Master of Science in Informatics at Grenoble
Master Informatique
Specialization Graphics, Vision and Robotics

Procedural Stylization

Isnel Maxime

June 2019

Research project performed at MAVERICK

Under the supervision of:
Romain Vergne
Joëlle Thollot

Defended before a jury composed of:
James Crowley
Dominique Vaufreydaz

June 2019

Abstract

Your abstract goes here...

Acknowledgement

I would like to express my sincere gratitude to .. for his invaluable assistance and comments in reviewing this report... Good luck :)

Résumé

Your abstract in French goes here...

Contents

— 1 —
Introduction

1.1 Background

1.2 Problem Statement

A part of the computer graphics is create non-photorealistic images. A method to do this is to stylize 3D objects and 3D scenes. Stylize an object means create an image that imitates the style of an artist who would have drawn it on sheet of paper. There exist many different styles like hand drawing, brush painting, pointillism painting, stippling, watercolor painting, etc.

The main problem of stylizing a 3D object in an animation is the *temporal coherence*. The *temporal coherence* problem in non-photorealistic rendering encompasses both spatial and temporal aspects of the marks. The effect given by the stylization has to be kept if the object is moving, rotating and scaling. Many research has been done to solve this problem of *temporal coherence* [28, 7, 3]. Bénard et al. separate this problem is three sections inspired by previous work[19, 11, 5, 8] the ideal solution (Figure 1.1 a) could correspond to something drawn by an artist at each frame. Neglect one of this three goals provide artifacts (Figure 1.1 b-d).

1.2.1 Flatness

The impression of drawing on a flat surface gives the *flatness*. The stylization has a good *flatness* if the image rendered has a good 2D appearance. In order to keep this effect, the size and the distribution of the marks of your stylization have to be independent of the distance between the stylized object and the camera.

1.2.2 Motion Coherence

Motion coherence is a correlation between the motion of marks and the motion of the 3D object. Bad *Motion coherence* will give the impression to see the scene through a semi-transparent layer of marks, this is called *shower door* effect [19], an example to illustrate what happens when there is a bad *Motion coherence* is the movie *Loving Vincent*[2]. The goal is to provide in 2D screen space a perceptual impression of motion as close as possible to the 3D displacement in object space.

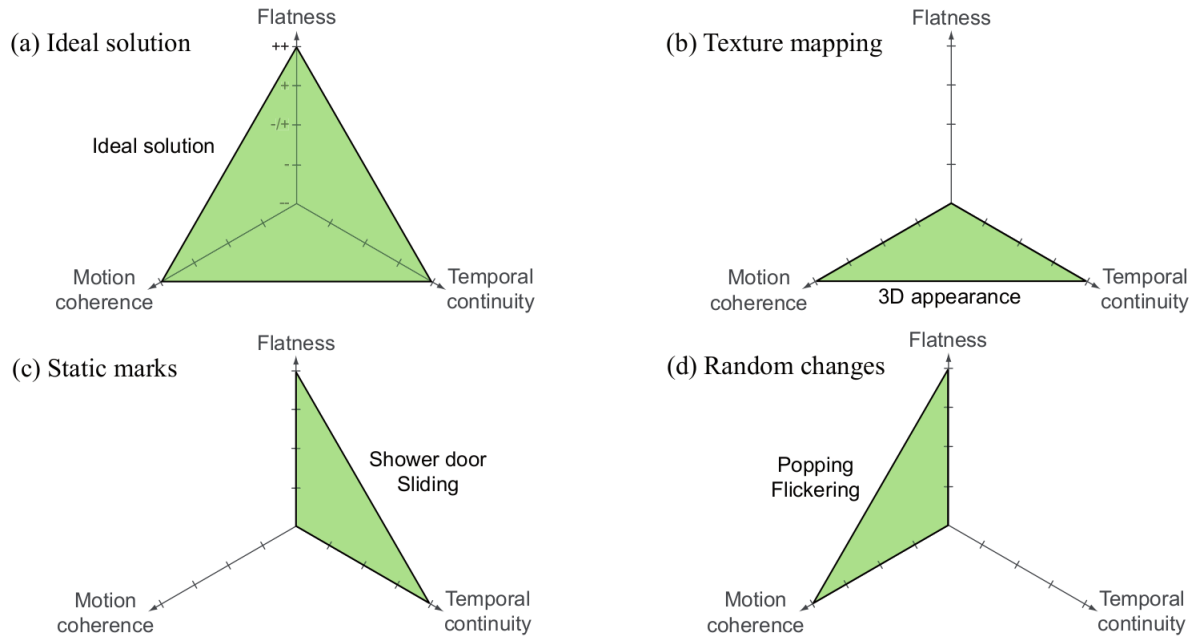

Figure 1.1: Problem involve by temporal coherence depending of the flatness, temporal continuity, motion coherence.

1.2.3 Temporal continuity

Temporal continuity is the quality of minimizing changes from frame to frame to ensure fluid animations. In order to have good *temporal continuity*, the marks of the image have to fade slowly during the animation. Human perception is very sensitive to *temporal incoherence* according to some perceptual studies[29, 25].

The problem introduced in the works of Bénard et al.[8] is the difficulty to have a ideal solution, the one that has a good *flatness*, a good *motion coherence* and a good *temporal continuity*. These three goals are inherently contradictory when you improve one you neglect one or maybe more. So researchers work to find solutions that make *trade-offs* between these three goals. We built our approach trying to take the better of each different approach, looking at what work well and what can be integrated in our method.

1.3 Scientific approach

1.4 Contents of this report

— 2 —
Previous Work

Many research have been done in stylizing 3D scene[24, 23, 16, 7, 6, 13, 8] trying to propose solutions or trade-offs for the problem of *temporal coherence*. In this part of this report, we will present you techniques to stylize 3D scenes and we will show their advantages and disadvantages. In order to render an image of a 3D object in the screen, a graphics program goes through several steps that compute some different information like the gradient of the image, the shadows made by the object, the amount of light received by the object, etc. The gathering of all these steps is called **graphical pipeline rendering**. In this graphical pipeline, there are two moments when we can stylize the objects. The first is when we computed information about the geometry of each object in the scene, we call it *object space*. The second moment is when we gather the previously computed images of the scene in order to make for example shadows, global illumination, ambient occlusion, etc. we call it the *image space*. We will treat these two space separately and with the two different types of methods to stylize.

2.1 Object Space

In the object space, we work on the surface of the object and so we have all the knowledge about the geometry.

