

# Communications Latency Hiding for Distributed SoS

Loren Peitso

Computer Science Department  
Naval Postgraduate School  
Monterey, CA, USA  
lepitso@nps.edu

**Abstract** – *Communications latency in a real-time distributed system of systems makes it difficult to ensure the participating systems all have sufficiently common shared world-views. If the world-views are divergent, decision making can be compromised. Latency is also variable, making it difficult if not impossible to predict or repeat a set of communications and resulting actions exactly. We propose a means of hiding network latency for a real-time distributed system of systems which will enable a common, consistent, shared world-view. Latency hiding also supports use of other remote network services such as web services or cloud computation.*

**Keywords:** Command and control, system of systems, distributed simulation, network latency, latency hiding.

## 1 Introduction

Communications latency in a distributed system of systems (SoS) makes it difficult to impossible to ensure the participating systems all have a common shared world-view. If the world-views diverge, decision making may be compromised. Finding effective solutions to latency is difficult because of variability that makes it impossible to predict or exactly repeat a set of communications messages and their resulting actions, or even conclusively verify that a solution is correct. This leads to design decisions that minimize the importance of the shared world-view in favor of local information, thereby sacrificing overall accuracy and trust in the shared world-view.

Command, Control, Communications, Computing and Intelligence (C4I) systems such as cooperative engagement capability (CEC) exhibit latency related issues. It is essential for correct and efficient operation to eliminate as much latency as possible from the SoS world-view and decision-making loop. It is not possible from an economic and safety standpoint to experiment on a deployed C4I SoS, so a proxy is necessary. Distributed simulation SoS architectures have numerous similarities, in fact work on harmonizing C4I systems and simulation has been underway in projects such as the Battle Management Language [1] the Test and Training Enabling Architecture (TENA) development effort [2].

Both domains are profoundly limited by network latency difficulties, to the point that textbooks such as [3] state all network-shared information is out-of-date and [4]

calls the problem a fundamental property of distributed simulations. These latency problems must be solved to allow effective use of remote cloud-based services and massive distribution to increase scalability and accuracy in the distributed SoS simulation domain. A simplified scenario illustrating the latency problems of distributed SoS simulation is presented below.

### 1.1 Latency Breaks Decision-Making

Trainees Blue and Red are participating in a distributed combat training simulation. The Blue trainee is physically located in San Francisco and Red in the Los Angeles area, with an average 30-50 milliseconds of network latency between them. At the start both trainee's avatars are stationary, facing each other and the trainees react simultaneously. Red applies control inputs to run left and Blue shoots at Red, see Figure 1 below.

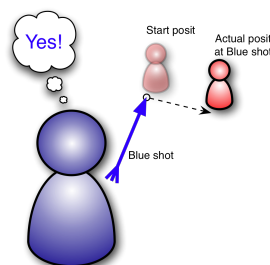


Figure 1. Simulation start and initial actions

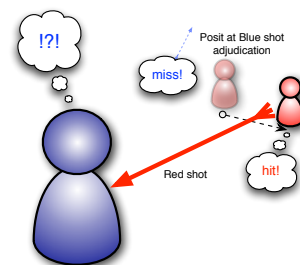


Figure 2. Simulation endgame

Trainee Blue believes it hit Red for a scenario win because on its monitor in San Francisco it appears that Red is square in Blue's crosshairs. But once the hit is adjudicated in Los Angeles based on accurate Red position information, it is assessed as a miss because Red has moved, see Figure 2. To Blue it appears that Red is immune to the effects of Blue's weapon! The situation is made worse by Blue remaining stationary believing it had won the scenario. This allows Red to attack a motionless target, without any negative effect from network latency.

### 1.2 Reconnection to the C4I SoS

C4I SoS latency arises from physical network transmission times and transmission cycle queuing, similar to the network transmission latency and time step

scheduling of simulations. C4I systems are also increasingly being connected to simulations to enable Live-Virtual-Constructive (LVC) training scenarios, further blurring the distinction of where the real world sensor inputs end and the simulation starts [2].

We see real-time distributed SoS simulation providing a safe and economically viable platform for examining the latency problem and testing potential solutions. Also that solving the “fundamental property” of latency in distributed SoS simulation can show a path to solving the latency-caused divergent world-view problems in C4I SoS. The solution to be presented in this paper should not require the C4I linking system to be redesigned. It should be implemented above the physical link layer just as the simulation version is implemented in the application layer above the physical networking architecture.

