Automatic Secondary Motion with Dynamic Kelvinlets

JASMEET SINGH and DAVE PAGUREK

Fig. 1. Frames of deformations due to generated Kelvinlet forces superimposed over primary motion frames.

Dynamic Kelvinlets [De Goes and James 2018] model the time-varying elastic deformations of objects in response to input forces. Videos showing Dynamic Kelvinlets in action demonstrate how it adds an extra level of realism to animations by introducing a secondary motion. A system using Dynamic Kelvinlet deformation is implemented to verify that it can achieve visually plausible secondary motion at real-time simulation rates. A framework to generate Dynamic Kelvinlets automatically is also implemented. It adds secondary motion to objects given skeletal animation keyframes for a model using linear blend skinning.

1 INTRODUCTION

Physically-based animations are widely used in computer graphics due to the realism of the resulting motion and their ease of use compared to manual animation of the same level of detail. However, physically-based simulations are computationally cumbersome due to the necessity of numerical solves and the accompanying stability conditions. There is additionally a monotonous setup phase for artists employing such techniques, usually involving the generation of an appropriate volumetric mesh for the simulation. Controlling the output of such a simulation is difficult. Generally, several simulations need to be run with different sets of input parameters to get the desired result. Hence, a simpler method to simulate secondary deformations would streamline the process of generating animations.

Our goal is to provide a system through which artists can automatically add secondary motion to standard keyframed animation using linear blend skinning bones. In this paper, we implement such a system based upon *Dynamic Kelvinlets* [De Goes and James 2018], an extension of elastostatic regularized Kelvinlets [De Goes and James 2017]. The paper proposing Dynamic Kelvinlets derives novel fundamental solutions of elastodynamics for spatially regularized and time-varying forces applied to an infinite continuum. This results in wave-like deformations through a medium. The method has the following prime advantages over conventional physically-based simulation methods:

- No geometric discretization is required
- The solution is closed-form, so no computationally intensive solve is required
- Displacement at a time t_i does not depend on the previous time step t_{i-1}, so there are no stability conditions due to accumulated error over time
- As the displacements of vertices are not correlated, they can be calculated in parallel on the GPU in real time

We implement the *impulse Kelvinlet* and *push Kelvinlet* responses of an object, which correspond respectively to the responses to Dirac delta function and Heaviside function force distributions over time. We use the accelerations of vertices in the input keyframes to generate Kelvinlets. We then examine how this technique can generate visually accurate secondary motion in real-time. The technique is specifically suited to scenarios involving jiggling, denting, ripples, and blasts.

2 BACKGROUND

Deformation of an infinite 3D medium formed by an isotropic and homogeneous elastic material are considered in the Dynamic Kelvinlets paper. Fundamental solutions are derived for the linear elasticity equation, where b is a time-varying external body force, m is the mass density, μ is the elastic shear modulus indicating the material stiffness, and v is the Poisson ratio that controls the material compressibility:

$$m\partial_{tt}u = \mu\Delta u - \frac{\mu}{1-2v}\nabla(\nabla\cdot u) + b$$

This equation is also called the elastic wave equation as it resembles the wave equation. It has an additional divergence term that restricts large volume changes. The point of application of force acts as the source for the waves, from which they radiate in all

directions towards infinity. Two kinds of waves are generated. The first is the pressure wave, which represents the volume oscillation in space and time. It depends on both the Poisson ratio and the elastic shear modulus. The pressure wave travels as a longitudinal wave through the medium. The second wave is the shear wave. It produces divergence-free displacements in the transverse direction. This wave only depends on the shear modulus of the medium. The combined response of these two waves gives us the material response to a regularized force in 3D space.

