Prof. G. Zachmann
C. Schröder (schroeder.c@cs.uni-bremen.de)

University of Bremen
School of Computer Science
CGVR Group
January 17, 2018

Winter Semester 2017/18

Assignment on Virtual Reality and Physically-Based Simulation - Sheet 6

Due Date January, 28 2018

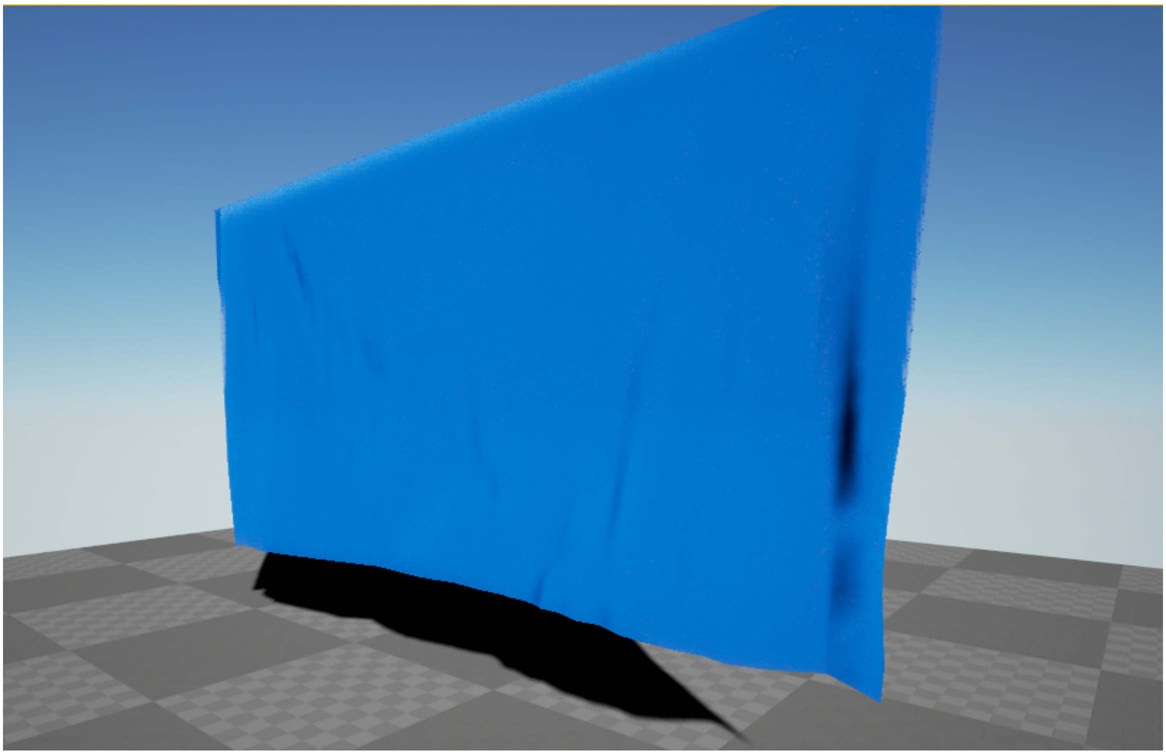

Figure 1: Final mass-spring system.

Exercise 1 (Mass-Spring-Systems, *8 Credits*)

The goal of this exercise is to implement a mass-spring system. On our website, you can find an Unreal project with most parts already implemented.[1] When you recall the definition of a mass-spring system from the lecture slides, you will recognize the two main components in the Spring.{h,cpp} and MassPoint.{h,cpp} files. The SpringMassActor.cpp glues the system together. It initializes the mass points and springs, calls the update methods, and further creates a mesh to visualize our system (`ASpringMassActor::initSpringSystem`). In the provided level, you can press `F` to apply a force to the center of the mesh. The logic behind it is implemented in the `ASpringMassActor::Touch` function. By pressing the keys `F1` and `F3`, you can switch the rendering between wire-frame and lit mode.

[1] `http://cgvr.cs.uni-bremen.de/teaching/vr_1718/uebungen/spring_mass_v4.18.zip`

a) Implement the force calculation for each spring and add it to the mass points (`Spring::Tick`). You can access the members of the connected mass points (`m_m1`, `m_m2`) directly, as `class Spring` is a friend of `class MassPoint`.

b) Add a gravitational force to each mass point in `MassPoint::updateGravity`.

c) Implement a perturbed gravitation vector[2] instead of a constant one to each mass point.

d) Change the integration method in `MassPoint::updateCurPos` to use the approximate midpoint method from the particle system slides.

[2] You can add a small, random vector offset to the gravitation vector.