The rest of this paper is organized as follows: We start by outlining our solution to the network latency problem – a form of latency hiding. Then we cover three supporting areas that are combined in a novel way to enable this solution. The first area is simulation time advance, showing how the traditional techniques relate to asynchronous and adaptive methods. Then we examine physical motion evaluation and finish with a high level overview of a collision detection technique that enables the overall composition to integrate smoothly and safely into a distributed SoS simulation.

## 2 Proposed Solution: Latency Hiding

We propose that communicating entity state over the network not only include the current state, which is already time late by over-the-network transmission time, but also includes the predicted motion over the interval between the current and next computation. We make use of standard adaptive numerical techniques to compute entity motion and schedule computation using the adaptively computed time advance. This maximizes the time advance interval and we use this maximized interval to hide the network latency. A secondary benefit is that total network bandwidth requirements may also be reduced by reducing the number of entity state transmissions required.

### 2.1 Related work

The proposed technique is roughly analogous to CPU-memory bus latency hiding where instructions are advantageously issued and predictive prefetching techniques are used to provide data before it is “requested” for the next computation [5]. The closest related work on network communications latency hiding applies to cluster computing. That work maximizes throughput by scheduling computation during periods that would otherwise be wasted blocked-on-transmit periods [6] and related recapturing of unused CPU cycles. The problem is superficially similar to our distributed SoS latency problem. While enhancing the

efficiency of a single distributed computation, these cluster solutions stop well short of what is required to adequately hide latency in our domain. Related work in distributed simulation does not exist, as noted in [3][4], distributed simulations do not mitigate network latency, rather dead reckoning [7] or attribute extrapolation [8] techniques visually “paper over” latency caused artifacts.

## 3 Time Advance Methods

Time management in a distributed SoS is the critical obstacle to enabling world-view consistency. This differs from time-synchronization, we assume that the distributed SoS has an adequate physical clock synchronization mechanism between the distributed nodes. Simulation uses three primary time advancement methods [9]. Time step is the most prevalent method used in distributed SoS simulation with fixed and adaptive synchronous time step being the primary sub-methods. Adaptive asynchronous discrete event, is an example of the second method and is prevalent in analytical and constructive simulation. The final type, independent advancement is not suitable for this domain and won’t be covered here.

Short increment time advances may be used to increase computational accuracy but do not help system latency problems. Short time advances may also contribute to increasing bandwidth requirements. Many methods to reduce bandwidth both make latency worse and generate intentionally inaccurate shared world-views [3][4][7][8]. Long fixed time advances do not affect latency, but both local and remote world-view accuracy may suffer if objects change motion and the change is not reflected until the end of the interval, or when accuracy is sacrificed by applying an inappropriately long time advance for the numeric method used. Our latency hiding techniques will take advantage of adaptive asynchronous discrete event time advance which is not as widely used as the synchronous methods, so a quick description of each methodology with basic strengths and weaknesses is provided.

### 3.1 Fixed Synchronous Time Step

Fixed synchronous time step is the typical time advancement methodology used in distributed real-time simulations. All entity updates are evaluated at a fixed interval with wall and simulation clocks progressing identically. To support the hard scheduling requirement that all time step computations must be completed before simulation time advances/steps, implementations optimize for spatial and temporal locality of both code and the relationships between simulated entities.

When simulations use physically-based methods to represent the world, numeric stability drives an update rate of 100Hz or higher, making 10ms the maximum usable time advance interval. The tradeoffs between numerical stability and requisite timeliness favor use of low fidelity numerical methods. The penalty for violating timeliness is

profound artifacts or catastrophic failure, trumping the slight inaccuracies of low fidelity computations. Fixed time advance also provides simulation designers with a very clean and understandable simulation progression that maps visually to Figure 3. The  $\Delta t$  time step remains constant between the arrays representing the synchronously computed states.

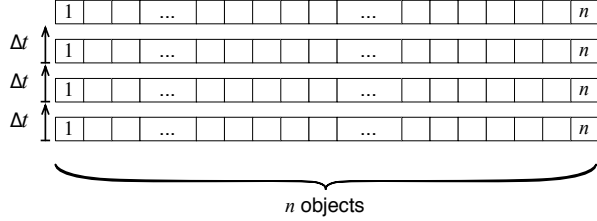


Figure 3. Fixed synchronous time step.

