COS 426 Computer Graphics - Final Project Interim Report
e-Motional e-Music

Charles Liu (cl43@), Nicholas Sum (nsum@)
https://github.com/nicksum107/emotionalemusic

Abstract

Our project, e-Motional e-Music, brings together graphics and music into an interactive musical simulation. Inspired by our interest in collaborative music and marble-based musical contraptions, we created a musical simulation involving the interaction of a piano and bouncing marbles, simulated using physically realistic physics and collisions. When a marble collides with the piano keys, it creates sounds corresponding to the key played, so that the user can make fun and interesting music using these balls. The user interacts with the piano through modification of the settings as well as through creating their own marbles and beautiful melodies. Long term goals of the project includes adding other musical instruments, refining the simulation to be more physically realistic, and allowing the user to use the simulation to create their own music with a looping or editing system.

Introduction

Goal

The goal of this project is to create a realistic interactive simulation combining computer graphics and music making, with physically accurate interactions between the objects in the scene. While this project was first inspired by our interest in music making and our fascination in marble based musical contraptions, our project and its more general extensions can also be used in realistic musical visualization videos. Instead of manually creating animations to match the music after it is finished, a musician can instead create music with a real-time updating visualization based on the music. This animation can also inform the musical creation process.

Previous Work

Although there has not been work that we are aware of attempting simulation-based music creation, there is much work in the more general field of post-composition musical animations. The most prominent of these is the work done by Animusic in the early 2000s (Lytle). In their two DVDs released in 2004 and 2006, they created many intricate musical visualizations of MIDI-based music. For example, their first released visualization “Pipe Dream” at SIGGRAPH 2001 involved many pipes and balls hitting instruments moving through the screen (Park).

However, unlike our project, the music came first. Using their proprietary MIDImotion software, they inputted the MIDI music scores as controllers for the animation. While the music did influence the animations further than simply moving characters to the beat, our project goes in the reverse direction. using a simulation to produce the music in the first place.

There has also been other work in creating graphical dashboards for creating music in music composition software that mimic instruments such as loop stations using touch screens, keyboard instruments, or USB-attached instruments. This is not what we have in mind for this project. In this project, we seek to

create a physical simulation based on marbles and the interactions of objects in motion rather than simply playing back a visualization of an instrument like how we play instruments in real life.

Approach

Our approach for the simulation was to use/find a piano model without keys and add our own keys as simple bounding box meshes with (invisible) springs underneath, to simulate the motion of real piano keys. We would add sound files for each key, to be played when they are hit sufficiently hard (and with varying volume, if possible). Marbles would be sphere meshes which can collide with the piano model, as well as with the keys and the ground. We also intended for the collisions to be simple, elastic collisions (or an approximation if necessary).

We think that the approach should work well with marbles with mass fairly small (but not too small) relative to the keys, and we think that some level of clamping or damping for the keys, as well as potentially a little damping for the marbles would be best. We think the mass difference is best in order for the marbles to bounce in an aesthetically pleasing and realistic manner (as opposed to having too large a mass such that the marbles barely bounce), while still displacing the keys enough to make a wide range of sounds. The clamping/damping of keys seems very important in order for the keys not to oscillate infinitely (since we model them as springs), and the damping of marbles may be useful due to floating point/approximation errors which may end up introducing more energy into the system at different points in the simulation (so the damping would help cancel out this effect).

Methodology

We implemented our project in Three.js and Node.js to create our simulation and dat.GUI for the creation of the sidebar UI. We created our scene, camera, event listeners, and objects using Three.js's built-in functions and classes as used in the starter pack. We use an out-of-the-box renderer and camera.

We have three main objects that we simulate in the beginning minimum viable product for this project: the piano, the keys, and the marbles. Although the piano model has keys in the mesh, we create separate key objects in order to simulate the keys separately from the piano, since keys have the ability to play notes. We hide the original piano keys underneath the ones that we create.