Texture-based methods

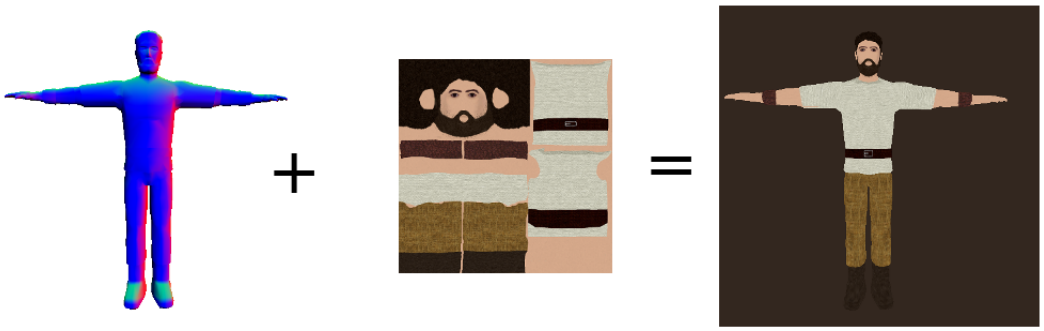

Figure 2.1: Texture mapping: example

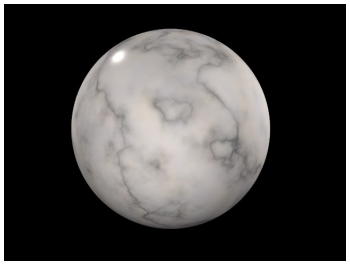

Figure 2.2: Marble with procedural texture

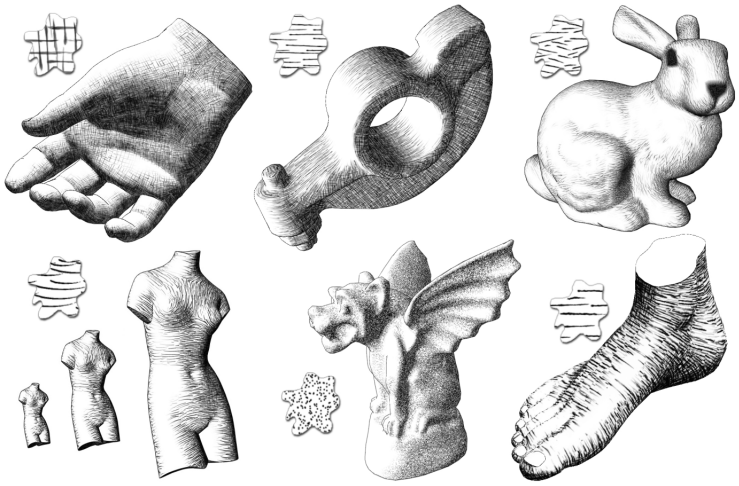

Figure 2.3: Real time hatching rendering [23]

Figure 2.4: Watercolor stylization with procedural textures [7]

Figure 2.5: OverCoat: implicit canvas [24]

One of the most used ways to colored object in 3D is the *texture mapping*. It consists of mapping an image on the object (Figure 2.12.3). This technique is widely used in video games because it is easy to implement, it can be implemented for GPU and it needs low computation. As shown in the example, texture mapping can be used with images as a texture in order to stylize the object[23, 16, 13]. Texture mapping can also be done with *procedural noises*[22] in this case called *procedural textures*. Procedural textures are mathematically computed from coordinates the most famous is Perlin but there exist others like Gabor noise, Worley noise, etc. With these textures, we can create images looks like marble (Figure 2.2) or realist wood. Bénard et al.[7, 6] use this method of texture mapping with noise in order to make watercolor stylization (Figure 2.4).

This method has a problem of *foreshortening* on the silhouettes which makes an area with more elements than in the center of the object and when zooming in/out the size of the elements in the mapped texture vary depending on the distance from the camera, if we get closer to the object the elements on the texture become bigger and bigger going up to pixelization sometimes. These two problems make texture mapping bad in term of *flatness* because an artist does not draw bigger strokes if the object is close and does not draw more details at the silhouettes than in the center of his objects. Texture mapping is also bad in term of variety of style that you can do because you are depending on the texture chosen you cannot, for instance, do a rendered image that looks like painted with brushes and especially you cannot change the shape of the object in the rendered image, a perfect sphere will always appear as a perfect sphere. On the other hand, this method naturally ensures *motion coherence* and *temporal continuity* because the texture is mapped directly on the surface of the object.

Mark based methods

The natural way to stylize 3D objects is to as an artist apply paint strokes on the object. These paint strokes can be represented with smalls images also called splat. Some works [19, 1, 10](more in the state of the art [8]) use point distribution in order to make anchor points for splats. These point distributions are often computed in image space and then are projected on the model. Anchor these splats to the model improve the *motion coherence* because each splat will follow the motion of the 3D model. These splats are rendered in the image space as a 2D sprites so preserved the *flatness*. The problem is how to have the point distribution and how can we control it in order to have a uniform, not too sparse and not too dense distribution. Moreover, these point distribution does not provide control over the *temporal continuity*. In our

Figure 2.6: Image filter for watercolorization [20]

method, we use procedural noise to anchor the splats.

Daniels[12] and Schmid[24] propose to project splat on the geometry and stored them in nodes but anchored to the geometry of the model but this technique is expensive in term of storage. OverCoat[24] is an interactive software that permit to the user to stylize manually a 3D object (Overcoat demo). Their techniques are the closer techniques to our proposed solution. In their solution, they use 3D splines drawn by the user and they display images along these lines. Each spline is stored in nodes that contains the image to splat along this line, the color, the thickness and the transparency that permit the user to edit the stylization after drawing the splines. These splines are anchored on a proxy generated by the program this has the disadvantage of placing the stylization at the beginning of the graphical pipeline. It has a problem of *foreshortening* of the silhouettes that make the rendering blurred at some place of the image. Overcoat has a big problem of flatness when rescaling the stylized object, this is because the user draws marks at a specific scale so when we zoom in the scene the marks looks bigger or smaller and that affect the *flatness* of the rendered image. All the marks drawn are anchored to the geometry so it ensures the *motion coherence* and because they use a proxy geometry the shape of the object can be changed, some details can be added.

The mark based methods have a problem of foreshortening has said before and the non-constant size of the marks when rescaling of the object make the impression of bad *flatness*. But the *motion coherence* is ensured by anchoring the marks to geometry that makes them follow the motion of the stylized object.

2.2 Image space

In image space, we work on images with 2d coordinates that means working on the pixel of the image directly. In a normal pipeline rendering, the image space correspond to the *compositing* which is the moment when many information have all ready be computed and are stored in images (like shadow map, image filter, global illumination, etc.), during the *compositing* we combine the computed images to render the final image<insérer screen gratin>.

Texture-based methods

Many methods to stylize in image space used texture based approaches. It consists to apply

Figure 2.7: Hertzman's algorithm on an image [26]

the texture to the entire image (Figure 2.6) [8, 20] but in the case of stylizing animated scenes, the problem is how do we deform the texture to minimize the apparition of sliding artefacts. Sliding artefacts are some parts of the image that do not move according to scene motion and give the impression that they are sliding. We can distinguish two families of approaches to solve this problem. The first family of approaches use an approximation of the 3D camera motion with 2D transformations of the texture[11]. This gives a nice trade-off between *motion coherence* and *flatness* but it is limited to static scenes and a set of few camera motions. Moreover, sliding artefacts still occur with strong parallax so Fung et al.[14] and Breslav et al.[5] improve the approximation of the scene motion in order to reduce sliding artefacts.

