Master's Project Report: Animated Lineart Tool in Houdini

Daria Tolmacheva
Student ID: s5000836

August 2024

Abstract

With the rising popularity of 2D stylisation in 3D animation the need for tools to assist artists in achieving the look they want in a short time frame rises rapidly. And with linework being a fundamental part of many different art styles, a versatile tool capable of helping to create it is a welcome addition to the pipeline. The idea of a tool that generates lineart in the form of a 3D curve from the geometry and camera supplied while allowing for fine artistic control and producing a natural look is inspired by Kismet developed by Sony Pictures Imageworks for their film Spiderman: Across the Spiderverse. Like Kismet, this project aims to leverage procedural nature of Houdini to create a tool which would generate convincing animated lineart. Unlike many existing methods of stylisation this solution provides versatility and comfortable user experience for the artist using it.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

In recent years animation has been embracing the visual style normally seen with multimedia traditional art forms such as painting, inking, collage and many others. However, when used in mainstream animation CGI tools are used to achieve similar effects to complement the 3D animation. One of such traditional artistic effects is linework which is very popular due to its use in many different mediums.

Line, being one of the fundamental parts of any visual art form together with shape, is used in a variety of media from basic sketching to 2D cell-shaded animation and graphic novels. Therefore, as modern animation takes inspiration from those styles, a multitude of methods and tools to ease the process of creating linework with different looks have been developed in recent years.

With that in mind, this project aims to create a Houdini Digital Asset Lineart Tool based on the ideas behind Kismet, a tool developed by Sony Pictures Imageworks for their film Spiderman: Across the Spiderverse. The emphasis of its development has been placed on the balance between the convincing effect of hand-drawn lineart and accessible user controls while utilising the procedural nature of the Houdini environment.

1.2 Aims

1. Create a tool which generates a convincing lineart in 3D space.
2. The generated lineart should convincingly redraw itself without being fully stuck to the model.
3. The tool should have user-friendly controls supporting freedom of artistic expression.

1.3 Objectives

The objective of this project is to create a tool in the Houdini environment which would generate a linear curve in 3D space based on the supplied model(s) and camera. It should be able to achieve a messy appearance to convey a hand-drawn look and feel both static or animated. When animated it should avoid a smooth CG look gotten when lines are stuck on the model. The tool should also break down the available ways of editing the result into logical controls arranged in a user-friendly interface enabling a smooth workflow for the artists using it.

Overall, the aim is to develop a tool which achieves a hand-drawn look of the lineart with an interface that keeps the balance between variety and accessibility of user controls.

Chapter 2

Previous Work

Various methods for creating different lineart looks have been developed over the years. In this section let's look at some of the most notable approaches to this problem.

The edge detection approach used in many areas of computer vision including restoration and vectorisation of older 2D animation (Mao et al. 2015)(Zhang et al. 2009). When it comes to 3D, it is the primary approach used for creating 2D cartoon-style visuals with toon shading in the past thirty years (Danner and Winklhofer 2008). This method uses the ambient image, the normals buffer and the z-buffer to generate a silhouette and crease edges that make up the overall outline of the objects in the image (Decaudin 1996). This means that this method includes rendering multiple passes of an image and running the edge detection algorithm on the textures produced in the first render run to generate the final image. This approach results in a very distinct style where outlines are attached to the model similarly to cell-shaded 2D animated cartoons. Alternations to this basic algorithm lead to some very interesting variations of stylised looks like a rendering mimicking Chinese painting (Yuan et al. 2007).

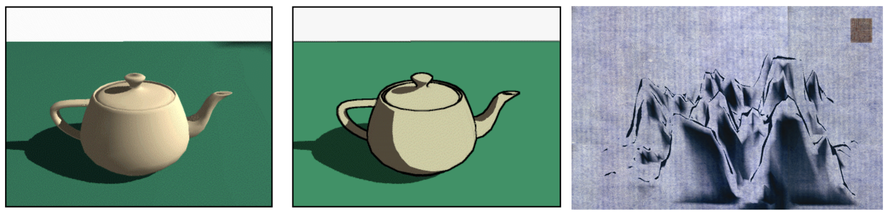


Figure 2.1: A teapot rendered by a photo-realistic algorithm (left) and by "cartoon-looking" algorithm (center) (Decaudin 1996), A mountain, automatically rendered with Chinese Painting Cartoon shader (Yuan et al. 2007).

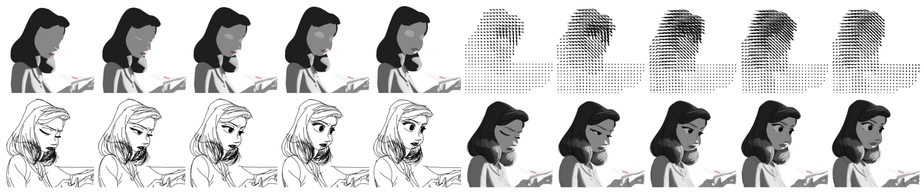


Figure 2.2: Example sequence showing the CG renders, motion fields, hand-drawn stroke centerlines, and the final line renders done for Paperman (Whited et al. 2012).

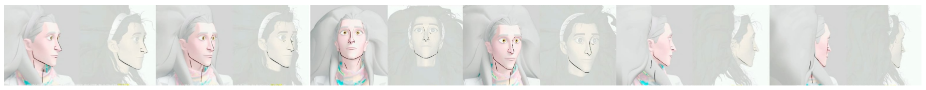


Figure 2.3: Example of data used for training of and generated by the machine learning inking tool used for Spiderman: Into the Spiderverse (Seymour 2019).

While this method is fast, allowing use even in real-time such as computer games, since shaders are fine-tuned to produce a specific style of an image, this method is, unfortunately, not very versatile. Since this method depends on rendering, it is far removed from the animators in the pipeline leading to less artistic control from them. Moreover, as it does not allow easy control over lines being in- or outside of the silhouette of the model, replicating realistic imperfectly drawn pen strokes around the object is challenging, so it does not work for the tool created in this project.