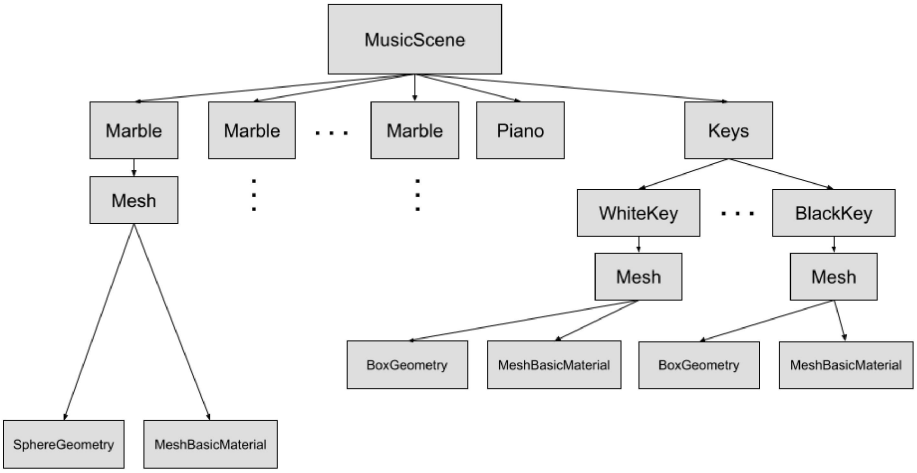


Fig. 1: Scene Architecture

Meshes

Keys and marbles lend themselves to very simple geometries; outside of simple boxes and spheres, the only potential improvement would be to model white keys as actual L-shaped keys wrapping around the black keys. This would be slightly more realistic, but we expect that there will rarely, if ever, be a noticeable difference (as the box geometries are almost identical in terms of surface area), and would be somewhat more complicated to implement. The simplest way to add a piano is using a mesh, although more complicated methods such as parametric surfaces or sweep models are possible. The benefit of other models may be more realism, although the focus is not on the piano itself but rather its keys, so we chose the simpler but still realistic mesh option. The piano mesh we found came from Google's Poly database, which we imported out of the box as a GLTF. The piano model is a simplified 4-octave polygon piano that is unrealistic but is a good starting point as a good visual model of a plausible piano.

Marbles and Pianos can be easily added to any scene through the basic Object3D and Group functions. Keys are slightly different. We create a Keys Group which creates an array of Keys corresponding to all the keys on the piano. We create BlackKeys and WhiteKeys within this group, which have different colors, locations, and meshes depending on the type of key. We used simple boxes to simplify simulation, collisions, at the expense of some realism (e.g. keys' meshes intersecting with other keys' meshes).

Fig. 2: Piano mesh

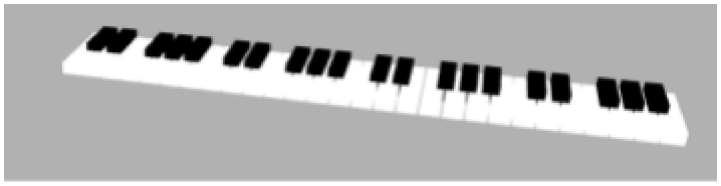


Fig. 3: Key meshes

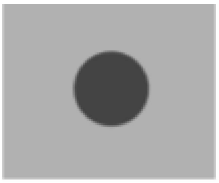


Fig. 4: Marble Mesh

Simulation

The main possibilities we considered for marble simulation were forward Euler integration and Verlet integration, both reasonable approximations for integration the equations of motion. The benefit of Euler integration is a closer approximation of the true velocity of an object, since Euler integration actually uses the velocity to update a position, and also updates the velocity. The benefit of Verlet integration is simplicity, but since we use velocity for collisions, we considered it to be easier and more realistic to use Euler integration. Marble simulation is thus done through forward explicit Euler integration. Other considerations are air resistance and friction, which would be beneficial for realism, but we liked the more "animation" feel of our idealistic simulation without these forces and thus did not add air resistance or friction. Finally, in terms of deleting marbles that are far from the piano, most schemes for deciding when to delete a marble are fairly arbitrary, so we allow the marble to bounce a fixed number of times (3) on the ground before deleting/disposing of the marble.