A problem with all time step methods is that the simulation state only exists at the time step intervals, there is no state during the time interval  $\Delta t$  “between” computed steps. This problem allows small fast moving objects to pass through thin objects between time steps and never register the physical contact that should have happened. This is covered in detail in Section 4.1 and Figure 7. None of the methods used to compensate for this are both real-time efficient and arbitrarily accurate, they are visually good for games or offline render farms, but not real-time engineering quality simulation.

### 3.2 Adaptive Synchronous Time Step

A variation on fixed time step methods is adaptive synchronous time step. In this method, standard adaptive numeric methods determine the longest computationally stable time step for each entity motion being computed [10].

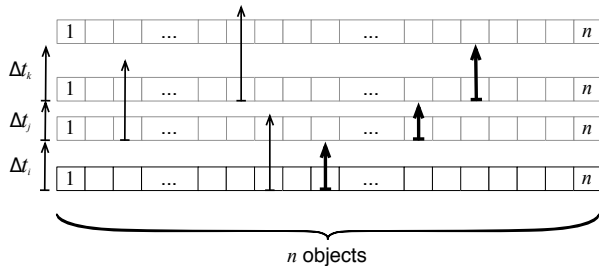


Figure 4. Adaptive synchronous time step.

The narrow arrows in Figure 4 depict the longest potential time step and bold depicting the shortest. The shortest is selected for the time advance and all entities are re-evaluated synchronously at that selected interval. The time steps in Figure 4, illustrate the unequal time advances matching those shortest interval choices. The work expended computing the potential time advances for all the other entities becomes expensive overhead. Adaptive time step attempts to trade larger advances than fixed time step against the need to do approximately twice the work per

time advance. This could pay off if every entity in the simulation can stably advance more than twice as far as the conservative fixed time step method, but in practice this is rarely occurs making this a less efficient method.

### 3.3 Adaptive Asynchronous Discrete Event

Adaptive asynchronous discrete event simulation is sometimes called continuous time simulation because the notion of a time step is eliminated altogether, entity state exists continuously with no holidays. The simulation state is no longer the set of entity locations at the end of a time step, it becomes the set of active events in a simulation.

Every entity event has a start time and an expiration time, for movement events we also include the linear approximation vector for the entity motion. The expiration time and linear velocity approximation are generated by the adaptive numeric method computations used in Section 3.2. This allows us to leverage the classic parametric equation of motion, Equation (1), relating original position ( $\mathbf{p}_0$ ), velocity ( $\mathbf{v}$ ) and time ( $t$ ) to be able compute the position ( $\mathbf{p}_t$ ) of an entity at any arbitrary time between our event start an expiration times.

$$\mathbf{p}_t = \mathbf{p}_0 + \mathbf{v}t \quad (1)$$

Discrete event implementations using higher order numeric methods provide equal or better error tolerances than low order synchronous time step implementations. We can see this illustrated graphically in Figure 5 where the depicted maximum stable advances combine to move the simulation state farther into the future than the fixed time step ( $3 \times \Delta t$  baseline) or adaptive synchronous time step ( $\Delta t_i + \Delta t_j + \Delta t_k$  baseline).

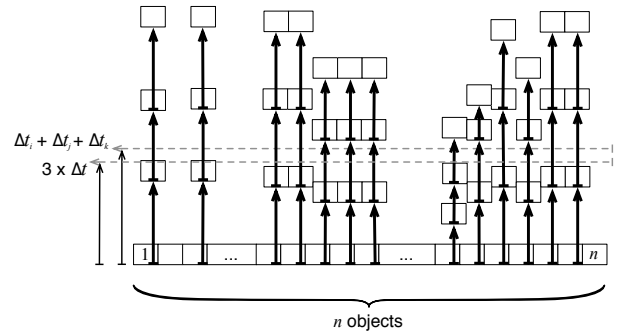


Figure 5. Adaptive asynchronous discrete events.

Adaptive asynchronous discrete event time advance also compares directly to the synchronous time advance method. The degenerate case of the adaptive asynchronous discrete event method is where all events are evaluated at the same time, with identical event durations. This is the same situation as seen in the synchronous time advance methods. The degenerate case also identifies the worst case performance situation for discrete event time advance because the additional work in choosing the time advance will make discrete event time advancement less efficient than fixed time step methods.

