases quadrupling the size of active memory taken by the system and therefore, damaging the overall performance of the system. Addressing these issues formed the basis for our current technique which is used in the current version of CPA. This technique is based on keeping the strokes' integrity. In this technique, we treat each stroke as a whole unit, without breaking it apart. Strokes of the source keyframe are compared to their corresponding ones from the target keyframe to eventually be mapped with their fittest counterparts. Figure 4.8

shows a better comparison between the former technique and the current technique of mapping source and target strokes.



(a)



(b)

Figure 4.8. ***CPA's former and current mapping techniques. In the First technique (a), each stroke was broken down to a number of sub-strokes and mapped accordingly to other sub-strokes from multiple target strokes. In the current technique (b), strokes are mapped as a whole unit and transform accordingly, keeping their unity.***

Grid-Based Search

To help limit and minimize the stroke movements through the time, CPA uses a grid-based search, by superimposing a grid window on the reference keyframes. Strokes are labeled based on the grid window they fall in. So each stroke gets a series of labels which are actually the grid window number of the windows their control points fall in. While searching for candidate eligible strokes for a reference source stroke, these window labels are used to filter out the strokes which have at list one grid window in common with the reference source stroke. Just like region, the grid window is used at the very first step of analysis, to determine the eligible strokes for further comparisons and analysis. Strokes which are not from the same semantic region or does not have any grid window in common are not even eligible for similarity score calculation (Figure 4.9).

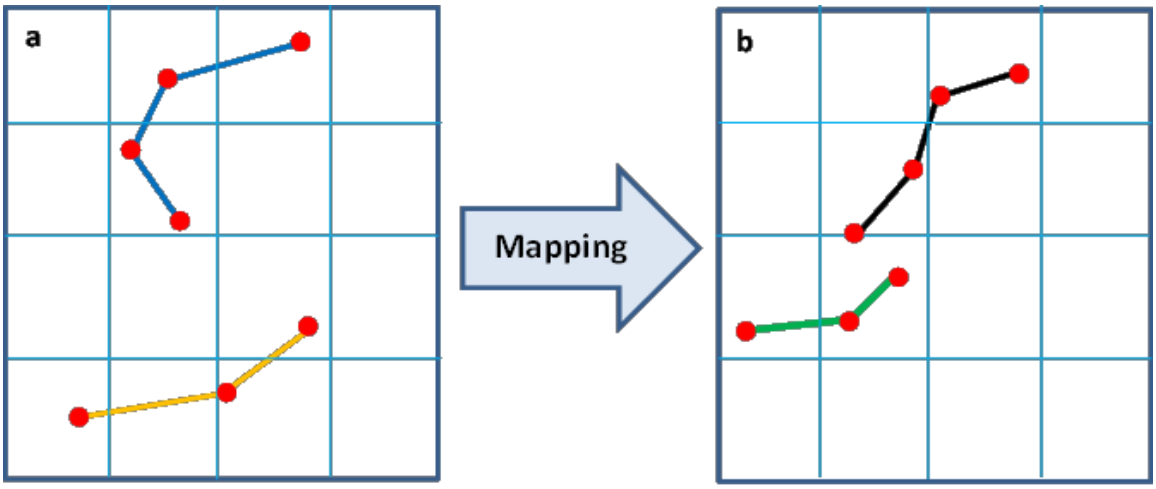


Figure 4.9. ***Grid windows are superimposed on source (a) and target (b) keyframes. The black stroke in the target keyframe is an eligible candidate stroke for the blue stroke in source keyframe, since these two strokes have common window labels for their control points. The green stroke is not eligible for further analysis for the orange source stroke, because they don't share any common window.***

This grid-based search has two main advantages. Firstly, it speeds up the search to find the candidate strokes for mapping, by shrinking the size of Eligibility-Set (See Section 4.3.2) and accordingly the number of the eligible strokes that should be examined for similarity score calculation. Secondly, it directly helps to increase the coherency level. By framing the area in which the system should look for an eligible

candidate stroke, this grid search physically limits the allowed position transformation, since strokes can be mapped only if they fall in the same window. The grid window value is an integer number that determines the number of evenly distributed columns and rows across the frame. Therefore, any frame will eventually be divided into ‘grid value x grid value’ numbers of windows. This parameter is applied to every individual pass separately, so each image can be processed with multiple grid window size values. The size of these grid windows are eventually determined by the user. More detail about this parameter is given in Sections 5.8.2 and 6.1.2.

4.3.3. Transforming/Generating Phase

The most challenging goal of this phase of our solution is to transform the content of source keyframe towards their corresponding in the target one. This transformation is done based on the mapping result from the previous phase. Just like mapping, transforming needs to be performed on representative elements of the frames’ contents, which are elements of the source and target XML files. The Transforming/Generating phase takes on the responsibility of reflecting all the necessary changes to morph each mapped element from their states in the source XML towards their states in the target file. Strokes, as the smallest high-level constructs of the painting, are the entities which undergo this procedure. As these strokes are ultimately represented and referred to, as the numbers and property values in CPA’s internal procedures, the transformation is applied to almost all their properties (i.e. position, color, opacity and order). The rate and time course of transformation for these properties depends on the 'Interpolation-Style' settings in CPA (See Section 5.8.4).

The final Generating aspect of the phase is conceptually more straightforward, as described in ‘Generating Sub-phase’ sub-section in the current Chapter.

Three Types of Stroke Transformation

There are three main types of transformations. Correspondingly, they happen in any cases of three possible mapping scenarios, articulated in Section 4.3.2.

Normal

A *Normal* transformation is a simple morphing, happening in cases of one-to-one mapping between corresponding strokes from source and target XML files. This transformation is performed on the style-properties of a reference stroke, to gradually alter them from their source-values towards the target-values. This whole process follows a dominant interpolation-style, which can be determined and authored to the model, through system-parameters (See Section 5.8). The stroke-properties which undergo this morphing process are position, opacity, color and order.

The remaining two transformation types can be construed as special modifications of the *Normal* transformation type.

Fade Out

This transformation type is used in the scenario when the source stroke has not been mapped to any corresponding stroke in target image (*Not-Matched Source)*. Hence, to prevent any sudden disappearance of that stroke, it is faded out instead by gradually reducing its opacity; changing it from its current value to zero. The reference stroke will go through the Normal transformation on all properties, except for the opacity.

Fade In

This transformation type is used in cases when the target stroke is not matched and mapped to any corresponding source stroke (*Not-Matched Target)*. To avoid any abrupt appearance of such a stroke, at any point during the sequence, the stroke is set to gradually appear among other strokes. This is done by gradually increasing its opacity value from zero to its current value. Apart from the opacity, all other style-properties go through a Normal transformation.

Transforming Strokes as Whole Units

In order to keep the morphing of images as smooth as possible, CPA treats each and every stroke as a whole entity, without breaking it apart; unlike the former technique (See Section 4.3.2). The source stroke will move, rotate, change its color and opacity and even its order in the hierarchical tree structure of XML script, in order to metamorphose to the target stroke. Figure 4.10 shows a better depiction of this transition

of the source stroke's control points along the transformation procedure. Middle states of the morphing stroke are the state of that particular stroke in its corresponding IB-frame. The number of these 'transitional states' is an open-parameter of CPA, defined by the user. It actually is the number of IB-frames the user wants to make between two successive keyframes.

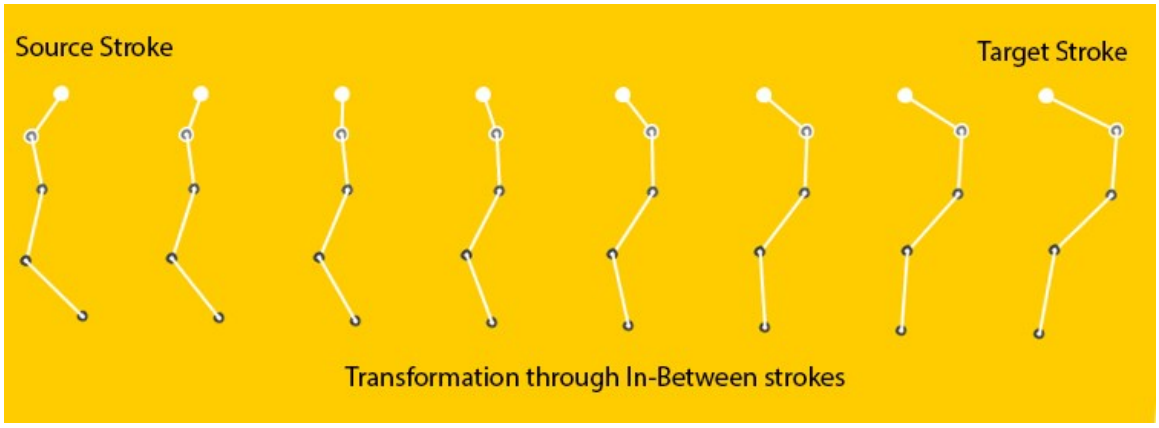


Figure 4.10. The spatial changes of the control points of a stroke, transforming from its source configuration to the target configuration (Photo credit: Nahid Karimaghalou).

Equalizing the Number of Control Points

One consequence of keeping strokes as whole units is that the transformation procedure may include an extra step of changing the number of their control points. The two mapped strokes – source and target - might differ from each other in the number of points (e.g. one has 3 and the other has 7 points). Therefore, equalizing the number of control points is the first step in the morphing process. By adding the necessary number of point to the smaller stroke, we guarantee that both mapped strokes will have the same number of points, which eventually makes the transformation process easier (Figure 4.11).

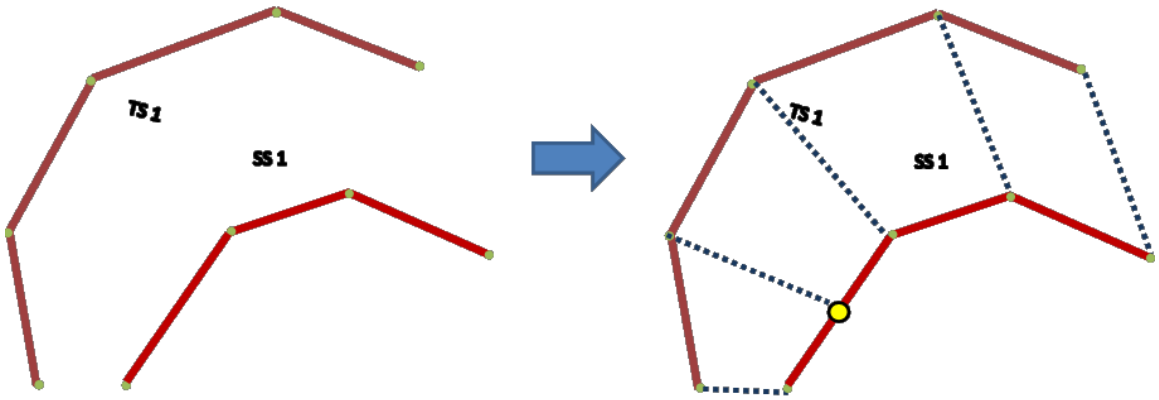


Figure 4.11. Equalizing the number of control points in two mapped stroke. The yellow point is the new control point added to the smaller stroke - SS 1 - to equalize the number of its points to TS 1

Accounting for Stroke Order

One other important change we made in the earlier version of CPA, to get to the current state, was to consider the 'order' by which a stroke is painted onto the digital canvas, in the transformation procedure. This order number is crucial in determining the final look of the resulting image. Eventually, all the strokes are put on top of each other on the virtual digital canvas and consequently, they are covering or overlapping with each other. The colors of strokes with lower opacities combine with the colors of strokes underneath them. Some other strokes may get completely covered by the strokes painted on top of them. Thus, it was important to take into account the order of matched strokes and interpolate this value in the transformation process as well. The current method of morphing this order value follows the overall interpolation-style and is currently linear across the generated IB-frames. For instance, if a stroke with an order number of 400 (meaning that it is the 400^{th} stroke that is painted in that pass) is mapped to a stroke with the order of 600, the order of source stroke is changed gradually and linearly towards the order of target stroke. In this example it will increase from 400 to 600. Applying non-linear order-transformation has not been explored in current thesis work. However, it certainly is an interesting practice that will create more abstract image results. It can be explored in future research works.

The rendering order of strokes is not considered in the mapping phase since it is not an actual style-property of the source and target strokes. Therefore, this property is not a matter of salience in finding the fittest map for the reference source stroke. Nonetheless, it is in the rendering procedure that this order becomes important, due to multi-layer nature of the painterly rendered image. Accordingly, to keep the transformation process as smooth as possible and reduce the amount of abrupt changes, CPA, also morphs this property value in the transformation phase.

Transforming Colors

As part of morphing a source stroke to its mapped target stroke, we need to transform their color values as well. As mentioned previously in Section 4.3.2, mapping was done based on strokes' position and color similarity. Because of using CIECAM02 color space, we needed to normalize the strokes' color values based on their color information extracted from the reference frame, so that we could map the strokes together (See Section 5.5). Taking this extra step of normalizing, however, was an advantage in morphing color values in the transformation phase. As a result, each color point could be treated as a control point, in a three dimensional space. Therefore, the transformation process was similar to a normal position morphing, from the source color point towards the target color point.

Interpolation-Style Parameter: Affecting Transformation Style and Motion Pattern

The Interpolation-style, which dominates the transformation procedure for all stroke-properties, is determined by a user-defined parameter. This parameter controls the rate and pattern of position and color morphing, as well as order transformation. Therefore, the style of transformation is directly affected through changing this parameter. It can be set to a constant value for a linear transformation or can be exponential or even the result of any desired mathematical equation to generate more un-realistic and abstract interpolations between keyframes. However, aiming at solving the coherency issue, this parameter has not been the focus of current thesis work, since it mainly affects the style of generated outputs. Nonetheless, the style of generated outputs can affect the perceptual level of coherency, as flickering is more visible, noticeable and sometimes more distracting and annoying in some styles comparing to

some other ones. Ultimately, in this thesis work, interpolation-style parameter has set to have a constant value and eventually create linear transformation, in all style-properties of reference strokes. Exploring different variations of interpolation style, through different values of this parameter is certainly an interesting task, which can be pursued in more future research works.

'Generating' Sub-phase

This step of Transforming/Generating phase is responsible for producing the actual IB-frames. Conceptually, the transformation data which has been computed in the latter sub-phase is applied and integrated to result in the final stroke data for each frame. Notable details beyond this occur at the implementation level, where an internal XML data-structure is instantiated and populated with the new values; this is explained in Section 5.2.2.

5. Implementation

In this Chapter we explain CPA in further detail. We start with its internal data structure, design/component models, process model and dataflow. We then, precede to a more detailed review of CPA's control structures and provide samples of the scripting input/output of the system. We continue this Chapter with some general information about the framework, programming languages and libraries used for implementing CPA. Finally we will conclude with a quick review of CPA's open-parameter and their impact on the outputted results.

5.1. Data Structures

Figure 5.1 shows the main data structures used throughout CPA. The XML script file is read and parsed by a *DOM* object and is transformed to an internal data structure digestible by CPA. Overall, there are three main sub-structures nesting into each other to form a more unified and sophisticated construct. Each *StrokeData* object represents a stroke and consists of numeric values of its style-properties. This structure is somewhat the smallest and most atomic entity of all. *RegionData* objects represent semantic/cognitive blobs/regions of the painting. As everything eventually comes down to computational operations, numeric values and number crunching in parametric systems, cognitive regions of the image are represented by some nested data structures consisting of some other data constructs. Each *RegionData* object carries a number of *StrokeData* objects. Going another level up, every *PassData* object contains a number of *RegionData* objects plus some header nodes. These header nodes carry some common information about the brush properties or other globally required information that all of the strokes belonging to the same *PassData* share with each other. *XMLData* objects represent an explicit XML stroke log file. Each *XMLData* object contains a number of *PassData* objects equivalents to the number of passes in the XML script file, a header

node, carrying the main information about the XML file itself (e.g. the version of the scripting language) and some other attributes with their numeric values (Figure 5.1).