Before the recent resurgence of popularity of the 2D look in 3D animation, Disney had developed a novel approach to creating a flat look with linework in the compositing stage of the pipeline for their short Paperman (Whited et al. 2012). This method includes rendering the motion fields rather than the final frames as part of the pipeline and drawing the key frames of 2D linework manually in vector software. Then the in-between linework is generated based on the motion field and the key frame position and orientation. The result is a temporally coherent and spatially stable mix of 2D and 3D animation.

While this method produces interesting results and gives a lot of freedom to the artists, it only assists by generating in-betweens based on key flames drawn by the animators themselves. This approach is very helpful as an interpolation technique to speed up the process of hand-drawing animation, but does not generate the linework itself which is the objective of this project.

As combining hand-drawn linework with 3D animation continued to be the primary approach to create this mixed look, creating tools to speed up the process of generating spatially and temporally coherent animated in-betweens

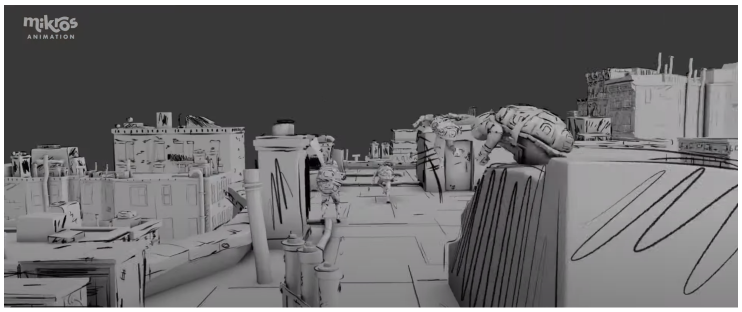


Figure 2.4: VFX breakdown of the shot from Teenage Mutant Ninja Turtles: Mutant Mayhem featuring some of the linework (Mikros Animation 2023).

based on the hand-drawn lines and animated 3D models rather than the main linework itself has continued to be the trend. To solve this problem for their 2018 film Spiderman: Into the Spiderverse, Sony Pictures Imageworks have trained a machine-learning model to edit linework curves based on the character's position, orientation and deformation in the shot (Seymour 2019). To achieve this, artists have created the curves for the linework of the faces and hands of all relevant characters. Then they manually edited the curves depending on the camera angle and expressions. This has become the original training data for the model, with original curves and data about the character's position in relation to the camera and their expression being the source and the transformation applied to the curve - the target.

This approach helps speed up the process of animating the linework while giving animators full artistic freedom to set the rules of the linework's look and motion. The only major drawback of this solution is its lack of scalability. Since a lot of manual labour must be done to create the training data, in addition to the very niche specialisation of the machine learning model. To produce linework of different styles for different objects rather than of one specific style only for faces and hands, multiple specialised models must be trained similarly to the development of different shaders in the toon shader approach, which is not feasible for such requirements. As such, on their film Mitchels vs the Machines, Sony Picture Imageworks opted out of using the machine learning tool used on Spiderverse due to a different style (SIGGRAPH Conferences 2021).

This being said, with the popularity of the stylisation of 3D animation growing and more films having a wider variety of requirements with linework being a fairly small part of the chosen style, manual linework animation still holds its place as one of the popular approaches. As such when tasked with implying teenage imperfection with the look of Teenage Mutant Ninja Turtles: Mutant Mayhem, Mikros Animation has used a variety of ways to achieve the result, including curves being added to the models themselves to represent drawn lines

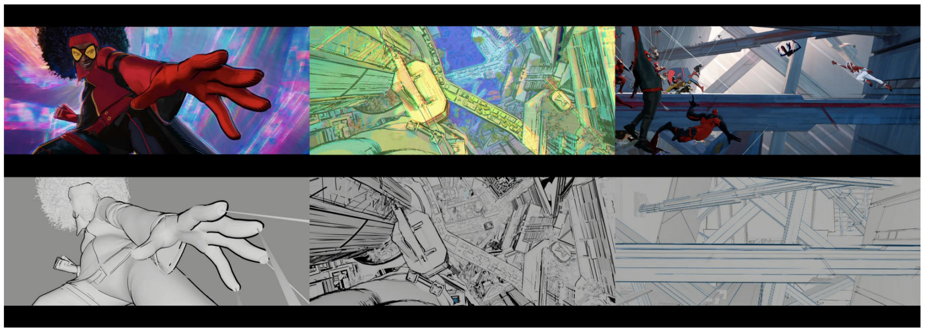


Figure 2.5: Linework breakdown of some shots from Spiderman: Across the Spiderverse (Grochola et al. 2023).

(Failes 2023a). This allowed the linework to be animated alongside the characters rather than be an additional layer added on top and worked in 3D rather than creating a 2D line drawing effect.

Following the success of the first Spiderverse movie, the team behind the sequel, Spiderman: Across the Spiderverse, got a new and challenging task of bringing a variety of characters and universes to life, each with its unique art style. Being based on comic books it involved creating a lot of distinct styles of lineart for scenes of varying complexity. To solve this problem the team at Sony Pictures Imageworks has created Kismet - a Houdini-based tool which primarily serves the purpose of a versatile outliner, however capable of generating a variety of linework styles (Grochola et al. 2023). The tool generates the linework in the form of a curve based on the given geometry and artistic input by leveraging the procedural nature of Houdini. As Pav Grochola, FX supervisor on the film, highlights (Pav Grochola interviewed by Ian Failes 2024): ”Generating linework from a 3D model is not a new idea, but the way we approached it was novel. In the past this was achieved by an edge detecting gag, in comp or in a shader.”

