A Lightweight Rigid-Body Verlet Simulator for Real-Time Environments

Ben Kenwright

***Abstract*—In this paper, we present a real-time rigid-body simulation technique based upon the popular position-based integration scheme (Verlet). The Verlet technique has gained popularity due to its intuitiveness and simulation stability (e.g., coupled soft-body systems, such as, cloths). We explain a simplified technique based-upon the Verlet approach for creating a robust rigid-body solution for dynamic environments (e.g., objects flying around while interacting and colliding with one another). What is more, we take the traditional particle-Verlet scheme and expand it to accommodate both angular and linear components. With this in mind, we formulate simple constraints (e.g., ball-joints and collision-contacts) to reconcile and resolve coupled interactions. Our algorithm works by approximating the rigid-body velocities (angular and linear) as the different between the current and previous states. Constraints are enforced by injecting corrective transforms that 'snap' violating positions and orientations out of error. The coupled rigid-body system is iteratively solved through relaxation to help convergence on an acceptable global solution. This addresses the issue of one constraint fighting with another constraint. We estimate corrective measures and iteratively apply updates to ensure the simulation correlates with the laws-of-motion (i.e., moving and reacting in a realistic manner). Our approach targets visually plausible systems, like interactive gaming environments, by reducing the mathematical complexity of the problem through ad-hoc simplifications. Finally, we demonstrate our rigid-body system in a variety of scenarios with contacts and external user input.**

Keywords–simulation, verlet, interactive, real-time, rigid-body, collisions, constraints

I INTRODUCTION

Interactive Environments Virtual environments are rich with motion. Highly interactive worlds contain dynamic and complex bodies. Bodies that fly around and bounce off one another. Typically, these scenes need to 'appear' detailed and move in a life-like way while running at real-time frame-rates. These action packed scenes have characters falling, stones flying, dust particles swooshing, and trees swaying. While we have an abundance of moving objects, it is also important that the user and objects are able to interact with one another. A lightweight solution is sufficient, as we are concerned with aesthetic properties, when buildings collapse, debris and dust is flying around in a dynamic and realistic way. Combined with forces from high-winds and impact explosions to produce an overall visual effect that is satisfying and captivating. Hence, we do not need an approach that is highly accurate but aesthetically pleasing.

Interesting & Important An intuitive, stable, and efficient rigid-body simulator is a valuable and important tool for any real-time interactive environment. This enables us to create emmersive dynamic worlds. The challenge is to synthesize coupled and uncoupled motion in a realistic manner efficiently with less emphasis on physical accuracy. We address engineering trade-offs, that trade scientific precision for simplicity and speed, while maintaining the fundamental rigid-body motion mechanics (visually). The Verlet integration scheme for calculating the equations of motion for rigid-bodies might be less accurate than other integration methods, but possesses favourable characteristics (i.e., it is inexpensive, stable and does not require the computation of velocities).

Currently At the present time, rigid-body systems primarily use Newtonian mechanics to solve the equations of motion (e.g., velocity-impulses, penalty-forces, constrained Lagrangian mechanics) [1]–[4]. The Verlet technique used in this paper has primarily been used for point-mass systems and soft-body simulations (cloths). For interactive media, however, we are less concerned with accuracy but aesthetic qualities. In dynamic situation (e.g., vehicles colliding, collapsing building, and flying debris), approximations and errors often go unnoticed (intersections and penetrations). Our lightweight approach focuses on these situations.

Components A simulation is the imitation of the real-world processes and their changes with respect to time. This includes a number of different components that work in synergy, such as, collision detection, resolution, coupled constraints and process parallelisation. We concentrate on the dynamic equations for simulating real-time interactive situations. The approach is able to solve a variety of common constraint and produces motions that correlate with physically-accurate models (i.e., classical mechanic formulations). Techniques include, velocity impulses, penalty forces, constraint solving methods. Each rigid-body is subject to a forces and torques (gravity and wind) and constraints (ball-joints and contacts) to prevent shapes interpenetrating with each other and the environment. A relaxation technique is used to resolve these 'concurrent' constraints (help the system converge on a global solution and reduce constraint fighting).