The regularization of the Kelvinlet force produces a density function ρ based on the amount of regularization ϵ and the radius r away from the force centre:

$$\rho(r) = \frac{15\epsilon^4}{8\pi(r^2+\epsilon^2)^{7/2}}$$

Plots of different amounts of regularization are shown in Figure 2. Regardless of the value of ϵ, the total amount of force applied to the continuum is preserved:

$$\iiint_{\mathbb{R}^3} \rho\left(\sqrt{x^2+y^2+z^2}\right) dx\, dy\, dz = 1$$

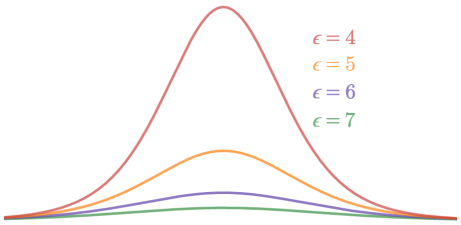

Fig. 2. Force density functions using different amounts of regularization.

We chose to implement these fundamental solutions for impulse and push response on a body. The force for an impulse is modeled as a regularized Dirac $\delta(t)$ function while the force for push is modeled as a Heaviside function $H(t)$.

3 METHOD

3.1 Framework

Our implementation [Singh and Pagurek 2019] is written in C++ using the OpenFrameworks library [Ope 2018], a wrapper for OpenGL that provides mesh data structures. Starting from an initial mesh, we apply an impulse or push force to it at a given location using our `DisplacedMesh` class, which is responsible for simulating and displaying the result. It implements the solution to the linear elastodynamics equation in C++ for calculation on the CPU. Since the displacement for each vertex has no dependence on the displacement of any other vertex, the calculation is also implemented as a step in the vertex shader on the GPU. In both cases, it produces offsets for the original mesh vertices given their initial locations and the elapsed time. These offsetted vertices are then passed through the rest of the OpenGL pipeline to be rendered to the screen.

3.2 Automatic Secondary Motion

Given the ability to simulate the response of a mesh to a Kelvinlet force, we then need a way to generate these forces automatically from an input animation. Each frame, our system follows four steps:

- For each vertex X in the mesh, find its location x for the current frame using the transformation $\phi(X)$ from the frame's bone positions.
- Approximate $\ddot{x}$ for all vertices using their locations from the past two frames of animation.
- Generate an impulse Kelvinlet for each x given its corresponding $\ddot{x}$.
- Iteratively merge Kelvinlets that plausibly would have resulted from a single source impulse.

These steps form the algorithm described in Algorithm 1, which represents our system at a high level.

The generation step creates one Kelvinlet per vertex. The iterative merging step is a performance optimization, reducing the number of new Kelvinlets added each frame to a more reasonable number. Impulse Kelvinlets may be removed once they have been simulated for a sufficient length of time, as their influence has likely reached a steady state. Our tests pick merging parameters to maintain a maximum of 100 Kelvinlets at a time.

When generating initial per-vertex Kelvinlets, we refer to Newton's Second Law, $F = m\ddot{x}$, to obtain a direction and magnitude for the impulse force vector. We approximate that the user-provided mass of a mesh is evenly distributed across its vertices when picking a value for m. We pick a level of regularization ϵ such that there is a force exactly equal to F at the location of the vertex:

$$\begin{aligned} \rho(0) &= F \\ F\frac{15\epsilon^4}{8\pi(0^2+\epsilon^2)^{7/2}} &= F \\ \epsilon &= \left(\frac{15}{8\pi} - 1\right)^{4/7} \\ \epsilon &\approx 0.8 \end{aligned}$$

We then merge Kelvinlets. We only want to merge them if they are nearby and have near-parallel force vectors. To check for closeness, we compute an approximate radius of the Kelvinlet's influence, only allowing Kelvinlets to merge if there is an intersection between the spheres produced by these radii. We pick the radius at which force magnitude drops to a low value (we use 0.1):

$$\begin{aligned} \|F\rho(r)\| &= 0.1 \\ \frac{15\epsilon^4\|F\|}{8\pi\left(r^2+\epsilon^2\right)^{7/2}} &= 0.1 \\ r &= \sqrt{\left(\frac{15\epsilon^4\|F\|}{0.1\cdot 8\pi}\right)^{2/7} - \epsilon^2} \end{aligned}$$