Among other features, Kismet is capable of generating linework animated at variable speed based on the rate of change of the part of the object moving. It also supports the inclusion of imported curves and redrawing the base curve based on those additional curves. This was implemented with Blender and its Grease Pencil in mind, with the curve interpolation tool developed by the same team to allow artists to use both software together. Being part of the big production, the tool is fully integrated into the pipeline with preset functionality being available and batch processing of the shots supported.

Overall, Kismet is a very versatile tool that perfectly balances the procedural generation and artistic control aspects of linework creation, which is why it inspired this project.

Chapter 3

Technical Background

3.1 Kismet Algorithm

In the paper ”Linework in Spider-Man Across the Spider-Verse” released by Sony Picture Imageworks, creators of Kismet break down the algorithm it implements into four stages (Grochola et al. 2023):

1. Generate all possible line positions based on the camera angle relative to the 3D mesh, creating the base curve.
2. Based on user input, create secondary lines on top of the base curve. These are the lines to be rendered.
3. Stylize secondary lines by assigning various properties such as varying tapering and offsetting to achieve a complex, natural look.
4. Output curves to Katana for rendering.

While they do not reveal the specifics of the process for the first two steps, it is stated that linework generated exists in the form of curves existing in 3D space (Pav Grochola interviewed by Ian Failes 2024) which allows manipulations such as offsetting them from the model to create lines that are not attached to the models or edit them during the lighting stage of the pipeline by changing the width or removing parts of them (Pav Grochola interviewed by Ian Failes 2024). This is also the part of the algorithm where curves are animated to fit the hand-drawn aesthetic.

The linework is animated with a ”dual-rest” approach (Grochola et al. 2023). This method involves two sets of slightly different curves working in tandem - with one being drawn while the other being erased and vice versa. The process is done at a redraw rate which looks most natural for a given animation. Determining the best redraw rate is a tricky task which was enhanced in Kismet by adjusting it based on the redraw rate of relevant parts of the models.

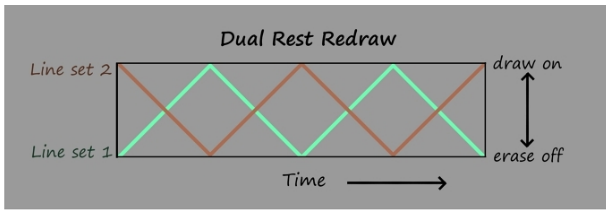


Figure 3.1: Dual rest redraw (Grochola et al. 2023)

In the third part of the process, the secondary curve is edited to fit the requirements of the scene based on the artistic requirements. Some of the examples of such edits mentioned include basing the lines' look on the attached light and the shadow it cast (Failes 2023b), creating hand-drawn messy appearance via extending lines away from the model (Grochola et al. 2023) and adding random variations (Pav Grochola interviewed by Ian Failes 2024) among others.

Due to the scale of production Kismet is used in, the final step is important to ensure all the relevant information for further pipeline stages such as lighting, rendering and compositing is preserved or generated and the linework itself is exported appropriately. One example of such attributes being used is the aforementioned editing of lines' width in lighting stages (Grochola et al. 2023).

3.2 Outlines

3.2.1 Dot Product Method

To create the initial base curve outlines of the given mesh need to be generated. Outline in this case is a subset of points on the mesh whose normals are perpendicular to the vector between the camera and those given points. Since two perpendicular vectors always have dot product equal to 0, set of points belonging to the outline (O) can be defined as follows:

$$P_p \in O, \text{ if } N_p \cdot (P_p - C) = 0,$$

where P is the position of the point tested, N_P is the normal of point tested and C is position of the camera.

Therefore, casting rays from the camera to each of the points on the mesh and checking the direction of the normals relative to the ray cast will produce the full outline of the mesh. However, it will include all the parts of the mesh which are not visible to the camera. Therefore, either extra pre-processing is necessary to remove occluded points or the points on the outline need to be checked for visibility against the original mesh.

3.2.2 Pure Houdini Method

Alternatively, Houdini grouping functionality can be used to achieve similar results. When creating a group in Houdini, a group of unshared edges can be created which will only include the edges belonging to a single primitive. This means a Houdini algorithm for creating an outline can look like this instead:

1. Cast rays from the camera and filter only primitives facing the camera based on the normals.
2. Remove all the invisible parts of the mesh leaving only those facing the camera.
3. Select unshared edges of the resulting mesh.
4. Delete all unselected points to leave only the outline.

This method leverages Houdini build-in functionality, also allowing to take advantage of additional options presented by the Houdini nodes involved such as enabling shadows when casting rays from the camera to remove all invisible parts of the mesh, not only those not facing the camera.

3.3 Imitating Hand-Drawn Linework

While analysing the qualities of the hand-drawn lifeworks, three parameters were identified which largely contribute to its messy and lively look.

Firstly, the width of the line varies with different patterns depending on the tool used as well as pressure and speed of the stroke. This can be replicated with the width of the curve changing both with noise or manually while layering larger changes with high-frequency noise, producing especially interesting results.

Secondly, when drawing with lines artists often only emphasise important parts of the overall shape instead of outlining everything, therefore placement of the lines is important. Gaps can be introduced both with noise or manually to break up the single curve and to give it a more natural feel of multiple strokes being combined rather than a monolithic structure.

Lastly, when hand drawn, the lineart often leaves the edges of the object drawn either in- or outwards due to looseness of the artist's stroke motion. This is a very important feature which sets the Houdini solution apart from the shader one. This can be implemented with displacement being introduced both with noise or manually to create a looser and messier look.

Together these three parameters combined with stylised animation allow to alter the base curve enough to reach convincing and varied linework. Therefore, this tool mainly focuses on them when it comes to artistic controls and lineart stylisation step of the process.

Chapter 4

Tool Architecture and Development

4.1 Base Curve Generation

The first step in generating the natural-looking lineart is to generate the original curve which would then be used as a base for the stylized version. In this case the starting point is creating the outlines and adding manually selected edges of the original mesh that are chosen by the user.