Contributions The key technical contributions of this paper are: (1) the formulation of the Verlet integration scheme for use in a real-time three-dimensional rigid-body simulator; (2) show that a high precision solution is not essential for most gaming situation and an uncomplicated second order velocity-Verlet or position-Verlet works well enough; (3) demonstration of the Verlet method as an effective solution for interactive environments (creating realistic motion for real-time systems and supporting arbitrary interactions); simulation and (4) the parallelisation of the technique using the graphical processing unit (GPU).

II RELATED WORK

Quantum Mechanics to Interactive Gaming Environments The Verlet integration scheme was born in the molecular dynamics field [5], but later gained recognition in other domains, such as, computer graphics [6]. During the 2001 Game Developer's Conference, Thomas Jakobsen [7] presented

a simulation technique for point-mass systems, such as, clothes and other soft bodies problems. The particle-Verlet was proven as a viable commercial solution in the video game Hitman: Codename 47 [8] - providing an efficient and stable ragdoll technique (i.e., dead human bodies). Of course, due to the simplicity and stability of the Verlet technique, the particle-Verlet method was expanded to include rigid-bodies. Porcino [9] presented a working application of a Verlet-Based physics engine. At the same time, Baltman and Radeztsky [10], presented a Verlet integration and constraints system for a six degree of freedom rigid-body physics simulation.

On a side note, it can be said that the Verlet technique is a modified impulse solution, since it instantaneous changes the position and velocity (i.e., velocity impulses). Another key thing to remember is, the basic Verlet integration scheme does not handle variable frame rates well. However, with a small modification, we can adapt the technique to account for changing time-steps.

Verlet Flavours The Verlet integration technique comes in three basic types (see Section VII for further details):

- Position-Verlet [11]
- Leapfrog-Verlet [12], [13] (Leap-frog Verlet method which is explained clearly by Hut et al. [14]).
- Velocity-Verlet

Our Approach We present a method for efficiently modelling complex rigid-body interactions in dynamic situations through lightweight approaches. We employ an ad-hoc solution based on the popular Verlet-system to reduce overheads yet maintain core aesthetic qualities. Our lightweight approach eases the memory and computational overhead for less accurate dynamic situations.

The Newton-Euler formulation [15] for a 6 degrees-of-freedom (dof) rigid body is given by Equation 1:

$$\begin{aligned} f &= ma \\ \tau &= \omega \times (I\omega) + I\alpha \end{aligned} \tag{1}$$

where f is force, τ torque, m mass, a acceleration, I inertia tensor, ω angular velocity, and α is the angular acceleration. For our simplified approach, we neglect the angular component from the Coriolis ($\omega \times (I\omega)$) leaving us the simpler Equation 2:

$$\begin{aligned} f &= ma \\ \tau &= I\alpha \end{aligned} \tag{2}$$

III METHOD

Verlet vs Euler (or Runge-Kutta) The advantages and disadvantages of a technique depend upon its target use. As an aesthetically pleasing simulation solution is more useful compared to a more accurate scientific model for interactive gaming environments. The different integration methods for interactive physics-based simulation possess numerous advantages and disadvantages. We show the limitations of a position-based approach (i.e., modified Verlet technique), including experiments to support and software engineering workarounds. We investigate the trade-offs between a motion appearing 'plausible' but not being realistic due to numerical simplifications - but provides improved stability in general [15], [16].

