Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation

THOMAS Y. YEH
IEnteractive Research and Technology
GLENN REINMAN
University of California, Los Angeles
SANJAY J. PATEL
University of Illinois at Urbana-Champaign
and
PETROS FALOUTSOS
University of California, Los Angeles

The error tolerance of human perception offers a range of opportunities to trade numerical accuracy for performance in physics-based simulation. However, most prior work on perceptual error tolerance either focus exclusively on understanding the tolerance of the human visual system or burden the application developer with case-specific implementations such as Level-of-Detail (LOD) techniques. In this article, based on a detailed set of perceptual metrics, we propose a methodology to identify the maximum error tolerance of physics simulation. Then, we apply this methodology in the evaluation of four case studies. First, we utilize the methodology in the tuning of the simulation timestep. The second study deals with tuning the iteration count for the LCP solver. Then, we evaluate the perceptual quality of Fast Estimation with Error Control (FEEC) [Yeh et al. 2006]. Finally, we explore the hardware optimization technique of precision reduction.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*

General Terms: Algorithms, Design, Human Factors, Performance

Additional Key Words and Phrases: xxxx

ACM Reference Format:

Yeh, T. Y., Reinman, G., Patel, S. J., and Faloutsos, P. 2009. Fool me twice: Exploring and exploiting error tolerance in physics-based animation. ACM Trans. Graph. 29, 1, Article 5 (December 2009), 11 pages. DOI = 10.1145/1640443.1640448 http://doi.acm.org/10.1145/1640443.1640448

1. INTRODUCTION

Physics-Based Animation (PBA) is becoming one of the most important elements of interactive entertainment applications, such as computer games, largely because of the automation and realism that it offers. However, the benefits of PBA come at a considerable computational cost. Furthermore, this cost grows prohibitively with the number and complexity of objects and interactions in the virtual world. It becomes extremely challenging to satisfy such complex worlds in real time.

Fortunately, there is a tremendous amount of parallelism in the physical simulation of complex scenes. Exploiting this parallelism for performance is an active area of research both in terms of software techniques and hardware accelerators. Prior work such as PhysX [AGEIA], GPUs [Havok], the Cell [Hofstee 2005], and ParallAX [Yeh et al. 2007] have just started to address this problem.

This work was partially supported by the NSF (grant CCF-0429983) and the SRC. Any opinions, findings and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the NSF or SRC.
The work was also supported by Intel Corp. and Microsoft Corp. through equipment and software grants.
Authors' addresses: T. Y. Yeh, IEnteractive Research and Technology; email: tomyeh@ienteractive.com; G. Reinman, Department of Computer Science, University of California, Los Angeles CA 90095; email: reinman@cs.ucla.edu; S. J. Patel, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL 61820; email: sjp@illinois.edu; P. Faloutsos, Department of Computer Science, University of California, Los Angeles, CA 90095; email: pfal@cs.ucla.edu.