4.1.1 Outlines

The first method to be implemented in an attempt to create clean outlines was the dot product approach. In this method the following algorithm needs to be executed over every point:

Algorithm 1 A basic algorithm for checking if point must be included in the base curve.

```
$direction \leftarrow cameraPos - pointPos$
$dot \leftarrow pointNormal \cdot direction$
**if** $dot < threshold$ **or** $dot > threshold$ **then**
    remove(point)
**end if**
```

The threshold is used due to the nature of the mesh not allowing to only check the zero values. Only finite set of points of the given mesh are checked rather than continuous sequence therefore it is unlikely that even the points closest to the border are perfectly perpendicular to the ray from the camera, therefore an adjustment must be made.

However, this results in all of the outlines of the parts of mesh being included, both visible and occluded. To combat this issue the following solution has been developed:

Algorithm 2 An algorithm for checking if point is occluded.

```
intersect_point = first intersection of direction vector with original mesh
if there is an intersection and intersect_point is outside threshold for at least one component then
    remove(point)
end if
```

This ensures the removal of the occluded point by checking if the first intersection with the original mesh results in the point itself or not. A small threshold is used again due to the same reason as in the above algorithm.

One other issue arises even after this, since the threshold is absolute and therefore has to be tweaked depending on the model and its size. Meaning to make the algorithm more flexible the threshold needs to be relative to the size of the model, for example by surveying the edges which include the current point and making the threshold a percentage of the average of their sizes. However, the closest point to the current one that is in front of it might not share an edge with it, so they should also be accounted for. Therefore the threshold can be calculated as a percent of the weighted average between the average edges linked to the current point and its closest point if it is in front. Resulting in the following algorithm:

Algorithm 3 Final algorithm for checking if point must be included in the base curve.

```
$direction \leftarrow cameraPos - pointPos$
$dot \leftarrow pointNormal \cdot direction$
if $dot < threshold$ or $dot > threshold$ then
    remove(point)
end if
intersect_point = first intersection of direction vector with original mesh
calculate average distance to all neighbours
if the closest point is closer to the camera than current point then
    threshold = (neigh_avg + closest_point_dist) / 2 * threshold_percent
else
    threshold = neigh_avg * threshold_percent
end if
if there is an intersection and intersect_point is outside threshold for at least one component then
    remove(point)
end if
```

Unfortunately, even with all the extra threshold tuning, the outlines pro-

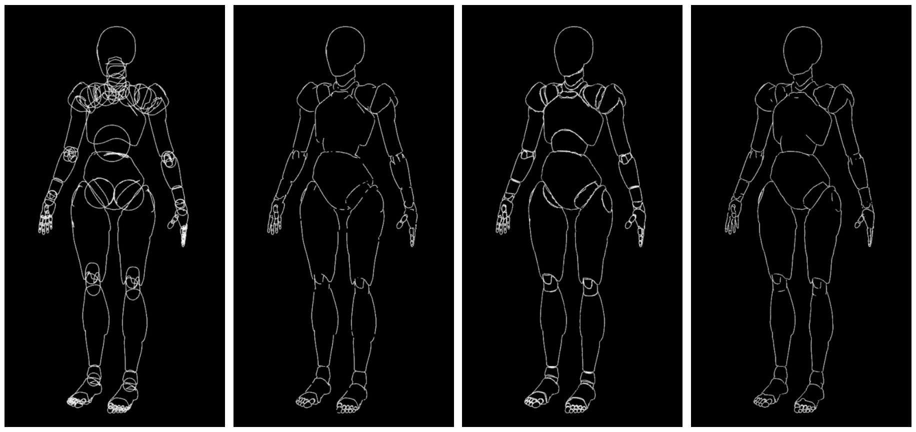


Figure 4.1: From left to right: 1) Outline produced by dot product method before the intersection check, 2) outline produced by dot product method with the intersection check, 3) outline produced by Houdini SOPs method before clean up, 4) outline produced by Houdini SOPs method cleaned.

duced by this method still need to be adjusted for different models and have artefacts on the models with various quad sizes. Due to this, a decision to switch to a different method had been made and the Houdini SOP approach described in section 3.2.2 was developed instead.

To generate full outlines of the given geometry, the Mask By Feature node is used which allows to cast rays from a point (camera position in this case) onto the geometry to eliminate back-facing parts of the mesh, followed by selecting only boundary edges of the remaining geometry and removing everything else. However, while creating a full outline of an object in Houdini is fairly simple when familiar with the nodes it has to offer, removing the occluded lines is a more complex task. The first step to achieve this is letting the aforementioned Mask By Feature node cast self-shadows, therefore eliminating the occluded lines. The problem arises from boundary edges now including the borders of the shadows, which due to the typical topology of the models coupled with unpredictable positioning result in the artefacts as seen in Figure 4.2.

Since with this setup the artefacts appear where two lines overlap - the one of the object casting a shadow and one of the shadow itself - removing the shadow line which is jagged due to topology is the goal here. The approach to removing it is to create two masks from the Mask By Feature node used earlier, one mask with self-shadows being cast and one with only back faces being unmasked. The overlap between the two are the lines that should be kept while the difference are either the artefacts or occluded parts which should be cut. Therefore, subtracting the masks and choosing a threshold for this value leaves the outlines clean. With some testing, it has proven to work best with

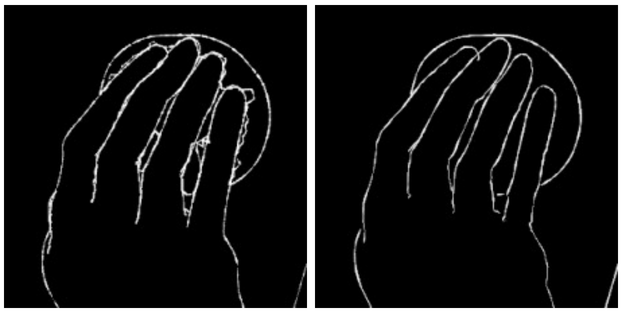


Figure 4.2: Example of the outline with (left) and without (right) artefacts created by Houdini SOPs method.

