Jake Crouch and Jake Crabtree

CS 395T: Physical Simulation

May 10, 2019

Physically Based Sound Generation

Our goal for the project was to generate physically correct sounds for motion and collisions of deformable bodies that synchronizes with a 3d-animation of the deformable bodies' motion. The sound generation and physical simulation form the simulation phase, which generates sound and positional values to be rendered into a real-time animation. The approach to the simulation phase consists of two main components: the simulation of motion and collisions of soft-bodies, using the simulated deformations to approximate the acoustic pressure waves the objects emit into the environment.

For soft-body simulation, the first component, we utilized a finite element method that discretizes the bodies in the system into tetrahedrons using TetGen. Each tetrahedron has four vertices that contain a position in template coordinates, m, a position world/deformed coordinates, and velocity in world coordinates, v. The functions x(u) and xdot(u) map the template coordinates to their world positions and velocities, respectively, and are based on their barycentric coordinates. The derivatives of these functions

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{m}_{[1]} & \boldsymbol{m}_{[2]} & \boldsymbol{m}_{[3]} & \boldsymbol{m}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \qquad \begin{aligned} \boldsymbol{x}(\boldsymbol{u}) &= \boldsymbol{P}\,\boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} & \frac{\partial \boldsymbol{x}}{\partial u_i} &= \boldsymbol{P}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \\ \dot{\boldsymbol{x}}(\boldsymbol{u}) &= \boldsymbol{V}\,\boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} & \frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} &= \boldsymbol{V}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \end{aligned}$$

$$\boldsymbol{\delta}_i = [\delta_{i1}\ \delta_{i2}\ \delta_{i3}\ 0]^{\mathsf{T}} \quad \delta_{ij} = \begin{cases} 1 & : \quad i = j \\ 0 & : \quad i \neq j \end{cases}.$$

can be used to calculate the Green's strain tensor ε, and the strain rate tensor ν, and are constant due to the linear nature of x(u) and xdot(u). From the strain and strain rate tensors, we compute

$$\delta_{ij} = \begin{cases} 1 & : \quad i = j \\ 0 & : \quad i \neq j \end{cases}. \qquad \epsilon_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} \right) - \delta_{ij} \qquad \nu_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \dot{\boldsymbol{x}}}{\partial u_j} \right) + \left(\frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} \right)$$

the stress tensor, σ, which combines the strain and strain rate with the material properties of the body. Under the assumption that the material is isotropic, the material properties are controlled by the Lamé parameters, λ and μ, that control the changes in volume and rigidity and the kinetic energy dissipating constants, ψ and ϕ. The stress tensor is used to compute a penalty force, f,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{(\epsilon)} + \boldsymbol{\sigma}^{(\nu)} \qquad \sigma_{ij}^{(\nu)} = \sum_{k=1}^{3} \phi \nu_{kk} \delta_{ij} + 2\psi \nu_{ij} \qquad \sigma_{ij}^{(\epsilon)} = \sum_{k=1}^{3} \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}$$

exerted from a tetrahedron on each of its four vertices and depends on the tetrahedron's volume, vol, which is computed by taking cross products of the tetrahedron's vertices' template

$$\boldsymbol{f}_{[i]}^{\text{el}} = -\frac{\text{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl} \beta_{ik} \sigma_{kl} \quad \text{vol} = \frac{1}{6} [(\boldsymbol{m}_{[2]} - \boldsymbol{m}_{[1]}) \times (\boldsymbol{m}_{[3]} - \boldsymbol{m}_{[1]})] \cdot (\boldsymbol{m}_{[4]} - \boldsymbol{m}_{[1]})$$

coordinates. The penalty force is applied component-wise to each of the vertices and combined with gravitational force in a per-vertex force matrix. The per-vertex mass, used in gravitational force and in velocity verlet time integration, is calculated by iterating over the tetrahedrons and adding the body's density times one-fourth of tetrahedron's volume to the mass of each of its vertices[2]. The velocity verlet and collision algorithms are similar to their implementations in Rigidbody/FuriousBirds project.

