CG1 WS 23/24 - Exercise 6: Animation & Simulation

Technische Universität Berlin - Computer Graphics
Date 01. February 2024 **Deadline** 21. February 2024
Prof. Dr. Marc Alexa, Tobias Djuren, Hendrik Meyer, Markus Worchel

Animation & Simulation (7 points)

In this exercise, you are going to implement Linear Blend Skinning (LBS) and a double pendulum. We kept the two exercises seperate but they are implemented in the same web application. You are able to switch between them by selecting the respective task in the GUI. Make sure that the correct exercise is shown in the window depending on the selection in the gui.

Linear Blend Skinning

In this part of the exercise we want you to animate the elephant using the linear blend skinning.

The skeleton code provides you the mesh of the elephant, the bone indices and weights for each vertex, as well as three different animation. The bone indices are stored as `Array<Array<number>>`. The outer array is associated with the different vertices in the geometry. Each inner array contains the bone indices which influence the vertex. The weights are stored in a similar fashion but instead of the bone indices it represents the bone weights. The length of the inner arrays of the indices can vary, but matches the length of the weights arrays. The weights sum up to 1. Example: Indices: [[0,1],[0,1,3],[1,2],[3]], Weights: [[0.3,0.7],[0.3,0.2,0.5],[0.2,0.8],[1]].

Each animation is an object with two `key: value` pairs, the `restpose` and the `frames`. The `restpose` is an `Array<Array<number>>` where the inner array represents a flattened local to world matrix of a single bone. You can convert it to a `THREE.Matrix4` by calling `matrix.fromArray(...)`. The outer array is therefore a list of bones representing the skeleton. Each frame is structured the same way but all frames are joined in another array resulting in an `Array<Array<Array<number>>>` object. The order of the bones is consistent between animations and individual frames and corresponds to the bone indices given in `indices`.

The tasks in detail are:

1. (Setup) First of all we need to be able to visualize the elephant and its skeleton. Load the elephant mesh using `getElephant` in `helper.ts` and add it to the scene. For the skeleton, use the restpose of any of the three animations. You can access the respose using the key `restpose` on the respective animation object. Given the local to world matrices in restpose you should be able to compute the bone positions. Visualize the origin of each bone as a single sphere (you do **not** have to connect them). Toggle the visibility of mesh and skeleton through the GUI. *(1 points)*

2. (Skeleton) Before we go into linear blend skinning we want to make sure that the we can animate the bones. Depending on the selection in the GUI, choose the respective animation. Iterate through all frames of the animation and update the bones according to the local-to-world matrices of each frame. You can access the frames using the key `frames` on the respective animation object. Make sure that the checkbox triggers the animation or rest pose. *(1 point)*

3. (Skinning) Now you should implement linear blend skinning such that the mesh starts to move around according to the bones. Using the local to world matrices of the restpose and the current frame as well as the bone indices and weights to update each vertex position. The indices object contains the bone indices. The first element in the array describes the bone indices for the first vertex in the mesh. Note that the number of bones a vertex depends on is not fixed. The respective weights are stored in the same format. *(1.5 points)*

Pendulum

In this part of the exercise you have to implement a double pendulum. It consists of a root element, which remains fixed, and a two freely movable elements. The pendulum is subjected to gravity and spring forces.

The tasks in detail are:

4. (Setup) Create a simple pendulum. Initialize the endpoints of the pendulum such that it can swing in a meaningful way (not just up and down). Visualize the pendulum as box for the fixed end and a sphere representing the freely movable end of the pendulum. You can create them using the functions `getBox` and `getSphere` in `helper.ts`. For the spring of the pendulum add a line (`getLine` in `helper.ts`) connecting the two endpoints. You can modify the line by updating the position attribute of its BufferGeometry. *(0.5 points)*

5. (Simulation) Implement the simulation loop. Given the current state (position and velocity) of the pendulum compute the gradient at the current time step. The gradient should consider gravity and a spring force. As parameters for the forces use the values defined in `Settings` in `helper.ts`. Update your current state for a single step using explicit euler integration using the computed gradient. Make sure that the visualization stays up to date (including spring). *(2 points)*

   Hint: The *up*-direction in threejs is defined as the positive y-axis. Make sure that the gravity points along the negative y-axis.

6. (Trapezoid) Instead of using the current gradient to update the state of the pendulum, use the trapezoid formula from the lecture. The solver type should be selectable through the GUI. *(0.5 points)*

   Optional: The GUI also provides dropdown elements for the midpoint and runge kutta methods. Given the implementation for the trapezoid method you should be able to easily extend it to these two methods. However, it is only a voluntary exercise.

7. (Double Pendulum) Extend your pendulum to two segments. The new element should extend the pendulum from the previous tasks. The double pendulum should be toggled based on the GUI selection. *(0.5 point)*

Requirements

- Exercises must be completed individually. Plagiarism will lead to exclusion from the course.
- Submit a `.zip` file of the `src` folder of your solution through ISIS by **21. February 2024, 23:59**.
- *Naming convention*: {firstname}_{lastname}_cg1_ex{#}.zip (for example: jane_doe_cg1_ex6.zip).
- You only hand in your `src` folder, make sure your code works with the rest of the provided skeleton.