DOI 10.1145/1640443.1640448 http://doi.acm.org/10.1145/1640443.1640448

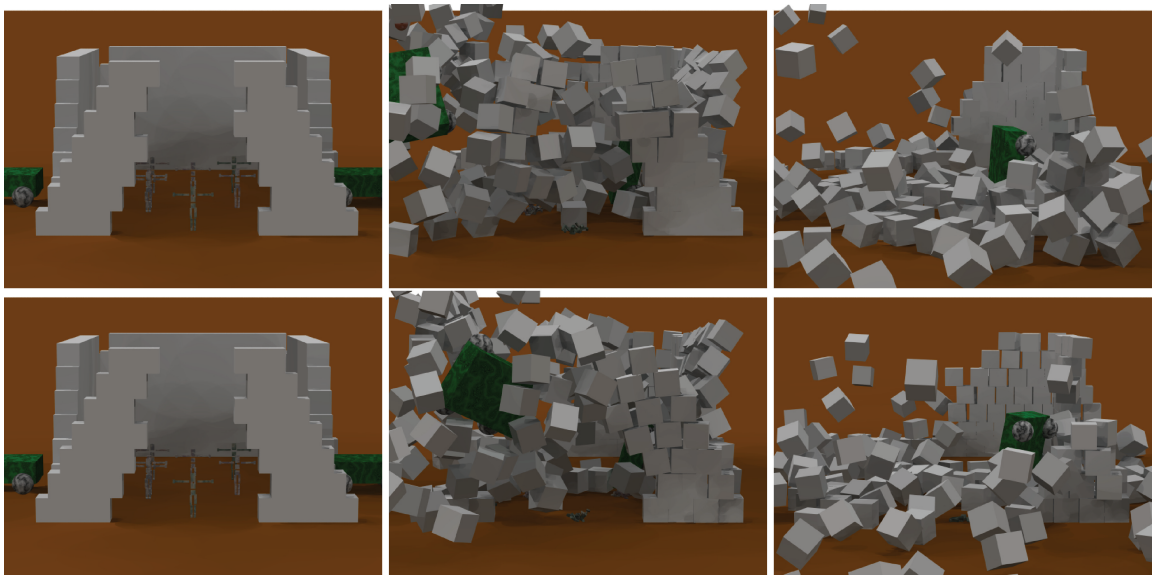

Fig. 1. Snapshots of two simulation runs with the same initial conditions. The top row is the baseline, and the bottom row is the simulation with 7-bit mantissa floating-point computation in *Narrowphase* and *LCP*. The results are different but both are visually correct.

While parallelism can help PBA achieve a real-time frame rate, perceptual error tolerance is another avenue to help improve PBA performance. There is a fundamental difference between *accuracy* and *believability* in interactive entertainment; the results of PBA do not need to be absolutely accurate, but do need to appear correct (i.e., believable) to human users. The perceptual acuity of human viewers has been studied extensively in graphics and psychology [O'Sullivan et al. 2004; O'Sullivan and Dingliana 2001]. It has been demonstrated that there is a surprisingly large degree of error tolerance in our perceptual ability.[1]

This perceptual error tolerance can be exploited by a wide spectrum of techniques ranging from high-level software techniques down to low-level hardware optimizations. At the application level, Level of Detail (LOD) simulation [Carlson and Hodgins 1997; McDonnell et al. 2006] can be used to handle distant objects with simpler models. At the physics engine library level, one option is to use approximate algorithms optimized for speed rather than accuracy [Seugling and Rolin 2006]. At the compiler level, dependencies among parallel tasks could be broken to reduce synchronization overhead. At the hardware design level, floating-point precision reduction can be leveraged to reduce area, reduce energy, or improve performance for physics accelerators.

In this article, we address the challenging problem of leveraging perceptual error tolerances to improve the performance of real-time PBA in interactive entertainment. The main challenge is to establish a methodology which uses a set of error metrics to measure the visual performance of a complex simulation. Prior perceptual thresholds do not scale to complex scenes. In our work we address this challenge and investigate both hardware and software applications.

Our contributions are threefold.

—We propose a methodology to evaluate physical simulation errors in complex dynamic scenes.

—We identify the maximum error that can be injected into each phase of the low-level numerical PBA computation.

—We explore software timestep tuning, iteration-count tuning, fast estimation with error control, and hardware precision reduction to exploit error tolerance for performance.

[1]This is independent of a viewer's understanding of physics [Proffit].

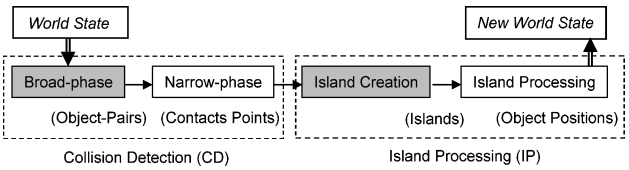

Fig. 2. Physics engine flow. All phases are serialized with respect to each other, but unshaded stages can exploit parallelism within the stage.

2. BACKGROUND

In this section, we present a brief overview of Physics-Based animation (PBA), identify its main computational phases, and categorize possible errors. We then review the latest results in perceptual error metrics and their relation to PBA.

2.1 Computational Phases of a Typical Physics Engine

PBA requires the numerical solution of the differential equations of motion of all objects in a scene. Articulations between objects and contact configurations are most often solved with constraint-based approaches such as Baraff [1997], Muller et al. [2006], and Havok ‖. Time is simulated in intervals of fixed or adaptive timestep. The timestep is one of the most important simulation parameters, and it largely defines the accuracy of the simulation. For interactive applications the timestep needs to be in the range of 0.01 to 0.03 simulated seconds or smaller, so that the simulation can keep up with display rates.