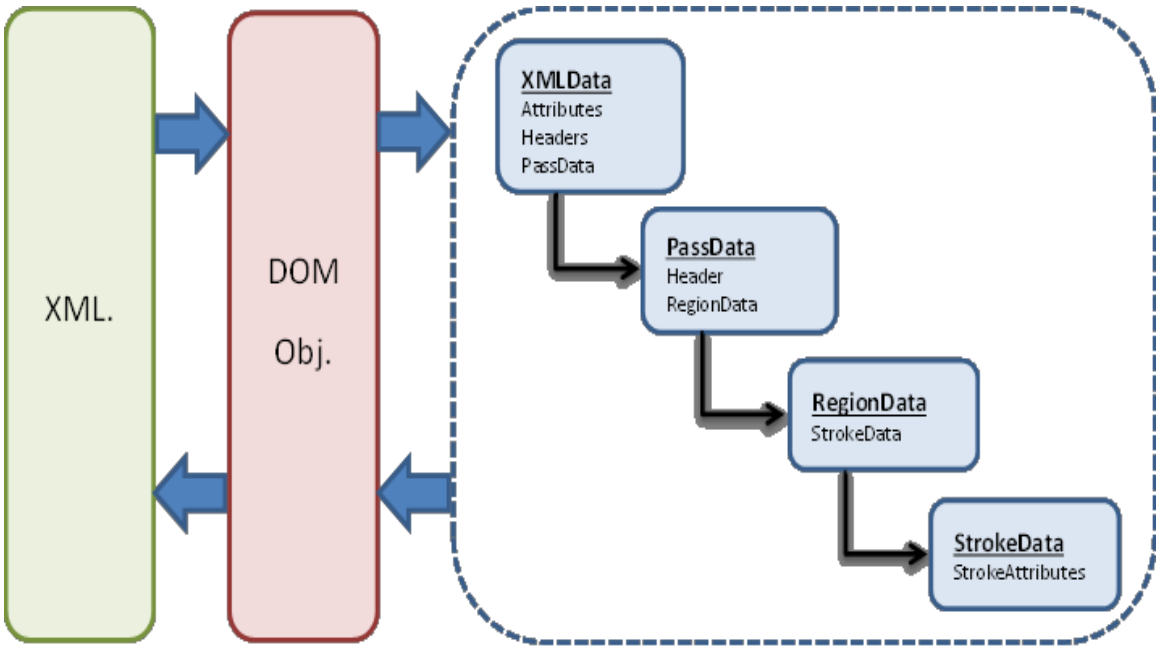


Figure 5.1. CPA's main data-structure and data-flow. The returning arrows show the conversion that happens after each round of frame-generation, to finally output another XML stroke log file.

As showed in Figure 5.2, *MapData* objects contain a number of *StrokeData* objects and a header node. In this sense, this object is somewhat similar to a *PassData* object. The difference is that in a *MapData* object, two *StrokeData* objects are paired together, representing two strokes from source and target XML files that have been mapped together.

Figure 5.2. MapData object's structure

5.2. Component Model

CPA's structure consists of four main components which are *Reader*, *Analyzer*, *Generator* and *Dumper*. However, the main work is done by the two middle components, while *Reader* and *Dumper* are technically responsible for managing input/output streams and reading from/writing to XML scripts. *Reader* intakes the actual XML stroke log file, parses it and populates an *XMLData* object based on that. *Analyzer*'s job starts with breaking down source and target XMLData structures, passed to it by *Reader*. It then analyzes and compares them to find the best mapping between corresponding strokes of source and target XML scripts. The mapping criteria are dictated to the system by CPA's open-parameters, defined by the user. *Analyzer* creates a dataset of all mapped strokes. This dataset will then be used by *Generator* to create the final IB-frames (Figure 5.3).

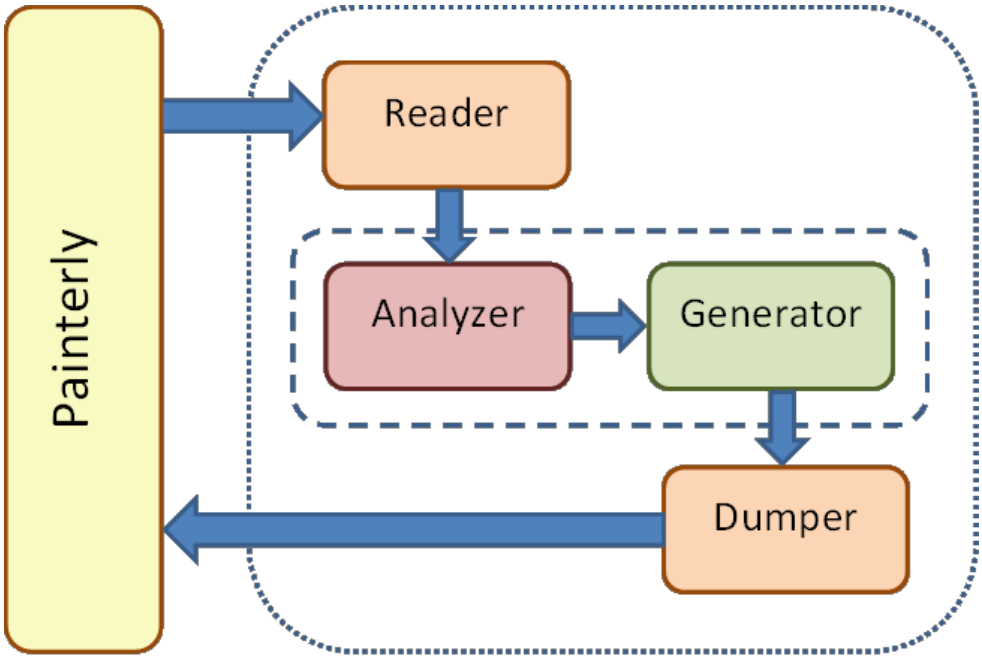


Figure 5.3. CPA's main components. The inputs and outputs are coming from and returning to Painterly.

In the following, we explain the internal structure of our two main components, Analyzer and Generator, in more details.

5.2.1. Analyzer

This component is responsible for analyzing, comparing and mapping between *XMLData* objects. The whole idea behind analyzing two input XML scripts is to find the best mapping between the strokes of each two corresponding semantic blobs from their corresponding passes, based on 'similarity' of strokes in position and color. As mentioned in Section 4.3.2, in this context, we define 'similarity' as the degree of fulfilling all the necessary criteria that makes a stroke an eligible mapping candidate among the rest of the strokes. These criteria are the internal rules and constraints of the system. They are in fact system's open-parameters determined by the user (Figure 5.4).

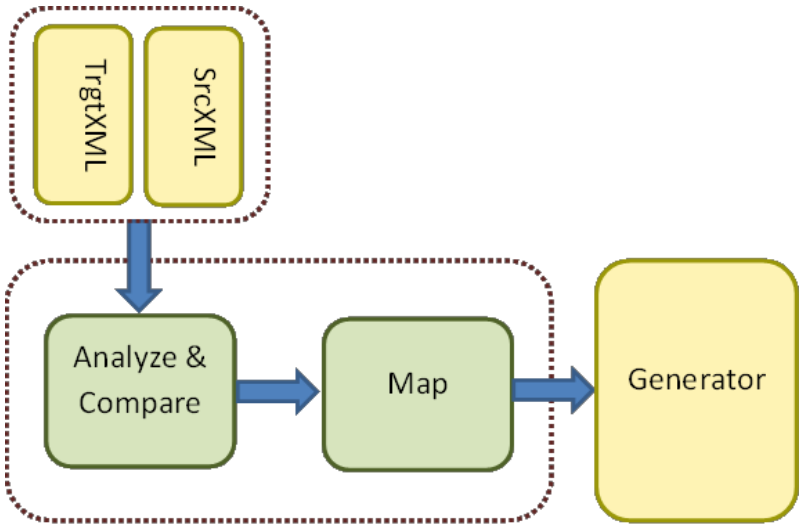


Figure 5.4. ***Analyzer component; zoomed in. Source and target XML scripts are provided through Reader component. Analyzing and comparing is a pre-step before finding an optimal map between source and target XML elements. Generated map is then passed to the Generator component to be processed respectively.***

5.2.2. Generator

This component is responsible for morphing the mapped strokes from their source configuration to their target ones, based on the mapping created by *Analyzer (transforming sub-phase)*, and generating the actual IB-frames afterwards (*generating*

sub-phase). *Generator* also, applies and leverages user's preferences about the number of IB-frames and the style of interpolation. These preferences are also the system's open-parameters. The produced results of Generator are then passed to *Dumper* to be flushed out in the form of XML scripted stroke log files (Figure 5.5).

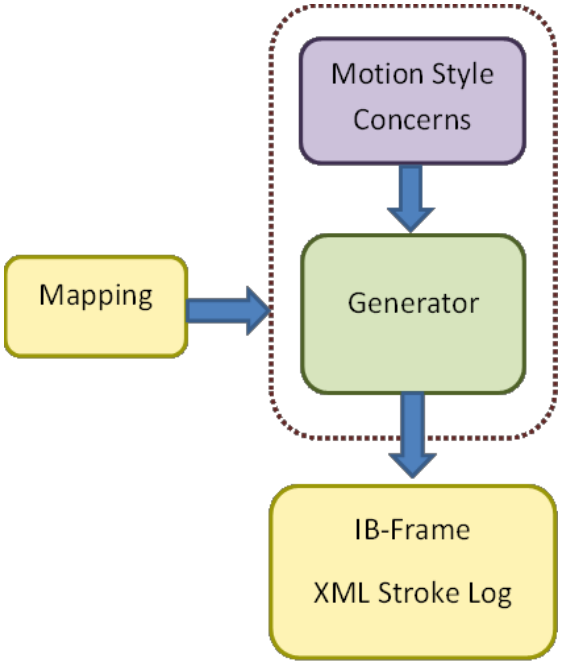


Figure 5.5. Generator component; Zoomed in. The mapping generated in Analyzer component is used in Transformer, together with motion preferences. The newly transformed strokes are put back together to populate an XMLData objects in Generator.

These outputted XML files are fed back into *Painterly* to be rendered and painted onto the digital canvas. The resulting image sets - which comprise all the generated IB-frames for each two successive key-frames, together with all the original keyframes - are eventually stitched together to create a QuickTime movie sequence.

5.3. Process Model - Data Flow

Here, we look at the whole process, from beginning to end, following the flow of data objects in CPA. The input XML stroke log file is read in through *Reader* component and an *XMLData* object is instantiated and filled upon that. This *XMLData* object is then passed to *Analyzer* from the *Reader* component. *Concerns* which are a set of rules to

determine the criteria of mapping based on user's preferences are then initiated and applied in the analysis process of each *PassData*. Consequently, a mapping would be created between each two corresponding *PassData* objects of the reference source and target *XMLData* objects. This mapping is represented as a *MapData* object which is then passed to the *Generator* component. Another set of *Concerns* are initiated here and used by *Generator* to apply user's preferences about the interpolation style in the final IB-frame. *Generator* instantiates a new *XMLData* object, and applies the necessary changes based on the provided MapData, to transform the source XML's content and accordingly create the new IB-frame's XML. This XMLData is then passed to be outputted by *Dumper* (Figure 5.6).

5.4. Control Structure in Details

In this sub-section we provide a short tour through the actual implemented modules and functions of the four main components of the system, from beginning to the end of a job.

5.4.1. Reader

Load

When 'Load' function from the *Reader* component is called, it instantiates an *XMLData* object. Using a DOM object, Load function parses both input XML scripts (source and target) and fills the reference *XMLData* object. As described in Section 5.1, an *XMLData* object is a hierarchical tree structure of *PassData*, *RegionData*, *StrokeData* objects. These objects are linked together based on their relative position and order of appearance in the reference XML script. By superimposing a grid window on top of the reference image, the *Reader* also groups instances of *StrokeData* objects based on the grid window in which their corresponding control points fall, across the digital canvas.

This step makes future accesses and comparisons faster and more efficient. The looseness/tightness of this grid for each *PassData* is an open-parameter of system defined by the user. The two *XMLData* objects created here are then passed to the *Analyzer* to be analyzed and compared in order to create the MapData. This Map Data

contains the information about which two *StrokeData* objects of the corresponding passes from Source and Target XML files are mapped together.

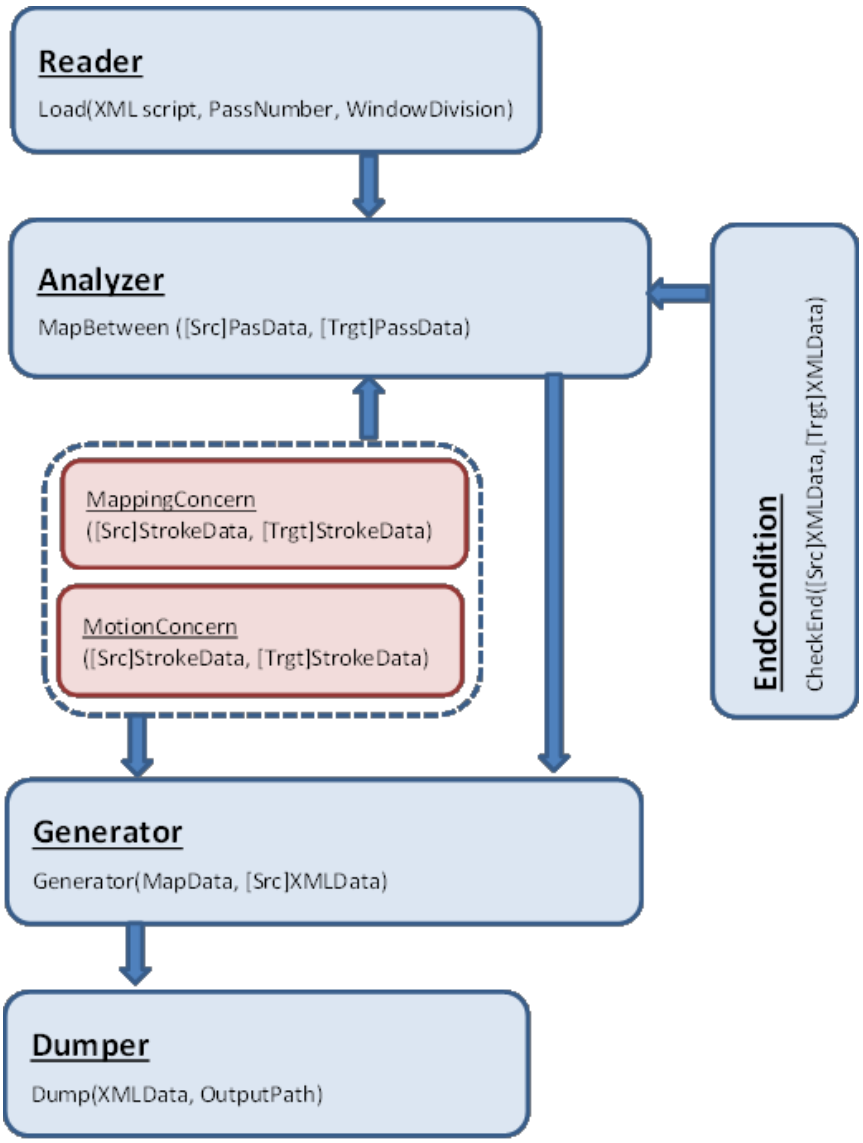


Figure 5.6. CPA's Process model and Data flow.

