02561 COMPUTER GRAPHICS DTU COMPUTE

Worksheet 12: Animation and interaction

Reading	Lasseter, J. Principles of traditional animation applied to 3D computer animation. *Computer Graphics (SIGGRAPH '87)* 21(4):35–44. 1987. Angel: Sections 3.2-3.11
Purpose	This worksheet takes inspiration from the fundamental principles of animation to create a jumping ball with a cartoonish character. We will use simple physically inspired models to make the ball move and use buttons and sliders to interactively set model parameters. Furthermore, we will use the mouse (or touch) interface to interact with the ball.
Part 1	We first use the principles of **arcs** and **timing** to get natural movement. Start from the bouncing ball in Part 5 of Worksheet 1. • Put the canvas in the left cell of an HTML table with one row and two cells. Create another HTML table in the right cell for setting up a user interface. • Add a button to the user interface for switching the animation on/off and a button for stepping to the next frame when the animation is off ($\Delta t = 1/60$ s). • Position the ball on the floor and give it an initial jump velocity in the y-direction. • Use simple kinematics with semi-implicit Euler steps to animate the ball. Reset the ball when it returns to the floor. • Add a slider to the user interface for setting the jump velocity. Use the `"input"` event instead of the `onchange` function if you prefer immediate response while using a slider (instead of only getting response when the slider is released).
Part 2	The next step is to use the principle of **squash and stretch**. We achieve these effects by non-uniform scaling of the circle in the vertex shader. • Introduce two uniform floating-point variables for scaling the ball in the x- and y-directions. These variables need to be available both in JavaScript and WGSL. In the vertex shader, they should scale the x- and y-coordinates of the incoming vertex position. • Stretch the ball by using the speed of the ball and a user-defined stretch factor to modify the scale variables when the ball is jumping. • Squash the ball by switching mode when the ball intersects the floor. In squashing mode, use a squash duration and a squash factor as variables to modify the scale variables in directions opposite to the stretch. Let the squash animation follow a sine function and ensure that the bottom of the ball touches the floor throughout the animation. Reset the ball when the squashing animation concludes. • Add sliders to the user interface for setting the stretch and squash factors as well as the squash duration.

Worksheet 12: Animation and interaction

Part 3	We now turn to the principle of **follow through and overlapping action**. We do this using a simplistic spring model instead of fully separated animations for squash and stretch. • Replace squash factor and squash duration with stiffness and damping variables for a spring (in the sliders too). • Introduce a spring velocity and y-coordinate. When the ball intersects the floor, transfer the ball velocity and stretch (stretch factor times speed) directly to the spring velocity and y-coordinate, respectively. The spring y-coordinate replaces the ball stretch in this mode. • In squashing mode, use the spring stiffness and damping to continue the animation until the spring velocity is positive while the y-coordinate of the spring is nearly zero (e.g. > -0.05). Then, reset the spring and the ball and move the ball the first step upwards.
Part 4	To touch upon the principles of **solid drawing**, **timing**, and **slow in and slow out**, we increasingly flatten the ball based on its proximity to the table when squashing and leave it at the lowest point for a few frames before jumping back up. • When the spring velocity first becomes larger than zero, the ball is in its flattest state. Set a flag and hold this state for a number of frames set by a flat pause duration variable. • Modify the vertex shader to detect when a squashing animation is ongoing (when the scale in the x-direction is larger than one). In this case, use the distance relative to the diameter that the vertex is below $y = 1$ to compute a contact flattening factor. Multiply the stretch part of the ball's scaling factors by this flattening factor, while ensuring that the ball maintains contact with the floor. • Add sliders to the user interface for setting the flat pause duration and a flatness controlling the computation of the contact flattening factor.
Part 5	Finally, we leave it up to the user to exercise the principle of **exaggeration** by pulling the sliders and enable **straight ahead action** by implementing control of the ball when clicking the canvas with the mouse. • Attach event handlers to the `mousedown`, `mousemove`, and `mouseup` events. Center the ball at the tip of mouse cursor with no velocity when the left mouse button is down and when the mouse moves. Continue the animation when the mouse button is released. • If points are offset from the tip of the mouse cursor, get the bounding rectangle of the canvas in the client area using `event.target.getBoundingClientRect()` and correct the mouse position using the `left` and `top` coordinates of this rectangle. • Use the spring model to squash the ball if the user moves the ball center closer to the floor than the circle radius.