The second family of approaches use deformations to animate the texture[4]. These deformations are computed from the optical flow of a video. This is an extension of the methods used in vector field visualization by Neyret[21]. These deformations can distort the texture and alter the original pattern. The method of Bousseau et al.[4] is very effective with stochastic textures as the fractalization process but creates artefacts with structured patterns.

These texture based methods for 3d scenes have issues with *motion coherence* and *temporal continuity* because of the independence of each frame and also these techniques suffer to make a poor variety of style, you cannot do for example hairs, brushes painting, pencil drawings. But they can do some different styles with good *flatness*.

Mark based methods

A method very used to stylize in image space consists to draw strokes/splats at some place of the image[3, 28, 9, 30, 15]. It is like adding others images in an image (like you can do in Photoshop or Gimp, Figure 2.7). The question of these mark based method is where do we place these marks in order to have a stylized rendering without loosing the meaning of the scene and without loosing details. A first approach is to extract lines that are relevant like the silhouettes, etc. [28, 15, 17] and then stylize the image with this information, like keeping only the extracted lines and change the shape of each line or apply strokes along these lines as

Figure 2.8: Example segmentation image[8]

	Motion coherence	**Flatness**	**Temporal continuity**	**Style variation**
Object space				
Texture-based methods	++	- -	++	- -
Mark based methods	++	+/-	+/-	+/-
Image space				
Texture-based methods	-	++	-	- -
Mark based methods	-	++	- -	+

Figure 2.9: Summary of trade-offs made in different approaches

Vergne et al.[28] did try to have a good *temporal coherence*. The problem of these techniques is the popping marks due to a bad *temporal continuity* because we do not know if in the next frame there will be more details or not or the color to display is different from the current one. A second approach is to segment the image (Figure 2.8) in order to have the different parts of the scene[30, 18]. Thanks to this segmentation, they apply different strokes for each part of the image with the corresponding colors. The work of Lin et al.[18] is about videos so they use the optical flow of the videos in order to have a good *temporal coherence*. These mark based methods have a good impression of *flatness* thanks to the splatting in image space, this is something that we will use in our approach.

These methods of splatting have almost the same problem that the texture based method they suffer of bad *motion coherence* and *temporal continuity*. Temporal continuity that can be improve using a motion flow which cannot be possible in our case. In the other hand, changing the type of splat, the size and their orientation we can make many different styles, like brushes painting, pencil drawings, stippling, hairs. These method draw 2d strokes/burshes in the image so it looks like drawn by a person on a flat surface which makes a good *flatness* impression.

The figure 2.9 is a summary of the previous works. The interesting point is mark based method can make a wider variety of styles than the texture based method. We can see that for *motion coherence* and *temporal continuity* doing the stylization in object space is better than in image space but in image space we can do stylization that have a good *flatness* impression contrary to the object space.

— 3 —
Realisation

3.1 Overview

As explains above in the state of the art and in the figure2.9 each approach has its advantages and its disadvantages. That is why in our solution we tried to take the better of the two worlds. We stylize the 3D scene in image space (screen space) but with all the information about the 3D object and the camera (camera matrices, position, normals, tangents, UV coordinates, distance from the camera). This solution permits to apply something like 2D images on the screen so have a good *flatness* while keeping the information on the silhouettes, the orientation, the depth, etc. This solution permits also to easily integrate the stylizing of a scene in a pipeline rendering because it can be done at the end during the post-processing rendering pass.

We chose to use mark based methods to stylize our scene because texture based methods in

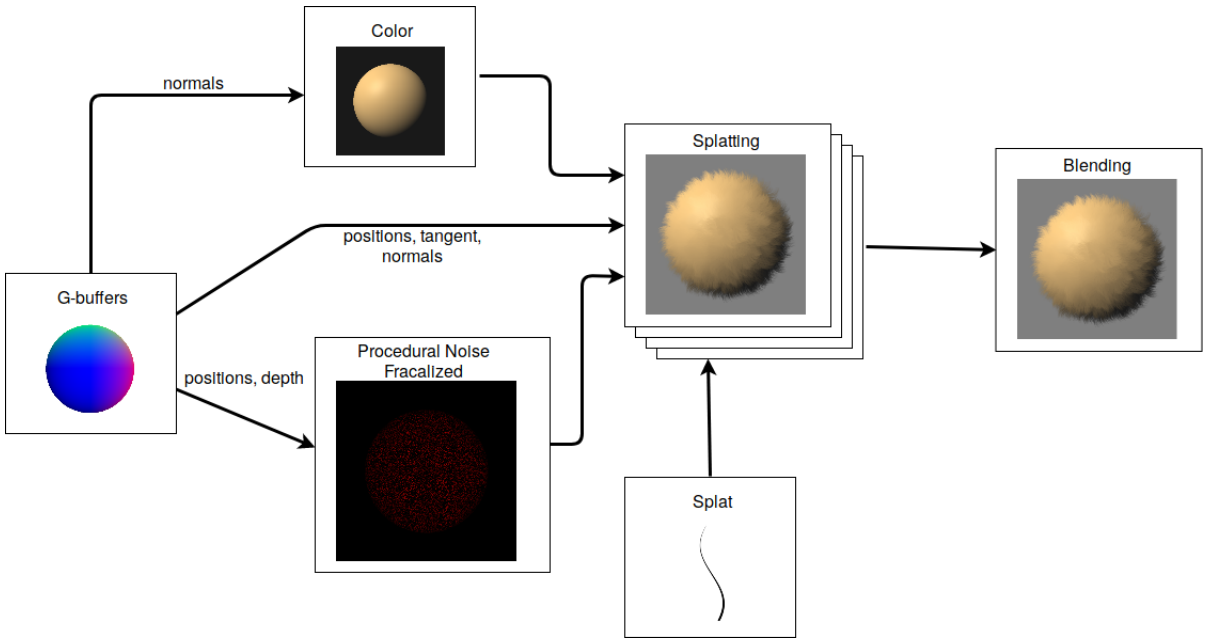

Figure 3.1: Overview of our solution.

Figure 3.2: Examples of procedural texture: *left using perlin noise, middle using Gabor noise,right using worley noise*.

image space give a poor variety of styles as said in the work of Bénard et al.[7]. This mark based method implies to decide where in the image the splats will be drawn. In our problem, the goal is to anchor these splats with the objects in order to have the same motion for the splats and the object. This avoid the problem of *shower door effect* and ensure the good *motion coherence*. So we needed anchor points depending on the position of our object. Therefore in our approach, we used procedural noise[22] as a texture of our 3D object. The procedural noises are easy to implement, fast to compute and easy to manipulate. Like every texture computed in object space, it has a good motion coherence. Each value different of zero of this texture represents an anchor point for a splat.

3.2 Procedural noise and fractalization

We compute procedural texture in order to create anchor points. Procedural noises are *pseudorandom* gradient of grid point. In computer grapchis, they are usually used in 2d 3.2 as an image or in 3d for texturing a 3d object, in our case we use it in 3d. This texture are computed from procedural noise with a mathematical process. There exist many procedural noise such as Perlin noise, Worley noise, Gabor noise, Value noise, Gradient noise, etc. In order to map the procedural texture to the 3d object, we compute it with the vector position of each vertex of the object. The frequency of a noise control how many details there is in the texture. With a Worley noise, increasing the frequency will increase the number of black area in the texture.