This simply means adaptive asynchronous discrete event simulation is not good for every type of simulation. It can be inefficient for evaluating fixed-grid, fixed time step, finite element problems or extreme contact density simulations such as particles in an avalanche. For simulations that mirror the real world as a command and control system sees it -- sparse and predictable at millisecond scales -- adaptive asynchronous discrete event simulation can be more efficient than time step methods.

## 4 Motion Evaluation

The quality of a system that evaluates entity motion can be evaluated on three characteristics: safety, correctness and progress [11]. Safety guarantees that simulated objects representing solids do not interpenetrate. Correctness provides objects adhering to the basic laws of physics such as causality and conservation (of mass, momentum, energy, *etc.*). Progress describes completing computations, not entering an infinite loop or artificially halting. We extend progress and correctness for real-time systems by requiring computations to be completed before the next scheduled update and within the desired tolerance.

### 4.1 Time Step-Based Collision Detection

Time stepped simulations generally adhere to the cycle depicted in Figure 6, which is updated from [10] to include handling events that arrive between time steps. The most computationally complex portion of this process is the collision detection subsystem, and it has theoretical worst case performance  $O(n^2)$ . In actual practice most collision detection algorithms have an effective performance of  $O(n)$  due to culling methods, optimizations and the extreme rarity of the worst case [12]. The culling method is the primary means to minimize how many entity pairs are subject to computationally expensive, detailed collision detection.

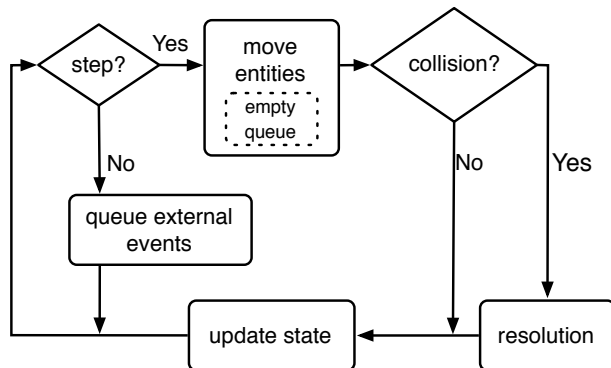


Figure 6. Time stepped physical simulation loop.

There is a triangular tradeoff space between collision detection computational cost, fidelity/accuracy, and time step length. This normally motivates entity motion computations to be done using lower fidelity first order differential equations, which reinforces selecting shorter time steps. This results in time stepped physics calculations

typically evaluated at 100Hz or higher, yielding maximum 10ms time steps.

Time step-based collision detection methods are highly optimized to leverage the temporal and spatial coherence of both simulated entities as well as code and data structures. *But they sacrifice safety for speed*, allowing some interpenetration as an optimization and often missing collisions between high velocity entities and entities that are “thin” compared to the closing velocities. This type of safety violation is illustrated in Figure 7. We return to the Red vs. Blue scenario, where Red actually misses Blue because his 254.508 m/s .45 cal projectile travels 2.545m per 10ms time step.

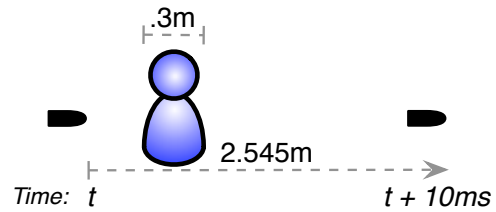


Figure 7. Ghostly time step passage.

At one time step the projectile is to the left of Blue and at the next it is at the right position where it effectively passed through Blue like a ghost, never registering a hit *because there is no state between the time steps*. The probability this happens is the ratio between the entity motion and entity depth, for Figure 7 approximately an 88% probability we will register a false miss.

Special case handling can be used to avoid this, but it is computationally expensive and requires even smaller time steps, a problem that only grows and further degrades performance as projectiles get faster (Red’s .45 cal projectile is slow for a military projectile). Entertainment and game physics middleware are unacceptable for engineering quality results, they avoid the above problem by selectively changing physics, artificially reducing projectile speeds, using invisible mega-size projectiles, or laser beam-like non-physical impact prediction.