We additionally enforce Kelvinlet alignment by only allowing them to merge when $\hat{F}_1 \cdot \hat{F}_2 > 0.5$. If these conditions are met, then we compute the properties of the merged Kelvinlet. We aim to preserve the total amount of force, and since the space integral of regularized force varies only based on the scale of the force applied at the centre, this constrains the combined force magnitude. We

make combined force direction a linear combination of the input force directions, weighted by the total amount of force in each. We use a similar linear combination to produce the new force centre x'. This gives us:

$$\|F'\| = \|F_1\| + \|F_2\|$$

$$\hat{F}' = \frac{\|F_1\|\hat{F}_1 + \|F_2\|\hat{F}_2}{\|F_1\| + \|F_2\|}$$

$$x' = \frac{\|F_1\|x_1 + \|F_2\|x_2}{\|F_1\| + \|F_2\|}$$

Finally, we also need to pick a level of regularization ϵ' for the combined Kelvinlet. Ideally, it should regularize more the farther away the two force centers are, allowing the influence of the Kelvinlet to spread out and reach both. This leads us to the following formula, which augments a linear combination with a distance term:

$$epsilon' = \frac{\|F_1\|\epsilon_1 + \|F_2\|\epsilon_2 + \min\{\|F_1\|, \|F_2\|\}\|x_1 - x_2\|}{\|F_1\| + \|F_2\|} \quad (1)$$

Figure 3 shows how the variation in the waveforms of two input Kelvinlet forces (in red) affects the resultant combined Kelvinlet force (in blue).

4 RESULTS

4.1 Stability

One of the benefits of Dynamic Kelvinlet based motion is that there is a closed-form solution for displacement, meaning there will not be any error accumulation or instability over time due to the integration scheme. However, within a single frame, there are still issues of stability due to numerical precision. This is largely due to the fact that some of the intermediate steps in calculating displacement include division by a factor of r^3. The paper includes a formula for the limit of displacement as $r \to 0$ to avoid division by 0 directly at the force application centre, but *near* the centre, if one uses the limit formula, it is equivalent to treating the whole neighbourhood around the centre as a rigid body. If one does not use the limit formula, one must deal with significant loss of precision. Using double precision floats and a single fourth-order Runge-Kutta (RK4) step, the paper treats a radius of 1e-4 around the centre as a rigid body. Since our implementation runs on graphics hardware in a shader, we are limited to single precision floats. We find that with a radius of 1e-2 and four RK4 steps, there is no noticeable precision error.

4.2 Performance

Multiple versions of the Dynamic Kelvinlet system were implemented to assess how truly real-time it can be. As noted in Section 4.1, different implementations require different numbers of RK4 steps depending on the precision of the numbers used in the calculation. Profiling a CPU-based implementation using Linux's `perf` showed that most of the time was being taken by power function computations. Hence, a GPU-based version was also implemented. We only compare implementations without visual artifacts.

Table 1 shows the resulting frame rate when our different implementations are run on a late-2015 Macbook Pro. Our CPU-based implementations use a single core, and the GPU-based implementation