5.4.2. Analyzer

MapBetween

This function of *Analyzer* component accesses every *StrokeData* from the reference *RegionData* of the source *XMLData*. It then tries to find the best and fittest candidate among corresponding *StrokeData* object of the target *XMLData*. In the course

of comparisons made between the source and target *StrokeData* objects, 'IsEligible' function is called to determine the eligibility of selected target stroke. 'IsEligible' in its turn, creates a set of eligible candidate strokes, called '*Eligibility-Set*', which is then used by 'FindBestMatch' function.

FindBestMatch

This function intakes the *Eligibility-Set* and selects the best and fittest candidates for the given *StrokeData* object of the corresponding pass. The selection is affected by the *Concerns* rule-set, authoring the constraints and mapping preferences of the user to the *Analyzer* component. This functions instantiates and fills a *MapData* object to be used in the *Generator* component.

5.4.3. Generator

MakeInBetweens

This function generates the in-between states of the given mapped source and target strokes based on the user's required number of in-between frames. Meanwhile, it injects *Concerns* rule-set in the process of crafting stroke motion. This rule-set is the channel through which the desired interpolation-style is dictated to the *Generator.* The interpolation-style is an open-parameter of the system defined by the user. The result set which is a set of *StrokeData* objects is passed to 'SetTheValues' function which in turn instantiates an *XMLData* object to be populated with new *StrokeData* objects and flushed out through the *Dumper*.

5.4.4. Dumper

Dump

This function of *Dumper* component transforms the *XMLData* object to a DOM object, manages the output streams and writes out the in-between XML Stroke Log to the output path.

5.5. Color Normalizing

Despite working to our advantage, using CIECAM02 color space raised one rather important challenge which addressing it was not straightforward. In the course of mapping and morphing source and target strokes, we needed to compare their color values and transform them. The problem was that the values of ‘J’, ‘a’ and ‘b’ were not properly normalized. Without normalizing them no valid comparison could be made. Since there was no standard way of doing this, we addressed this issue in a rather content-based way.

What we did was to normalize these values per frame and based on the data extracted from that frame. We extracted the minimum and maximum values that each one of ‘J’, ‘a’, and ‘b’ correlates could have had in that particular frame and used these values to normalize them. However, in practice, we extracted this information once for every batch of frames belonging to the same scene, instead of doing it per frame. Our reason was that successive frames of any one given scene were very much similar to each other regarding color range values.

5.6. Scripting Examples

What come bellow are excerpts of *Painterly*’s Input XML scripts, the Header and the Body, followed by CPA’s input and output script, the XML Stroke Log file. The scripts are indented, shortened and commented for legibility.

In the first two scripts, the main process starts at the Pass blue block. Each Pass block consists of Thinker, Painter and Concern sections. The main dominant information through all processed passes are provided in Header file (e.g. source image and region map file directory) whereas the details about how each pass should be processed and rendered is given in the Body XML script. In Previous versions of *Painterly,* Header and Body used to be combined in one unified script file, which has been changed in the current new version. This separation reduces the amount of redundant header codes and eventually, the possibility of errors in reading/writing the XML stroke log script. By

adding one header node to the script file which carries all the common style-properties, we can dominate all sibling nodes of passes that belong to the same tree root.

5.6.1. Header Script:

```
reference-path  resources/TOICam3/MasterBeauty161.jpg
output-path  o/TOI/Cam3/TestCase2/MasterBeauty161StrokeLogImage-
testing
painting-path dev/strokeLog/TOIstrokeLog-TestCase2-testing.csv

variable resources/TOICam3/MasterBeauty161.psd name  region-map

variable dev/paintingHeader/green           name  palette-folder

variable name  hertzmannPainterExtras1
write-stroke-output o/TOI/Cam3/TestCase2/TOIstrokeLog161-
testing.xml
   %This  tells  HertzmannPainter  to  output  a  log  of  all  the
strokes it makes for pass 1
variable name  hertzmannPainterExtras2
   write-stroke-output o/TOI/Cam3/TestCase2/TOIstrokeLog161-
testing.xml     %This tells HertzmannPainter to output a log of
all the strokes it makes for pass 2
variable name  hertzmannPainterExtras3
   write-stroke-output o/TOI/Cam3/TestCase2/TOIstrokeLog161-
testing.xml          %This  tells  HertzmannPainter  to  output  a
log of all the strokes it makes for pass 3
variable 1  name  thinker-blob-complexity          %  This  affects
how finely the portrait is detailed
variable 0.2  name  gradient-filter-size          %  This  affects
how closely brush strokes will follow the contours of the image;
at large values the strokes will be greatly distorted.
variable 30 name  painter-stroke-length           %  This  affects
the length of brush strokes
variable 0.05  name  painter-grid-size            %  This  affects
the size of brush strokes
```

5.6.2. Body Script

```
% script demo
region-map  % load 2d maps
   file
variablename   region-map
pass   - begin a pass
    thinker   class   BlobThinker
blobs-from-regions TRUE
   leaf-blob-size  0.05
```

```
   max-blob-size 0.5
   blob-complexity  0.08
   blob-blur-size   70
   density 1
   detail   0.9
 painter    class    StrokePainter
opacity 0.5
   grid-size   0.04
   brush-scale    0.9
   brush class eduPainter.toolkit.jogl.GLTextureBrush
      brush-file resources/brushes/spatter.gif
      variablename  hertzmannPainterExtras1
palette    class    eduPainter.palette.RelativeJChPalette
map-file
      variablename  region-map
   palette-folder
      variablename  palette-folder
   palette  class eduPainter.palette.RelativeJChPalette
      ignore-duplicate-palcolors             FALSE
```

5.6.3. Stroke Log Script

Painterly also outputs another XML scripted file, referred to as 'Stroke Log' in this context, which is the input/output script of CPA. The stroke log files generated by CPA would later on be fed-back to *Painterly,* to be painted onto the digital canvas. Below is another excerpt of the mentioned script file which will be discussed afterwards. This code block is also indented and commented for readability.

```
  <?xml version="1.0" ?>
  <stroke-list width="938" height="400">
    <brush-def id="0">
      <brush class="eduPainter.toolkit.jogl.GLTextureBrush">
         <brush-scale>1.0</brush-scale>
  <!--Canonical                                          Path:
'C:\Users\root539\Desktop\paintOutStrokes\trunk\resources\brushes
\spatter.gif'-->
<brush-file>resources\brushes\spatter.gif</brush-file>
      </brush>
   </brush-def>
   <pass>
       <stroke      region="bg"      brush="0"      size="38.294605"
alpha="0.0128058195">
         <color J="19.449165" a="0.19258387" b="0.11884503"/>
   <p x="333.17840576171875" y="218.58920288085938"/>
   <p x="269.790283203125" y="358.0365905761719"/>
   <p x="157.702880859375" y="462.4398498535156"/>
```

```
      </stroke>
     <stroke     region="bg"     brush="0"     size="38.294605"
alpha="0.30496562">
       <color J="19.449165" a="0.19258387" b="0.11884503"/>
       <p x="229.76763916015625" y="76.5892105102539"/>
        <p x="222.14756774902344" y="229.57797241210938"/>
        <p x="316.4951477050781" y="350.25177001953125"/>
     </stroke>
```

The main process starts by initiating a Stroke-List. The header node, ‘brush-def’, provides the required information about the brush class and its image file for all the proceeding passes and their corresponding strokes. Users may change the brush definition for any of the passes. This would add another brush-def node along the way which would dominate all the pass nodes and their child stroke nodes appearing after the newly added brush-def node.

Each Pass contains a number of strokes. This number depends on the ‘density’ parameter defined by the user in the Body XML script. The total number of strokes is virtually infinite; therefore it can go as high as users’ preference for more elaborate and detailed looks. The number of passes also depends on the user’s liking of the output result. Each Stroke element contains a set of control points, with X and Y values, determining its position on the canvas along the X and Y axis. The units of these axes are pixels. Since we are mainly focusing on 2D images the Z value has been considered *0* by *Painterly* for all of the control points.

5.7. Framework

Since Painterly has been developed in Java, we used Java 1.7.0 as our programming language, in order to make the two systems more compatible. CPA is developed in IntelliJ IDEA 12.1 Community Edition, Jetbrain’s Java development environment. This IDE has been claimed to be the most intelligent Java IDE by the provider’s website. It provides the developers with lots of helpful features like fast and

smart code completion, version control support and refactoring tools. The Community Edition of this IDE is available for download from their website[11] and it is free to use.

Painterly uses Java Advanced Imaging (JAI) version 1.1.3 libraries which is an Application Programming Interface (API). This library is freely available for download via their website[12]. *Painterly* also uses Java Media Framework API (JMF) version 2.1.1.e. It is used for incorporating time-based media data such as audio and video, into Java applications. According to their website, version 2.1.1 of this API provides support for capturing and storing media data, controlling the type of processing and performing custom processing on media data stream.

Moreover, to stitch jpeg images together for making an animation sequence, we used the aforementioned version of JAI library. The media library of JAI provides a simple and low-level function to create QuickTime movies from a collection of jpeg images. The frame-rate of the resulted movie is an open-parameter of this function which can determine the speed of the final sequence.

5.8. System's Open-Parameters; Quick Review

We have declared and discussed CPA's open-parameters and their impact on the outputted results along the latter Sections. In this Section we quickly review them collectively, before moving on to the next Chapter and discussing the details about how different values of these parameters can have different impacts on the results.

5.8.1. Number of IB-Frames

This parameter determines the number of IB-frames created between each two consecutive keyframes. Generating a higher number of IB-frames will increase the smoothness of movement and transformation; and vice versa. However, this number needs to be balanced with the number of original frames which have been skipped between each two successive keyframes. For instance, if the user has chosen frame

[11] http://www.jetbrains.com
[12] http://www.oracle.com

number 1 and frame number 6 as the reference keyframes, the number of IB-frames should be in balance with the original 4 frames that have been skipped between their selected keyframes. Going over 4 will slow the animation down but create a smoother transformation. Whereas, going under 4 will speed the flow up but generate a rougher transformation. Whatever this number is, the speed of the final animation piece can also be modified by frame-rate value in the movie making algorithm used in CPA (See Section 5.7).

5.8.2. Grid Window

This parameter determines the size of the grid window which is superimposed on the source and target keyframes. This grid is used for 1) speeding up the search to find all the candidate strokes, in the process of mapping source and target XML files. And 2) increasing/decreasing the coherency by framing the area in which the system should look for an eligible candidate stroke. This parameter will be discussed in further detail in Section 6.1.2.

5.8.3. Position-Color Ratio

Considering color 'similarity' in optimizing the mapping between source and target XML files, raises the challenge of finding a balance point between these two factors in the competition of color similarity vs. position similarity. Simply favoring one over the other would affect the aesthetics of the results. So we should find a ratio between these two values. This ratio will indicate how much the color value will be weighted comparing to the position, in the course of calculating the overall similarity score for a given target stroke. Pragmatically, position similarity score is summed up with color similarity score, in order to calculate the overall eligibility score. So each one of these scores – position similarity and color similarity – are multiplied by a number (integer or float) before being summed up together. The optimal range of ratio between these two parameters is utterly content-based and depends on the quality of the reference keyframe[s]. This parameter's estimating method will be discussed in more detail in Section 6.1.1.

5.8.4. Interpolation-Style

This parameter dominates the transformation procedure for all stroke-properties by controlling the rate and pattern of position, color and order morphing. In other words, this is the parameter which determines *how* each stroke should move and change through time to transform from its source state to the target state. Therefore, it directly determines the style of transformation. This parameter can get many different values. It can be set to have a constant value for a linear transformation or it can be the result of any desired mathematical equation to generate exponential or un-realistic and abstract interpolations. The flickering and other possible undesired visual artifacts might be more noticeable and distracting or disturbing in some styles more than the others. For instance, in a linear interpolation wrong movements might appear to be more noticeable comparing to an abstract form of transformation; since in the latter one, there is not any known stereotype of the interpolation and movements that the viewers are used to and therefore expect to see. As a result, this parameter can indirectly affect the perceptual level of coherency in the final result. Nonetheless, since this thesis work was set out to attend to coherency issue in the *Painterly* rendered sequences, and this parameter does not have a direct impact on the coherency level, it has not been the focus of current thesis work. We have set a constant value to this parameter to create linear transformation, in all of the style-properties of the reference strokes.

6. Parameter Calibration, Tests and Examples of Final System Output

In this Chapter we present and discuss results from different stages of qualitative testing we have carried out as part of the development, refinement and validation of CPA. We begin with parameter calibration tests, which let us understand the behavior of the system under various settings of the parameters left open in initial system design. When then move on to tests which demonstrate the performance of the current system on real-world, industry-relevant data obtained from our NSERC Engage Collaboration (See Section 1.2). All the video sequences which are provided in this Chapter are accessible through author's website[13].

6.1. Impact of System Open-Parameter Settings and Demonstration of Potential Output Artifacts

Being a parametric system, the quality of CPA's results highly depends on the values set for its parameters, and furthermore the optimal setting of parameters may depend on the content of the images being processed. For example, as mentioned in Section 5.8, the number of IB-frames can change to the users' liking. However, the 'fitness' of this number is directly affected by the number of skipped frames between each two consecutive keyframes and the desired speed of the resulting movie sequence. Moreover, providing the proper position-color ratio range and grid window scale is crucial in the aesthetics of the results and the overall processing time

As part of CPA system development we devised various test scenarios to qualitatively explore the impact of different parameter settings. In this Section, we present results showing how different values of each of the CPA's open parameters can

[13] http://ivizlab.sfu.ca/research/PainterlyAnimThesis/

affect the aesthetics of the output. We observe certain failure modes and types of output artifact that can result when parameter settings are not optimal. These findings represent progress toward understanding the best default settings or ranges for the parameters, and toward understanding how to improve the parameterized system with further heuristics or AI techniques in the future.

For a better and clearer demonstration of what actually happens to the strokes throughout the generated IB-frames, we chose to work mostly with coarse rendering styles, using low numbers of strokes. This way, each brush stroke is visually distinguishable and the movements and interpolations are subsequently easier to follow. In some examples, we have cropped the specific part of the image to emphasize more on the point being discussed.

6.1.1. Position-Color Ratio

The effect of a parameter like position-color ratio is heavily content-based, and technically can vary for every single frame. However, considering the fact that successive frames are highly similar in content and change gradually, the ratio value can be approximated for almost all frames in a particular scene. Unfortunately there is not any standard way or rule of thumb to guess the most proper range of this parameter in a realtime manner. Therefore, we took the step of performing a number of test-runs in order to estimate the most suitable range for this parameter. These test-runs help narrow down the wide range of this parameter to find the fittest range which works for that specific frame set.

Figure 6.1 and Figure 6.2 together illustrate the effect of this parameter on the generated IB-frames, when not in a suitable range. Figure 6.1 demonstrates the source and target keyframes that we have used to create the example IB-frames. These keyframes are rendered with *Painterly*, using a low density value, large brush size, and one single pass, to create a coarse result with sparsely distributed strokes across the canvas. This way, the brush strokes are more distinguishable and therefore, their movement and transformation is easier to follow.

Figure 6.2 illustrates three individual IB-frames, generated between the aforementioned keyframes. Each of these IB-frames, a, b and c have been generated separately, with a different value for position-color ratio. Moreover, these frames are rendered with grid size value of 1. This means that the effect of grid size parameter has been eliminated, to isolate the impact of changing position-color ratio. According to our observations on this test data, **the most desirable settings for position-color contain values in the range [10, 13] for position and the range [2, 3] for color.** Instead of using a float number, we found it easier to multiply each of position similarity score and color similarity score by a number from the corresponding range mentioned above, and then sum up the two scores together, to find the overall similarity score for any given stroke. In this figure, parts a, b and c each show the results of settings not employing the proper range. Images (*a)* and (*b)* (the top and middle ones) show results of position-color ratios 14/4 and 20/7 respectively (position is higher than the optimal range). Image c shows the result of the ratio 6/4 (position is lower than the optimal range).

As previously explained in Section 5.8.3, position-color ratios indicate how much the position similarity score will be weighted comparing to the color similarity score, in calculating the overall similarity score. The generated IB-frames show distortion and visible imperfections which is a result of assigning an improper value for position-color ratio.

(a)

(b)

Figure 6.1. ***(a) and (b) are respectively source and target keyframes, used for generating the IB-frames of Figure 6-2.***

Figure 6.2. ***(a), (b) and (c) are single IB-frames generated between keyframes of Figure 6-1. (a) is generated with 14/4, (b) is generated with 20/7 and (c) is generated with 6/4 position/color ratios. Improper position/color ratio has affected the aesthetics of the results.***

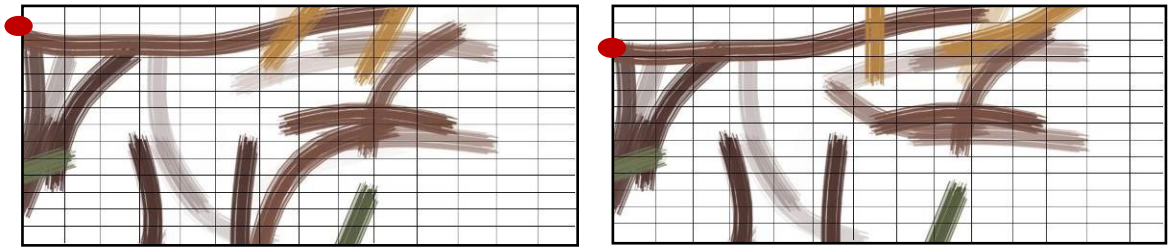
6.1.2. Grid Window Size

Calibrating grid window size also is another important step for enhancing the quality of output frames. The choice of grid value, on one hand, affects the time complexity of algorithm by determining the size of the list of candidate strokes in the mapping phase. On the other hand, it determines how far CPA should look for a candidate eligible stroke and therefore directly affects the coherency level of the final results, because a tighter grid would limit the scope of allowed position transformations and consequently the amplitude of movements. This parameter shows the number of equally distributed columns and rows on both width and height of the frame. Accordingly, any frame would eventually be divided into 'grid value x grid value' number of windows. Grid window is assigned to every individual pass separately, so each image can be processed with multiple grid window size values. The value of this parameter highly depends on its corresponding passes' configuration; how big or small the brush sizes are and how dense or sparse the strokes are distributed across the canvas. If grid window size is not assigned in the right range, undesirable transformations and stroke movements may occur in the results. This will eventually cause more flickering in the final movie. In the following, we discuss three possible scenarios that can happen and their consequences: the grid size value being higher or lower than the optimal range, or being in this range. Moreover, despite the fact that there is not any standard or official definition for grid window size, we found it easier to give a better sense of this parameter's size by comparing it relatively to the minimum or maximum average distance between the mapped strokes.

Tight Grid Window Size

When the grid window size is relatively smaller than the average Euclidian distance between the control points of two closest strokes in the pass, the grid window is too *tight*. A tight grid size shrinks the size of the Eligibility-Set and accordingly decreases the number of eligible candidate strokes for mapping (See Section 4.3.2 for term definition). As a result, it may cause non-optimal mappings. This will make some strokes undergo a series of unnecessary transformation or behavior (e.g. fading in/out). Eventually it will deteriorate the output quality and damage the coherency level.

In Figure 6.3 and Figure 6.4, we investigate the behavior of two strokes in the course of transforming from their source states in the first keyframe to their target ones in the next keyframe. This series of images are rendered with a very sparse density value and a relatively large brush size. They also have been severely cropped to give a closer and better view of each single stroke's movement and transformation. The suitable grid window range for these two keyframes is [2-3]. Figure 6.3 shows the two reference keyframes with a tight grid window which is 14. We have generated 10 IB-frames between these two keyframes which are shown in the following Figure. In Figure 6.4 the first and last images, marked by a red border, show the aforementioned source and target keyframes, without showing the grid. The rest of the images are the 10 IB-frames produced between those two keyframes. The two strokes we wanted to focus on are marked by red and blue dashed circles. The longer stroke, marked with a red circle, is almost at the same place in both source and target keyframes. Therefore, we expect this stroke to remain relatively the same, throughout the generated IB-frames. However, following the red circle across the IB-frames shows that this stroke undergoes unnecessary movements and deformations. The shorter stroke, marked with a blue dashed circle, disappears in the second keyframe. Therefore, we expect it to fade out throughout the IB-frames, without changing its original shape. However, following the blue circles show that, this stroke also, undergoes some deformation, before completely fading out. We have not marked these two strokes in all of the frames, to avoid excess clutter and keep the images cleaner.



(a) (b)

Figure 6.3. ***(a) and (b) are respectively source and target keyframes showing the superimposed grid windows. Despite being relatively close, due to tightness of the imposed grid, the long strokes do not fall into the same grid cell, and therefore do not get mapped to each other.***

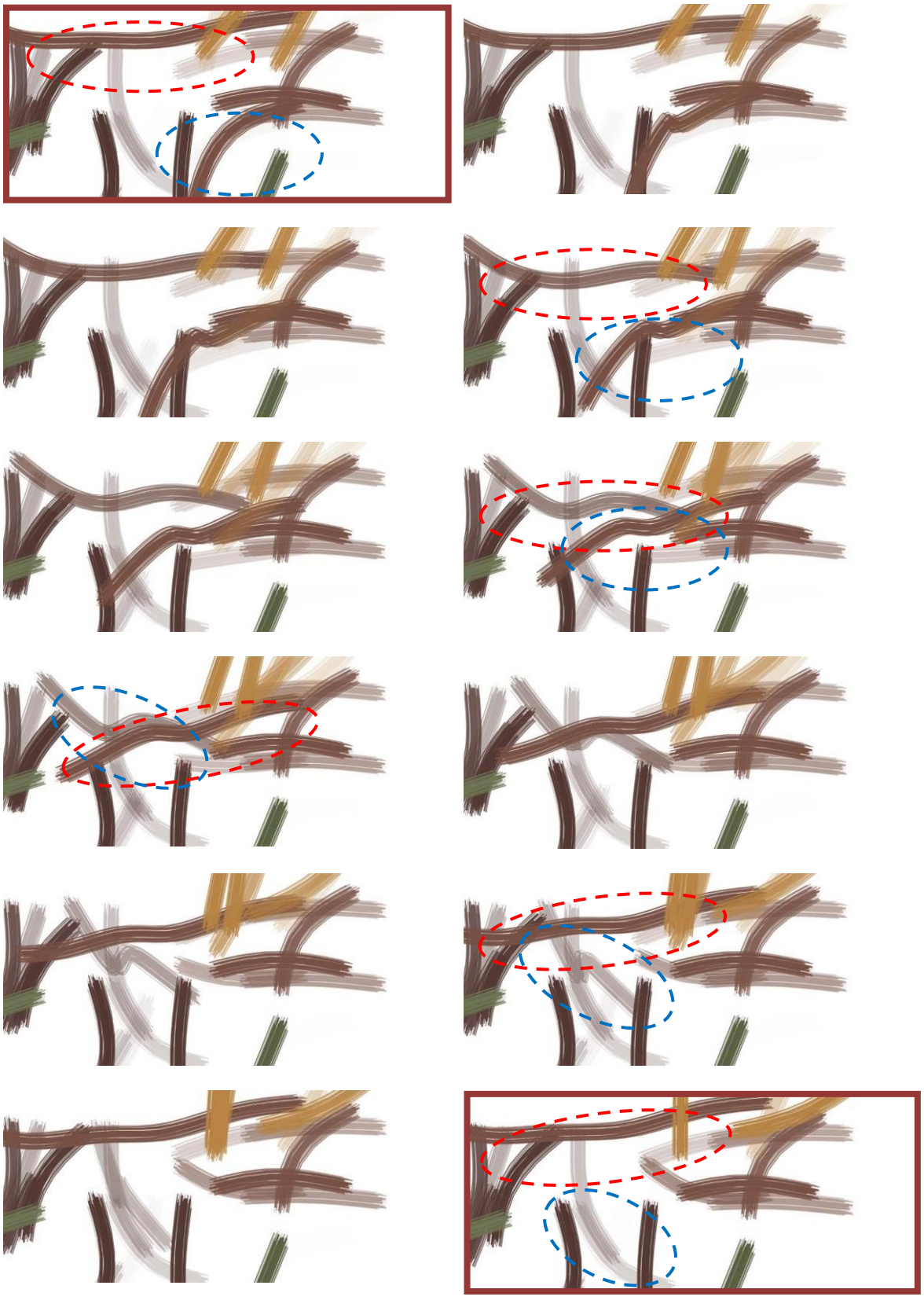


Figure 6.4. Improper grid window size has caused unnecessary transformations of the strokes. The first and last images marked by a red border are the source and target key frames. The rest are all generated IB-frames between these two keyframes. Note the behavior of strokes marked by the red and blue dashed circles.

The following is a URL to the video of this transformation, to better illustrate the impact of tight grid window size.

- **Sequence #6.1-TightGridWindowSize**

Loose Grid Window Size

An unexpected phenomenon that can occur when the grid window size is bigger than the proper range is what we call "*migrating strokes*". A grid window is *loose* when the grid size is relatively bigger that the average Euclidian distance of two farthest source and target strokes. A loose grid window increases the size of Eligibility-Set (See Section 4.3.2) and as a result, increases the number of eligible candidate strokes for mapping. This fact by itself would increase the time complexity of the algorithm, by increasing the search and analysis time, happening over the Eligibility-Set. Apart from time complexity, enlarging the scope of eligible strokes will increase the chance of strokes for getting mapped. But it also will increase the chances of non-optimal mappings between strokes. In particular, the phenomenon can appear when semantic regions are distributed discretely across the keyframe (Figure 6.5). Theoretically, any two strokes from the same semantic areas would be eligible for mapping on the first level. However, not any mapping between the strokes of a region makes sense visually. In this particular example, mapping a stroke from the beard area to the moustache area is allowed (all the strokes belong to the same semantic region which is 'hair') but not optimal.

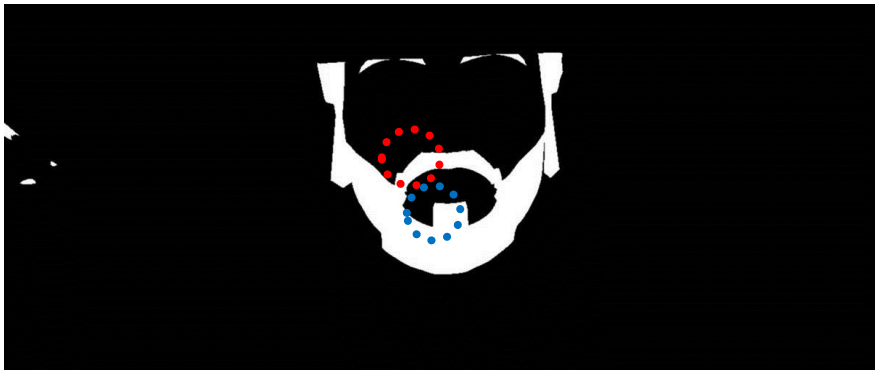
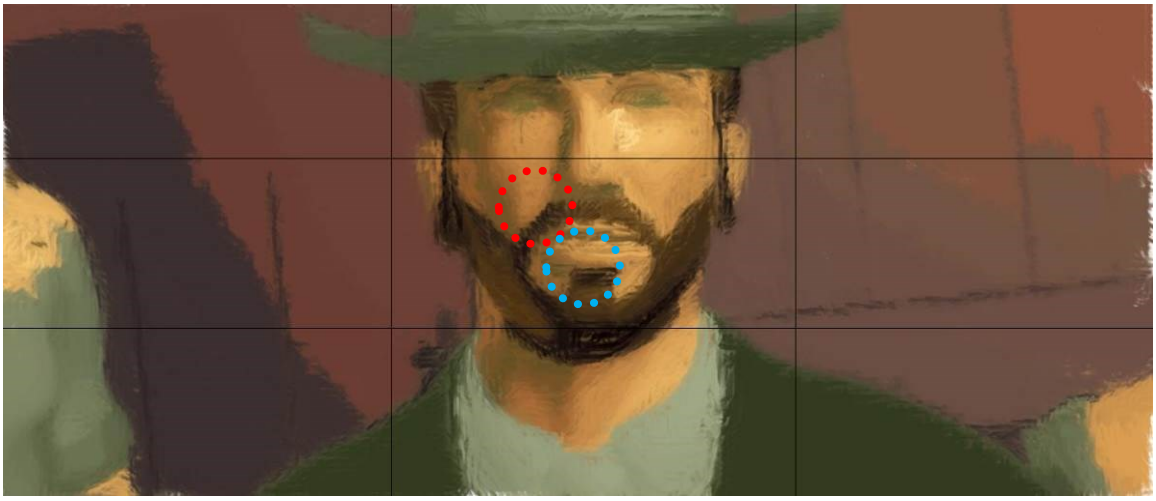


Figure 6.5. Two marked areas belong to the same semantic region, but they are located discretely across the image. If a stroke from the red circle gets mapped to a stroke from the blue circle (or vice versa), despite being allowed, the mapping will not be an optimal one.

Such non-optimal mapping might cause the strokes to leave their local eligible region in the transformation process and 'migrate' to another non-local area, from the same semantic region. This undesired movement takes away from the coherency of transformation. Figure 6.6 illustrates the source and target keyframes with a loose grid window. While the suitable grid window range for these two keyframes is [9-12], the superimposed grid has the value of 3. Note that the areas marked by the red and blue dashed circles remain in the same grid window in both source and target keyframes. As a result, any stroke from the red circle area is eligible to get mapped to a stroke from the blue circle area, if both belong to the hair region. However, such mapping would not be optimal. Figure 6.7 demonstrates how some small strokes move across the lip area, going from the beard to the moustache of the character due to the non-optimal mapping.

This phenomenon is more apparent in frames with smaller strokes, in which the size of the grid window is significantly bigger than the maximum distance between the two farthest brush strokes; it is also more apparent when semantic regions are distributed discretely throughout the frame. For these reasons, the example set chosen to illustrate this undesired visual artifact is finer in rendering and has not been cropped or zoomed. This way, the movement of the migrating strokes can be easily followed.



(a)

(b)

Figure 6.6. ***(a) and (b) are respectively the source and target keyframes used for generating the IB-frames of Figure 6-7. Note the big size of the grid window compared to the size of the strokes. The red and the blue circles belong to the same semantic region of hair and also remain in the same grid window in both (a) and (b).***

Figure 6.7. ***A series of 5 IB-frames generated in between keyframes of Figure 6-6. Note the migrating strokes moving from the bottom left side of the character's beard to the top left side of the moustache, throughout these frames.***

Below is URL to the video of this transformation, to better depict the effect of loose grid window on the output results.

- **Sequence #6.2-ThirdScene-MigratingStrokes**

Suitable Grid Window Size

A suitable value range for the grid size parameter keeps the scope of eligible candidate strokes at an efficient size. This means that it neither reduces the chances of

strokes to get mapped, nor does it allow for a high number of non-efficient mappings. In other words, it optimizes the mapping process and keeps the unnecessary movements and transformations to a minimum.

Figure 6.8 and Figure 6.9 are based on the same set of keyframes studied in Figure 6.3 and Figure 6.4. This time, we will investigate the behavior of the same two strokes over time and through successive generated IB-frames, using a suitable grid window value. Figure 6.8 demonstrates source and target keyframes marked with this grid window size, while Figure 6.9 show the 10 generated IB-frames between these two keyframes, using the new grid window size. The frames marked by a red border show the source and target keyframes. The blue dashed circle marks the stroke, which remains almost unchanged in the source and target keyframes. Ideally, we expect it to stay relatively unchanged throughout the generated IB-frames as well. Following the blue circles, we see that this stroke remains almost the same and does not move or change its shape during the transformation. Similarly, the stroke marked with a red circle is the one which changes significantly from the source keyframe to the target – since it disappears. Therefore, we expect to see a smooth transformation to gradually change its source state to the target state. Following the behavior of this stroke through IB-frames shows that the desired transformation and movement are obtained for this stroke.

