Masaryk University
Faculty of Informatics

Maya2CellVIEW: 3D Package Integrated Tool for Creating Large and Complex Molecular Scenes

Master's Thesis

David Kouřil

Brno, Fall 2016

Masaryk University
Faculty of Informatics

Maya2CellVIEW: 3D Package Integrated Tool for Creating Large and Complex Molecular Scenes

Master's Thesis

David Kouřil

Brno, Fall 2016

Replace this page with a copy of the official signed thesis assignment and the copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

David Kouřil

Advisor: Ivan Viola and Barbora Kozlikova

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

Scientific illustrators communicate the cutting edge of research through their illustrations. There are numerous software tools that assist them with this job. The aim of this thesis is to push abilities of illustrators working on a large scale molecular scenes. This is done by connecting two software packages - Maya and cellVIEW - combining the rendering possibilities of cellVIEW and modeling tools of Maya which results in more effective and efficient workflow.

Keywords

keyword1, keyword2, …

Contents

List of Tables

List of Figures

1 Introduction

[**In what field are we? What are the people in this domain doing?**]In this day and age, scientists come to new findings everyday. Unfortunately, not all of these are ever shown to the general public. This can be because of several reasons. New facts might not be easily visualizeable. Or the things that they found out are a piece of a bigger picture and hence cannot be easily communicated. On top of that, scientists are not usually trained to expose their results to the public eye.

This is the job of scientific illustrator. These artists are usually, by themselves (nebo first, proste zduraznit, ze jsou to predevsim vedci, kteri do oblasti maji ohromny insight), experts in their scientific field but on top of that they have focused on honing their artistic skills as well. They use their skill to visualize the science in their domain with easily understandable images, animations or other forms of media. To name just a few examples - Drew Berry, Graham Johnson, Janet Iwasa, ... (TODO: references to their sites or projects).

There are many ways how they can do their job. Few years back, illustrations have been done by hand, drawings and paintings. This is a very timely process, as individual illustration can take months to make (reference David Goodsell, feature video na youtube). If we take into consideration that science is moving way faster what could end up happening is that before an illustration is finished a new finding emerges, rendering the illustration effectively obsolete. This is of course undesirable and scientific illustrators have been looking for ways how to accelerate this process.

[**Can we use computer graphics for this?**]Then came the age of computer graphics. As graphics tools have become more and more common and sophisticated in a field of computer generated movies (reference to Pixar?). Tools have become more accessible, easier to use and so scientific illustrators adapted and tried to use these tools for their benefit. Tools like Maya, Cinema4D, or XXX are nowadays the industry standards when it comes to authoring 3D content. Inside these programs there usually exist a software called renderer. With this two parts - modeler and renderer - the user is able to create images or animations (movies) from the 3d data that he creates in the software. Even with just this, the modern computer graphics brings us the

possibility to create amazing imagery and enables illustrators to show ordinary people fenomena from all kinds of science disciplines.

[**Are there any problems that we could solve?**]But this workflow isn't very effective. One of the obstacles in the work of illustrators is that the software they are using is made for somebody else. Games and movies industry are the leading industries that push computer graphics tool creators forwards and provides most of the revenue for them. This means that these tools, no matter how much they try to be versatile, are being skewed towards the use cases of movies and games industries. That means that people who want to create scientific content might struggle to use the tools sometimes.

For this, there are other tools that help them to make the pipeline more effective. This doesn't mean that these tools completely replace Maya or Cinema4D. The domain specific tools are more commonly implemented in a form of a plugin that is developed and loaded into the concrete software packages.

In the end - the features that 3D authoring package provides for modeling 3D scenes are hugely beneficial to the creators. Another things is - people that used these software are used to that and switching from one to another is not a matter of a short time. That's one of the things that we need to take into account. The pipeline of each artist might be a little different and rather than suggesting a solution that completely replaces their pipeline, it's better to make a tools that doesn't break their workflow but rather the artist is able to incorporate the tool, benefit from it but doesn't have to change the way they work a lot.

Odstavec mozna namisto toho dalsiho: On the other hand, the fact that science is progressing forward, constantly bringing new data of various formats, techniques how to visualize, render, show this data in the most optimal way have been developed. Visualization research is an on-going and booming branch of science. This means that every year, new techniques of visualization of data are invented and presented. But as we already foreshadowed, developers of 3D modeling software aren't always up to date with these techniques. This also illustrates the fact, that scientific illustration isn't their primary focus. What that means for us is that we do have new, innovative and better ways how to visualize and display our specific data but these

are rarely integrated into the tools that scientific illustrators use. This is the gap that we set out to fill in this project.

