Simulation of Curly Hair

5th semester Project Laboratory report

Barnabás Börcsök
borcsok.barnabas@simonyi.bme.hu
Budapest University of Technology and Economics

Dr. Szécsi László
Advisor
Budapest University of Technology and Economics
Computer Graphics Group
Department of Control Engineering and Information Technology

Figure 1. The achieved visual look

Abstract

A self-assessment of the 5th semester Project Laboratory Project is presented in this article. My goal was to achieve hair simulation of acceptable quality both in terms of look and performance. Multiple approaches were considered before arriving at a Position Based Dynamics based solution. The simulation was implemented in C++ with the Open Graphics Library (OpenGL [1]).

[1] http://www.opengl.org

CCS Concepts: • **Computing methodologies** → *Physical simulation.*

Keywords: hair simulation, position based dynamics, OpenGL

1 Introduction

In the 5th semester of their undergrad studies, BSc students from Budapest University of Tehchnology and Economics embark on their first journey of scientific research. I was always interested in and fascinated by computer graphics, it came naturally to choose a subject in this area. As I had

little hands-on experience in this field, a long time had to be dedicated to research and trying out different simulation methods.

The first of this paper reflects this, giving an overview of considered methods, and other possible routes that could have been taken to implement hair simulation.

The implementation and all of the code mentioned is avaiable at http://git.sch.bme.hu/bobarna/brave-2.

2 Overview of considered methods

There were mainly three methods considered, two of them being substantially different:

2.1 Mass Spring System

The whole idea of doing hair simulation as my project laboratory came after reading Iben et al. [1]. They outline a method for the artistic simulation of curly hair for use in their film production. The method was featured in the 2012 movie Brave.

This approach models the hair as a chain of particles with given mass, each connected via springs. This results in a mass-spring system, which is then modelled by considering well-known physics formula's such as Newton's second law of motion ("$F = m * a$", and Hooke's Law, which states that the force (F) needed to extend or compress a spring by some distance (x) scales linearly with respect to that distance. The paper by Iben et al. [1] created an elaborate system of springs, with additional core springs making the simulation stable and resulting in the desired look and feel.

The Iben et al. [1] paper also features great further readings in their citations, which gives a great glimpse into related previous works.

The main characteristic of this approach is that a Mass Spring System deals with forces, and tries to stay true to the laws of physics. This will not be the case with Position Based Dynamics.

2.2 Position Based Dynamics (PBD)

Umenhoffer et al. [2] Position Based Dynamics (PBD)

2.3 Follow the Leader (FTL)

The Dynamic Follow-The-Leader method outlined in *TODO reference article*

Acknowledgments

To Dr. László Szécsi, associate professor at the Computer Graphics Group (Department of Control Engineering and Information Technology), whom I had the chance to consult with during the semester.

References

[1] Hayley Iben, Mark Meyer, Lena Petrovic, Olivier Soares, John Anderson, and Andrew Witkin. 2013. Artistic Simulation of Curly Hair. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Anaheim, California) *(SCA '13)*. Association for Computing Machinery, New York, NY, USA, 63–71. https://doi.org/10.1145/2485895.2485913 http://graphics.pixar.com/library/CurlyHairB/paper.pdf.

[2] Tamás Umenhoffer, Artúr M. Marschal, and Péter Suti. 2016. Simulation methods for elastic and fluid materials. http://cg.iit.bme.hu/~umitomi/publications/GRAFGEO2016_PhysicsSimulators.pdf.

A Supplementary development

A.1 OBJ Reader

An OBJ reader utility was implemented as part of the project. As the OBJ file format [2] describes a wide range of properties for objects. The present OBJ reader handles only the subset of these describtion options needed for the project. Lines containing other type of object descriptions were ignored, resulting in a successful file read if possible.

Table 1. Supported OBJ Data Types

Type of data	Format
Comment	`# comment`
Geometric vertex data	`v x y z`
Texture coordinates	`vt u v`
Vertex normals	`vn i j k`
Triangular faces	`f v1/vt1/vn1 v2/vt2/vn2 ... v3/vt3/vn3`

- "`v x y z`" defines a vertex with position (x, y, z).[3]
- "`vt u v`" defines a 2D texture coordinate with position (u, v).[4]
- "`vn i j k`" specifies a normal vector with components i, j and k.
- "`f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3`" Specifies a face with the given indices. For the first vertex, the *v1*st previously defined vertex position is used, the *vt1*st texture coordinate and *vn1*st normal vector is used.[5]

A.2 Recording the simulation on-the-fly

A recording utility was made to work in tandem with the application. In essence, the rendered images are written into `bmp` or `png` files if the capturing is on. In the implementation[6], toggling the capturing is mapped to the `C` key.

[2] http://paulbourke.net/dataformats/obj/

[3] The obj file format allows for a w coordinate (also called the weight) for describing rational curves and surfaces. The default value of w is 1.0. My implementation handles only the assignment of x, y and z values.

[4] The obj file format allows for 3D texture coordinates as well, but this implementation handles only 2D texture coordinates.

[5] The obj file format allows for face definitions far beyond only triangles, although this implementation handles only face definitions of the above format.

[6] https://git.sch.bme.hu/bobarna/brave-2

For handling the export of the rendered image into files, the stb_image_write.h[7] single-file public domain library was used.

The program outputs the image files in the `renders` folder at 24 FPS, with the naming convention renderXXXX[.bmp/.png], where XXXX is the number of the frame with left-padded zeroes.

After the image files are written, the user can start the `make_video.sh` script to assemble the `output.mp4` and delete all the rendered frames. This script uses the free and open-sorce ffmpeg command line utility[8]

[7] https://github.com/nothings/stb/blob/master/stb_image_write.h
[8] https://ffmpeg.org/