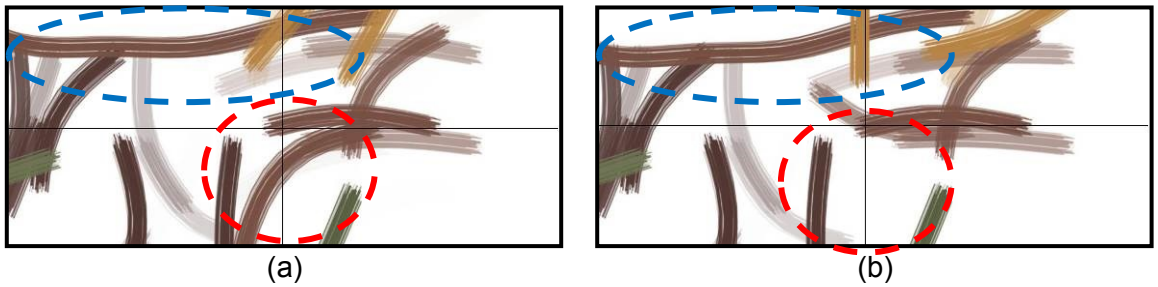
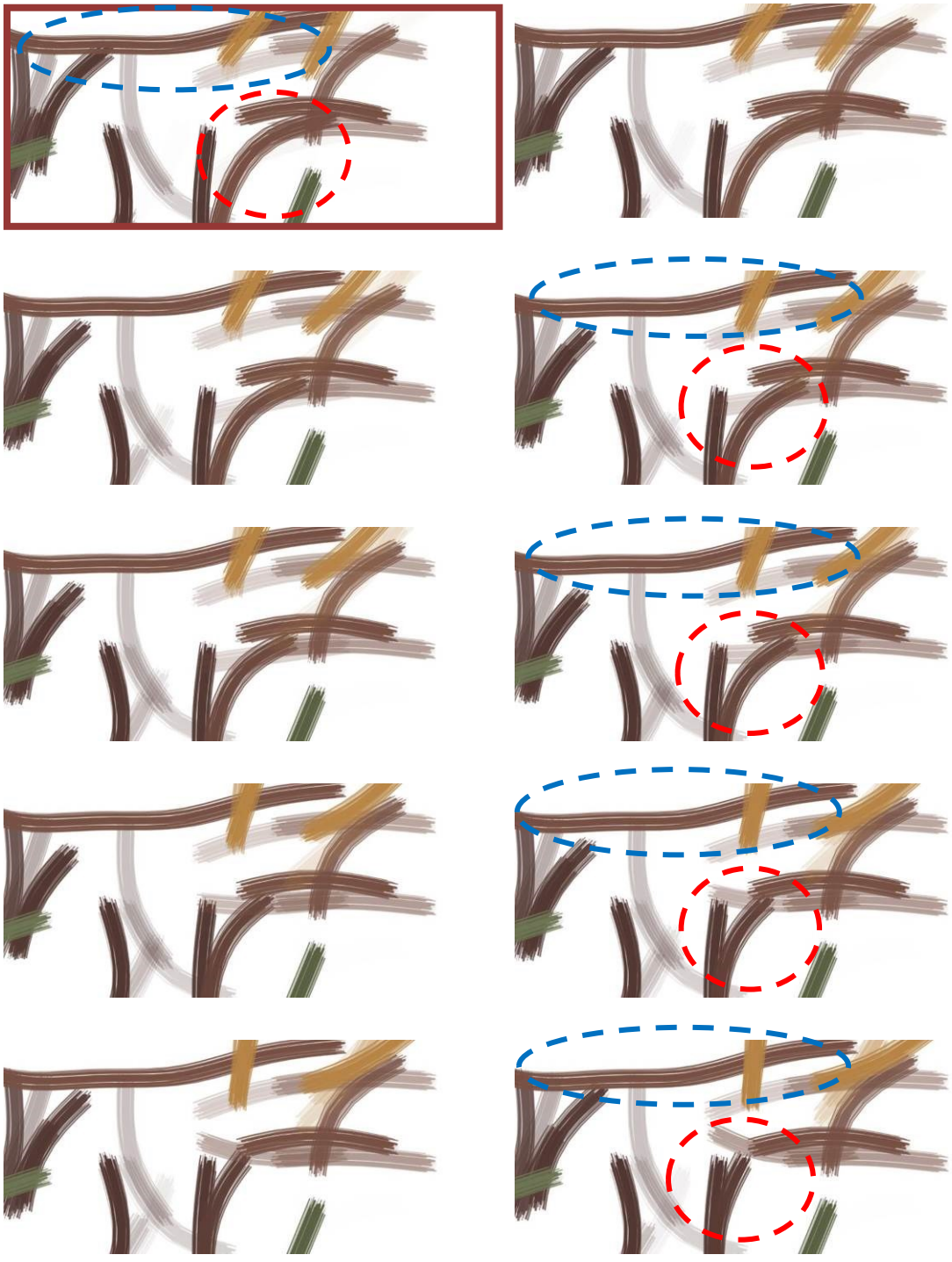


Figure 6.8. ***(a) and (b) are respectively source and target keyframes with the superimposed grid window. Note the position and shape of the marked strokes in both source and target keyframes and how the blue marked stroke remains almost the same from the source to the target keyframe while the red marked stroke disappears in the target keyframe.***



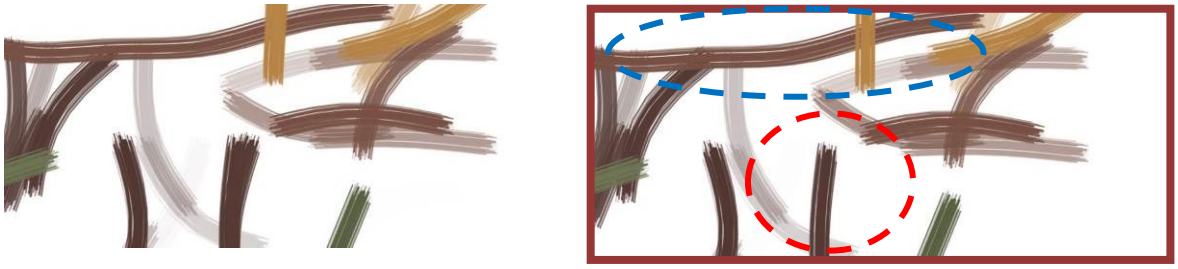


Figure 6.9. ***Choosing a proper grid window scale has optimized the mapping. The first and last images marked by a red border are the source and target key frames. The rest are all generated IB-frames between these two keyframes. Note the behavior of strokes marked by the red and blue dashed circles. The blue marked stroke remains almost the same while the red marked strokes transforms from its source state to its target state as expected.***

Following is a URL to a video sequence for better demonstrating the effect of grid window parameter, when chosen in a suitable range.

- **Sequence #6.3 - SuitableGridWindowSize**

6.1.3. "Thinning-Out Issue"

Increasing the number of passes in *Painterly* rendered keyframes, eventually causes a drop in the quality of the IB-frames generated by CPA. We call this phenomenon the *"thinning out"* issue. This issue is the result of CPA's less-than-optimal communication method with *Painterly*, which is an incomplete part of current CPA. We have suggested a solution for this problem in 7.1. *Painterly*'s Concerns component modifies the PaintActions directly, based on *Painterly*'s internal/external concerns (See Section 3.3.1). In this case, by increasing the number of passes in rendering the keyframes, the overall number of accumulated strokes in each semantic region also increases, considering the fact that all the strokes from different passes are piled up on top of each other, across the frame. This, consequently, causes the strokes to overlap with each other and cover up previously rendered strokes. CPA keeps the number of passes and their corresponding strokes in the generated IB-frames the same as the number of passes and strokes in the original *Painterly* rendered keyframes. Ultimately, the generated IB-frames are rendered with *Painterly*. Because of *Painterly*'s updated global view at each point of the process, *Concerns* will modify each stroke of the

rendering IB-frame and its style-properties based on that global view. Therefore, not all generated strokes in the CPA generated IB-frames end up getting rendered onto the canvas, and not all get rendered with the same style-properties, planned and assigned to them by CPA. In other words, some brush strokes might get omitted or trimmed and some might change their color or opacity values by the time they are rendered onto the canvas. This issue is easier to observe when the strokes are sparsely distributed across their corresponding passes and brush strokes can be singled out. Figure 6.10 and Figure 6.11 better demonstrates this phenomenon. Figure 6.10 shows the source and target keyframes which are used to generate IB-frames, shown in Figure 6.11. These keyframes are originally rendered with 6 passes of coarse-to-fine brush strokes. These frames are generated with a sparse stroke distribution in their passes, by lowering the density parameter in *Painterly*'s parameter values (See Section 3.3.1), and coarse brush strokes so that their movements are easier to follow. Figure 6.11 illustrates the source and target keyframes – marked by a red border – together with the two generated IB-frames between the aforementioned keyframes. Note that how the strokes in the marked areas do not get rendered onto the final frame.

(a)

(b)

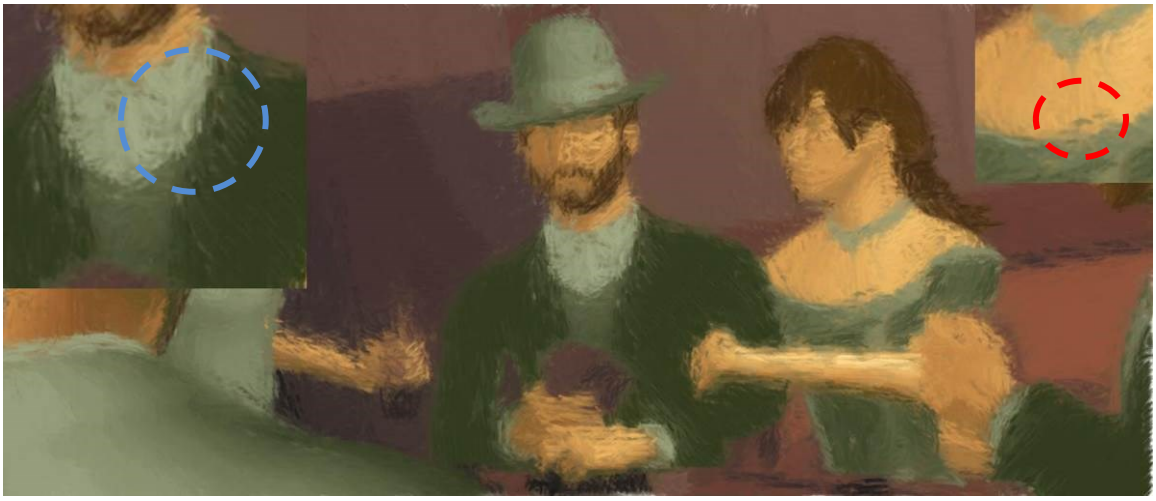
Figure 6.10. ***(a) and (b) are respectively the source and target keyframes, used for generating the IB-frames of Figure 6-11. These keyframes are rendered with 6 passes of coarse-to-fine brush strokes.***

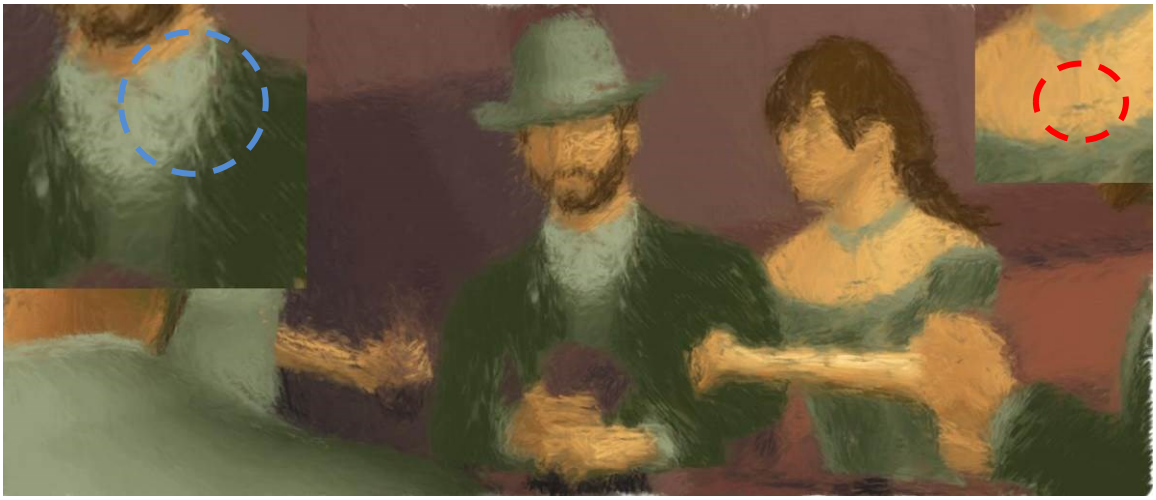
Figure 6.11. These IB-frames show the 'thinning-out' issue. The images marked by a red border are the source and target keyframes, shown in Figure 6-10 and the rest are 2 generated IB-frames between those keyframes. Note how the zoomed areas of the frames are thinning out in the number of rendered strokes, through the IB-frames.

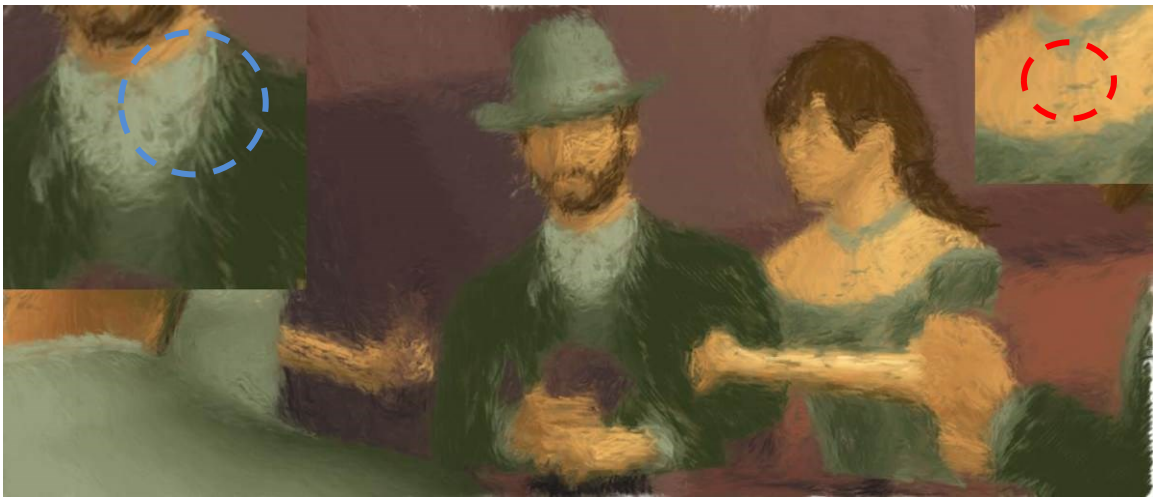
Below is a URL to a video sequence to better show this undesirable visual artifact.