[**what are the alternatives to what people are using already?**]There also exist programs that are completely separated from any of the mentioned authoring software. There is an active research in the domain of visualization, with many conferences every year and thousands (?) of research papers in this domain. Usually, as a biproduct, these paper generate software that showcases the proposed visualization technique or pipeline. Some of these developed into full featured visualization packages and thus provide a way how to illustrate something in a new way. While these programs might hugely benefit scientific illustrators, it's not always the case that they get used. This might be because of several reasons one of which is that the illustrators are simply used to a certain pipeline and incorporating a new software into this pipeline doesn't seems very beneficial to them. Another problem is that because these programs are developed for a certain use cases (showcasing the point of the paper) they might not be easily appliable to more than one purpose.

[**"mission statement" of the thesis**]In this thesis, we made an effort to solve these problems. Our domain of choice is molecular visualization.

In this domain we try to visualize living (?) organisms on the smallest possible scale. With this approach, the computer memory and performance requirements are very challenging even for todays available hardware. Still, people have been able to visualize for example model of HIV in blood plasma in it's full detail down do individual atoms of each protein. This however, is achieved by using a very customized rendering method and a custom data format. In 3D graphics, data are usually represented as meshes consisting of triangles (which in turn are made of vertices). If we wanted to represent each atom as a sphere, we would need at least <number> of vertices for one atom and that's only for the roughest level-of-detail. With such representation, a one whole protein could use up to X bytes of memory which already takes up most of the video cards' memory. Thus this representation is not suitable for this task. Instead, various other techniques have been developed for rendering on molecular data.

That being said, these techniques are not usually taken into account when 3D authoring software is developed. As was mentioned,

the primary users of these software are video games and movies industry, where mesh representation is the one used. That implies that people that use these programs don't have access to the cutting edge technology of rendering, visualization and illustration of molecular data and there is room for us to improve this situation.

In the next chapter, we will describe the state of the art tools that are used for molecular visualization today and we'll mainly focus on showing the gaps in intercompability of the available programs. Then, in chapter 3, we will propose a method of how the problem could be solved.

2 State of the art

I think the sections could be: 3D Modeling Software (Maya, C4D), Niche (Specific) Tools (cellVIEW, ...), Data Generation (cellPack) Popsat: Molecular Flipbook, Molecular workbench, molecular maya, mcell, cellblender, pymol, VMD, ePMV(scripps)

In this chapter we will introduce the software that is available for scientific illustrators to use now. Illustrators have gone a long way from doing these works by hand and nowadays there exist a variety of programs that help them do their job communicating the scientific findings.

[**Commercial 3D software is absolutely dominating**]First and foremost, the pipeline of almost all of these illustrators is strongly built upon a 3D modeling software of their choice. The biggest players right now are Autodesk Maya, Cinema 4D, Autodesk 3Ds Max Design, although other solutions have started to emerge - we can name open-sourced Blender or more and more popular MODO(https://www.thefoundry.co.uk/case-studies/viscira/), first two named (Maya and Cinema4D) being the dominating choices between illustrators. These programs are complex and aim to provide tool for all kinds of users. Their main functionality revolves around two activities - creation of the 3D scene which is most commonly a mesh model and rendering of this scene. These two stages will be referred to as modeling and rendering.

[**The problem of this programs for our case** - this probably should go somewhere else]From this brief description we can already see one problem with using a software like this for creating illustrations of molecular models - they make assumptions about the data. While most of the people using this software would actually be using meshes to represent their 3D scenes, in case of molecular models this is not the case. TODO: say something about "scene graph" which is how scene is implemented usually inside these programs.

[**We have an "entry point" which we can exploit to inject our custom functionality**]Luckily, it has become a convention that all of these programs provide some kind of scripting interface. This enables its users to extend the program with plugins that use the scripting interface. This way the user can implement functionality that he or she

is missing from the basic program. Thanks to that these 3D authoring programs are able to be adapted to more use cases.