Constraints, Collisions, and Contacts One way of reacting to collisions is to use a penalty-based system which basically applies a set force to a point upon contact. The problem with this is that it is very difficult to choose the force imparted. Use too strong a force and objects will become unstable, too weak and the objects will penetrate each other. Another way is to use projection collision reactions which takes the offending point and attempts to move it the shortest distance possible to move it out of the other object. The Verlet integration would automatically handle the velocity imparted via the collision in the latter case, however note that this is not guaranteed to do so in a way that is consistent with collision physics (that is, changes in momentum are not guaranteed to be realistic). Instead of implicitly changing the velocity term, you would need to explicitly control the final velocities of the objects colliding (by changing the recorded position from the previous time step). The two simplest methods for deciding on a new velocity are perfectly elastic collisions and inelastic collisions. A slightly more complicated strategy that offers more control would involve using the coefficient of restitution. Our rigid-body Verlet integration (in which velocity is implicit) corrects constraint violations by implicitly change the angular and linear velocities (analogous to an applied spring forces).

Connecting Newtonian & Verlet A first order simplectic Euler integration (sometimes called the Euler-Cromer) - the reason the we call the integrator simplectic is the modified velocity (v_{n+1}) is used to update the position - helping to improve stability (i.e., semi-implicit feedback).

$$\begin{aligned} v_{n+1} &= v_n + a\ \Delta t \\ x_{n+1} &= x_n + v_{n+1}\ \Delta t \end{aligned} \tag{3}$$

We approximate velocity as:

$$v = \frac{x_n - x_{n-1}}{\Delta t} \tag{4}$$

We substitute this into our simplectic Euler for v_n:

$$\begin{aligned} v_{n+1} &= \left(\frac{x_n - x_{n-1}}{\Delta t}\right) + a\ \Delta t \\ x_{n+1} &= x_n + v_{n+1}\ \Delta t \end{aligned} \tag{5}$$

Combining into a single representation for p_{n+1}:

$$\begin{aligned} x_{n+1} &= x_n + [\left(\frac{x_n - x_{n-1}}{\Delta t}\right) + a\ \Delta t]\ \Delta t \\ &= x_n + \frac{x_n - x_{n-1}}{\Delta t}\Delta t + a\ (\Delta t)^2 \\ &= x_n + (x_n - x_{n-1}) + a\ (\Delta t)^2 \\ &= 2x_n - x_{n-1} + a\ (\Delta t)^2 \end{aligned} \tag{6}$$

We are able to calculate the approximated velocities as $v_n = \frac{1}{2\Delta t}(x_{n+1} - x_{n-1})$, since having the velocities is frequently useful for some calculations.

Rotation & Translation We use quaternions to represent a rigid-body's rotation. This represention is intuitive especially for the calculation of rotational constraints. The concatenation and difference calculation of quaternions is performed through multiplication and conjugate. Using traditional quaternion angular integration (see Section VIII), we formulate the equations for the rigid-body Verlet:

$$\begin{aligned} &\text{Translation:} \\ &\quad x_{n+1} = x_n + (x_n - x_{n-1}) + a\,(\Delta t)^2 \\ &\text{Rotation:} \\ &\quad q_{n+1} = q_n\,(q_n\,q_{n-1}^*) + \frac{1}{2}(\alpha(\Delta t)^2)(q_n) \end{aligned} \tag{7}$$

where a is linear acceleration, $\alpha = \frac{\tau}{I}$ is the angular acceleration, x represents the translation and q the orientation as a unit-quaternion.

Penalties For traditional Newtonian mechanics, we have the well known connection between force and torque:

$$\tau = r \times f \tag{8}$$

where τ is the torque, r is the offset from the centre of mass to the point we are applying the force f. Equation 8, enables us to calculate penalty forces at specific points on a rigid-body, so we are able to inject corrective motions.

