Dynamic Kelvinlets

JASMEET SINGH and DAVE PAGUREK

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. 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 a 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

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:

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

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 1. Regardless of the value of ϵ, the total amount of force applied to the continuum is preserved:

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

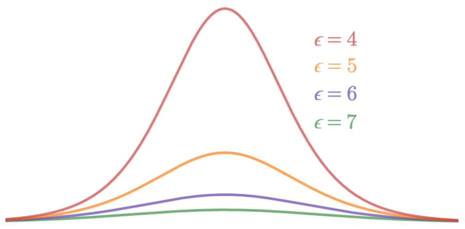

Fig. 1. Force density functions using different amounts of regularization.

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. Additionally, 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.

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 of exactly 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:

$$\begin{aligned} \|F'\| &= \|F_1\| + \|F_2\| \\ \hat{F}' &= \frac{\|F_1\|\hat{F}_1 + \|F_2\|\hat{F}_2}{\|F_1\| + \|F_2\|} \\ x' &= \frac{\|F_1\|x_1 + \|F_2\|x_2}{\|F_1\| + \|F_2\|} \end{aligned}$$

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:

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

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 $r \longrightarrow 0$ 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.

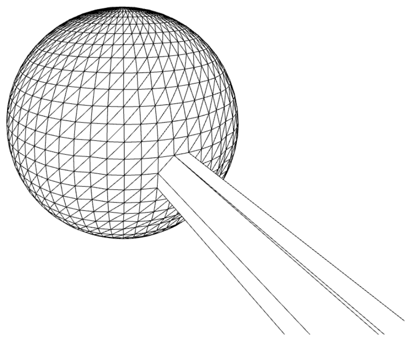
Fig. 2. Catastrophic precision error around the impulse centre.

Algorithm 1 Overall Mesh Generation algorithm

```
1:  procedure KelvinletGenerator::generate($Mesh\ \&M$)
2:      $v_i \leftarrow M$                          ▷ get current vertex locations
3:      $a_i \leftarrow A$                          ▷ get accelerations from animated mesh A
4:      $list < kelvinlet > potentialKelvinlets$
5:      for v ∈ $A$ do                              ▷ for all vertices v in A
6:          $kelvinlet_i \leftarrow a_i$
7:          $potentialKelvinlets.push\_back(kelvinlet_i)$
8:      for $k1 \in potentialKelvinlets$ do
9:          for $k2 \in potentialKelvinlets$ do
10:             if $distance(k1, k2) < threshold1\ \&\ angle(k1, k2) < threshold2$ then
11:                 $combine(k1, k2)$
12: procedure AnimatedMesh::update($float\ \&time$)
13:     $A_{t-2},\ A_{t-1},\ A_t \leftarrow animationLoader$        ▷ get keyframes
14:     $v_{t-2},\ v_{t-1},\ v_t \leftarrow A_{t-2},\ A_{t-1},\ A_t$   ▷ get vertex locations
15:     $a_{t-2},\ a_{t-1},\ a_t \leftarrow A_{t-2},\ A_{t-1},\ A_t$   ▷ calculate accelerations
16:     $Mesh\ M \leftarrow v, a$        ▷ create displaced mesh from vertex locations and accelerations
17:     generate($M$)                    ▷ generate kelvinlets for the mesh
18:     $display(M)$ ▷ display update mesh using OpenFrameWorks
```

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.

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.

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 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 [Wiki 2019]. 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.

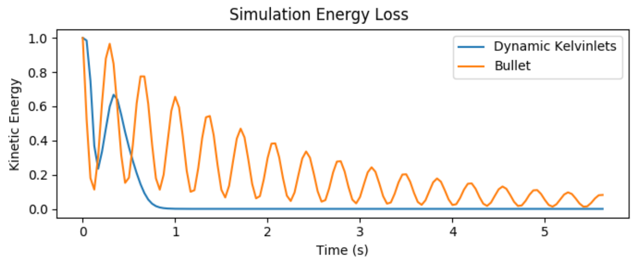

Fig. 3. Kinetic energy over time present in the Dynamic Kelvinlet and Bullet simulations of the application of a single regularized impulse.

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 3 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 rather than propagating 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

TODO

5 FUTURE WORK

TODO

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

OpenGL Wiki. 2019. Uniform (GLSL) — OpenGL Wiki,. http://dev.khronos.org/opengl/wiki_opengl/index.php?title=Uniform_(GLSL)&oldid=14539 [Online; accessed 7-December-2019].