[**Tools where people already used this plugin architecture**]Some of these plugins have actually already been implemented to help artists in molecular illustration. The most prominent is Molecular Maya which, as the name suggests, extends Maya with the ability load and manipulate models of macromolecules from Protein Database. Molecular Maya plugins gives its users the ability to load macromolecules based on its PDB ID or locally from pdb file. After the model is loaded, user is able to select the display representation (between TODO: vyjmenovat) and also select with how much detail the model should be rendered.

Similar plugin exists for Blender modeling program as well. It's name is cellblender This could be a nice reference: https://pdb101.rcsb.org/learn/resource/mole animation-q-and-a-interview [**How people normally render - offline**]The rendering stage takes place after the scene is modelled. Again, there are more options at hand. 3D packages usually come with a default rendering solution which is for the most part good enough to use. For more demanding artist, external renderers like vray, mental ray, corona or octane. What these renderes have in common is that they are so called "offline renderers". In practice this means that they are using ray tracing (or similar) technique to render the scene with a process that is very much close to how light works in real life. The disadvantage of this approach is that this process take more time. It usually isn't possible to achieve real-time rendering (fps at least 25).

[**Define the workflow - how it is now and how we could improve it (fasten it)**]We should mention here how the workflow of artists looks like. The individual stages - modeling and rendering - don't happen one after another. The workflow actually looks more like a loop with multiple interactions. The final rendered image is essential when creating the actual scene as the artist needs to tweak multiple attributes of the scene to make it look like he wants. In practice, this means that he performs this modeling/rendering iterations very frequently. Thus he needs this iteration loop to be as fast as possible. As the renderers are getting better and computer hardware more and more powerful, the iterations are getting shorter. (Octane-like) renderers are contributing to this as well. They work in a way that they first display some aproximation of ligthing of the scene (primary rays) and

iteratively increase the quality of rendering. User can therefore see the effect of changes he's made in the scene sooner. However, when we talk about molecular visualization we don't usually care about the rendering quality and physical correctness. What we usually use are simpler, almost illustrative rendering styles. Thus we can simplify the techniques to the point of being able to render molecular scenes in interactive frametimes (real-time) which brings us the possibility to create interactive molecular scenes. This is hugely attractive for scientific illustrators.

[**"visualization research tools that could be used but aren't"**]As we've said, today, most of the illustrators use commercial 3D package to create their works. However, there exists a huge group of programs that are made specifically for science illustration/visualization. These are usually bi-products of a ongoing research project or a scientific paper. Most of these share the property of solving some niche problem. They vary in the degree of how sophisticated they are. Most of them demonstrate a solution for a problem presented, and solved, in the paper which they accompany. However, there have been attempts to create solutions that provide its users the ability to author molecular content by themselves. As an example we can name Molecular Flipbook. It allow user to create simple molecular models and animate then with a simple, keyframe-like animation workflow. On the other side of the spectrum we have custom-built/ad-hoc-built tools coming out as a result of research in rendering and visualization community. These are usually tied to a specific use case, they are a demonstration of a solution. Because of the nature of the data that we are dealing with in molecular visualization, we are able to render our scenes using very effective techniques. The state-of-the-art approach to rendering molecules has been established impostor (or billboard) rendering (TODO: confirm and find references). This technique utilizes the fact that sphere looks the same from all view positions. Because of that we can, instead of rendering sphere consisting of many hundreds of vertices to get the desired level of detail, render sphere using just 4 vertices as a quad always oriented to the camera. This not only optimizes the rendering speed but also memory requirements.

Barely any of these tools is perfect. Some of the tools are very closed and tied to it's creator. The ideal case would be if any new user could

approach any of these tools and with a little training be able to use it. This is not the case, unfortunately.

[**cellpack**]An important part of any scientific illustration are the data one is trying to illustrate. In the case of molecular models, there is cellPACK which helps us with this. cellPACK is a software that assembles large molecular scenes from a description (a recipe) of how this scene should look, what it should contain and in what quantity.

[**interactive renderers like octane, how do they fit and why they are not the full solution**]If we only focused on the problem that the iteration loop is too long because of the rendering times, we would be able to find solution in "interactive renderers" - something like OTOY's Octane or whatever is in MODO. These are path tracers that show the intermediate result which enables the artists to see the progress of the renderer rendering the scene. The artist can then see how his actions are reflected in the final render. This solution however would not solve everything for us if we consider that we are rendering molecular scenes. These scene are usually very big and we still need to use different representation other then meshes to represent our scene and use this representation to effectively render the scene. The contribution of this thesis is not only the fact that we shorten the iteration loop time of the artist but also to sketch an idea of how interoperability of molecular programs can be done.