For the particle-Verlet constraint solution, we correct constraint errors by 'moving' (or snapping) each particle's position out of error. However, for our rigid-body solution, we have both a rotational and translational component. Hence, we use the founding formula given in Equation 8 to derive Equation 9 below (with reference to Figure 1).

$$\begin{aligned} \Delta\theta &= r \times \hat{n}\Delta d \\ \Delta x &= ((r \times \hat{n}\Delta d) \times \hat{n}) \cdot \hat{n} \end{aligned} \tag{9}$$

Δx provides the translational penalty, and $\Delta\theta$ provides the axis-angle penalty that we convert to a quaternion.

The penalty corrections are applied over multiple iterations in combination with a relaxed over time to provide stability at the expense of error. This enables the system to converge on an acceptable solution and reduce constraint fighting. As shown in Equation 10 below, we multiple the solution by a scalar c_r to help the simulation converge. This is especially important for coupled problems, multiple contacts and chains of items.

$$\begin{aligned} \Delta\theta &= (r \times \hat{n}\Delta d\,)\,c_r \\ \Delta x &= (((r \times \hat{n}\Delta d) \times \hat{n}) \cdot \hat{n}\,)\,c_r \end{aligned} \tag{10}$$

Velocity-Verlet & Position-Verlet In this paper, we focus on the 'position-Verlet' technique, however, there are different types of Verlet integration schemes. A related, and more commonly used, algorithm is the velocity-Verlet algorithm [17], similar to the Leapfrog method [14], except that the velocity and position are calculated at the same value of the time variable (Leapfrog does not, as the name suggests). This uses a similar approach but explicitly incorporates velocity, solving the first time step problem in the basic Verlet algorithm.

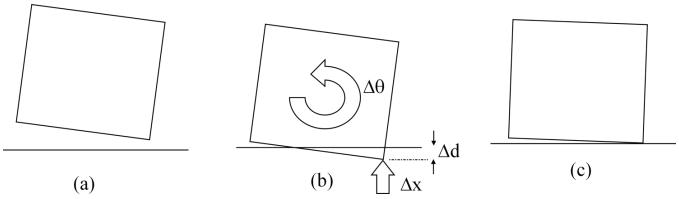

Figure 1. **Penalty Correction** - Corrective position and orientation penalty.

Time-Corrected Verlet A disadvantages of the Verlet method is it handles changing time steps badly, i.e., it is not a self-starter (it requires 2 steps to get going, so initial conditions are crucial), and it is unclear from the formulation how it handles changing accelerations. The modified Verlet integrator is referred to as the Time-Corrected Verlet (TCV) and is shown below with its original counterpart:

$$\begin{aligned} &\text{Original Verlet:} \\ &\quad x_{n+1} = x_n + (x_n - x_{n-1}) + a(\Delta t)^2 \\ &\text{Time-Corrected Verlet:} \\ &\quad x_{n+1} = x_n + (x_n - x_{n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + a(\Delta t_n)^2 \end{aligned} \tag{11}$$

Damping The Verlet equations can also be modified to incorporate a simple damping component (for instance, to emulate air friction):

$$\begin{aligned} &\text{Without Damping:} \\ &\quad x_{n+1} = 2x_n - x_{n-1} + a(\Delta t)^2 \\ &\text{With Damping:} \\ &\quad x_{n+1} = (2-\beta)x_n - (1-\beta)x_{n-1} + a(\Delta t)^2 \end{aligned} \tag{12}$$

where β is the damping coefficient between 0.0 and 1.0. For the quaternion implementation, we integrate in damping in a similar way to the translational damping, but use an interpolation calculation.

No Silver Bullet Each physics-based technique offers pros and cons, with respect to simplicity, speed, accuracy, and stability - and we find that there is 'no' one shoe fit all - a best solution for every situation. However, for interactive environments, such as, video games, the Verlet technique offers provides certain advantages, as we are interested in asthetic qualities - compared to physical accuracy. To summarize, the key points:

- Pros
  - Quick to process
  - Practically any geometry can be simulated
  - Well suited for soft-body systems
  - Integration with other areas of physics, such as, pool balls and rope/cloth
  - Time-invariant (play-back)
- Cons
  - Has been described as 'bouncy' with few iterations

- Inaccuracies develop from relaxation
- Requires user intervention to get the collision detection and response to feel right

Pros

Parallelisation We distribute the work load of the simulation by double buffering the simulation parameters. Each rigid-body updates itself and writes its modified position and orientation to the next buffer. Once every rigid-body has updated the next buffer, the current and next buffer are swapped and the system is integrated forward in time (as shown by Westwood et al. [18]).

