Simulation & Animation - SS 2025

Binding of glass Mini golf 2
Can you hit it?

Florian Winston, 11727495, winston@student.tugraz.at
Leon Tiefenboeck, 11919874, tiefenboeck@student.tugraz.at

May 26, 2025

1 User Documentation

1.1 Gameplay Overview

1.2 Controls

So far we have implemented two levels, where each showcases a different technique, however for the levels to be really complete and make sense to play the second technique is needed (per level).

The game is started by opening `main.html` which opens a start screen where one can choose between the two levels. The main concept is the same in both levels: In the bottom of the game area there is a blue ball that can be moved by clicking it, dragging somewhere and then releasing the mouse button. The ball will move in the opposite direction of where was dragged and the momentum of the ball depends on how far back was dragged, much like a spring. Somewhere else in the game area there is a green circle. This is the hole where the ball should go into. If the player manages to get to the ball into the hole the level is completed and they are put back into the main menu. On the left side of the screen there is the control panel where level specific techs and the overall animation update rate can be adjusted. Here one can also toggle the needed visualizations for different techs. Now below we will outline the different levels and their techs.

1.3 Level 1

This level so far features the Path Interpolation tech. However this will really only make sense once Rigid Bodies are also implemented, since the moving objects should act as obstacles where the ball bounces off of. For this reason we only added two splines to showcase, one that forms a wave and the other just a straight line. As for now the movement only loops but we plan to add the option to make them move back and forth.

The code for this is located in the PathInterpol.js file. To construct a spline we only need the control points and the traversal speed. Then we set the t values (basically how far along the spline we are) to 0 and precompute the arc length table. Then the spline is ready and waits until `update` is called. In `update` we calculate the new value based on the current t value and the speed and then we make the lookup in the arc length table and evaluate the spline at that value. We also have a render function that just draws a circle at the current position (though in the future we also want to add different shapes).

1.4 Level 2

Level 2 showcases the particle dynamics. Here there are two repelling bodies in red and two attracting bodies in purple. Once the ball is shot, they start affecting it. The forces become more visible when a higher frame rate is selected. The force field created by the bodies and trajectory of the ball over the last few seconds can be visualized through the button. One can also change the integration method to choose between the RK4 and Euler methods. At the moment, only the ball is effected by the forces, while there is no effect on the other object from each other or from the ball. This might be changed in the future for more enjoyable level design.

The code for this is located in the ParticleDynamics.js file. There, the particle class with the necessary members is defined. It also holds the defineForces function, which takes all the particles into account, to calculate the effect they have onto the ball. The Rk4 an euler functions are used for the integration step and apply the results to the ball. The visualize and drawArrow functions are used to draw the particles, the trail and the force field. Note: Pushing the visualization button once visualizes the trail, pushing again adds the field. Pushing a third time only visualizes the field and pushing a fourth time turns the visualization off again. A vector class is also defined, which will be integrated into the other techs.

2 Technical Documentation

2.1 Path Interpolation

2.2 Particle Dynamics

2.3 Voronoi Fracture

We implemented our game very modular way. We have one main game class that controls the game logic and everything that is the same in each level. This class also includes one main loop that updates the position of objects and renders them. We can very easily add other objects from other parts in the code to this loop. We can adjust the animation update rate here by simply adding a delay to when the next iteration of this loop is called. We achieve the framerate independence also here, by calculating how long a frame took to produce and then adjust all updates by this delta.

Next we have an individual class for each of the techs. Here everything should also be confined and other parts of the game need only interact with the constructor and the `update` and `render` methods.

Finally we have a file for each level that interacts with the main game class and the different techs. Here we set up the objects and their techs in the respective level and the controls that are level specific (visualization, speed adjustment, etc.). If everything is set up we add the `update` and `render` methods of the objects (that come from the techs) to the main game loop and the start the game by calling the loop.