[TODO: write about the cooperation between two programs - usually importing/exporting files, what are the standards (there are none, only pdb for molecule data which we take advantage of, maybe describe packing result data file format)]

2.1 Maya

Autodesk Maya is not a tool for molecular visualization per-se.

2.2 cellVIEW

3 Method

Outline: [**two options, we choose third**] [image: modeler, renderer, illustrate the connection through shared memory, data streaming, camera position] [**overview of the method: two ends, shared memory, what needs to be transfered, ...**] [**advantages of this method**]

[**two options, we choose third**] What this boils down to is that we have two programs and we want to use some functionality from the first one and other functionality from the second one. We could implement our desired functionality in the software that is missing it. In our case, that would mean we could either implement the fast and visually appealing rendering into our modeling software, or we could do the opposite and implement the desired modeling tools into our renderer.

We've chosen a third option. We don't want to reimplement from scratch something that is already available to use. We want to avoid this development overhead. Instead, we went for a different approach and tried to connect the two parts in a way that would allow us to use both features at the same time. We do this by sending the data via shared memory.

[**overview of the method: two ends, shared memory, what needs to be transfered, ...**] On the Figure X, you can see an overview of the system. On one side we have a modeling software while on the other we have a domain-specific tool, in our case it's a high performance renderer. Our vision was to have both of these programs running at the same time. The illustrator could have his scene opened in his modelling software. This way he would be using the tools he knows and is accustomed to for modelling the actual scene. Then on the renderer side he would be able to see how the changes he's made are reflected in the rendered final image. This all would happen in real-time thanks to writes and reads into and from shared memory.

[**simplifications, what do we count for in the scene**]As was already mentioned, when dealing with molecular data we can take advantage of several simplifications. A scene like this consists of macromolecules: proteins and lipids. First simplification - we consider the shape of the molecule to be static. This means that inside each molecule, the positions of all the atoms doesn't change in time. The position of

the atoms is taken from the PDB file. The second simplification is that all the instances of certain molecule type look the same. They only differ in position and rotation.

[**what data do we need to transfer**]With these facts in mind, we can define a molecular scene as a set of molecules, where each molecule is defined by its position, rotation, and type, which says what kind of molecule this instance is.

This means that we need to stream this data - positions, rotations and types - of all molecule instances from the modeling software to the rendering tool. Because of the technical implications we write the data in a format described on Figure X. This is more efficient that writing the data in a format position/rotation/into for each molecule.

3.1 old

TODO: there probably shouldn't be any mention about Maya or cellVIEW here (it belongs to implementation) I have introduced the field. I have sketched the problem that we are trying to solve (improving the workflow of animators/illustrators). Now it's time to describe the suggested solution. We talked about improving the workflow of scientific illustrator that uses Maya as an authoring software. The way that we are going to accomplish this is by taking advantage of extensibility of both Maya and Unity. There are basically two stages of authoring a scientific visualization. These are modeling stage and rendering stage.

[**high-level description of the method**]In the previous chapter, we sketched out the problem that scientific illustrators face in their day-to-day work - the fact that they don't have domain specific tools. Namely that the software that they use don't support newest results from scientific visualization field. We aim to solve this problem by extending their software by effectively connecting them to software which does implement newest visualization techniques. We want to achieves this by using shared memory and this way enable real time data streaming. TODO: image of the pipeline with time estimates of how long each stage takes, compared to our new approach

[**what are other options how the problem could be solved**]If we looked at this problem of having two pieces of software - one that illustrators use and one that is able to perform state-of-the-art visu-

alizations - we would be facing deciding between two choices. First, we could take the state-of-the-art visualization functionality and implement it into the modeling software. Or, we could implement the desired functionality from modeling tool into the visualization tool. The second option would be really hard to do. As we've said in the previous chapters, modern modeling tools are products of hundreds of people and big companies are driving its development. Because of that, reimplementing all the functionality into our visualization tool would be impossible. On the other hand, implementing newest rendering algorithm into the modeling software might not be that hard to do. The problem, however, is that the modeling software might not be able to extend in the way we want and need. Because a big company stands behind the software, that means that the source code is closed and only a small part is exposed in the form of API. Implementing some functionality, not to say a state-of-the-art realtime rendering, might not be possible without the knowledge of the whole software architecture. Thus we need to take a third option - connect these two programs together. This way we are able to use modeling tools and function from modeling software and at the same time we can use modern rendering/visualization capabilities of visualization software.

