Innovations Report: Procedural Fur System

Joe Withers*
National Centre for Computer Animation

Abstract

For this project I developed a procedural fur system, with the aim of exploring the feasibility of offloading computation onto the GPU within artist tools. The final artefact is an application which serves to interface with the fur system API that I have developed. This report primarily documents the implementation of the API, as well as my findings.

1 Introduction

Introduction Text.

1.1 Existing Solutions

Existing Solutions.

2 Implementation

Method Text.

2.1 Resources

I decided to develop the API side of the fur system using C++, primarly as it most commonly used when developing computationally heavy artist tools, but also because it is the language I am most comfortable using. I opted to use OpenGL over other APIs (CUDA, OpenCL) for offloading of computation onto the GPU, simply because my final artefact is a tool with a graphical interface, and OpenGL is capable of handling both arbitary computation using compute shaders and rendering of geometry.

To handle the user interface I used the Qt framework within C++. Qt is commonly used for artist tools within visual effects as it is cross-platform, and applications can be configured to run within other applications that make use of it, such as Autodesk Maya.

I wanted to include a node-graph style interface within my application, as it is commonly used within existing artist tools (Autodesk Maya, Unreal Engine), and would encourage modularity within my API. Qt does not natively support this, so I made use of NodeEditor[et al 2017] an existing Qt-based library that provides this functionality.

3 Results

Results Text.

References

ET AL, D. P., 2017. Qt5 node editor. `https://github.com/paceholder/nodeeditor`. Accessed 25 Jan 2018.

*e-mail:joewithers96@gmail.com