PBA can be described by the data-flow of computational phases shown in Figure 2. Cloth and fluid simulation are special cases of

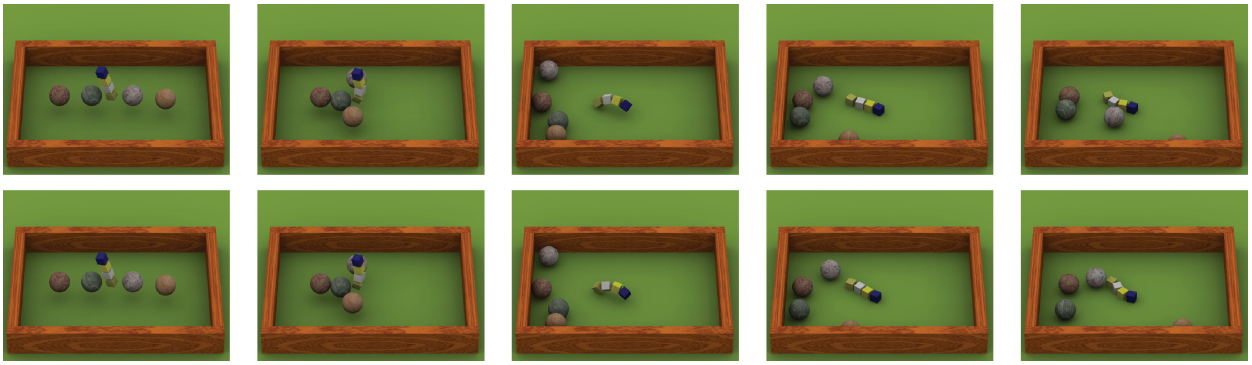

Fig. 3. Snapshots of two simulation runs with the same initial conditions but different constraint ordering. The results are different but both are visually correct.

this pipeline. Next we describe the four computational phases in more detail.

Broad-phase. This is the first step of *Collision Detection* (CD). Using approximate bounding volumes, it efficiently culls away pairs of objects that cannot possibly collide. While *Broad-phase* does not have to be serialized, the most useful algorithms are those that update a spatial representation of the dynamic objects in a scene. And updating these spatial structures (hash tables, kd-trees, sweep-and-prune axes) is not easily mapped to parallel architectures.

Narrow-phase. This is the second step of CD that determines the contact points between each pair of colliding objects. Each pair's computational load depends on the geometric properties of the objects involved. The overall performance is affected by broad-phase's ability to minimize the number of pairs considered in this phase. This phase exhibits massive Fine-Grain (FG) parallelism since object-pairs are independent of each other.

Island creation. After generating the contact joints linking interacting objects together, the engine serially steps through the list of all objects to create islands (connected components) of interacting objects. This phase is serializing in the sense that it must be completed before the next phase can begin. The full topology of the contacts isn't known until the last pair is examined by the algorithm, and only then can the constraint solvers begin.

Simulation step. For each island, given the applied forces and torques, the engine computes the resulting accelerations and integrates them to compute each object's new position and velocity. This phase exhibits both Coarse-Grain (CG) and Fine-Grain (FG) parallelism. Each island is independent, and the constraint solver for each island contains independent iterations of work. We further split this component into two phases.

—*Island processing*. which includes constraint setup and integration (CG).

—*LCP* which includes the solving of constraint equations (FG).

2.2 Simulation Accuracy and Stability

The discrete approximation of the equations of motion introduce errors in the results of any nontrivial physics-based simulation. For the purposes of entertainment applications, we can distinguish between three kinds of errors in order of increasing importance.

—*Imperceptible*. These are errors that cannot be perceived by an average human observer.

—*Visible but bounded*. There are errors that are visible but remain bounded.

—*Catastrophic*. These errors make the simulation unstable which results in numerical explosion. In this case, the simulation often reaches a state from which it cannot recover gracefully.

To differentiate between categories of errors, we employ the conservative thresholds presented in prior work [O'Sullivan et al. 2003; Harrison et al. 2004] for simple scenarios. All errors with magnitude smaller than these thresholds are considered *imperceptible*. All errors exceeding these thresholds without simulation blow-up are considered *visible but bounded*. All errors that lead to simulation blow-up are considered *catastrophic*.

An interesting example that demonstrates *visible but bounded* simulation errors is shown in Figure 3. In this example, four spheres and a chain of square objects are initially suspended in the air. The two spheres at the sides have horizontal velocities towards the object next to them. With the same initial conditions, two different simulation runs shown in the figure result in visibly different final configurations. However, both runs appear physically correct.This behavior is due to the *constraint reordering* that the iterative constraint solver employs to reduce bias [Smith]. