Procedural real time live drawing animation

Sofiane Ben Embareck
ESGI
Paris, France

Julien Boutet
ESGI
Paris, France

Félix Cantet
ESGI
Paris, France

Figure 1: Final frame output of our procedural drawing method.

ABSTRACT

This work presents a real-time method to create a procedural drawing animation given a simple image and a set of parameters. The resulting animation, based on a GPU particle simulation, respects the input image structure and dynamic to draw and move brushes. Our work could be helpful for both creating live drawing animation and, more generally, to create a stylized image reproduction. The set of parameters allows the user to achieve a wide range of artistic styles.

CCS CONCEPTS

• **Computing methodologies** → **Procedural animation**; **Motion processing**; **Non-photorealistic rendering**; **Image-based rendering**.

SIGGRAPH '21 Posters, August 09-13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8371-4/21/08…$15.00
https://doi.org/10.1145/3450618.3469141

KEYWORDS

rendering, particles, procedural drawing, GPU, real time

ACM Reference Format:
Sofiane Ben Embareck, Julien Boutet, and Félix Cantet. 2021. Procedural real time live drawing animation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Posters (SIGGRAPH '21 Posters), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450618.3469141

1 INTRODUCTION

In computer graphics, an image is often considered as a static matrix and animations come from a sequence of static images. Our work is motivated by this statement: how could a texture be a living and evolutionary process in a real time rendering context? In video games and computer graphics, evolutionary and dynamics representations are often based on particle systems. Our work tries to reconcile these two opposite contexts: one that is static and the other one that is animated by nature. Our approach is to animate the steps that lead to the static texture input, resulting in a live drawing animation. Furthermore, this method gives the ability to create stylized and unique images by tweaking parameters.

The principle of our method is to provide the program with an input image and generate a vector field from it. This vector field is then used to guide particles in a 2D space in accordance

with the input image structure. Particles are rendered in a texture that accumulates every frame, resulting in a believable live painting animation. To provide a more accurate control over parameters, this process is separated into multiple particles layers that are composed in a single texture at the end of each frame

2 ALGORITHM

The method is based on a layered simulation approach. Each particle layer is simulated and rendered independently, and the total number of layers is adjustable by the user.

Our method consists of six steps:

(1) Load the Input image and all particle layers in GPU buffers.
(2) Bake a vector field from the Input image in a texture. This vector field corresponds to the local anisotropy of the input image [Akl and Germain 2014] that we compute with the eigen vector of the smoothed structure tensor of the input image

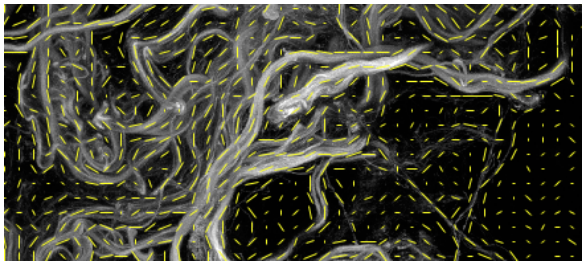

Figure 2: Visualization of the vector field

(3) Instantiating particles directly in a compute shader to avoid useless data transfer from CPU to GPU. This instantiation process sets up all initial per particle data: position, color, size, speed... The initial color is based on the input image. The position could be driven by many methods, from a fully random initialization to a prebaked control map-based approach, like a sobel filter to identify the edges of the input image.

Figure 3: Example of an instantiation control map computed with a Sobel Filter

(4) The particle simulation is stepped forward for each frame. This simulation is based on the vector field texture and is responsible for updating particles data like position, color and speed.
(5) Render each particle layer in a texture which accumulates the rendering of every frame using the layer brush texture and an indirect instance drawing command.
(6) For every frame, compose all individual particle layers in one image with a custom, user defined blend function. This blend function alters the final look of the image and the changes in the view of the desired output.

3 NEXT STEPS

While the system is robust and tolerant, we have identified some potential weaknesses:

(1) The parameters could be hard to setup and a small variation could totally change the final result. Thanks to the real time rendering context, we can iterate quickly to find the desired parameters.
(2) The blend function is also a very sensitive step of the algorithm and it cannot be exposed as a simple parameter. Because there is no universal blend function to achieve a specific kind of style, a shader programmer is required to address this issue, breaking the artistic pipeline consideration.

To address the parametrization complexity, a Neural Network could be used to setup all parameters based on higher level and artist friendly parameters. This way, the user trades a little bit of control for a more pleasant iteration context. Our implementation provides some simple sequencing tools to control the simulation timing of each layer. There is much more work to achieve in this domain to provide a complete toolset to an artist.

4 CREDITS

Original model/artwork: Splash fox, designed and created by Daniel Bystedt.
https://www.artstation.com/dbystedt
https://twitter.com/3DBystedt

REFERENCES

A. Akl, C. Yaacoub, M. Donias, J. Da Costa and C. Germain, "Structure tensor based synthesis of directional textures for virtual material design," 2014 IEEE International Conference on Image Processing (ICIP), Paris, France, 2014, pp. 4867-4871, doi: 10.1109/ICIP.2014.7025986.

David Vanderhaeghe, John Collomosse. Stroke Based Painterly Rendering. Paul Rosin; John Collomosse. Image and Video-Based Artistic Stylisation, 42, Springer, London, pp.3-21, 2012, Computational Imaging and Vision, 978-1-4471-4518-9. (10.1007/978-1-4471-4519-6_1). (hal-01342483)