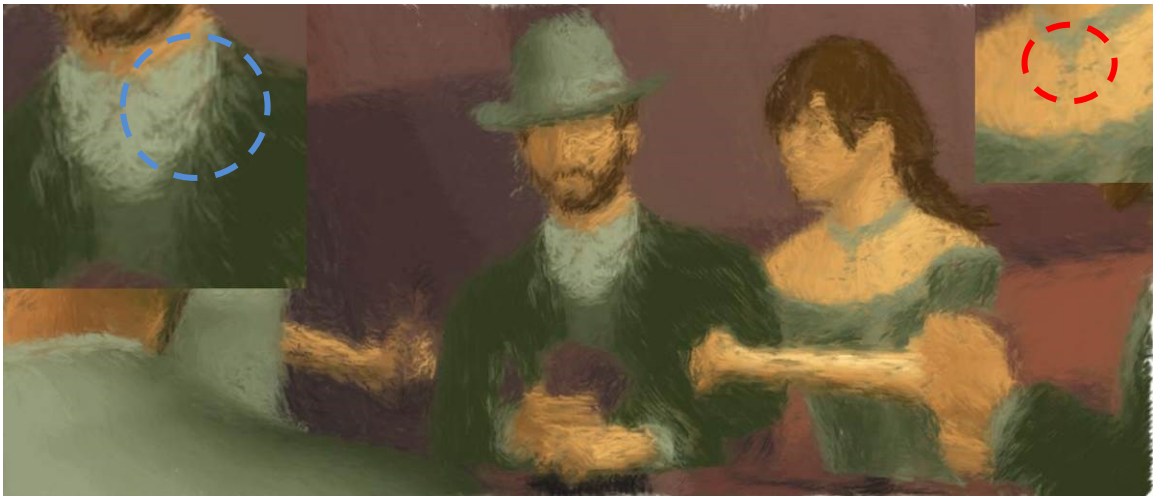
- **Sequence #6.4-ThirdScene-ThinningOut**

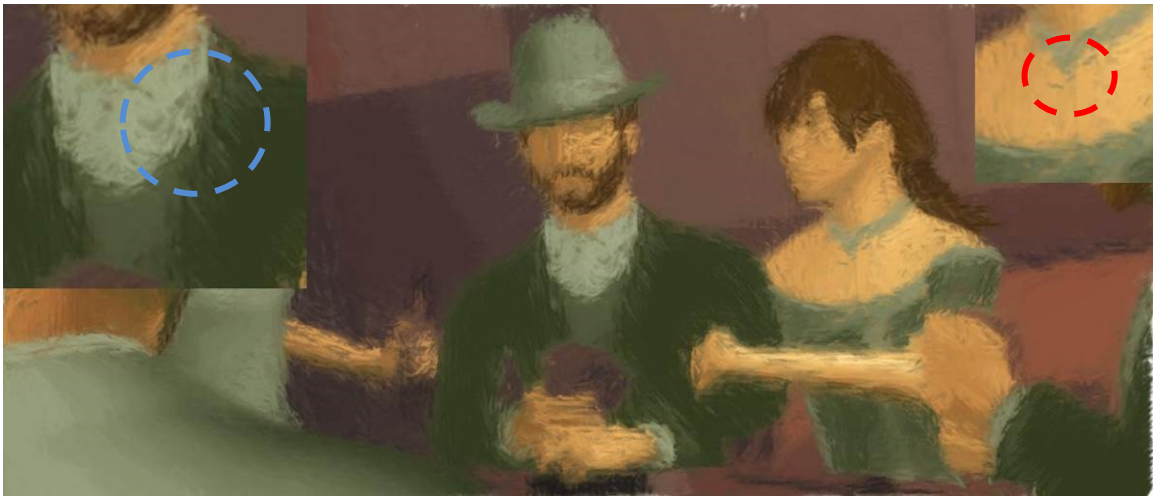
Figure 6.12 shows both “migrating strokes” and “thinning out” problem, in another set of 6 generated IB-frame from a different scene. The source and target keyframes are not shown in this series. The red marked areas show the migrating strokes while the blue marked areas demonstrate the thinning out issue throughout these IB-frames.











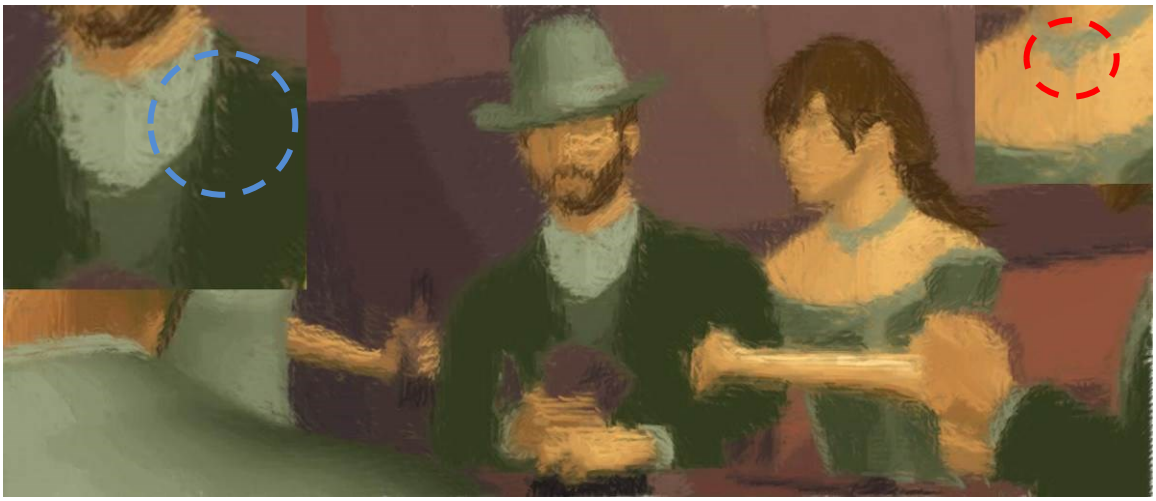


Figure 6.12. 6 generated IB-frames, selected from a different scene. This series of generated IB-frames illustrate 'thinning out' and 'migrating strokes' issues. The red circle shows the movement of the migrating strokes while the blue marked area shows the thinning out issue

Following is a URL to a video sequence of the above stills, demonstrating these two undesirable visual artifacts.

- **Sequence #6.5-FirstScene-MigratingStrokes_ThinningOutIssue**

At the end, all the above instructions are given towards a better understanding of CPA's parameters' range and default settings to facilitate any possible future enhancement via AI techniques or heuristic methods.

6.1.4. General Aesthetic Effects of CPA's Open-Parameters on the Final Results

In the latter subsections, we reviewed CPA's open-parameters and their effect on the generated results. These parameters were isolated and tested in a rather goal-oriented way, to investigate the effect of each single parameter on the look of the generated frame and the overall cohesiveness of the final result. However, we should mention that flickering is not a totally undesired artifact, since people are used to perceive some specific amount and type of flickering in the traditional hand drawn painterly animation. Therefore, the flickering artifact can even make the result of a computer NPR more plausible to the eyes of human viewers. In fact, the purpose of CPA's parameters is to control the flickering – as well as other visual artifacts – rather than to eliminate it. As a result, flickering and incohesiveness can also be controlled and used as a new aesthetic dimension in the computational NPR animation.

CPA's parameters provide the user with an overall control over the generated artifacts which were reviewed in the latter sub-sections. These artifacts can be controlled to achieve a more CGI look or modified and set to create a hand-drawn looking result, or even more abstract and new styles. The interpolation-style is a parameter which controls and dominates the whole interpolation process, and therefore is the channel to dictate the users preferred style in the frame synthesis process.

6.2. Examples of System Output using Real-World Input Data

In this Section we present and discuss system output results obtained by applying the current version of CPA, and our best known usage practices, to real, industry-sourced animation data. We created fine-tuned painterly rendered keyframes with *Painterly* and also tweaked CPA's parameters to create the best possible results to show the capability of CPA in generating higher quality and cognitive-based animation/video sequences. These results can be qualitatively compared to other NPR animation techniques, for example our earlier pixel-based interpolation technique (which we also present for comparison). In this manner we provide some informal evidence that the CPA system as a whole has made advances toward the stated goals of a more temporally coherent, cognitively motivated animated NPR output. By making these sample videos as well as the CPA source code publicly available, we allow the community to judge the merit of the current CPA system performance, and potentially to validate system performance on a broader selection of input data.

The example animation sequences were obtained from 3D pre-visualization design firm, Twenty One Inc., with whom we collaborated on an NSERC Engage project (See Section1.2). We selected 3 different sequences of the source animation. These sequences are referred to as First Scene, Second Scene and Third Scene; we chose these three scenes based on their different characteristics. More detail about these scenes and their specifications is given under the following sub-sections.

Shown below are 3 original animation frames (prior to rendering via CPA) which provide representative examples of the 3 source animation sequences, respectively (Figure 6.13).

Figure 6.13. TOI Collaboration stills, selected from three different scenes of the reference animation sequence. These three scenes were used as inputs of CPA. Stills respectively belong to the First Scene (top), Second Scene (middle) and Third Scene (bottom).

Each set of frames from the input scenes was keyframed using a uniform spacing of 3 regular frames between each two successive keyframes. Semantic region maps of these frames were automatically created by the TOI Inc.'s 3D authoring tool - Autodesk Maya. Keyframe sets and their corresponding semantic region maps were used to create elaborate and refined looking images, by using *Painterly* with manually chosen painting script parameters. *Painterly*'s outputted XML stroke log scripts were then used by CPA, to generate a set of middle IB-frames. Five IB-frames were generated between each pair of successive keyframes. The entire frame set, consisting of the original renders and the synthesized IB-frames, was stitched together to create a QuickTime movie sequence (See Section 5.7 for more information on the stitching method).

We generated 5 IB-frames between each two consecutive keyframes, while the original frame count between keyframes was 3. We did that to be able to smooth the interpolation and also be able to better monitor the course of transformation. As a result, we needed to have a higher frame rate (i.e. higher than normal which is 24 fps) to keep the speed of the sequence in a normal and plausible range. The final frame rate was chosen manually and independently for every batch based on test-runs.

6.2.1. Preparing Fine-Tuned Sequences

To create our fine-tuned output sequences, we performed a number of test-runs to design scripted painterly rendering parameters for use by *Painterly.* Figure 6.14 illustrates the look of the rendering style we arrived at for use in the first 3 CPA video results presented below. Note that a common color palette was used, to create a more unified look among all the scenes. Two other sets of rendering parameters (not shown here) were designed for the FirstScene and SecondScene, and applied to the final 2 CPA videos presented in Section 6.2.5.

Figure 6.14. Top, middle and bottom images are respectively taken from First Scene, Second Scene, and Third Scene batches. All three batches are Painterly rendered using the same color palette to create a more unified look.

6.2.2. CPA Video Result #1 - First Scene

Video description

In First Scene, aside from one moving character, a large portion of each frame, especially in the background region, remains still throughout the sequence. The reason for choosing this scene was to test how well CPA performs on input combining static and dynamic areas. We wanted to achieve a temporally cohesive result on both areas.

Frames of this sequence contain 4 passes, and a stroke count averaging approximately 76000 on the finest passes (e.g. for skin and clothes). The IB-frames were generated with 10/1 position color ratio (i.e. position weighted 10 times more than color) and grid size values of 1, 0.5, 0.1 and 0.1. The rendering frame rate was 40 fps.

Video Link

- **Sequence #6.6-FirstScene-CPA-FineTuned-p10_c1-fr40**

6.2.3. CPA Video Result #2 - Second Scene

Video Description

Second Scene was chosen mostly because it contains a character moving into the scene from outside. We wanted to see how CPA would perform with the sudden appearance of an object which wasn't previously there.

This sequence was rendered with 4 passes in total, with an average stroke count of approximately 60000 in the finest passes (e.g. for skin and clothes). IB-frames were generated via a 9/1 position-color ratio and grid size values of 1, 0.5, 0.1 and 0.1. The sequence was made at a rate of 30 fps.

Video Link

- **Sequence #6.7–SecondScene-CPA-FineTuned-p9_c1-fr30**

6.2.4. CPA Video Result #3 - Third Scene

Video Description

In the Third Scene sequence, the movement happens in the form of a camera zooming into the character's face, so all areas of the image are moving constantly and steadily. The reason for choosing this scene was to test CPA's performance with a set of frames with constantly moving contents.

The sequence was rendered with 4 passes and an average stroke count of 76000 in the finest passes (e.g. for skin and eyes). The generated IB-frames used a 10/2 position-color ratio and grid window values of 1, 0.6, 0.2 and 0.1. The sequence was made at 35 fps.

Video Link

- **Sequence #6.8–ThirdScene-CPA-FineTuned-p10_c2-fr35**

6.2.5. CPA Video Result #4, #5, and #6 – Alternate Styles for First Scene

Here we present one more outputs generated from the First Scene batch of frames. This sequence was rendered by *Painterly* using different parameter settings and the OriginalPalette color palette - different from the previous test sequences.

Video Description

This video is generated using the FirstScene keyframe set. The frames are rendered using 3 passes, and an average number of approximately 55000 brush strokes in the finest pass (eyes). The sequence is generated with a position-color ration of 10/2, grid window size of 2, 1 and 0.1, and frame rate of 40 fps.

Video Link

- **Sequence #6.9–FirstScene-CPA-OriginalPalette-p10_c2-fr40**

Comparative Videos

The following video sequences are generated by stacking sequence #1.3 and #6.9 to present a better comparative view of CPA's performance on reducing the amount of flickering in the reference video sequence.

Video Link

- **Sequence #6.10-FirstScene-OriginalPalette_BiggerStrokes/CPA-OriginalPalette-p10_c2-fr40**

Video Description

The above sequence demonstrates a full view of the Sequence #1.3 followed by video sequence #6.9

Video Link

- **Sequence #6.11-FirstScene--OriginalPalette_BiggerStrokes/CPA-OriginalPalette-p10_c2-fr40[OneHeadShot]**

Video Description

The above sequence demonstrates a close up of the woman character's head from sequence #1.3 followed by the same head cut from sequence #6.9.

Video Link

- **Sequence #6.11-FirstScene--OriginalPalette_BiggerStrokes/CPA-OriginalPalette-p10_c2-fr40[TwoHeadShots]**

Video Description

- The above sequence illustrates a close up of the man and woman characters' heads from sequence #1.3 followed by the same head cut from sequence #6.9.

6.2.6. Results Discussion

In the accompanying videos, we demonstrated CPA's performance on the materials (computer animation sequence with automatically generated region maps) accumulated from our NSERC Engage Collaboration with Twenty One Inc. We evaluate CPA's performance qualitatively by generating sample sequences from three selected scenes of the aforementioned TOI Collaboration video sequence. Generated sequences are provided on the author's website[14], and also discussed here. The videos are labeled by the Chapter and video number. We should mention that the version of *Painterly* which we used for generating the old TOI results – which we delivered to the company at the time - is different from the current version of this toolkit. For generating the results and movie sequences of this thesis work, we have used the newer version of *Painterly* toolkit. *Painterly* has moved from CIELAB JCh color space to CIECAM02, in its current version. We have discussed the merits of using this new color space in Section 4.3.2. Some more modifications have also been done on the newer version of *Painterly*, regarding XML scripting and parameters set. However, these updates have not affected the aesthetic qualities of the generated results; since they have been mostly architectural updates to make *Painterly* more robust. With all that been said, it was impossible for us to recreate the exact old TOI Collaboration results with the new version

[14] http://ivizlab.sfu.ca/research/PainterlyAnimThesis/

of *Painterly*, or re-use those old results in CPA. Therefore, we needed to re-render the selected scenes' frames (FirstScene, SecondScene, and ThirdScene) with the new version of *Painterly* and use them in CPA to generate our results, which have been presented in this thesis work. Since re-rendering an exact replication of the old TOI Collaboration keyframes was technically impossible, we *Painterly* rendered the keyframe in the most similar style and color palette.

Apart from the keyframe-set, we have compared CPA's results with the old pixel-based interpolation method – which we used to generate TOI Collaboration results, delivered to the company. The following points are overall comparisons between CPA's and previous pixel-based interpolation method's results and performance.