### 4.2 Continuous State Collision Detection

Explicitly managing the state continuously between time advance computations avoids the safety problems noted in Section 4.1, without adding additional time steps. This method uses a sphere that fully envelopes the entity, and sweeps it along the linear motion approximation vector discussed in Section 3.3. The result is a 3-dimensional geometric capsule as seen in Figure 8, representing the entity motion through 4-dimensional space-time. The Figure 8 vector  $\mathbf{p}_0 \rightarrow \mathbf{p}_t$  is the Equation (1) representation of the linear motion approximation.

To date the challenge has been using capsules efficiently. The overhead of maintaining and manipulating the capsule in time stepped collision detection is very large compared to the benefits [12], but eliminating the

synchronously evaluated time step removes the duplicative overhead. Maintaining the capsules in an efficient dynamic data structure such as an R-Star tree [13] provides an efficient collision detection culling algorithm. The potentially colliding pairs are processed by existing pairwise time-of-impact detailed collision algorithms in much the same manner as they are currently used in time step-based collision detection.

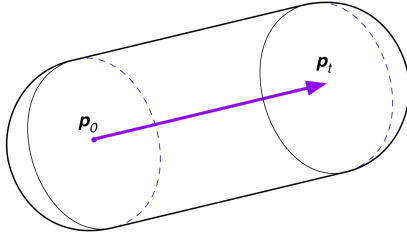


Figure 8. Capsule.

## 5 Hiding Network Latency

We can leverage the efficient continuous state representation to hide network communications latency. This is possible when all participating systems are using algorithms, models and constant data that verifiably produce results within a system-wide specified tolerance. We can then say they are sharing identical models, even if they are implemented in different products, by different vendors or even on different hardware architectures. For example, in an engineering quality simulation, distance results that fall within a 0.5 micron tolerance will be able to discriminate between paint-blemishing near misses and actual structural collisions. Consistent generation of this fidelity is well within the hardware abilities of existing general purpose CPUs under the IEEE 754 Standard for Floating-Point Arithmetic.

### 5.1 Informed Follow-on Calculations

With the above consistency in computation, this first technique contributing to latency hiding is available. We include in the network transmitted entity state update all necessary information for remote systems to make follow on calculations when the event expiration time arrives. Those calculations will result in tolerance-identical results throughout the distributed simulation SoS. Further, as long as no other entity controlled by externally generated behavior creates a potential collision, that follow-on calculation can be followed up again and again.

As an example, a single firing event for a long range projectile can provide the necessary information for all remote systems to do tolerance-identical computations over the entire time-of-flight. Even simulation randomizations during time-of-flight can be handled identically if the firing event update identifies the distribution and seed value. This example is impossible in a distributed system that cannot maintain a shared consistent state.

### 5.2 Pre-Computation

This is the second technique contributing to latency hiding. Entity spatial and temporal coherence, adaptive computations, asynchronous evaluation and the collision detection data structure of the R-Star tree make it possible to safely project the effects of follow-on events ahead of time. Spatial coherence and inertia mitigate perturbations on upcoming events due to external control inputs.

The next advance can be computed anytime after the current event is posted if we know the entity is not externally controlled. The result of the computation can be transmitted, taking expected transmission latency into account, to arrive before the current event expires. Even in the case of some unexpected input making the next computed event obsolete, the maintenance of the time sorted event queue prevents the obsolete event from occurring. This technique will not hide arbitrarily long latencies, but it will hide latencies which are less than the adaptively computed time advance for a specific entity motion. Nearly all events not subject to external control should be able to use latency hiding effectively.

The remainder of the events to be handled are those subject to external control. External control can be from inputs which capture human decision-making and reaction, or could be from algorithms which are not allowed to be simulated on the remote distributed SoS platforms. In the case of human inputs, they are very slow in relation to the sub-millisecond paced execution of a distributed SoS, so there is a telegraphing of incoming changes. The results of those inputs also take time before they yield state changes in the entity being controlled. The input may be fast, but physical reaction to the input is still slowed by factors such as system processing time, actuator response and inertia.

We can consider this time delay in system reaction to be analogous to lash in a gear train. This lash and the entity response timing delay associated with it allow us to generate an event which can be used to ensure the overall distributed system does not get more than an event or two ahead of itself. Technically, having to remove already computed events from a simulation is called roll-back and it can be expensive if numerous events are affected. But real-time systems through the properties of spatial and temporal coherence inherently provide reasonable limitations to leverage against blindly computing ahead, thus rollback costs can be tightly constrained.