[**What are the things from each program that we want**]What we basically did is use the fact that modern GPUs are able to render what we want in a decent visual quality and that they can do it really fast (real time). This way we can shorten the time it takes to the illustrator to create his artwork. From Maya we aimed to use it's various modeling tools - tools that enable translating, rotating, scaling of objects in scene. With that we wanted to use particle system to generate randomly molecular scene (not scientifically correct but close enough for illustration purposes).

[**what was the initial vision**]The vision was that there would be cellVIEW which renders the scene on one side and Maya on the other. In Maya the scene would be appropriately simplified. At the beginning we had an idea that there would just be cubes with maybe textures that would give the user at least some visual information about the protein type. We ended up using very rough meshes that represent the proteins in the scene in Maya.

[**are there any initial requirements?**]As this project wasn't meant to be software intended for certain user group, we work with very

little input requirements and so the design and user experience side of things have been explored over the course of development. We however had initial input from one of the most respected domain experts in scientific illustrator - Drew Berry. From this persona came the initial idea to use Maya as the modeling package as that's the software that Drew Berry uses. He expressed his interest in rendering style that cellVIEW provides and suggested that if we were to make cellVIEW usable in his workflow with Maya, it would greatly improve his productivity and shorten the production of his movies. His input sparked our interest in making cellVIEW usable as a part of a pipeline rather than a standalone tool. This then lead to research of which resulted in the following proposed method. This general method is presented as an idea of an approach that tools like cellVIEW could take in making themselves usable in art authoring pipeline. In this chapter we present this idea. Further in implementation chapter we focus on our use case - using Maya and cellVIEW but we believe that this method should be appliable on any two modeling and rendering tools (with possibility to be used in other applications as well).

[**are there any standardized file formats in this field?**]In computer science there are not a lot of ways to translate information from one program to another. The majority of this communication is done via files that are exported on one side and imported on another. This lead to a need for standardized format in which this data are formatted so that both of these programs understand each other. TODO: write about standard format in molecules (not a lot - pdb, we are taking advantage of that, other than that, maybe write about packing result data file format) This simple approach is fine for most of the application where it's used. In our case however this isn't sufficient. There is one big problem. If user needs to perform the actual export and import step, this greatly reduces his productivity. Another problem that is found in our domain has already been mentioned - rendering times.

[**we don't want to use files** TODO: why?]Our method tries to solve these two problems. The problem of import/export is solved by using shared memory instead of files managed by the user. The rendering times problem is solved by using modern rendering techniques which enable us to render what we want in an interactive framerate. The reason why it's possible to take this approach of using shared memory comes from the character of data that we work with. Molecular scenes

consist of molecules. Molecules are made of atoms which can be, and usually are, represented as spheres. Atoms of different elements are visualized by having different radii and colors. So for every molecule we need to keep track of what atoms it consists of, positions of those atoms and type of each of those atoms. This is even more leveraged by Protein Data Bank, which stores data about various proteins that have been found through out the years. For us, this means that we only need to keep track of the type of molecule and by fetching data from Protein Data Bank we get information about the protein cataloged under the protein identifier. Our scenes are (simplified) made of macromolecules. That means that in the end for rendering the scene we need a list of molecules, where for each molecule instance we need to save it's position, rotation and type. That's all the info that we need to be able to render our large molecular scenes. Even though this is still a lot of data, we are slowly getting to being able to render this on commonly available hardware. Because of the fact that this data size is "manageable" by modern computers we are able to store the data in shared memory. This method is mostly valuable as an example of how interconnection of software can be done and what improvements to the users this can bring. There is some prerequisities - both programs need to be extensible to the extend of the programmers must be able to develop extension that are able to read from and write to the shared memory using operating system api calls.