Key simulation is similarly done through Euler integration of Newton's Laws, which created plausible key movement with the same benefit/drawback as for marbles. We do not move the x or z coordinate of the key and only move the key in the y up-down direction, simulating how real piano keys only move in the y direction.. We simulate gravity as well as an upwards spring force that forces the key back up to rest position. Clamping is very important for plausibility, and we clamp the key's position to a range between the highest and lowest position the key can reach so that the keys do not fly off of the piano or oscillate around an invisible spring. When the key passes a certain threshold as it goes downwards, we play the note using a pre-recorded sound, and the volume of the sound scales linearly with the velocity of the key.

This is a simplification of the real life key but estimates the true mechanism of piano keys. Internally, piano keys are effectively levers with two sides preventing the key from passing a particular point, much like a see-saw. This lever action then causes the damper on the string to lift and the hammer to hit (AATuners). However, since we would have to create a whole internal physical mechanism that would be incredibly difficult and error-prone, we decided to create a physically plausible but approximate mechanism. We also decided not to use a lever in order to simplify the collision code; we only needed to worry about the y-coordinate of the key in our mechanism. The sound scaling linearly with velocity is also an approximation, but when testing, it seemed plausible, and the alternatives would involve more complicated physical simulations such as string vibrations which similarly would be error-prone and likely not a large improvement worth the complication.

Collisions and Other Interactions

Marble collisions with the simulated keys are the main interactions in this simulation. Since our keys are axis-aligned and never move in the x or z directions, we can simplify the interaction to a simple check of the ball hitting the top of the key in terms of y-coordinate. If the bottom of the sphere is below the top of the key and the sphere within the key in the x and z directions, we resolve a collision. Since we are planning to do collisions with the piano, we do not need to enact collisions with the bottom of the key which should always be protected by the piano.

For collisions, there are two broad categories: elastic and inelastic collisions. Elastic collisions provide the benefit of simplicity with a reasonable level of physical realism. Inelastic collisions provide greater

realism, as energy is released in each collision in the form of sound and compression (if it were a real marble), at the expense of a much more complicated simulation. We resolve collisions using elastic collisions between the ball and the key, as the added benefit of inelastic collisions is exclusively realism, but the cost of calculating energy lost via sound and compression would still be approximate while being quite complicated. Also, our clamping of the keys causes a loss of energy, so there already is a loss of energy in the system, and further loss may be detrimental to the user experience anyways.

We are planning on performing collisions between the marble and the piano mesh itself. This is an important extension that would be a proof of concept showing the ability to use our marbles to play instruments with a more general mesh than simply an axis-aligned box. There are a couple different options for checking collisions with an arbitrary mesh. Naively, we can check the intersection of the sphere with every polygon in the mesh and return an intersection if found. However, this is too slow as it takes O(num_polygons) for each marble and mesh. Alternatively, we can choose a few cardinal directions on the sphere or choose a few sample directions on the sphere to raycast in order to estimate collisions, which would work best for small spheres. We can create rays from the center of the sphere to each direction, and use the built-in, optimized ray tracer to search for intersections with any mesh. If the collision is close enough, then we would perform a collision.

We resolve this collision with a simple damped reflection, as we assume the mass of the piano is much bigger than the mass of the marble. The piano can play a percussion sound for the bounce.

Marble Creation and User Interaction

The current design for user interaction is through the dat.GUI side bar and through key press events. When a user presses a key corresponding to a note (A-G, shift used for accidentals), depending on whether we are directly playing the note or not, we either directly give an impulse to the key to play the note or spawn a marble above the corresponding key. Octaves are set through a slider bar. P spawns a marble that is above the lid of the piano to create the piano's percussion notes once this feature is complete.