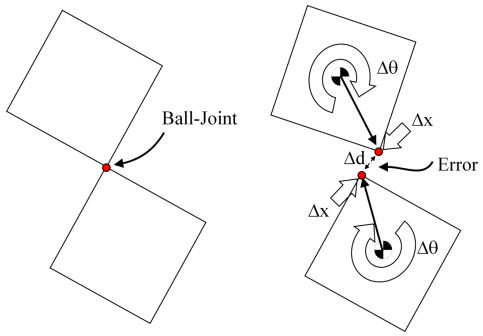

Figure 2. **Joint Constraint** - Connecting relative rigid-body offsets.

IV EXPERIMENTAL RESULTS

Simulation test-cases:

- single rigid-body falling and bouncing with ground
- number of rigid-bodies interacting
- chain of interconnected objects (e.g., bridges and ragdolls)
- stacking objects
- parallelisation of the constraint algorithm by ping-ponging updates between two copies of memory on the GPU

The technique is simple to implement and moves in a realistic manner (see Figure 3 and Figure 5). However, as shown in the simulations, the approach has some issues, for example, long chains of objects settle in a configuration that appears incorrect (see Figure 2). In our implementation, we did not cache contact points between frames, hence we observed a small amount of jittering, especially when multiple constraints influence a single rigid-body. For dynamic situations, with lots of interacting rigid-bodies flying around (see Figure 6, the simulation produced acceptable results.

V DISCUSSION

Overall the technique is simple to implement and produces asthetically pleasing results in the majority of cases for dynamic situations. To create more accurate solutions and solve specific cases, such as, stacks and resting bodies, the approach would require user intervention and tuning of parameters. We did not include any advanced techniques to help improve the simulations realism and performance, for instance, shock-propagation, sleeping, and contact caching.

We have concentrated on a rigid-body system, on the other hand, the approach we present in this paper is easily combined with other techniques (particles, deformations, and soft-bodies).

VI CONCLUSION

We propose a lightweight rigid-body simulation method that is stable, fast, and flexible enough for used in real applications, like computer games. Future work would be the development of additional constraint types. We also considered the translational and rotational components separately, while a more esoteric solution to investigate would be the unification of the mathematics using dual-quaternions. The creation of a more complex joints/constraints types using the Verlet integration model would be useful for simulating intricate systems.

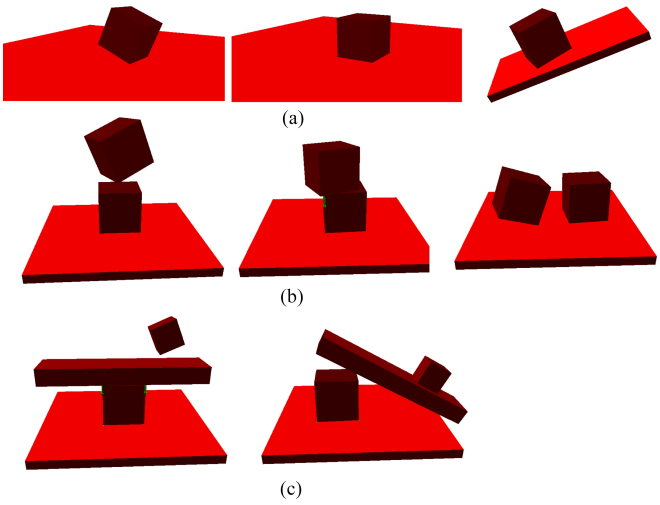

Figure 3. **Simple Test Cases** - A set of basic simulation tests to show the working mechanics of motion.