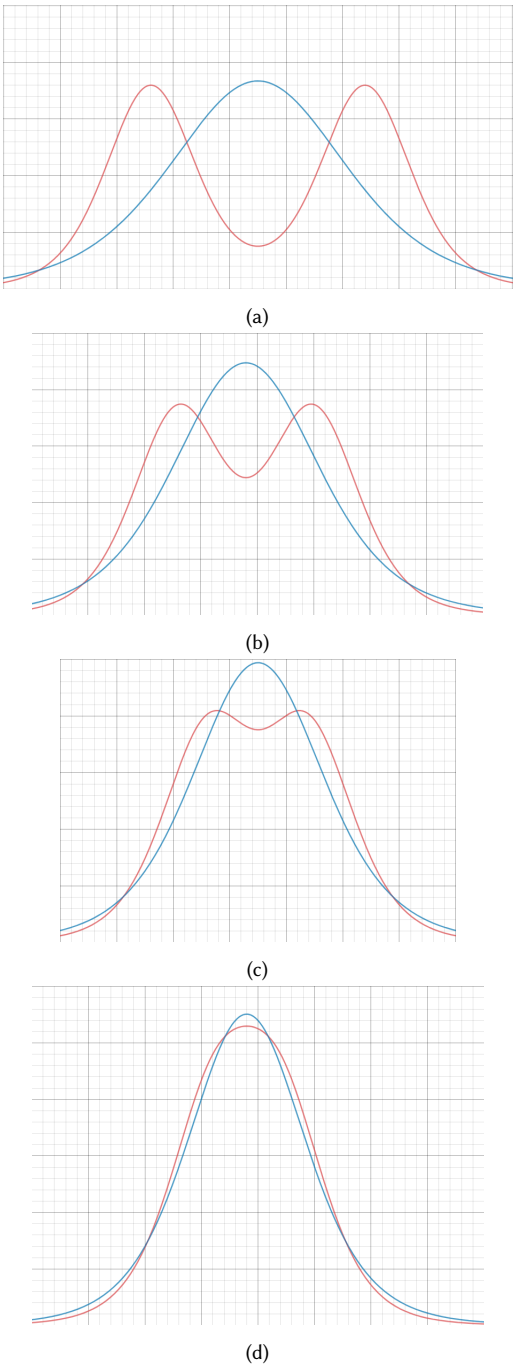

Fig. 3. Demonstration of combining Dynamic Kelvinlets. Two Kelvinlets, whose sum is shown in red, are combined and approximated to form whe Kelvinlet in blue. The value of regulariztion ϵ for the initial Kelvinlets is 1. The value of regularization for the resultant combined Kelvinlet is obtained by solving Equation 1. The location of the centre of the combined force is taken as the weighted mean of the initial forces. (a)-(d) show how the initial force waveforms affect the resultant Kelvinlet force.

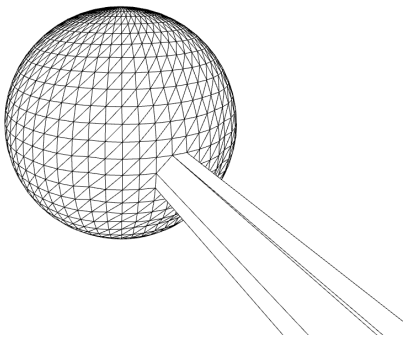

Fig. 4. Catastrophic precision error around the impulse centre.