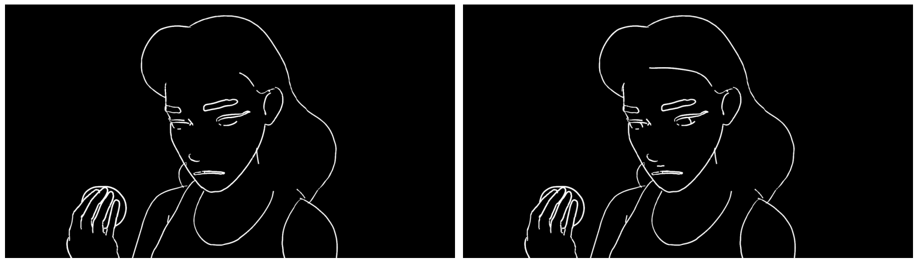


Figure 4.3: Base curve example without (left) and with (right) additional manually added lines.

shadowed mask blurred for more distinct difference.

4.1.2 Additional Lines

In addition to the generated outlines the user can select edges of the mesh they want to keep as an additional part of the base curve as seen in Figure 4.3. This allows extra details to be kept but works best when the topology of the object is kept in mind with either edges in question having been already placed where the lines should go, or the lines must be chosen with current topology in mind for an organic look when animated.

This feature is implemented via a paining interface which can be enabled once the option to add extra lines is toggled.

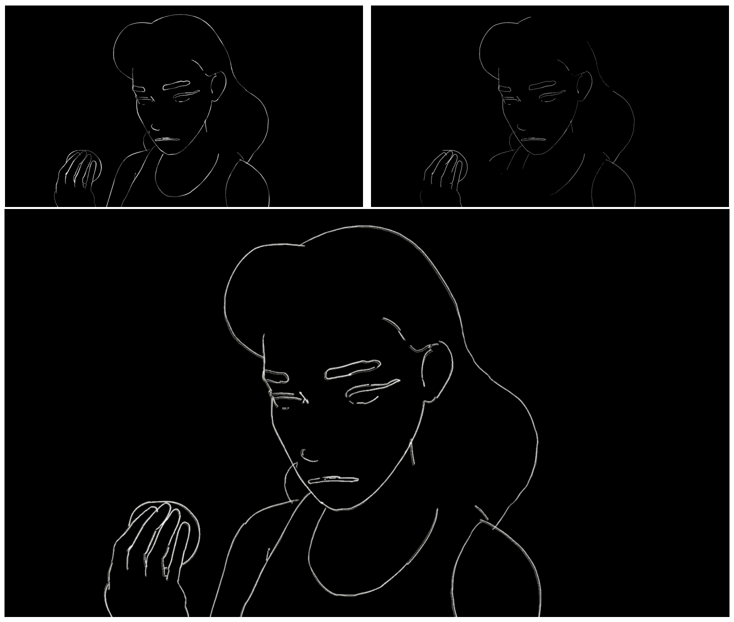


Figure 4.4: Noise used to stylize lineart via altering width (top left), gaps (top right) and displacement (bottom) where lighter lines on the image with displacement altered show the base curve.

4.2 Lineart (Secondary Curve)

4.2.1 Width, Gaps and Displacement

With three main parameters contributing to the natural look of the lineart being the width of the curve (and its variation), the placement of the lines (and gaps between them) and the displacement of the secondary curve from its base position, the user interface features a tab dedicated to each of them, with the contents of each being almost identical to make it more familiar and therefore user friendly. There are three main ways to edit each of those parameters: add noise, add a map file and draw a map.

Noise is a necessary feature due to it being the primary source of the natural messy look. Enabling it gives the user a variety of ways to control it which anyone familiar with noise in Houdini would know: mainly the type of the noise, element size, seed and the values variation settings. These are more varied for width and gaps attributes, due to displacement being a vector rather than float and therefore having more restrictions on actions available on it. Figure 4.4 showcases examples of noise use on each of the three of the available parameters.

Adding a texture map to drive the parameters is useful when the tool is used on a single model. A good example of a use case of this option is when lines must avoid certain areas of the model in all scenes (like avoiding the face to deal with it separately) or if the lines must be extra thick on certain parts of the model like chin or clothes.

Drawing a texture map allows users to correct specific parts of the scene or fully control the look they want. Implemented with Houdini's attribute paint node it has the usual Houdini painting interface giving a lot of control to the artist. One of the notable examples of use is for the artist to paint areas to be displaced to create an effect of pen trailing off the silhouette of the object as it leaves the paper.

4.2.2 Raycasting

The raycasting feature is a special case of controls for width and placement based on a "light source". The "light source" in question is an object which casts rays onto the geometry and affects the look of the lineart generated based on artist inputs. The light can be moved via a translate handle, its radius of effect and angle of the spotlight can be changed along with the falloff pattern. Together it creates an effect similar to a light lighting the lineart only in certain areas. Whether the light enables or disables the lineart can be switched as seen in Figure 4.5, allowing for some creative use cases such as either creating a rim light effect with white lineart or having thicker more prominent dark lines in the shadow with nothing in the light instead or even a combination of both.

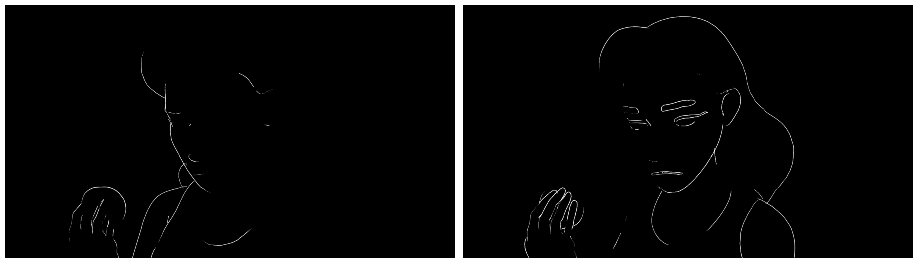


Figure 4.5: Lineart produced by reycasting mode on with linework being generated in the lit areas on the left and linework in the shaded areas on the right.