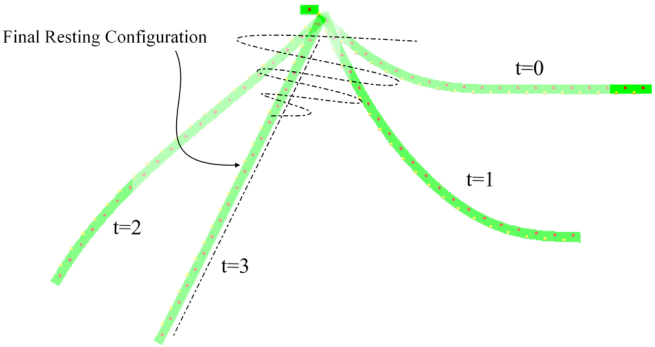

Figure 4. **Chain Rigid-Bodies** - 21 rigid-bodies connected via ball-joint constraints. The figure shows how the chain is dropped horizontally and oscillates back-and-forth due to gravity and damping. However, at the end of the simulation the chain settles in a straight line (i.e., not the ideal curved arc due to the weight distribution). We emphasis it here - but point out that in the majority of cases would not be noticed by the user (e.g., in dynamic situations or a systems with only a few links).

References

[1] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia. ACM, 2007, pp. 281–288. 1

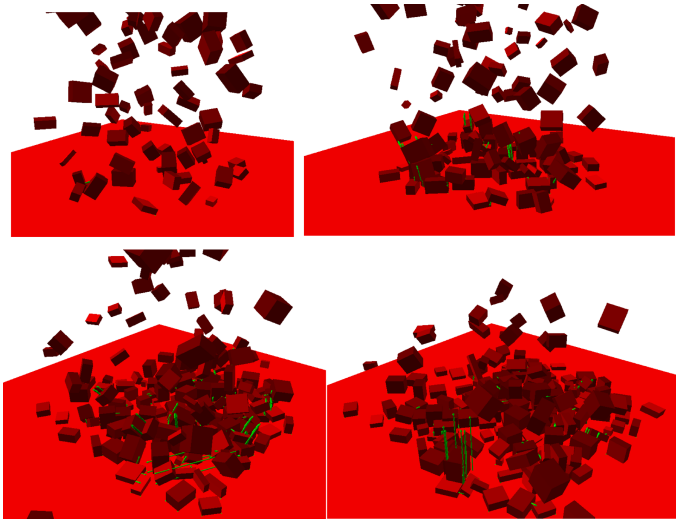

Figure 5. **Random Cubes** - 200 rigid-body cubes falling.

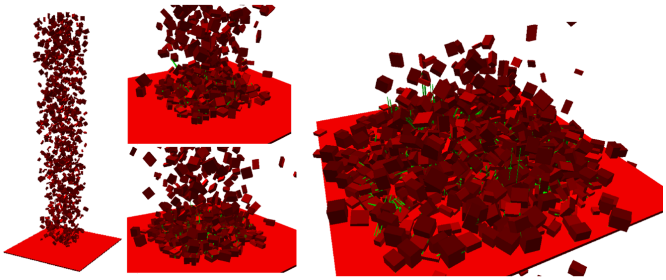

Figure 6. **Random Cubes** - 1000 rigid-body cubes falling. Our lightweight technique is well suited to 'dynamic' situations (multiple rigid-bodies flying around) - capturing the mechanics of motion and the responsiveness during impacts and collisions.