Algorithm 1 Overall secondary motion generation algorithm

procedure GenerateKelvinlets(M : *Mesh*)
  $X : [x_1, ..., x_n] \leftarrow$ GetVertexPositions(M)
  $\ddot{X} : [\ddot{x}_1, ..., \ddot{x}_n] \leftarrow$ GetVertexAccelerations(M)
  $K \leftarrow \{\}$ ▷ *Initialize set of new Kelvinlets*
  for $i \in \{1, ..., n\}$ **do** ▷ *Add a Kelvinlet for each vertex*
    $k_i \leftarrow$ MakeKelvinlet($x_i, \ddot{x}_i$)
    $K \leftarrow K \cup \{k_i\}$
  $P \leftarrow \binom{K}{2}$ ▷ *Get pairs to possibly merge*
  for $(k_a, k_b) \in P$ **do**
    if dist$(k_a, k_b) < d_{\max} \wedge$ angle$(k_a, k_b) < a_{\max}$ **then**
      $k' \leftarrow$ MergeKelvinlets(k_a, k_b)
      $K \leftarrow K \setminus \{k_a, k_b\}$ ▷ *Remove original Kelvinlets*
      $P \leftarrow \{P : (k_c, k_d) \mid k_c, k_d \in K\}$ ▷ *...and their pairs*
      $P \leftarrow P \cup \{k'\} \times K$ ▷ *Add pairs with merge result*
      $K \leftarrow K \cup \{k'\}$ ▷ *Add the merge result itself*
  return K
procedure Update(T : *Timeline*, $t : \mathbb{N}$)
  $X_{t-2}, X_{t-1}, X_t \leftarrow$ GetVerticesAtTimes($T, \{t-2, t-1, t\}$)
  $\dot{X}_{t-1}, \dot{X}_t \leftarrow X_{t-1} - X_{t-2},\ X_t - X_{t-1}$
  $\ddot{X}_t \leftarrow \dot{X}_t - \dot{X}_{t-1}$
  $M \leftarrow$ MeshWithAccelerations($X_t, \ddot{X}_t$)
  $K_{\text{all}} \leftarrow K_{\text{all}} \cup$ GenerateKelvinlets(M)
  Display(M, K_{all}) ▷ *Display mesh using displacement shader*

Device	Precision	RK4 Steps	**FPS**
CPU	double	1	**11.8**
CPU	single	4	**3.8**
GPU	single	4	**55.6**

Table 1. Comparison of frame rates, in frames per second (FPS), of different Dynamic Kelvinlet implementations, with two Kelvinlets active.

uses the device's onboard Intel Iris Graphics 6100 chip. The reduced-size Stanford Dragon [Laboratory 1996] mesh is used, which has 5205 vertices. Two Kelvinlet forces, one push and one impulse, are active on the mesh. The simulation frame rate is capped at 60 frames per second due to the limitations of the platform.

Even with 4 RK4 steps, the GPU-based implementation achieves significantly higher frame rates, nearly keeping up with the maximum platform refresh rate. This result supports the original paper's claim that the system can run in real time.

We also tested performance with multiple Kelvinlets influencing the mesh, which happens in practice when automatically adding Kelvinlets to an animation. If too large a number of Kelvinlets are required, the shader will not compile, as only 1024 registers will reliably be present when compiling the shader [**?**]. In our implementation, we can support 203 Kelvinlets before, in conjunction with other uniforms required by the shader, this value is surpassed. Using an NVIDIA GeForce GTX 1660 GPU, a target frame rate of 60fps is maintained with 203 active Kelvinlets.

4.3 Simulation Quality

Although Dynamic Kelvinlets use physics to compute displacements over a field, real meshes are not infinite fields: they have boundaries and are not connected by an invisible, uniform continuum. To assess how plausible results look despite this assumption, we compare Kelvinlet forces applied to a continuum with a more traditional physics simulation of the application of a roughly equivalent force on a deformable body without this assumption.

We used the Bullet Physics [Bul 2018] simulator packaged with Blender 2.8 [Ble 2019]. This models the mesh as a mass-spring system with springs along the edges of the mesh. A direct comparison with Dynamic Kelvinlets is not entirely possible because the two systems use different models. A single Dynamic Kelvinlet has a position, force, and amount of regularization; in Bullet, there is a force, position, and force field shape. The field shape defines the direction and magnitude of force for each point in space. Field shape options include vectors facing away from a point, vectors facing away from a line, vectors normal to a plane, and vectors pointing away from an arbitrary surface. The best analogue to a regularized Kelvinlet force in Bullet is a force field emanating from a point with an exponential distance falloff. For material properties, the Dynamic Kelvinlets system uses stiffness and compressibility; Bullet uses mass, spring stiffness, and friction. Once a force is created in Bullet, the material properties are manually tweaked to attempt to best match the Dynamic Kelvinlet response.