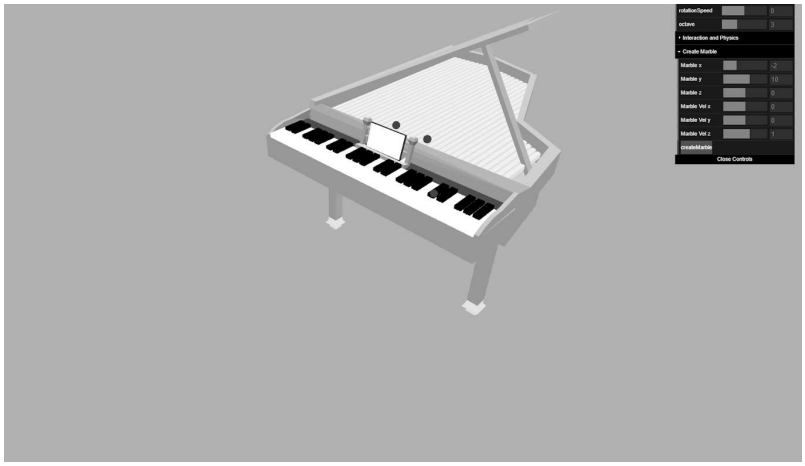
We also have a section of the GUI that can arbitrarily create marbles with a given location and velocity, although it is primarily for testing and we intend to make a more user-friendly way to create marbles. However, it can create some fun effects such as a marble bouncing down the keyboard and playing a cool scale. Future work could allow the user to save, copy, and modify a particular marble spawn location and velocity so that they can easily click a button on the GUI to generate the same marble again. A natural extension of this is being able to parse a predefined scene (likely in JSON format).

More user interaction ideas for next steps are discussed below.

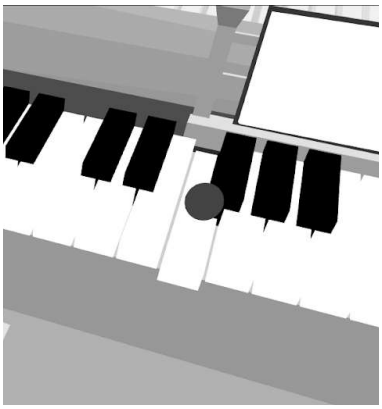
Results

Below are some screenshots showing the results of our simulation. Although the main results of this simulation are the objects in motion, requiring video, this report format does not allow us to include video. However, we have included a link to a quick video hosted on Google Drive here https://drive.google.com/file/d/1e-8YX6lpveucuiG12RJJyqnvck1oL0FQ/view?usp=sharing.

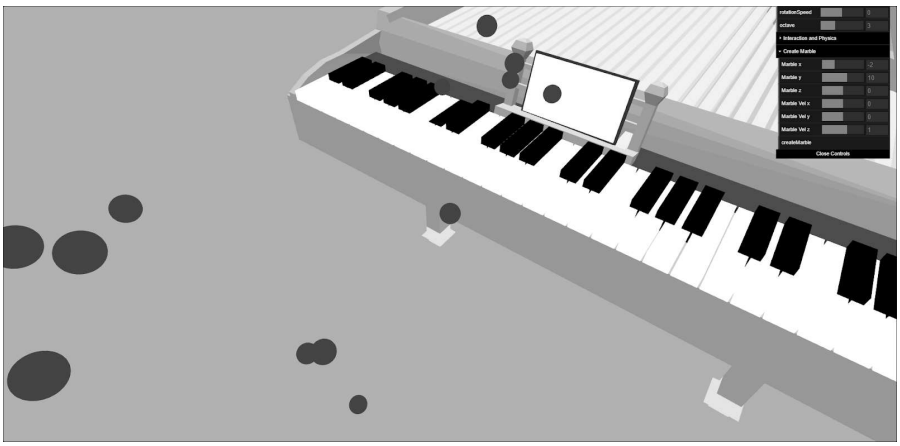
An example of a custom created marble that goes down the keys