- The previous pixel-based interpolation method as discussed in Section 1.2.1 was not 'smart'. It was unable to acquire or perceive any knowledge from scene contents (e.g. semantic regions of the frames) and/or utilize it in the process of generating in-between frames. Whereas, CPA leverages *Painterly*'s knowledge space in the process of frame synthesis.
- As a result, the generated IB-frames are real '*painted frames*', equivalent of any other frame which is rendered by *Painterly*. This means that CPA generated IB-frames have real stroke and other structural components of a painterly rendered. → say more
- CPA system has fairly good edge preservation. Whereas due to the nature of morphing process, the pixel based algorithm made blurry edges and contours (See Section 1.2.1).
- The previous pixel-based interpolation method was a simple linear method which could only generate on type of interpolation style, whereas, CPA supports more different interpolation styles. Through its parameters, the user can author their style preferences to CPA. This will eventually be fully affecting the look and style of the generated in-between frames.
- CPA is significantly slower than the previous method, in the overall frame synthesis procedure. Depending on the complexity of the keyframes (the number of passes and strokes in each pass), CPA requires 10 minutes to 5 hours per frame, on a modern desktop, with CPU between 3.4 and 3.8 GHz. Apart from implementation characteristics, CPA's low speed is due to the amount of computational work that is done for generating the IB-frames, since, as mentioned before, each of the newly generated IB-frames are real painted frames, equal to any other *Painterly* rendered frame, with the necessary structural components. Whereas the pixel-based interpolation method generated only morphing images between two keyframe which were merely image files without any structural build. Nonetheless, CPA's 'temporal performance' is somewhat similar to some of the previous systems from other research works. For instance, Hertzmann's system (Hertzmann & Perlin,

2000) took 3-4 hours to render one single frame, using a 3.4 GHz CPU. Completing longer sequences took weeks to complete. We believe that CPA (and for that matter *Painterly*) can eventually be sped up by implementing more of the code based to GPU acceleration.

The newly rendered keyframe-set differ from the old TOI Collaboration keyframe-set, in the degree of finesse. The new set is more elaborate and finer regarding the stroke sizes, especially in the regions of eyes and skin. Accordingly, brush strokes are harder to distinguish in #1, #2 and #3 sequences. However in sequence #2, there are some strokes around the *characters neck and collar* which are more distinguishable. Following the behavior of these strokes shows a fair amount of temporal cohesiveness throughout the generated IB-frames. Also, in sequence #3, the eyes of the character move fairly cohesive and the shades on the skin keep the temporal coherence. In sequence #4, which looks similar to one of the old pixel-based method's result (**Sequence #1.3-FirstScene-OriginalPalette_BiggerStrokes**) (See Section 1.2), strokes are more distinguishable. The sequence shows that the movement of the blobs is fairly coherent across the generated frames, in comparison to the older pixel-based-interpolated sequence. There are some visual imperfection and noise occurring around the edges of the characters, which do flicker to some extent in the in-between frames. But overall, the edge preservation is far better than the pixel-based method.

While evaluating and comparing the interpolation methods themselves is possible, evaluating the generated painterly rendered movie sequences is a nontrivial process, since painterly movie sequences are forms of art, at some point Noted NPR scientists Hall and Lehmann argue NPR artistic results "experiments are at best difficult to design, and even the Turing test is of limited value because we are not asking whether a piece has been produced by a human but whether it possesses artistic merit regardless of its source" (Hall & Lehmann, 2013). Therefore, presenting a method for evaluating the resulted sequences remains elusive.

7. Conclusion and Future Works

In this Chapter, we conclude this thesis by articulating the contributions and limitations of the current thesis work, providing some suggestions on the directions which can be taken in the future and giving a short summary of the entire work at the end.

7.1. Contributions

Following the main design goals of this thesis work, articulated in Section 4.1, this thesis work have fulfilled them to a full extent, while making several contributions to the body of NPR animation field. Our Design for CPA is:

1. A novel system architecture for generating animated NPR. Through carefully designed two-way communication between *Painterly* and the other components of CPA, it maintains and extends key capabilities of *Painterly* which are:
   a. The capability of rendering painterly animation/movie sequences based on a cognitively-informed model of human artistic practise.
   b. The incorporation of high-level knowledge such as image region semantics, allowing for future elaboration of computational intelligence in the system.
   c. Allowing the users for creatively explore different styles of output through a parametric design.
   ►This will allow for controlling and using of visual artifacts as another dimension in the aesthetics of the generated results – as well as the overall body of NPR research; since controlling the flickering as a visual artifact and using it as a possible aesthetic factor is a goal of computer generated NPR animation.
2. A novel algorithmic approach, based on keyframing and mapping/interpolation of strokes, for extending a still-based and stateless NPR system to a stateful and animation-inclusive one, while solving a number of key issues as follows:

a. Compared to a "naive"/"un-informed" approach to generating animation (rendering the frames independently and without knowledge of previous or future frames' configuration) our approach limits the amount of undesired visual flickering and enforces temporal coherency in the resulted sequences.

b. Compared to a pixel-based interpolation technique, our approach preserves the identity of strokes and thus each frame maintains a plausible "painted" look.

c. Our approach creates an interesting aesthetic interpretation of an "animated painting" by employing strokes which move, flow and transform smoothly over time

►This elevation that CPA has brought to *Painterly* certainly will expand the domain and scope of *Painterly, as* a research tool, in the interdisciplinary area of human perception, computational art and science. For instance, time-based animation/movie sequences that are created by *Painterly* and CPA can be used in 2 ongoing iVizLab researches. Firstly, they can be used in emotional control of game and entertainment sequences. Secondly, they can be used in the lab's research work in face to face autism communication. With knowledge gleans from passed down art techniques, CPA's results can be used to guide the viewer's eye, in order to better understand a face to face conversation and communication ("iVizLab - Simon Fraser University," 2013; H. Seifi et al., 2011; H. Seifi, DiPaola, & Enns, 2012; Hasti Seifi, 2010)

3. A flexible system architecture which can incorporate and/or synergically work with alternative still-frame-rendering modules (other than *Painterly*) in future investigations.

CPA also delivers a number of side contributions to the NPR community and other research communities.

4. Implementation of CPA together with its open source and publicly available source code, allows future research to further validate and improve on the design.
5. Our work also makes a potential contribution to research in psychology and cognitive science regarding human perception of art and visual stimuli.
6. Building on *Painterly*'s previous use in still-image-oriented perception research, CPA can now also be used in animation-oriented perception research.

7.2. Limitations

There are some limitations to the design and performance of CPA as well which are listed briefly as follows:

1. Current implementation of CPA is not optimal in terms of time and memory efficiency. It has yet lots of room to enhance.
2. Current system design of CPA creates a "thinning out artifact" due to *Painterly*'s re-application and authoring of '*Concerns*' rule-set, on interpolated stroke data (this drawback could be corrected through future development).
3. CPA requires the users to have a detailed understanding of the provided painterly rendering parameters; both *Painterly*'s parameters and CPA's parameters. Therefore it is not intended for ordinary users.
4. Setting the CPA's open-parameters, which are mostly content-based, depends on the user experience and requires performing a number of test-runs. This shortcoming also can be fixed through future works and further automations.

7.3. Future Works and Extensions to CPA

There are a number of improvements that can be performed to the design model or internal algorithms of CPA or on the *Painterly*-CPA interaction model to enhance and optimize CPA's performance, as well as CPA and *Painterly*'s synergy. This will consequently improve CPA's generated results and its overall user interaction experience.

Certainly, not having a user-friendly UI is a downside for both *Painterly* and CPA. The fact that users should deal with a relatively high number of parameters with different ranges and impacts on the output, makes the whole parameter tweaking phase a baffling process. Although, we should mention that the *Painterly team* is working on the Artificial Intelligence front end, using Evolutionary Systems, Deep Learning, to help the users control and modify *Painterly* parameters easier. This way, the users would be able to generate their desired results without having to have any required knowledge of *Painterly*'s internal processes and parameters.

Painterly and CPA are two individual components which run separately and not collaboratively, as one unit. This makes the user experience even more frustrating. Since, for making a painterly movie sequence, users need to go back and forth from *Painterly* to CPA and back to *Painterly* and finally to CPA again. Improving the workflow and designing a unified and easy-to-use UI is a simple but necessary future step to take.

To find a desirable parameter range, users need to tweak the parameter values in the input scripts, and conduct some test-runs to narrow down the range in both *Painterly* and CPA systems. Dealing with two parametric systems at the same time might make the whole experience rather daunting and frustrating for the user. As a possible future work, we suggest to build a side tool to help users decide on their preferred range of parameters for both systems. Such a tool can provide histograms of the number of passes, semantic regions, and strokes, together with the density of each pass, how the strokes are distributed across the reference canvas, etc. and a preview of the resulting painterly image. This enables users have a better evaluation of the inputs and make a better estimation of the parameters, and as a result, generate their desired outputs much faster and easier.

This thesis work has been focusing on enhancing and maintaining temporal coherency. Therefore, we did not explore different interpolation styles, induced by changing CPA's Interpolation-style parameter. A possible future work can be to expand the styles of our resulting movies/animations by adding a selection of predefined interpolation-styles. Moreover, exploring and examining different values for this parameter can result in new and abstract forms of interpolation and movement in the final animation/movie sequence.

Currently CPA works on CGI based source material, mainly because *Painterly* and CPA work best with known semantic regions (hair, face, bottle, etc.) which can be automatically produced by labeling regions in a CGI 3D model. However, nothing specific in the code precludes using live action video sequences as source. When image processing and pattern recognition techniques exist to label regions of live action video sequences automatically (similar to what we have now with CGI models), *Painterly* and CPA should work fine on live action video sequences. Our lab ("iVizLab - Simon Fraser University," 2013) is interested in using Deep Learning AI techniques to do such

automatic labeling. Already *Painterly* produces strong work on still photography or hand label sequences.

This thesis work concentrated on the temporal incoherency problem. We mentioned in this thesis that the other non-trivial challenge was the Shower Door Effect. We believe that *Painterly* and CPA could be enhanced to improve this problem by understanding the 3D topology nature of a scene from CGI source. And therefore not have strokes drift on the 2D picture plane but feel more affixed to the actual 3D surfaces in a scene. This however is a subject of future work where painterly concerns that have knowledge of depth could be passed on to CPA via the XML script.

The implementation of CPA as we presented in here was a research prototype. It was developed to explore and test our approach to achieve temporal coherency in *Painterly* rendered movie sequences. Therefore, the code can be optimized in several ways to reduce the time and memory complexity of the algorithm and accordingly enhance the performance of CPA. Some of these improvements are as follows:

- Using alternative technologies for reading in / writing to XML files
- Incorporating innovative and heuristic search method in finding the fittest stroke for mapping

In conclusion, this work was an effort in exerting and incorporating the cognitive knowledge model behind the creative process of painting by a human artist in a painterly NPR animation system. Our vision is to create cognitive-based and coherent animation/movie sequences. We incorporated aspects of human's cognitive knowledge in our results by using a cognitive and knowledge-based toolkit (*Painterly*) to process, plan and paint our keyframes and outputted frames. Moreover, we achieved a fair amount of coherency in our results, by controlling the frame synthesis process and making each frame cohesive with the previous and the next frame. We think this work is a beginning point for a much deeper exploration of the cognitive and perceptive aspects of painterly movies.

7.4. Summary

One of the most popular issues in the field of NPR animation is the lack of temporal coherency. *Painterly* rendered sequences tend to have poor temporal coherency due to undesired visual artifacts appearing in the form of unwanted flickering. This issue not only deteriorates the aesthetic quality of the results, but it also takes away from the visual and semantic effectiveness and expressiveness of the movie/animation sequence. Many different techniques and algorithms have been proposed to mitigate this problem. Some of these approaches have successfully reduced the amount of flickering to some extent; however, temporal coherency is still the main focus of many ongoing research works.

Past research has demonstrated that using scene semantics and human's perception knowledge in the painting process enhances the semantic effectiveness of the artistic narrative conveyed through that sequence as well as the visual expressiveness of the piece itself (Colton et al., 2008; DiPaola et al., 2013, 2013; Duke et al., 2003; Halper et al., 2003; Pelachaud & Bilvi, 2003). However, there still is a general lack of work on incorporating such perceptual knowledge in the video stylizing process.

Among the more notable and successful approaches for attending to the flickering issue, only a few research techniques have been taking the road towards a more 'informed' and knowledge-based process. Video segmentations and the use of scene semantics are some of the methods that these researchers have utilized in their works (Agarwala et al., 2004; M. Kagaya et al., 2011; L. Lin et al., 2010; O'Donovan & Hertzmann, 2012). Besides the lack of more research work on incorporating humans' perceptual and cognitive knowledge in the NPR video field, there also is a lack of automation in the majority of the existing literature. Providing the user with a fully interactive system has been so popular that there almost is not a relatively 'smart' and yet automatic system for painterly rendering video sequences. These two issues raise the need for a smart and automated system, which requires the least amount of user manipulation and effort, and/or can be adapted in an industrial movie/animation making pipeline.

Painterly is a knowledge-based and parameterized NPR toolkit, which incorporates humans' cognitive and perceptive painterly knowledge space inside the process of painterly rendering (DiPaola, 2007, 2009). However, *Painterly* is incapable of transmitting previous frames' rendering information to the next ones. Consequently, movie sequences which are rendered by *Painterly*, lack temporal coherency by a great deal. Using a software engineering approach, we proposed and developed CPA, which is a solution to enhance coherency in *Painterly* rendered movie/animation sequences. CPA uses *Painterly* as a base system, to exploit *Painterly*'s well-developed and cognitively inspired algorithms for semantic parsing and hierarchical, blob-based stroke filling, and enforces coherency to the resulting movie sequences. *Painterly* toolkit provides CPA with a painting process-plan in the form of XML scripts which encapsulate aspects of human's cognitive knowledge space. CPA uses these XML files as inputs. The content information of each keyframe and the encapsulated scene semantics, which are provided in these XML scripts, are used by CPA to coherently propagate and interpolate the brush strokes and other painterly elements through generating a number of in-between frames. All the generated in-between frames are eventually rendered by *Painterly*. Finally, all the keyframes and the generated in-between frames are stitched together to make a movie/animation sequence.

Ultimately, the resulting movie sequences generated by CPA were comparable and superior to the results of a low-level and pixel-bases interpolation algorithm[15] regarding the coherency of the piece. Moreover, by successfully incorporating *Painterly*'s knowledge space in CPA's interpolation and frame synthesis process, we created coherent cognitive-based computer movie/animation sequences. As a result, we expanded *Painterly*'s domain and scope and elevated it from a stateless and still-oriented system to a stateful and multipurpose one, without compromising its research-intended goals. Furthermore, we developed a system based on our proposed model, which can be used in the industrial animation/movie production pipeline. Our results were evaluated using a focused group method and through performing comparative evaluations. Lastly, this thesis work is a proof of concept for our proposed solution to

[15] The interpolation algorithm we used to deliver our results for NSERC Engage Collaboration (See Section 1.2).

mitigate temporal coherency issue in *Painterly* rendered animation sequences. Our source code and early comparative results are openly accessible to NPR computer science community and public to provide the means for future reproduction of our system and further studies, evaluations and extensions of our proposed model.