4 Implementation

Outline: [**maya side plugin**] [**unity side plugin**] [**unity side c++/c interoperability**] [**unity side script - how the data is used/rendered**] We have implemented this method using Autodesk Maya and cellVIEW. Maya provides its users an API which allowed us to implement a plugin which would provide desired functionality. Similarly, we have been able to create plugin for cellVIEW. This was because of the fact that cellVIEW is implemented using Unity engine which also allows custom plugins in a form of DLL to be used. [**Why Maya?**] We have chosen Maya as the modeling software because our collaborating partner, Drew Berry, is its prominent user and he's been a key person that we had in mind when implementing this method. Chosing cellVIEW as the renderer has been an obvious choice as its rendering capabilities are on the state-of-the-art level of what common hardware computers are able to render. Drew Berry has been impressed with the results of cellVIEW and expressed his interest in using it for his movies.

[**high-level overview of the architecture**]The key element of the implementation has been how to use all the different parts of the ecosystem, all the extensibility possibilities, and connect them together in a way that would show our method. The architecture of the system basically consists of three parts - a plugin for Maya, plugin for Unity and a Unity script. The resulting system consisting of 3 parts which are illustrated on a Figure X. There is only one way which the data flow which simplified the implementation but of course it would be possible to extend the interoperability to be two-way(?). On Maya side, a plugin using Maya API written in C++ has been implemented. The function of this plugin is to parse the 3D scene, looking for a molecular objects, and output their positions and rotations (along with info about the instance) into the shared memory. Unity side of the system consists of two parts - a C++ plugin and a Csharp plugin. The C++ plugin takes care of reading the data from shared memory, while the Csharp plugin, which is a part of cellVIEW, receives this data and uses them to render the final molecular scene. As you can see, the interoperability is achieved using (1) shared memory functionality and (2) interoperability between C++ and Csharp.

[**maya api**]Maya actually provides two APIs - one in Python and one in C++. There is also another way how additional functionality can be implemented - MEL scripting language. MEL is very similar to other scripting languages like Bash, ... However, it is meant to be used in a way where user automates tasks, not to access API and even system API calls. Because of that MEL was not suitable for our job. Python API is more mature then MEL. Its advantage is that Python, as a interpreted language, can be run without compilation. This means that there doesn't have to be a compilation step and this makes the iteration loop faster. However, we wanted to implemented a functionality that is real-time. This means that we needed every piece of performance we can get. Because of that, the C++ API has been chosen and thus our Maya plugin has been implemented in C++. There is also another reason why only the C++ route could be taken. We wanted to write into shared memory. The most basic way how to do that is via system API function calls. In our case, we implemented this on Windows platform. Thus the API we used has been WinAPI.

4.1 Implementation limitations

I think I should talk here about what are the limitation of this method - how much data can we fit into memory and stuff like that.

4.2 WinAPI

4.3 Maya side

Topics - Architecture of the plugin, what parts/classes of API are used, memory layout, what I tried and didn't work. What we needed to do is basically scan the 3D scene graph and output positions and rotations of each macromolecule object. There are two usecases which needed to be considered. When modeling the scene, the rate of change is not that high. Usually the scene only changes very little (moving or rotating usually just one or few objects). However, we had in mind also a usecase where the animation of the scene would be outputed as well. This means that artist would create a keyframe animation of (nearly) all molecule objects and we would want to transfer the

position/rotation data each frame for the animation. Obviously, for each of these use cases there are different ways how to achieve the optimal performance. In the end I settled for (todo: I don't actually know right now)

4.4 Unity side

Topics - Architecture of the plugin, very generally about plugins (it's just a basic C++ dll plugin), interface between C++ and C#

5 Demonstration

In this chapter we present two use cases of how the user can approach illustration with this new proposed system. Use case one - modified Janet's scene. Use case two - microtubulus (create a model in maya and then name it properly and we will get it in cellVIEW).

6 Discussion, future work

Limitations (of the tool) - what was done just for the use case and should be worked on for final production. The project has been presented to Drew Berry (I think). Also, it was mention in a talk in Utah, presentation by Peter Mindek. The method has tremendous potential in its application. The current implementation should be extended for both performance and actual use for artists. For this however, we will need additional input from domain experts. We have been fortunate enough that Drew Berry really liked this work and expressed his desire to develop this project further. It is highly probable that we will be working even more closely with him and that we would continue to improve this system so that he can use it for his movies production. The challenge will be how to design the system to be easily adaptable by other creators as well.

A An appendix

Here you can insert the appendices of your thesis.