A closer look at the key collision and simulation mechanism



Playing some black keys with many balls



Conclusion and Next Steps

Conclusions

The approach that we took in this project seems promising for future work in music creation and simulation. The simulation is fairly realistic and it seems quite possible to create an enjoyable song and scene, even with one marble playing multiple notes potentially. We believe that we have made significant steps towards achieving our goal, with our simulation being fairly realistic and aesthetically/audibly pleasing. Our next goal is to make it much more user-friendly and easy to work with, as well as adding more instruments/improving the simulation.

Limitations

As seen in the above screenshots, we have produced a plausible marble-piano simulation that looks good and can produce some good music. However, this simulation is not entirely physically accurate. As discussed above, our models are not physically accurate to real-life pianos and keyboards. Furthermore, while clamping has the physical basis of a piano with much larger mass than the marbles or keys, damping was more of an empirical approximation. We used these estimations to create a more plausible-looking visible product. Other limitations include some future technical debt, such as hard-coded values for positions of meshes and simulation parameters.

Challenges

Some challenges we encountered were correctly working with Three.js's conventions for positions of objects, meshes, groups, etc. and adding them correctly. Some of the integration details were tough to debug due to numbers not translating very intuitively, and a couple of issues were also encountered with floating point precision, which we solved with a constant EPS of error tolerance. Finally, determining parameters such as the spring constant of the springs for keys, optimal mass ratio of marbles to keys, as well as finding realistic locations and velocities for spawning marbles was challenging. To solve this, we did extensive empirical testing to find visually plausible constants that roughly model real-life pianos and marbles, while maintaining some idealistic motion that makes the simulation feel more like an animation.

Next Steps

Besides adding the general piano mesh collisions as discussed above, we also want to add more features; before the presentation date, we will add as many of these features as possible in the following order.

Saving, Playback, and Looping:

In order to make our simulation useful for music creation, the first thing to do would be to allow the user to save and playback a simulation that they created. We can add the option to begin recording all marble creation events at a given timestamp offset from the beginning of the recording. We can save these in a simple json file and load them using a similar file loading schema to other assignments in either a batch mode or using the GUI loader.

Looping, which is what many composers use to create beats, would be something cool to implement as well. We can use a similar schema to the playback mechanism described. When a button is pressed, it will save and load the file, and repeat it.

Additional Instruments

We can also add more instruments to the scene for a greater variety of sounds and possibilities for the user. With the general mesh collisions implemented for the piano, we can add instruments such as hi-hats and drums which would improve the toolkit of any users. Another fun idea would be to add a character to walk and jump on the keys to be controlled by the user.

Appendix

Contributions

Both: Debugging each other's work, brainstorming ideas and architecture, dat.GUI work
Charles: Marble creation and simulation; Marble-key interactions, elastic collision, sound
Nick: Minor setup and beginning architecture of the project, key event handler; Key creation and simulation; Marble-piano collision simulation.

Bibliography

AATuners, "How Does a Piano Work?" http://www.aatuners.com/how-it-works.html.
Cabello, Ricardo, Three.js, https://threejs.org/.
Dahl, Ryan, Node.js, https://nodejs.org/en/.
Gleitzman, Benjamin, "MIDI js Soundtracks," https://github.com/gleitz/midi-js-soundfonts
Lytle, Wayne, "Animusic," https://www.animusic.com/.
Oliviera, Bruno, "Piano," https://poly.google.com/view/5vbJ5vildOq.
Park, John Edgar, "The SIGGRAPH 2001 Computer Animation Festival: A Digital Odyssey," 2001 https://www.awn.com/animationworld/siggraph-2001-computer-animation-festival-digital-odyssey.
Webb, Edwin and Bova, Reilly, "Three Seed," https://github.com/edwinwebb/three-seed/.