After the motion of the soft bodies is computed, the acoustic pressure of each surface element/triangle can be computed based on its surface normal, velocity, and the environment's acoustic impedance (415 Pa · s/m). The velocity of the triangle is computed by averaging over

$$\bar{p} = z\bar{\boldsymbol{v}} \cdot \hat{\boldsymbol{n}} = z\left(\frac{1}{3}\sum_{i=1}^{3}\dot{\boldsymbol{x}}_{[i]}\right) \cdot \hat{\boldsymbol{n}}.$$

the velocities of its three vertices. The pressure of each triangle over time is then convolved with a low-pass filter K to remove high frequencies that would be outside of the human audible range.

$$K_i = \mathrm{sinc}(i\Delta t) \cdot \mathrm{win}(i/w) \quad , \; i \in [-w, \ldots, w] \tag{6}$$

$$\mathrm{sinc}(t) = \frac{\sin(2\pi f_{\max} t)}{\pi t} \tag{7}$$

$$\mathrm{win}(u) = \begin{cases} 1/2 + \cos(\pi u)/2 & : \; |u| \leq 1 \\ 0 & : \; |u| > 1 \end{cases}, \tag{8}$$

A DC-blocking filter is then applied to the result of the low-pass filter, g, that removes any constant factors and attenuates low-frequency values, decreasing noise in the final pressure.

$$\tilde{p}_i = (1-\alpha)\tilde{p}_{i-1} + (g_i - g_{i-1}),$$

The effect of attenuation between the sound's source and the listener, the camera position in our case, is applied to the final pressure based on the equation below, with xbar as the triangle center,

$$s = \frac{\tilde{p}\, a\, \delta_{\bar{\boldsymbol{x}} \to \boldsymbol{r}}}{||\bar{\boldsymbol{x}} - \boldsymbol{r}||}\cos(\theta),$$

θ as the angle between the surface normal and the vector from the xbar to the camera, and the delta term set to 1 to reduce computational time. The delay caused by sound propagation through the air is calculated alongside the per-triangle pressures, and is based on the distance between the

source and receiver over the speed of sound (343 m/s). When combined with the simulation time, this serves as an index into the sound accumulation buffer. A final Gaussian filter is applied to reduce any remaining artifacts, and this, along with the simulation data, is written out to file for the playback phase.

We generated two custom file formats a render file type and a faces file type. The render file type keeps the data to render back the simulation at 60 FPS with audio running at a sample rate of 44100 Hz. The render file type first has the number of frames, the number of vertices in the whole scene, the vertex matrices over the frames, the number of audio samples, and then all of the audio samples. The faces file type preserves the information about the faces and the colors of the faces by first listing the number of faces, the face matrix, and then the color matrix. Additionally we made changes to how scene files are laid out, we added parameters for Young's Modulus, the Poisson Ratio, phi, and psi per mesh.

When playing back the render's audio, we use an SDL audio callback to fill an audio buffer based off the accumulation buffer we wrote to the audio file. The positional data computed from the soft-body simulation is read from in the render file, and is used to update the renderF and renderQ matrices that libigl uses at a rate of 60 FPS.

Limitations of our code include high memory overhead, so long renders are unlikely in our implementation. Our implementation also suffers from slow computation of the physical simulation, leading to long wait times before being able to see results. Another small issue is that during playback of a render you should not pause the implementation, since we do not handle this.

References:

[1] James F. O'Brien, Perry R. Cook, and Georg Essl. "Synthesizing Sounds from Physically Based Motion". In Proceedings of ACM SIGGRAPH 2001, pages 529–536. ACM Press, August 2001.

[2] James F. O'Brien and Jessica K. Hodgins. "Graphical Modeling and Animation of Brittle Fracture". In Proceedings of ACM SIGGRAPH 1999, pages 137–146. ACM Press/Addison-Wesley Publishing Co., August 1999.