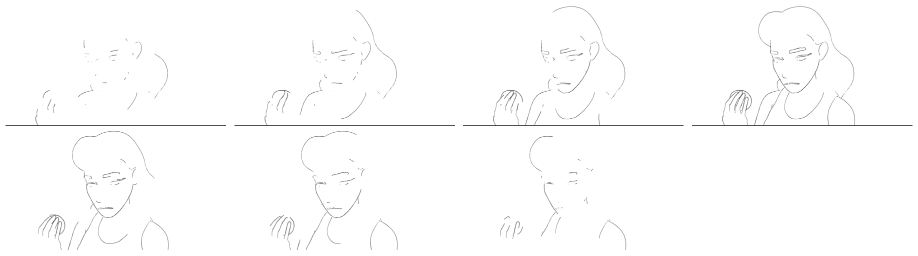


Figure 4.6: Sequence of 4 frames of two alternating linearts (top and bottom) being carved out at redraw rate of 4.

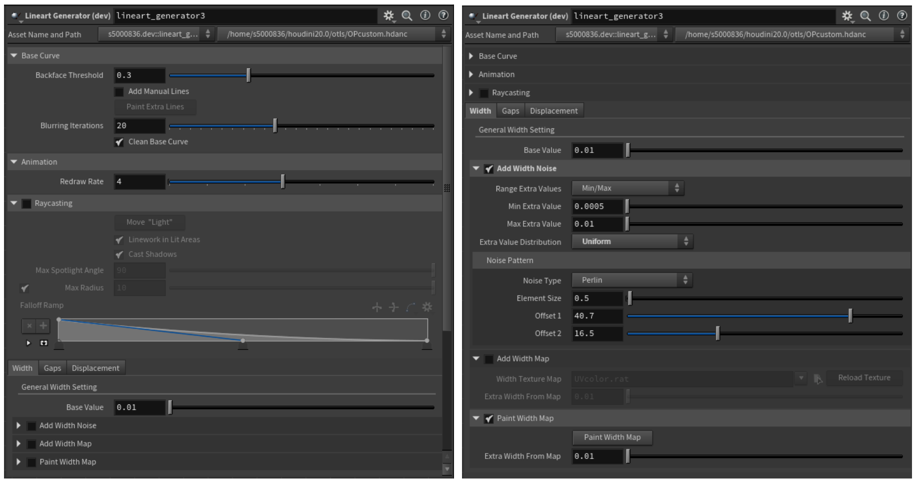
4.3 Animation

The first step to creating the desirable animation effect of lineart being redrawn by hand is to step the animation and set it to a desired frame rate. This is done by two Timestep nodes on the model just as it is supplied to the tool. After that, two slightly different versions of the secondary curve are generated and alternated at the desired rate. The first method to fade them in and out was to change the alpha channel, however, this resulted in unpleasant flickering artefacts so had to be scrapped. The final version features the lineworks being carved instead as seen s 4.6. The carving is controlled by two variables for each curve and is calculated by the following calculation in Algorithm 4.

The algorithm is reversed by switching if and else parts for the second lineart to alternate them. This produces a subtle yet effective result which can look even more convincing for static objects at a slower redraw rate.

Algorithm 4 An algorithm for calculating carving start and end values based on the frame.

```
if $frame\%(redrawRate * 2) < redrawRate$ then
    startU = 0
    endU = (frame+redrawRate)%redrawRate / redrawRate
else
    startU = (frame+redrawRate)%redrawRate / redrawRate
    endU = 1
end if
```



Lineart Generator (dev) lineart_generator3
Asset Name and Path s5000836.dev::lineart_g... /home/s5000836/houdini20.0/otls/OPcustom.hdanc

Base Curve
Backface Threshold 0.3
Add Manual Lines
Paint Extra Lines
Blurring Iterations 20
✓ Clean Base Curve
Animation
Redraw Rate 4
Raycasting
Move "Light"
✓ Linework in Lit Areas
✓ Cast Shadows
Max Spotlight Angle 90
✓ Max Radius 10
Falloff Ramp
Width | Gaps | Displacement
General Width Setting
Base Value 0.01
Add Width Noise
Add Width Map
Paint Width Map

Lineart Generator (dev) lineart_generator3
Asset Name and Path s5000836.dev::lineart_g... /home/s5000836/houdini20.0/otls/OPcustom.hdanc

Base Curve
Animation
Raycasting
Width | Gaps | Displacement
General Width Setting
Base Value 0.01
✓ Add Width Noise
Range Extra Values Min/Max
Min Extra Value 0.0005
Max Extra Value 0.01
Extra Value Distribution Uniform
Noise Pattern
Noise Type Perlin
Element Size 0.5
Offset 1 40.7
Offset 2 16.5
Add Width Map
Width Texture Map UVcolor.rat
Reload Texture
Extra Width From Map 0.01
✓ Paint Width Map
Paint Width Map
Extra Width From Map 0.01

Figure 4.7: Main menu in different configurations.

4.4 User Experience and Documentation

4.4.1 Helpcard

For a better user experience the tool features a full helpcard with every parameter and part of the menu explained with images included where beneficial. It also includes references to helpful materials which explain useful information regarding other processes involved in the workflow of creating lineart with this tool, such as Houdini presets and rendering out cryptomattes based on custom attributes.