In our case we used the procedural texture to anchor the splat. For each pixel of the final image we "paste" a splat if the current pixel correspond to black pixel in the procedural texture the splat is not displayed (see Figure 3.3). Thanks to this mechanism we can control the density of splat in the image. The value of our noise in the texture is between 0 and 1. The opacity vary according to this value of the procedural noise. One problem of working in image space is the aliasing, it creates some problem with *temporal continuity* and *motion coherence*. To reduce the aliasing in the procedural texture we make multiple samples with very small variation and we do an average to have the final value.

We add a threshold parameter to compute the noise in order to reduce the number of splat to display but especially to have small points in the procedural texture if not the splats convolve as you can see in the rendered image in the figure 3.3.

We mainly work with the Worley noise which is a cellular noise as you can see at the right

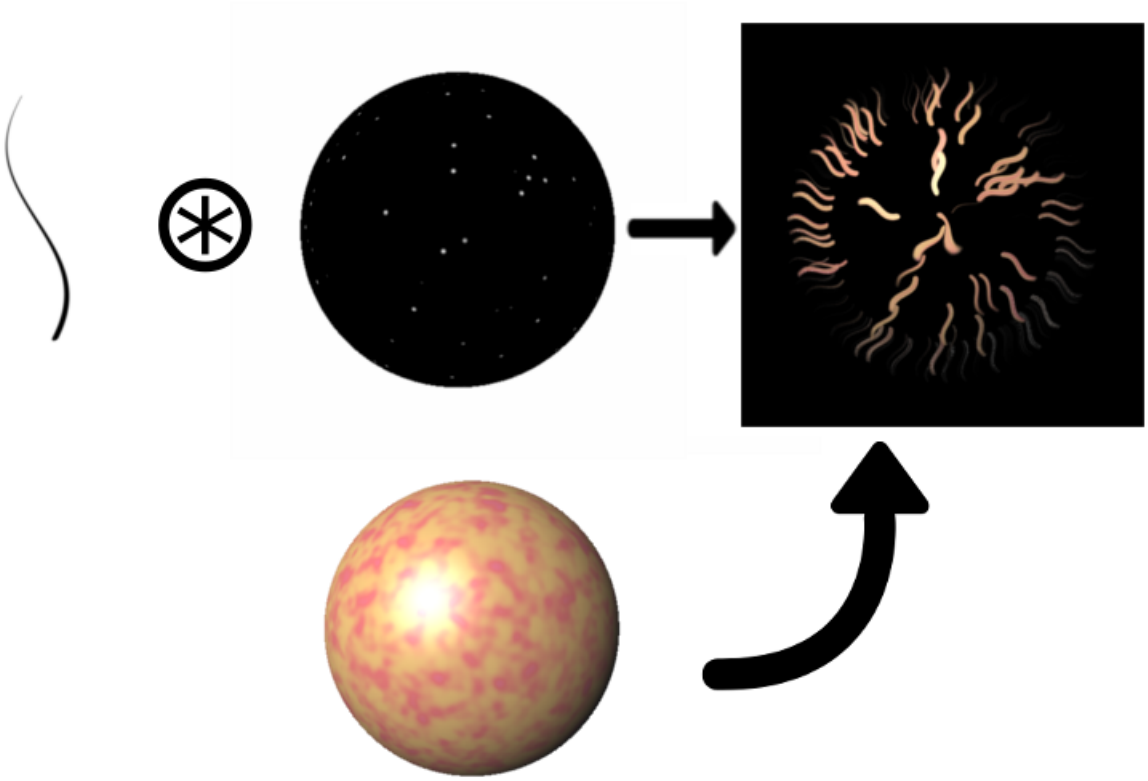

Figure 3.3: Usage of procedural texture to anchor splats: (left: splat image, middle: procedural teture, right: rendered image, bottom: color).

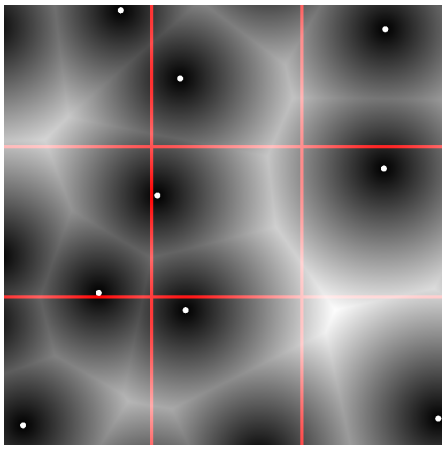

Figure 3.4: Worley noise in 2d.

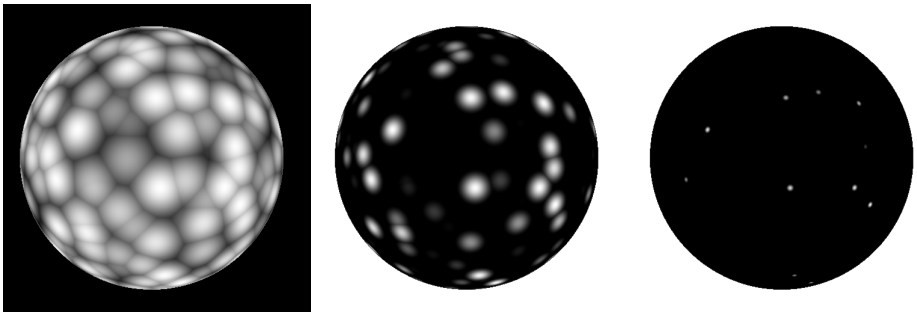

Figure 3.5: Use of threshold in the case of Worley noise with the same frequency.

in the figure 3.2. To create a procedural texture from a Worley noise, we divide the image in cells (squares with the same size in 2d and cubes with the same size in 3d, red lines in the figure 3.4), then in each square (cube in 3d) we compute a point with a seeded random (white dot in the figure 3.4) and finally at each pixel of the texture we put as value the distance with the closest point computed previously with the random seed. Increase the frequency of this noise correspond to divide the image with smaller cells and so have more cells in the image. In our pipeline we use the inverse of this noise (1-worleynoise) in order to have the white pixels of the procedural texture as anchor points.

As we said before we use a threshold for the purpose of having an another control on the noise. With the Worley noise this threshold correspond to the "size" of the points. If we increase the threshold the points become smaller as you can see in the figure 3.5. In order to facilate the control of this noise we adapt this threshold according to frequency and the distance from the camera. That means that the size of each point is constant if the frequency or/and the distance from the camera is changed.

Fractalization

Bénard et *al.*[6] introduce the mechanism of fractalization on textures. This technique creates an impression of infinite zoom effect (like in this example: ShaderToy of Neyret Fabrice). In our method of stylizing it improves the *temporal continuity* because there is always display even if you get very close to the object and it also improves the *flatness* impression because there is almost the same number of anchor points in the screen and their size is quasi-constant. This method alters the original pattern of the texture as you can see in the figure 3.6 it can be an issue because some pattern cannot be fractalized (like the checkboard pattern) but it works well with stochastic textures. To creates this fractalization, they use textures at multiple frequencies (not necessary procedural texture) (see figure 3.6) and they combine them with the transparency and they overlap them according to the distance from the camera as you can see in the results of the figure 3.6.

The figure 3.6 show 4 noises (octaves) at different frequency it is a zooming cycle. As you can see, if the distance from the camera is close to 0 the frequency of the noises is high and if we move away from the object the frequency decrease. In the figure, we have: `2*frequency(octave 1) = frequency(octave 2)` and `2*frequency(octave 2) = frequency(octav 3)` and etc. because in this case they divide the distance from the camera with a NON ! *log2* so when the distance from camera is twice as large the frequency of the noise displayed is multi-

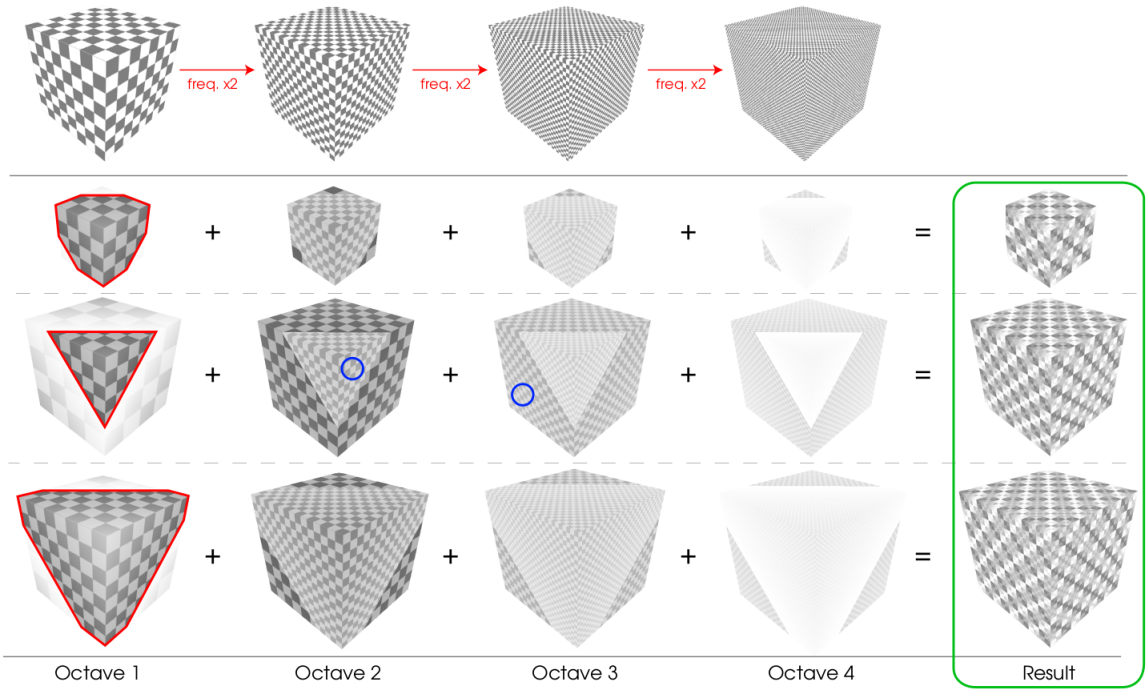

Figure 3.6: Principle of fractalization [6].

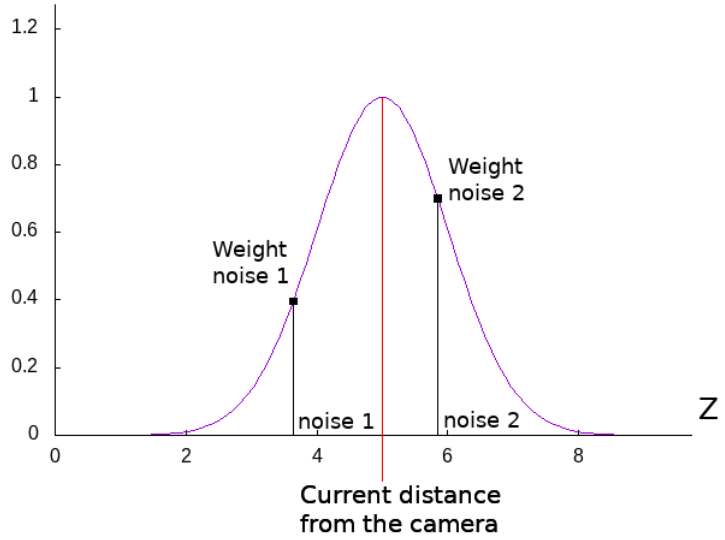

Figure 3.7: Combination of the noises with the weight value in this case with 2 noises.

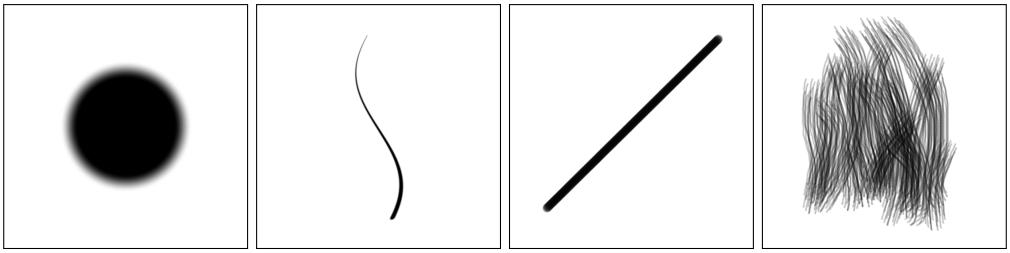

Figure 3.8: Example of what the splats can be.

plied by 2. During the zoom, we modulate each octave with a weight. To compute the weight of each noise we use Gaussian interpolation centered on our current distance from the camera (see figure 3.7) with a sigma depending on the number of noises that we want to combine.

3.3 Splatting

The marks based method try to imitate how the artists do when they stylize. They draw on a flat surface that gives a good impression of *flatness*. We use the same principle to stylize our scene. We put splats/marks directly on the image like an artist will do (illustrated in the figure 3.3).

In our solution, the user controls the splat image he can use anything. These marks can be leaves, hairs, dots, feathers, lines, paint brushes, etc. They can also be created from a procedural noise. The user has also the control on the size of these splats, he has a global control of the size and control on a specific part of the object through a texture. The rotation of the splats is also a parameter that the user can control. He can choose to rotate all the splat, for example, depending on the normals or depending on the tangents.

Artists control how their marks are combined for example if he is doing painting he could want to have non-transparent marks in order to cover the marks behind the new one. He could want more transparency if for example he wants to do watercolorization.

In our method, we take this into account during the blending of all marks. The user can control how many marks the stylization will mix and control the transparency (alpha blending). For example, the user can choose to only show the marks at the top of the flat surface (the closest to the camera) or he can choose to mix some marks in order to account some marks covered.

3.4 Stylization

As explained in the previous section, in our method, the user control the content of the splats, their size, their orientation, the density of splat on the object and how they are blended with the alpha blending. In this section, we will show you some examples of how we did specific style using our pipeline.