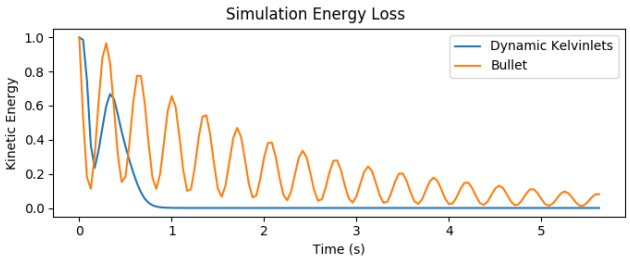

Fig. 5. Kinetic energy over time present in the Dynamic Kelvinlet and Bullet simulations of the application of a single regularized impulse.

Immediately after the application of an impulse, the displacement in the neighbourhood of the impulse centre deforms similarly in the two systems. As time progresses, Bullet and Dynamic Kelvinlet

responses diverge significantly. The most noticeable difference is that Bullet's mass-spring model continues to oscillate long after the impulse has been applied, whereas the Dynamic Kelvinlet response has a single pressure wave and a single shear wave. Figure 5 shows the kinetic energy over time for both simulations, communicating how the Bullet simulation continues moving for longer than the Dynamic Kelvinlet simulation. Note that since spring energy internal to the Bullet simulation is inaccessible when treated as a black box, only kinetic energy is shown rather than total energy of the system.

To assess the impact of the continuum assumption made by Dynamic Kelvinlets, we test on the Stanford Dragon [Laboratory 1996] as a true wave would have to travel through its winding body; it would be unable to propagate directly outwards from the force centre as it would in a continuum with no gaps in space. While the continuum approximation produces inaccuracies on sharp, targeted forces, with large enough regularization of the impulse force, there is not much noticeable difference in displacement wave propagation between the continuum approximation and the mass-spring model. This is because the regularization spreads the force out enough that it has some influence across spatial gaps directly, without needing to propagate through the volume of the mesh.

The response of force on soft bodies is usually perceived as a wave traveling through the body. Depending on the properties of the soft body, the dynamics might change, but the wave-like motion is well known. Spring-based models tend to give an oscillatory motion about the original location, which can differ from real life soft bodies when there are insufficient damping parameters. Dynamic Kelvinlets have an advantage over spring based methods for such cases, where erring on the side of having too few oscillations is preferable. This also results in more easily controllable responses, as the influence of a single force has a tighter bound in time, which can be preferable for artists and animators.

Another point of note is that the input to Dynamic Kelvinlets includes the material properties of the soft body rather than the mass and the spring constants. Not only does that make the setup of the simulation faster, it also makes it more intuitive as it is easier to identify material properties than to identify the spring constants for an object.

A final note is that the Bullet simulation ran at around 19 frames per second on the CPU. This type of simulation has the capability of achieving decent performance, but its ability to parallelize further is limited due to information dependencies in steps of the simulation algorithm.

4.4 Secondary Motion Quality

To analyze the quality of the generated Kelvinlet-based secondary motion, we produced a number of models with linear blend skinned primary animation. These models and animations were passed through our system to produce real-time output. To illustrate the effect of the generated Kelvinlets on the meshes, some frames are shown in Figures 1 and 6. The resulting animations themselves can be viewed in our implementation on Github [Singh and Pagurek 2019].

It is first important to clarify what "primary motion" entails. By analogy to other aspects of computer graphics, secondary motion is to primary motion as texture is to geometry. In John Lasseter's paper describing the 12 Principles of Animation [?], secondary motion is "the action of an object resulting from another action." Secondary motion is necessarily smaller than and subordinate to primary motion because its role in storytelling is to add depth and realism and not to take focus away from the acting, which is done by the primary motion. Lasseter states that if secondary motion "conflicts, becomes more interesting, or dominates in any way, it is either the wrong choice or is staged improperly." Larger actions that are arguably still the result of another action are likely not secondary motion and instead fall into the definition of *follow-through:* the termination of an action. Segments of a character that are attached to the part leading movement will "move at a slower speed and 'drag' behind the leading part of the figure," continuing to move after the leading part has stopped. In our system, this is considered part of primary motion and must be manually animated in the linear blend skin keyframes.