References

Agarwala, A., Hertzmann, A., Salesin, D. H., & Seitz, S. M. (2004). Keyframe-based tracking for rotoscoping and animation. In *ACM SIGGRAPH 2004 Papers* (pp. 584–591). New York, NY, USA: ACM. doi:10.1145/1186562.1015764

Collomosse, J., Kyprianidis, J. E., Wang, T., & Isenberg, T. (2012). State of the "Art": A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*. doi:10.1109/TVCG.2012.160

Collomosse, J. P. (2004). *Higher level techniques for the artistic rendering of images and video* (Doctoral Dissertation). Retrieved from http://epubs.surrey.ac.uk/600592/

Collomosse, J. P., Rowntree, D., & Hall, P. M. (2005). Stroke surfaces: temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics*, *11*(5), 540–549. doi:10.1109/TVCG.2005.85

Collomosse, J.P., & Hall, P. M. (2002). Painterly Rendering using Image Salience. In *Proceedings of the 20th UK conference on Eurographics* (p. 122–). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=787261.787788

Collomosse, John P, Rowntree, D., & Hall, P. M. (2005). Stroke surfaces: temporally coherent artistic animations from video. *IEEE transactions on visualization and computer graphics*, *11*(5), 540–549. doi:10.1109/TVCG.2005.85

Colton, S., Valstar, M. F., & Pantic, M. (2008). Emotionally aware automated portrait painting. In *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts* (pp. 304–311). New York, NY, USA: ACM. doi:10.1145/1413634.1413690

Cornish, D., Rowan, A., & Luebke, D. (2001). View-dependent particles for interactive non-photorealistic rendering. In *No description on Graphics interface 2001* (pp. 151–158). Toronto, Ont., Canada, Canada: Canadian Information Processing Society. Retrieved from http://dl.acm.org/citation.cfm?id=780986.781005

Daniels, E. (1999). Deep canvas in Disney's Tarzan. In *ACM SIGGRAPH 99 Conference abstracts and applications* (p. 200–). New York, NY, USA: ACM. doi:10.1145/311625.312010

DiPaola, S. (2007). Painterly rendered portraits from photographs using a knowledge-based approach. In *In Proc: SPIE Human Vision and Imaging, Int. Society for Optical Engineering, Keynote* (Vol. 6492, pp. 649203–649203–10). doi:10.1117/12.706594

DiPaola, S. (2008). The Trace and the Gaze: Textural Agency in Rembrandt's Late Portraiture from a Vision Science Perspective. *Proceedings of Electronic Imaging & Visual Arts*.

DiPaola, S. (2009). Exploring a parameterised portrait painting space. *International Journal of Arts and Technology*, *2*(1/2), 82. doi:10.1504/IJART.2009.024059

DiPaola, S. (2013). *Exploring the cognitive correlates of artistic practice using a parameterized non-photorealistic toolkit* (Doctoral Dissertation). University Of British Columbia, Vancouver.

DiPaola, S., & Gabora, L. (2009). Incorporating characteristics of human creativity into an evolutionary art algorithm. *Genetic Programming and Evolvable Machines*, *10*(2), 97–110. doi:10.1007/s10710-008-9074-x

DiPaola, S., Riebe, C., & Enns, J. T. (2010). Rembrandt's textural agency: A shared perspective in visual art and science. *Leonardo*, *43*(2), 145–151.

DiPaola, S., Riebe, C., & Enns, J. T. (2013). Following the masters: Portrait viewing and appreciation is guided by selective detail. *Perception*, *42*(6), 608 – 630. doi:10.1068/p7463

Du Buf, H., & Rodrigues, J. (2006). Painterly rendering using human vision. Retrieved from http://w3.ualg.pt/~dalmeida/publicacoes/pub/PainterlyRenderingUsingHumanVision.pdf

Duke, D. j., Barnard, P. j., Halper, N., & Mellin, M. (2003). Rendering and affect. *Computer Graphics Forum*, *22*(3), 359–368. doi:10.1111/1467-8659.00683

Gooch, A. A., Long, J., Ji, L., Estey, A., & Gooch, B. S. (2010). Viewing progress in non-photorealistic rendering through heinlein's lens. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (pp. 165–171). New York, NY, USA: ACM. doi:10.1145/1809939.1809959

Gooch, Amy A., Long, J., Ji, L., Estey, A., & Gooch, B. S. (2010). Viewing progress in non-photorealistic rendering through Heinlein's lens. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (pp. 165–171). New York, NY, USA: ACM. doi:10.1145/1809939.1809959

Gooch, B., & Gooch, A. (2001). *Non-photorealistic rendering* (1st ed.). Natick, MA, USA: A. K. Peters, Ltd.

Haeberli, P. (1990). Paint by numbers: Abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (pp. 207–214). New York, NY, USA: ACM. doi:10.1145/97879.97902

Hall, P., & Lehmann, A. S. (2013). Don't measure—Appreciate! NPR seen through the prism of art history. In P. Rosin & J. P. Collomosse (Eds.), *Image and Video-Based Artistic Stylisation* (pp. 333–351). Springer London. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4471-4519-6_16

Halper, N., Mellin, M., Herrmann, C. S., Linneweber, V., & Strothotte, T. (2003). Psychology and non-photorealistic rendering: The beginning of a beautiful relationship. In G. Szwillus & J. Ziegler (Eds.), *Mensch & Computer 2003* (pp. 277–286). Vieweg+Teubner Verlag. Retrieved from http://link.springer.com/chapter/10.1007/978-3-322-80058-9_28

Haro, A., & Essa, I. (2002). Learning video processing by example. In *16th International Conference on Pattern Recognition, 2002. Proceedings* (Vol. 1, pp. 487–491 vol.1). doi:10.1109/ICPR.2002.1044771

Hays, J., & Essa, I. (2004). Image and video based painterly animation. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering* (pp. 113–120). New York, NY, USA: ACM. doi:10.1145/987657.987676

Hegde, S., Gatzidis, C., & Tian, F. (2013). Painterly rendering techniques: A state-of-the-art review of current approaches. *Computer Animation and Virtual Worlds*, *24*(1), 43–64. doi:10.1002/cav.1435

Hertzmann, A. (1998). Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (pp. 453–460).

Hertzmann, A. (2002). Fast paint texture. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (p. 91–ff). New York, NY, USA: ACM. doi:10.1145/508530.508546

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., & Salesin, D. H. (2001). Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (pp. 327–340). New York, NY, USA: ACM. doi:10.1145/383259.383295

Hertzmann, A., & Perlin, K. (2000). Painterly rendering for video and interaction. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (pp. 7–12).

Isenberg, T. (2013). Evaluating and validating non-photorealistic and illustrative rendering. In P. Rosin & J. P. Collomosse (Eds.), *Image and Video-Based Artistic Stylisation* (Vol. 42, pp. 311–331). London: Springer London. Retrieved from http://www.springerlink.com/index/10.1007/978-1-4471-4519-6_15

Isenberg, Tobias. (2013). Evaluating and Validating Non-photorealistic and Illustrative Rendering. In *Image and Video-Based Artistic Stylisation*. Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4471-4519-6_15

iVizLab - Simon Fraser University. (2013). Retrieved November 1, 2013, from http://dipaola.org/lab/

Kagaya, M., Brendel, W., Deng, Q., Kesterson, T., Todorovic, S., Neill, P. J., & Zhang, E. (2011). Video painting with space-time-varying style parameters. *IEEE Transactions on Visualization and Computer Graphics*, *17*(1), 74–87. doi:10.1109/TVCG.2010.25

Kagaya, Mizuki, Brendel, W., Deng, Q., Kesterson, T., Todorovic, S., Neill, P. J., & Zhang, E. (2011). Video Painting with Space-Time-Varying Style Parameters. *IEEE Transactions on Visualization and Computer Graphics*, *17*(1), 74–87. doi:10.1109/TVCG.2010.25

Klein, A. W., Sloan, P. J., Finkelstein, A., & Cohen, M. F. (2002). Stylized video cubes. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (pp. 15–22). New York, NY, USA: ACM. doi:10.1145/545261.545264

Kyprianidis, J. E., Collomosse, J. P., Wang, T., & Isenberg, T. (2013). State of the “Art”: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, *19*(5), 866–885. doi:10.1109/TVCG.2012.160

Lee, H., Lee, C. H., & Yoon, K. (2009). Motion based painterly rendering. *Computer Graphics Forum*, *28*(4), 1207–1215. doi:10.1111/j.1467-8659.2009.01498.x

Lin, L., Zeng, K., Lv, H., Wang, Y., Xu, Y., & Zhu, S. C. (2010). Painterly animation using video semantics and feature correspondence. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (pp. 73–80). New York, NY, USA: ACM. doi:10.1145/1809939.1809948

Lin, T., Lin, L., & Wang, Q. (2012). Robust stroke-based video animation via layered motion and correspondence. In *Proceedings of the 20th ACM international conference on Multimedia* (pp. 729–732). New York, NY, USA: ACM. doi:10.1145/2393347.2396298

Litwinowicz, P. (1997). Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (pp. 407–414). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. doi:10.1145/258734.258893

Meier, B. J. (1996). Painterly rendering for animation. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (pp. 477–484). New York, NY, USA: ACM. doi:10.1145/237170.237288

O'Donovan, P., & Hertzmann, A. (2012). AniPaint: Interactive painterly animation from video. *IEEE Transactions on Visualization and Computer Graphics*, *18*(3), 475–487. doi:10.1109/TVCG.2011.51

Olsen, S. C., Maxwell, B. A., & Gooch, B. (2005). Interactive vector fields for painterly rendering. In *Proceedings of Graphics Interface 2005* (pp. 241–247). School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society. Retrieved from http://dl.acm.org/citation.cfm?id=1089508.1089548

Park, Y., & Yoon, K. (2008). Painterly animation using motion maps. *Graphical Models*, *70*(1–2), 1–15. doi:10.1016/j.gmod.2007.06.001

Pelachaud, C., & Bilvi, M. (2003). Computational model of believable conversational agents. In M. P. Huget (Ed.), *Communication in Multiagent Systems* (pp. 300–317). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-44972-0_17

Salesin, D. (2002). Non-photorealistic animation & rendering: 7 grand challenges. In *Keynote talk at NPAR*.

Santella, A., & DeCarlo, D. (2002). Abstracted painterly renderings using eye-tracking data. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering* (p. 75–ff). New York, NY, USA: ACM. doi:10.1145/508530.508544

Santella, A., & DeCarlo, D. (2004). Visual interest and NPR: an evaluation and manifesto. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering* (pp. 71–150). New York, NY, USA: ACM. doi:10.1145/987657.987669

Seifi, H., DiPaola, S., & Arya, A. (2011). Expressive animated character sequences using knowledge-based painterly rendering. *Int. J. Comput. Games Technol.*, *2011*, 7:7–7:7. doi:10.1155/2011/164949

Seifi, H., DiPaola, S., & Enns, J. T. (2012). Exploring the effect of color palette in painterly rendered character sequences. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (pp. 89–97). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. Retrieved from http://dl.acm.org/citation.cfm?id=2328888.2328903

Seifi, Hasti. (2010, December 3). *Emotion depiction: expressive character sequences using painterly rendering* (Thesis). Communication, Art & Technology: School of Interactive Arts and Technology. Retrieved from http://summit.sfu.ca/item/11479

Shugrina, M., Betke, M., & Collomosse, J. P. (2006). Empathic painting: Interactive stylization through observed emotional state. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (pp. 87–96). New York, NY, USA: ACM. doi:10.1145/1124728.1124744

Snavely, N., Zitnick, C. L., Kang, S. B., & Cohen, M. (2006). Stylizing 2.5-D video. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (pp. 63–69). New York, NY, USA: ACM. doi:10.1145/1124728.1124739

Vanderhaeghe, D., & Collomosse, J. (2013). Stroke Based Painterly Rendering. In Paul Rosin & J. Collomosse (Eds.), *Image and Video-Based Artistic Stylisation* (pp. 3–21). Springer London. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4471-4519-6_1

Zeng, K., Zhao, M., Xiong, C., & Zhu, S.-C. (2009). From image parsing to painterly rendering. *ACM Trans. Graph.*, *29*(1), 2:1–2:11. doi:10.1145/1640443.1640445

Zhao, M., & Zhu, S.-C. (2011). Portrait painting using active templates. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (pp. 117–124). New York, NY, USA: ACM. doi:10.1145/2024676.2024696

Appendix. *Painterly* Videos

There are 13 numbers of videos enclosed in this thesis work. Below is a list of the aforementioned video sequences, by the order they appear in this thesis and a short description on each video sequence. The video files are available on the author's website. They are also uploaded by this thesis to the Simon Fraser University's Library website.

Twenty One Inc. NSERC Engage Video Sequences (4 Videos)

Below are 4 video URLs to the video sequences of Chapter 1. These 4 video sequences have been rendered by *Painterly* toolkit and delivered to TOI Company, as part of the NSERC Engage Collaboration (See Section 1.2).

Video 1: Sequence #1.1. TOI-FirstScene-OriginalPalette

The above video is taken from the first scene of the original video and rendered with the original color palette with *Painterly* toolkit.

Video 2: Sequence #1.2. TOI-ThirdScene-PurplePalette

The above video is taken from the third scene of the original video and rendered with *Painterly* toolkit, with a purple color palette and a different style.

Video 3: Sequence #1.3. TOI-FirstScene-OriginalPalette_BiggerStrokes

The above video is taken from the first scene of the original video and rendered with the original color palette by *Painterly* toolkit and has big strokes.

Video 4: Sequence #1.4. TOI-FirstScene-OriginalPalette_Blurry

The above video is taken from the first scene of the original video and rendered with the original color palette, using *Painterly* toolkit, but has a blurry look.

Calibration Videos (5 videos)

Below are 5 video URLs to the video sequences of Chapter 6. These 5 videos depict parameter calibration as discussed in Chapter 6.

Video 1: Sequence #6.1. TightGridWindowSize

The above video is illustrating the effect of tight grid window on the output results (See Section 6.1.2).

Video 2: Sequence #6.2. ThirdScene-MigratingStrokes

The above video demonstrates the impact of loose grid window on the output results (See Section 6.1.2).

Video 3: Sequence #6.3. SuitableGridWindowSize

The above video shows the effect of suitable grid window size on the output results (See Section 6.1.2).

Video 4: Sequence #6.4. ThirdScene-ThinningOut

The video above shows the thinning out issue and the impact of higher pass numbers on the output results (See Section 6.1.3).

Video 5: Sequence #6.5. FirstScene-MigratingStrokes_ThinningOutIssue

The video shows both migrating strokes and thinning out issue on the output results (See Section6.1.2).

CPA Fine-Tuned Videos (4 videos)

Below are video URLs to the videos generated by CPA from the video sequence provided by the Twenty One Inc. to show the strength of CPA.

Video 1: Sequence #6.6. FirstScene-CPA-FineTuned-p10_c1-fr40

The above video is taken from the first scene of the original video and rendered with black-brown color palette.

Video 2: Sequence #6.7. SecondScene-CPA-FineTuned-p9_c1-fr30

The above video is taken from the second scene of the original video and rendered with the same black-brown color palette.

Video 3: Sequence #6.8. ThirdScene-CPA-FineTuned-p10_c2-fr35

The above video is taken from the third scene of the original video and rendered with the black-brown color palette.

Video 4: Sequence #6.9. FirstScene-CPA_OriginalPalette-p10_c2-fr40

The above video is taken from the first scene of the original video and rendered with the original color palette.

Video 5: Sequence#6.10-FirstScene-OriginalPalette_BiggerStrokes/ CPA-OriginalPalette-p10_c2-fr40

The above sequence demonstrates a full view of the Sequence #1.3 followed by video sequence #6.9Video Link.

Video 6: Sequence#6.11-FirstScene--OriginalPalette_BiggerStrokes/CPA-OriginalPalette-p10_c2-fr40[OneHeadShot]

The above sequence demonstrates a close up of the woman character's head from sequence #1.3 followed by the same head cut from sequence #6.9.

Video 7: Sequence#6.11-FirstScene--OriginalPalette_BiggerStrokes/CPA-OriginalPalette-p10_c2-fr40[TwoHeadShots]

The above sequence illustrates a close up of the man and woman characters' heads from sequence #1.3 followed by the same head cut from sequence #6.9.