Figure 3.9: Example of pointillism with our method.

Pointillism

In order to make pointillism style we use a dot as splat (gaussian fonction in 2d, the first in the figure 3.8), the procedural noise to control the number of splats has a high frequency with a high threshold in order to have small anchor points, the size of the splats can vay depending on which size of brush he wants to imitate but usually we use small splat size for this style and for the alpha blending it usually adapt to not mix many marks. In this case the orientation does not matter. The figure 3.9 shows the final rendered image.

Brush painting

In order to make an image made like a painter we use an image of a paint strokes (like the fourth in the figure 3.8), we do not need too much splat displayed so the frequency of the procedural noise do not have to be very high but a threshold high/medium, the size of the splats depend of the 3d model but usually we use large splat. For the alpha blending it depends on the effect wanted, in our case we mixed many splats. And for the orientation usually the artists align the strokes with the tangents but it works well if the strokes are horizontal. The figure 3.10 shows the final rendered image.

Hair rendering

In order to show that we have the control of the *flatness* impression we tried to create hairs with a 3d aspect like it can be done in digital painting (see figure 3.11). To do this we use a splat created mathematically that looks like a hair (the second in the figure 3.8) rotated according to the normals of the object. With the procedural noise that is used to anchor marks, the density can be varied impacting directly the density of hairs in the fur. The size of the splats varies depending on what size you want your hair to be. And for the alpha blending, you do not want to mix too many splats but you do not want to only show the last hairs drawn. The figure 3.11 shows the final rendered image.

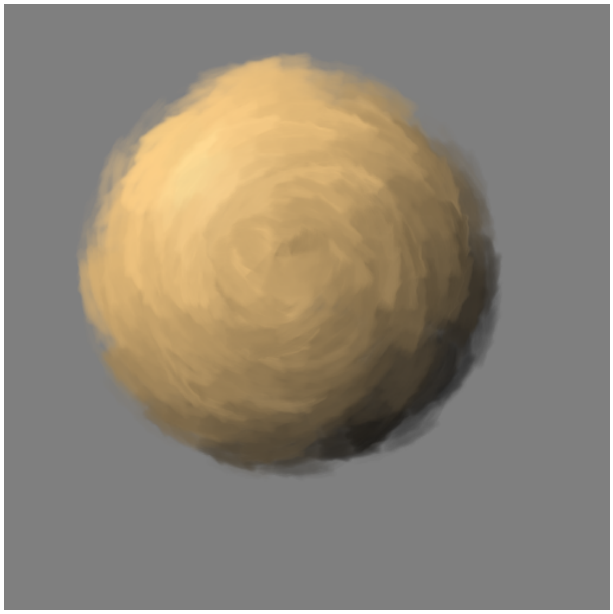

Figure 3.10: Example of painting with burshes with our method.

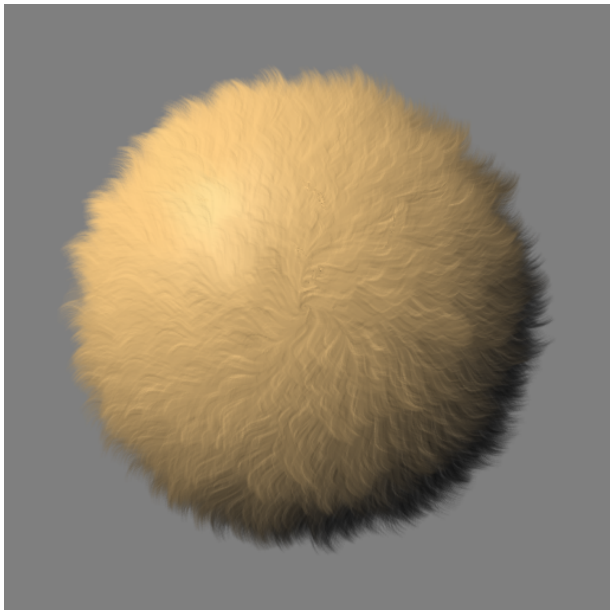

Figure 3.11: Example of hairs rendering with our method.

Here there are 3 examples of how we use the method to do some styles but the same styles can be done differently and others styles can be done as well. These 3 examples show what is possible to do with this method and show the control of the *flatness* we have.

— 4 —
Practical implementation

During our research on stylization, we used Gratin [27] which a pipeline rendering software using *OpenGL*. It permits to easily creates rendered images with *GLSL* combining previous images as textures <insert screenshot of gratin>. As we said before every texture used from our contribution can be computed in parallel on GPU including the noises, the fractalization of the noises, the alpha blending and the splatting. Our method use only images as input so it can easily be integrated into every pipeline rendering. For example, if you have a pipeline rendering with *path tracing* to compute the color of your scene you can use this rendered image as color input of our method without re-compute it.

For the splatting step, we draw on each pixel of the screen a square centered on the current pixel (see figure 4.1). Every square is treated independently on the GPU before in the *vertex shader* which manipulate the 4 vertices of the square which permits the resizing of the splat and permits the rotation of the splat. After this step of resizing and rotation, we pass in the *fragment shader* which manipulate all pixels of splat. It is in this step that we will decide if we display the splat or not thanks to the procedural texture previously computed.

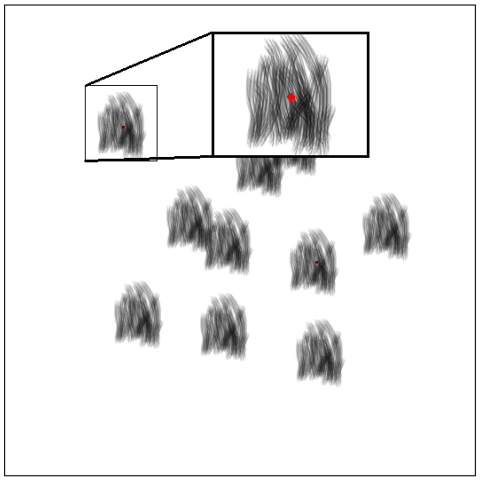

Figure 4.1: Splatting principle: red dots are the anchor points and the splat is centered and anchored to this point.