4.4.2 Interface Design

The interface for the tool has been designed with the possible workflow when using it in mind. The base curve controls, animation controls and raycasting are all set as collapsible sections at the top of the menu, open by default. With the base curve and animation controls being the main settings for the overall

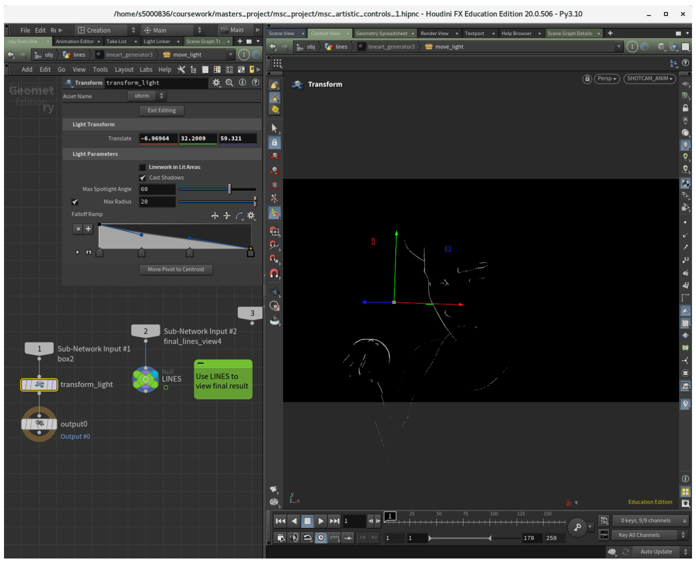

/home/s5000836/coursework/masters_project/msc_project/msc_artistic_controls_1.hipnc - Houdini FX Education Edition 20.0.506 - Py3.10
Transform transform_light
Asset Name xform
Exit Editing
Light Transform
Translate -6.96964 32.2009 59.321
Light Parameters
Linework in Lit Areas
Cast Shadows
Max Spotlight Angle 60
Max Radius 20
Falloff Ramp
Move Pivot to Centroid
Sub-Network Input #1 box2
Sub-Network Input #2 final_lines_view4
transform_light
LINES
Use LINES to view final result
output0
Transform

Figure 4.8: Interface for editing the "light" for raycasting lineart option.

lineart positioning and motion they are at the top. This invites the user to play around with them first before moving on to other sections. Once the user is done with those basic settings they can collapse them to remove the clutter while continuing with more detailed stylisation options.

The raycasting section providing different controls compared to other sections is also at the top. Since it provides a unique result, it is best if the user is aware of it first to decide whether it is relevant to their current goals. When used, the "Move Light" button opens an interface for "light" editing which includes both the handle for translating it as well as all the other controls from the main tool to let the user edit it in both interfaces as seen in Figure 4.8.

The width, gaps and displacement settings are split into tabs with almost identical controls available for each of them. The similarity of the controls provides a comfortable user experience while the collapsible nature of the tabs means an absence of clutter allowing the user to focus on one parameter at a time.

Overall the menu flows in a natural progression of building up the look of the lineart. Starting with base curve tuning and setting the redraw rate for animation, moving on to stylisation from the most extreme option to more thorough controls allowing tweaks and edits as well as adding noise to create a natural messy look.

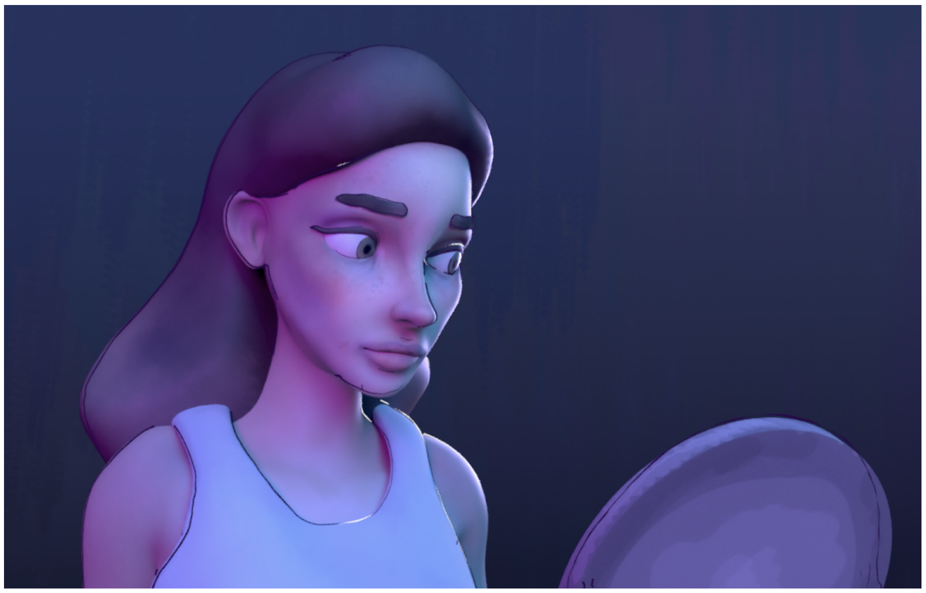


Figure 4.9: Shot from a fully rendered scene with lineart generated by the Animated Lineart Tool fully composited in.

4.5 Output and Use

The output of the tool is a 3D curve with a constant shader applied to it. It has position, normal and width point attributes, as well as any other attributes passed to it from the input geometry as part from the original materials. This means that the curve can be rendered as a solid colour mask and edited in the composition stage by adding texture and colour to it if needed. It also means that cryptomatte images can be rendered as long as it is possible for the original geometry.

The Houdini preset functionality is also supported, meaning once user is satisfied with the tool settings they can be saved as a preset to be loaded later in other scenes. This, however, does not extend to map painting and "light" moving.

Chapter 5

Conclusion

5.1 Reflections/Critique

There are many improvements - big and small - that would benefit this Animated Lineart Tool, including some features available in Kismet, which this tool is based on.