As noted when discussing the Bullet physics simulation, Kelvinlets on their own do not produce oscillations. While this is a technical limitation of Kelvinlets, the addition of large-scale oscillation where none is present in the input is likely not desired by animators. The oscillation of bones in character animation typically is the result of follow through action as leading bones slow down and stop, causing following bones to overshoot and bounce back. Being part of primary motion, this is something we expect animators would want control over and would prefer to specify in the input. The secondary motion we produce instead takes the form of small waves and ripples travelling through the mesh.

We find that our generated secondary motion is quite plausible for dramatic movements. In such inputs, the likelihood that any given part of the mesh surface will be undergoing some type of secondary motion deformation is likely. This masks the fact that Kelvinlet waves travel through empty space, so the continuum assumption does not produce any distracting artifacts. For subtler movements such as a walk cycle, artifacts from the continuum assumption become more noticeable, especially when material properties are chosen that exaggerate the squash and stretch.

5 FUTURE WORK

The largest issues with our system come from the continuum assumption made by Kelvinlets. This can potentially be overcome by discretizing the mesh into segments through which waves can propagate freely without travelling through space unrealistically. With each segment simulated with their own separate sets of Kelvinlets, no waves would be found travelling through empty space. Boundary conditions would need to be addressed by adding Kelvinlets to neighbouring segments as waves reach segment intersections.

Our current system assumes that the mass of a mesh is equal to the user-provided mass of the whole mesh divided evenly across its vertices. This firstly assumes that vertex density in the mesh is propoertional to mesh mass, which may not be true. Secondly, this makes the assumption that the mass of the mesh is spread across its surface, which also may not always mirror reality, where the movement on the surface is a result of the movement throughout the volume. A mechanism to allow for correction of both issues would