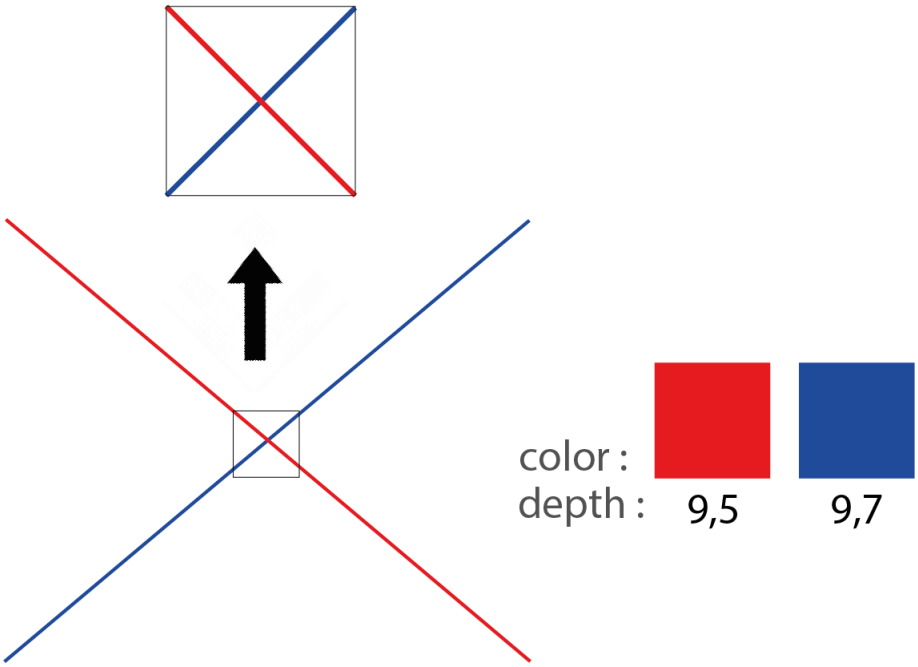

Figure 4.2: Example of usage of the order independant transparency.

In this *fragment shader*, we prepare the technique of *order-independent transparency* that consists a structure that permit to know the order of the splats depending on the depth. This structure is a matrix of array containing the color and the depth of the splats (in our case the depth of the vertex where the splat is anchored). The size of this matrix is the same that the images (height x width). In our example in the figure 4.1, we have some splats that cover others splats. How do we know which one is in front of the other if we treat them independently? For example, if there is a pixel where splat covers another one, we construct an array with 2 elements composed of color and a depth so we can know which one is in front of the other one (see example in the figure 4.2). We apply the same technique but in our case we have a lot of splats that cover others and as depth to put in this array we use the depth corresponding to vertex of the stylized object where the splat is anchored (the red dots in the figure 4.1 correspond to a vertex of the 3d objects).

After creating these array, we will in the last pass treat each pixel of the screen independently and (still on GPU) decide which color has to be displayed. In order to do this, we sort the array corresponding to the current pixel according to the depth. And finally, we apply our algorithm of alpha blending:

```
vec4 over(in vec4 c1,in vec4 c2) {
    float a = (c1.a+(1.-c1.a)*c2.a);
    return vec4((c1.rgb*c1.a + c2.rgb*(1.-c1.a)*c2.a)/a,a);
}

vec4 alpha_blending(vec4 colorArray[], int count, vec4
   backgroundColor, float gammaBlend){
    // colorArray is the list of color sorted: colorArray[0]
       correspond to the farest
```

```
    // count correspond to the number of splat that we will take
       account during the blending
    // backgroundColor is the color of the background our image
    // gammaBlend is a user parameter that will increase the
       transparency of the displayed color

    vec4 finalColor = vec4(0,0,0,0);

    for(int i=count-1; i>=0; i--) {
        vec4 currentColor = colorArray[i];
        finalColor = over(finalColor,vec4(currentColor.rgb,clamp(pow(
           currentColor.a,1./gammaBlend),0.,1.)));
    }

    return over(finalColor,backgroundColor);
}
```

After this algorithm we have the color to display in the current pixel of the image/screen.

— 5 —
Results and performance

Here there are some results of our method of stylization. The figure 5.1 show some rendering with a point as mark and with differents color and models. We use the dot splat (the first in the figure 3.8) with a small size in order to keep a good level of details. In the figure 5.2 we tried to stylized as painted with brushes so we used brushe stroke as splat. In the figure 5.3, we tried to render hairs with our technique we a simple hair as splat. We rotated it according to the normals and to increase the realism we changed the size of the splat depending of a procedural noise to have in some areas of the object smaller splat and in some others areas bigger splat. In our work, we did not focused on the color of rendered color because it depends on the 3d model and because our solution tend to be integrate in more complex pipeline rendering where the color is computed before.

Animations

During our analysis of the state of the art, we talked about what happened if the camera moves. We made some videos to show the results of our solution with an animated camera. The videos are in the Maxime's website (http://www.maximeisnel.fr/procedural-stylization/). It is difficult to create a good *flatness* effect while moving, the animations of the brush painting is a good example at each frame we have something that has good *flatness* but during the motion we can see problem of *temporal continuity*. This due to the alpha blending and also mainly due to the size of the splat because they are big and so to improve *temporal continuity* we have to make to splat disappear/appear more slowly in order resolve some popping effect of the splats. As you can see in the animations of hairs rendering the splats are smaller so we don't have this problem of *temporal continuity*.

Performance

Our pipeline is implemented as a set of (non-optimized) OpenGL fragment shader in the Gratin software ([27]). Its performance is affected by the number of sampling during the creation of the procedural noises to minimize aliasing problem (popping effects). But the pipeline is mainly affected by the splatting step and the algorithm of order-independent transparency because in order to stylize the whole object we draw a lot of splat on the image so we have many splat cover others and then in order to know the order of them depending on the depth we sort with a *bubble sort* an array that contains at most 300 elements (arbitrary limits fixed depending on the GPU used) for each pixel of the final image. In other words, the performance depends on how many splats are drawn and their size, if the size of splat is big the splats will cover more splats that if their size is small and the complexity will be higher. For the animation

Figure 5.1: Results of pointillism.

of brush painting (present in Maxime's website) with 300 frames with a resolution of 512x512 pixels, it takes approximately 4 minutes with an Nvidia GeForce RTX 2070 graphics card to render.

Figure 5.2: Results of brush painting.