### 5.3 Use of Remote Computation

Another benefit of latency hiding is that we can use some of that hidden latency to enable remote computations which are impossible to do with short fixed time steps. This may allow a simulation to request a high fidelity set of calculations from a remote supercomputer or cluster, have

those results returned to the requesting system and then integrated into the simulation.

For example in a notional CEC simulation tracking a Mach 3 contact, a high performance cluster could be used for electro-magnetic sensor propagation computations in tandem with high fidelity atmospheric data available from a dedicated weather facility. These computations may take 30ms each to deliver a set of propagation path contact intersection results back to the requesting system. In the 100Hz, 10ms per time step system, these high fidelity computations are too slow to be used real-time, so simulation designers resort to approximations or simple table lookups. The 30ms-to-delivery results are timely enough to yield useful real-time information to an adaptive asynchronous discrete event simulation as the contact only travels a known, pre-computed ~12m during those 30ms, a distance which is less than the absolute positional resolution of many EM search/track sensor systems. Even if the contact generates a mid-event control input, system lash will result in a delay before the contact begins to deviate from the existing projected motion. That delay allows the simulation to generate a new superseding event representing the upcoming control input before the entity motion changes can take effect, helping to eliminate stale and increasingly inaccurate states in the simulation.

## 6 Conclusions

We described a methodology where a distributed simulation SoS can transmit entity state updates that effectively hide network transmission latency, allowing the participating systems to maintain a consistent shared world-view. This methodology is designed for real-time systems with communication latency delays less than the time advances determined by common adaptive numerical methods. The consistent shared world-view enabled by this methodology can increase trust in the SoS simulation as a decision-support tool. The similarities between the operating constraints of real-time distributed simulation SoS domain and modern C4I SoS domain are indicative of the applicability of these simulation latency hiding techniques back into the larger SoS domain where time critical communications are necessary for decision-making and/or decision-support tools.

### 6.1 Future Work

A thorough experimental implementation is necessary to allow testing and validation of this non-traditional architectural combination. Follow on work should also quantify the latency hiding capability based on expected entity motion models, to provide a predictive planning tool. Additionally, distributed SoS with a common shared world-view enable a whole new set of C4I system possibilities, including work on integrating physics-based simulation into C4I systems to provide robust sanity checking, or generating what-if scenario queries in real-time. These C4I-

Simulation integration lines also dovetail into the push for larger, more complex and integrated LVC training exercises. The common shared world-view and an eventual open standard supporting it can provide significant business, training and operational benefits to extend C4I integrated simulation far beyond where it is today.

## References

- [1] W. Sudnikovich, J. Pullen, M. Kleiner, and S. Carey. *Extensible Battle Management Language as a Transformation Enabler*. SIMULATION, 2004.
- [2] K. Poch and J. Secondine. *TENA and JMETC, Enabling Technology in Distributed LVC Environments*. AIAA, U.S. Air Force T&E Days, Feb 2010.
- [3] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. ACM Press/Addison-Wesley Publishing Co., New York, 1999.
- [4] A. Steed and M. Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Morgan Kaufmann, Burlington, MA, 2009.
- [5] A. Gupta, J. Hennessy, K. Gharachorloo, T. Mowry, and W. Weber. *Comparative Evaluation Of Latency Reducing And Tolerating Techniques*. In Proc. ISCA '91, ACM, 1991.
- [6] V. Strumpen and T. Casavant. *Exploiting Communication Latency Hiding for Parallel Network Computing: Model and analysis*. In Proc. PDS'94, IEEE, 1994.
- [7] IEEE Standard 1278.1-1995 *Distributed Interactive Simulation (DIS) (and revisions)*, 1995.
- [8] IEEE Standard 1516-2010 - *Standard for Modeling and Simulation High Level Architecture (HLA) - Framework and Rules*, 2000
- [9] M. Baker. *Time Management in Distributed Simulation Models*. In Proc. SimTecT'99, Melbourne, Australia, 1999
- [10] M. Coutinho. *Dynamic Simulations of Multibody Systems*. Springer-Verlag, London, UK, 2001.
- [11] D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun. *Asynchronous Contact Mechanics*. SIGGRAPH '09: SIGGRAPH 2009 papers, ACM, 2009.
- [12] K. Erleben, J. Sporrying, K. Henriksen, and H. Dohlmann. *Physics-based Animation (Graphics Series)*. Charles River Media, Boston, MA, 2005.
- [13] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.