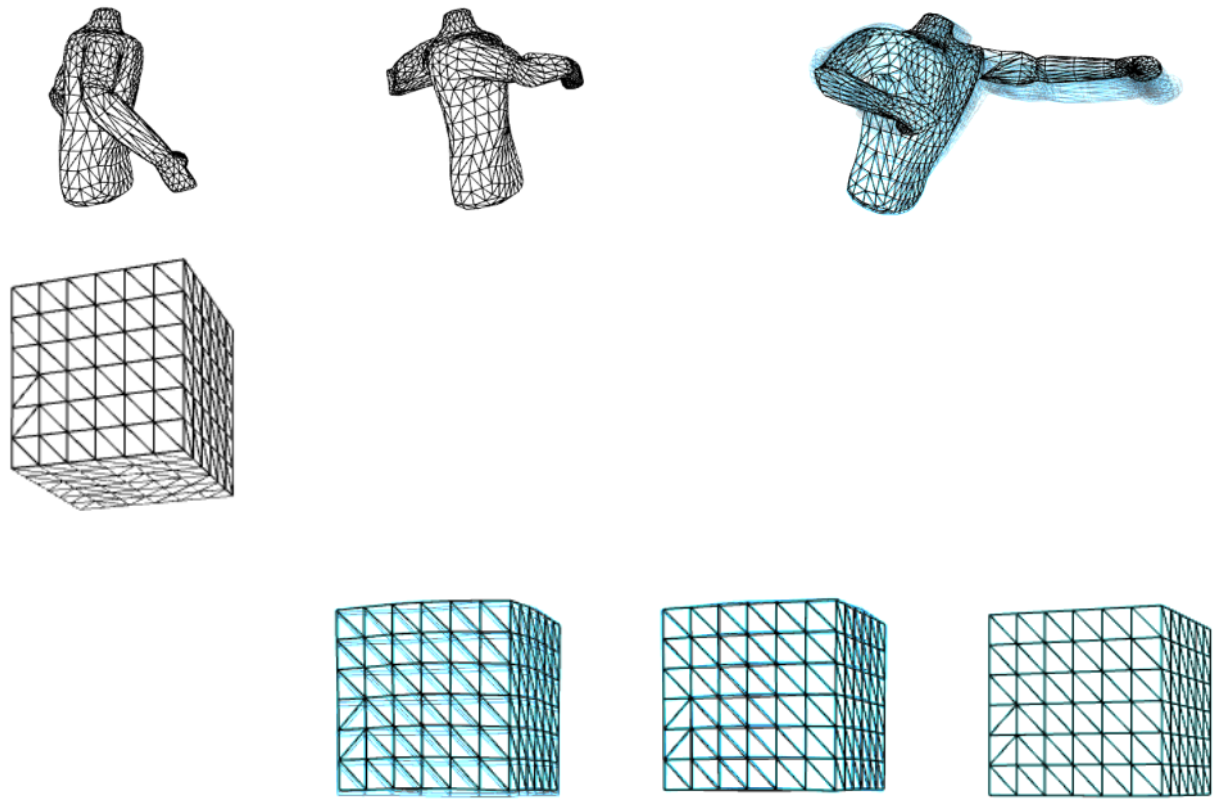

Fig. 6. Frames from linear blend skinned animations with frames of Kelvinlet-induced deformation motion superimposed.

be to specify the effective mass of a vertex as an additional vertex attribute, possibly through UV mapping. This allows mass to be easily redistributed. It does not on its own account for the volume beneath the mesh surface, but it would provide a mechanism to allow an artist to increase or decrease the effective mass at a vertex to more closely match the intended material properties.

It should also be mentioned that the system does not take into account undeformed objects. A deformed object may end up clipping through other objects in a scene, if any are present. There is the potential for work that builds off of this system while attempting to address this issue in a way that does not prevent efficient implementation of the GPU.

REFERENCES

2018. Bullet Physics SDK 2.88. https://github.com/bulletphysics/bullet3
2018. OpenFrameworks 0.10.1. https://openframeworks.cc
2019. Blender 2.81. https://www.blender.org/download/releases/2-81
Fernando De Goes and Doug L. James. 2017. Regularized Kelvinlets: Sculpting Brushes Based on Fundamental Solutions of Elasticity. *ACM Trans. Graph.* 36, 4, Article 40 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073595
Fernando De Goes and Doug L. James. 2018. Dynamic Kelvinlets: Secondary Motions Based on Fundamental Solutions of Elastodynamics. *ACM Trans. Graph.* 37, 4, Article 81 (July 2018), 10 pages. https://doi.org/10.1145/3197517.3201280
Stanford Computer Graphics Laboratory. 1996. Stanford 3D Scanning Repository. http://www-graphics.stanford.edu/data/3Dscanrep
Jasmeet Singh and Dave Pagurek. 2019. Dynamic Kelvinlets. https://github.com/davepagurek/DynamicKelvinlets

A CONTRIBUTIONS

Dave set up the framework for the project in C++ and GLSL, implementing impulse Kelvinlets in both. He created an equivalent test case both in the Dynamic Kelvinlets project and in Blender using the Bullet physics simulator. He set up the pipeline to process animated meshes and generate Kelvinlets from them each frame. He investigated the behavior of regularized Kelvinlet forces and created formulas for approximating two Kelvinlets with a single Kelvinlet. He modeled, rigged, and animated the animated meshes used for tests in this report using Blender.

Jasmeet implemented push Kelvinlets in C++, generalizing both types of Kelvinlets under a common interface. He later implemented push Kelvinlets in GLSL after profiling the C++ implementation to identify whether or not there were fixable bottlenecks. He added code to export image sequences and convert them to video files. When automatically generating secondary motion, he investigated the magnitude of the forces involved with common actions and experimented with the parameters in Kelvinlet generation to find ones that work reasonably for these actions.

Conception of experiments and analysis of their results was a collaborative effort.