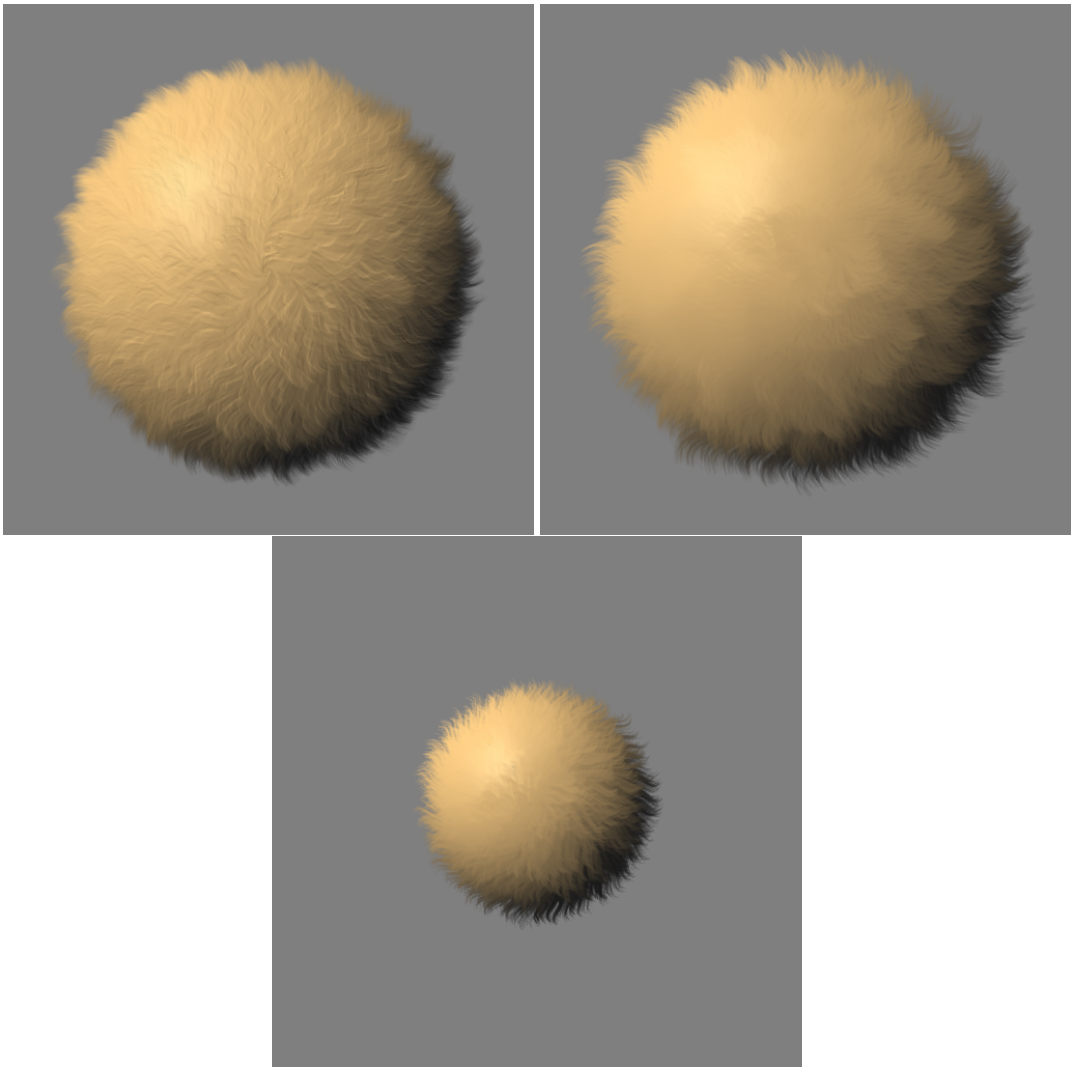

Figure 5.3: Results of hairs rendering.

— 6 —
Conclusion

6.0.1 Futur works

Bibliography

[1] *NPAR '00: Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, New York, NY, USA, 2000. ACM. ACM Order No: 434000.

[2] Loving vincent, 2017.

[3] Alexandre Bléron, Romain Vergne, Thomas Hurtut, and Joëlle Thollot. Motion-coherent stylization with screen-space image filters. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering - Expressive '18*, pages 1–13, Victoria, British Columbia, Canada, 2018. ACM Press.

[4] Adrien Bousseau, Fabrice Neyret, Joëlle Thollot, and David Salesin. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics*, 26(3):104, July 2007.

[5] Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, and Joëlle Thollot. Dynamic 2d Patterns for Shading 3d Scenes. page 6.

[6] P. Bénard, A. Lagae, P. Vangorp, S. Lefebvre, G. Drettakis, and J. Thollot. A Dynamic Noise Primitive for Coherent Stylization. *Computer Graphics Forum*, 29(4):1497–1506, August 2010.

[7] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. Dynamic solid textures for real-time coherent stylization. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*, page 121, Boston, Massachusetts, 2009. ACM Press.

[8] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. State-of-the-Art Report on Temporal Coherence for Stylized Animations. *Computer Graphics Forum*, 30(8):2367–2386, December 2011.

[9] Pierre Bénard, Jingwan Lu, Forrester Cole, Adam Finkelstein, and Joëlle Thollot. Active Strokes: Coherent Line Stylization for Animated 3d Models. page 10.

[10] Ming-Te Chi and Tong-Yee Lee. Stylized and abstract painterly rendering system using a multiscale segmented sphere hierarchy. *IEEE transactions on visualization and computer graphics*, 12(1):61–72, February 2006.

[11] Matthieu Cunzi, Joelle Thollot, Jean-Dominique Gascuel, Sylvain Paris, and Gilles Debunne. Dynamic Canvas for Non-Photorealistic Walkthroughs. page 10.

[12] Eric Daniels. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99 Electronic Art and Animation Catalog*, SIGGRAPH '99, pages 124–, New York, NY, USA, 1999. ACM.

[13] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. *Computer Graphics Forum*, 20(3):184–192, 2001.

[14] Jennifer Fung and Oleg Veryovka. "Pen-and-ink textures for real-time rendering". page 6.

[15] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Transactions on Graphics*, 29(2):1–20, March 2010.

[16] Allison W. Klein, Wilmot Li, Michael M. Kazhdan, Wagner T. Corrêa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 527–534, Not Known, 2000. ACM Press.

[17] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. Line drawings via abstracted shading. page 5.

[18] Liang Lin, Kun Zeng, Yizhou Wang, Ying-Qing Xu, and Song-Chun Zhu. Video Stylization: Painterly Rendering and Optimization with Content Extraction. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, page 13.

[19] Barbara J. Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, pages 477–484, Not Known, 1996. ACM Press.

[20] Santiago E. Montesdeoca, Hock Soon Seah, Amir Semmo, Pierre Bénard, Romain Vergne, Joëlle Thollot, and Davide Benvenuti. MNPR: a framework for real-time expressive non-photorealistic rendering of 3d computer graphics. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering - Expressive '18*, pages 1–11, Victoria, British Columbia, Canada, 2018. ACM Press.

[21] Fabrice Neyret. iMAGIS-GRAVIR† / IMAG-INRIA. page 8.

[22] Ken Perlin. Improving Noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. ACM. event-place: San Antonio, Texas.

[23] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, page 581, Not Known, 2001. ACM Press.

[24] Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W. Sumner. OverCoat: an implicit canvas for 3d painting. In *ACM SIGGRAPH 2011 papers on - SIGGRAPH '11*, page 1, Vancouver, British Columbia, Canada, 2011. ACM Press.

[25] Michael Schwarz and Marc Stamminger. On predicting visual popping in dynamic scenes. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*, APGV '09, pages 93–100, New York, NY, USA, 2009. ACM.

[26] David Vanderhaeghe and John Collomosse. Stroke Based Painterly Rendering. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, volume 42, pages 3–21. Springer London, London, 2013.

[27] Romain Vergne and Pascal Barla. Designing Gratin, A GPU-Tailored Node-Based System. *Journal of Computer Graphics Techniques*, 4(4):54–71, 2015.

[28] Romain Vergne, David Vanderhaeghe, Jiazhou Chen, Pascal Barla, Xavier Granier, and Christophe Schlick. Implicit Brushes for Stylized Line-based Rendering. *Computer Graphics Forum*, 30(2):513–522, April 2011.

[29] Steven Yantis and John Jonides. Abrupt visual onsets and selective attention: Evidence from visual search. *Journal of experimental psychology. Human perception and performance*, 10:601–21, 11 1984.

[30] Kun Zeng, Mingtian Zhao, Caiming Xiong, and Song-Chun Zhu. From image parsing to painterly rendering. *ACM Transactions on Graphics*, 29(1):1–11, December 2009.