First of all, currently, the tool would only work for individual artists or small productions. To successfully scale it up, some additional pipeline tools are needed. Most importantly a command line batch processing tool would be necessary. While at the moment it supports preset functionality, which can speed up the process of generating the lineart for the same characters and scenarios, the tool still needs to be added manually into the scene. This process can be greatly sped up by a script that can be run on multiple shot files with necessary information such as presets to be used as well as render pass options, which would generate necessary lineart and render it out as needed. This specific enhancement was a low priority for this project due to the nature of potential users not being a large-scale production.

Another big enhancement for this tool which is also present in Kismet would be a smart animation system which would automatically change the redraw rate based on the rate of change of the parts of the model. This would make the effect more natural while also saving the artist some time spent calibrating the redraw rate. This could first be approached by separating the redraw rate of different primitives in the scene (potentially providing the user with controls for each to be tweaked manually in the beginning), further improved by checking the position of the points from each primitive in previous frames to calculate the rate of change the prim and use it to drive the redraw rate value. While this would greatly improve the tool and the final result it generates, it is a fairly complex enhancement that would take considerable time to implement properly.

Moreover, there are various potential additions to the interface and artist controls that can be enhanced. This includes but is not limited to the following ideas:

- More types of lines such as hatching and squiggles.
- More raycasting options, such as promoting all lights that already exist in the scene as objects influencing the lineart.
- Finer controls for painting masks and noise, allowing to add multiple of layers of both, for example, layer noise of different frequencies, or paint maps for different width values and combining them.
- Painting (and importing) masks on the "frame"/camera plane rather than the mesh itself.
- Transparency option with different settings for the front and occluded parts of the object.
- More varied tapering options.
- Option to paint not only gaps positions but lines positions as well.
- Generating inlines in addition to outlines and manual inlines painting.

These types of extra enhancements would allow finer control for the artist while providing a larger variation of styles to be created. However, many of these can be achieved at the compositing stage even with the current tool, so these extras would only save time rather than introduce something completely new.

All of this being said, the base of the main tool has already been created and is not only useful on its own but also lays the groundwork for future improvements. For example, since the tool already passes through the original attributes from the given mesh, allowing parts of the model or different models to be tagged with unique IDs, these can be used to differentiate between parts of the model to track the individual rate of change to calculate the redraw rate.

5.2 Conclusion

This project had been concluded with a Houdini Animated Lineart Tool being created and tested. As aims set in the beginning of the project it does create a convincing lineart in 3D space, while redrawing it naturally without being stuck on the model and features a user-friendly interface with a variety of artistic controls. Overall, the project is a success with plenty of room for improvement.

Bibliography

Danner, Simon and Christoph Winklhofer (Jan. 2008). “Cartoon Style Rendering”. PhD thesis. DOI: `10.13140/2.1.3664.0323`.

Decaudin, Philippe (June 1996). *Cartoon Looking Rendering of 3D Scenes.* Research Report 2919. INRIA. URL: `http://phildec.users.sf.net/Research/RR-2919.php`.

Failes, Ian (2023a). *‘We wanted to imply teenage imperfection everywhere’*. URL: `https://beforesandafters.com/2023/09/07/we-wanted-to-imply-teenage-imperfection-everywhere/` (visited on 09/07/2019).

— (2023b). *This was the first CG animated movie I’ve ever heard of that actually had a dedicated inking team.* URL: `https://beforesandafters.com/2023/06/28/this-was-the-first-cg-animated-movie-ive-ever-heard-of-that-actually-had-a-dedicated-inking-team/` (visited on 06/28/2023).

Grochola, Pav et al. (2023). “Linework in Spider-Man Across the Spider-Verse: An artistic driven approach to linework generation”. In: *ACM SIGGRAPH 2023 Talks.* URL: `https://dl.acm.org/doi/pdf/10.1145/3587421.3595456`.

Mao, Xiangyu et al. (2015). “Region-based structure line detection for cartoons”. In: *Computational Visual Media* 1, pp. 69–78. URL: `https://api.semanticscholar.org/CorpusID:17258714`.

Mikros Animation (2023). *Mikros Animation - Teenage Mutant Ninja Turtles: Mutant Mayhem Breakdown Reel.* Youtube. URL: `https://www.youtube.com/watch?v=N4eKniLJxzM&t=9s`.

Pav Grochola interviewed by Ian Failes (2024). *Giving ‘Spider-Man: Across the Spider-Verse’ that crucial hand-made quality.* URL: `https://beforesandafters.com/2024/01/10/giving-spider-man-across-the-spider-verse-that-crucial-hand-made-quality/` (visited on 01/10/2024).

Seymour, Mike (2019). *Ink Lines and Machine Learning.* URL: `https://www.fxguide.com/fxfeatured/ink-lines-and-machine-learning/` (visited on 05/10/2019).

SIGGRAPH Conferences (2021). *‘The Handmade Look of ‘The Mitchells vs. The Machines”.* URL: `https://blog.siggraph.org/2021/12/the-handmade-look-of-the-mitchells-vs-the-machines.html/` (visited on 12/17/2021).

Whited, Brian et al. (2012). “Computer-assisted animation of line and paint in Disney’s Paperman”. In: *ACM SIGGRAPH 2012 Talks*. SIGGRAPH ’12. Los Angeles, California: Association for Computing Machinery. ISBN: 9781450316835. DOI: `10.1145/2343045.2343071`. URL: `https://doi.org/10.1145/2343045.2343071`.

Yuan, Manli et al. (2007). “GPU-based rendering and animation for Chinese painting cartoon”. In: *Proceedings of Graphics Interface 2007*. GI ’07. Montreal, Canada: Association for Computing Machinery, pp. 57–61. ISBN: 9781568813370. DOI: `10.1145/1268517.1268529`. URL: `https://doi.org/10.1145/1268517.1268529`.

Zhang, Song-Hai et al. (2009). “Vectorizing Cartoon Animations”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.4, pp. 618–629. DOI: `10.1109/TVCG.2009.9`.