[2] R. Smith et al., "Open dynamics engine," 2005. 1

[3] E. Coumans, "Bullet physics engine," Open Source Software: http://bulletphysics. org, vol. 1, 2010. 1

[4] A. Bond, "Havok fx: Gpu-accelerated physics for pc games," in Proceedings of Game Developers Conference 2006, 2006. 1

[5] E. Barth, K. Kuczera, B. Leimkuhler, and R. D. Skeel, "Algorithms for constrained molecular dynamics," Journal of computational chemistry, vol. 16, no. 10, 1995, pp. 1192–1209. 1

[6] D. Rozmanov and P. G. Kusalik, "Robust rotational-velocity-verlet integration methods," Physical Review E, vol. 81, no. 5, 2010, p. 056706. 1

[7] T. Jakobsen, "Advanced character physics," in Game Developers Conference, 2001, pp. 383–401. 1

[8] Eidos Interactiven, "Hitman: Codename 47," in Video Game, 2000. 2

[9] N. Porcino, "(LucasArts), Writing a verlet-based physics engine," Game Programming Gems 4, April 2004, pp. 231–241. 2

[10] R. Baltman and R. Radeztsky Jr, "Verlet integration and constraints in a six degree of freedom rigid body physics simulation," in Game Developers Conference, 2004. 2

[11] D. Ramtal and A. Dobre, "Numerical integration schemes," in The Essential Guide to Physics for Flash Games, Animation, and Simulations. Springer, 2011, pp. 443–459. 2

[12] C. Raymaekers, E. CUPPENS, K. CONINX, and L. VANACKEN, "A comparison of different techniques for haptic cloth rendering," 2005. 2

[13] G. van den Bergen and D. Gregorius, Game physics pearls. CRC Press, 2010. 2

[14] P. Hut, J. Makino, and S. McMillan, "Building a better leapfrog," The Astrophysical Journal, vol. 443, 1995, pp. L93–L96. 2, 3

[15] B. Kenwright and G. Morgan, "Practical introduction to rigid body linear complementary problem (lcp) constraint solvers," Algorithmic and Architectural Gaming Design, 2012, pp. 159–205. 2

[16] B. Kenwright, Computational Game Dynamics. Digital Media (ISBN: 978-1501018398), 2016. 2

[17] P. F. Batcho and T. Schlick, "Special stability advantages of position-verlet over velocity-verlet in multiple-time step integration," The Journal of Chemical Physics, vol. 115, no. 9, 2001, pp. 4019–4029. 3

[18] J. D. Westwood et al., "A gpu accelerated spring mass system for surgical simulation," Medicine Meets Virtual Reality 13: The Magical Next Becomes the Medical Now, vol. 111, 2005, p. 342. 4

VII APPENDIX A

Position Verlet Algorithm takes the form:

$$\begin{aligned} x_{n+1} &= 2x_n - x_{n-1} + a\Delta t^2 \\ v_n &= \frac{1}{2\Delta t}(x_{n+1} - x_{n-1}) \end{aligned} \tag{13}$$

Verlet Leapfrog Algorithm takes the form:

$$\begin{aligned} v_{n+1/2} &= v_{n-1/2} + a\Delta t \\ x_{n+1} &= x_n + v_{n+1/2}\Delta t \\ v_{n+1} &= \frac{1}{2}[v_{n+1/2} + v_{n-1/2}] \end{aligned} \tag{14}$$

Velocity Verlet Algorithm takes the form:

$$\begin{aligned} x_{n+1} &= x_n + v_n\Delta t + \frac{a_n}{2}\Delta t^2 \\ v_{n+1} &= v_n + (a_{n+1} + a_n)\frac{\Delta t}{2} \end{aligned} \tag{15}$$

VIII APPENDIX B: QUATERNION INTEGRATION

$$q_{n+1} = q_n + (\frac{dq}{dt})\, dt \tag{16}$$

$$\begin{aligned} \frac{\mathrm{d}q}{\mathrm{d}t} &= \lim_{h\to 0}\frac{q(t+h) - q(t)}{h} \\ &= \frac{1}{2}\omega\, q \end{aligned} \tag{17}$$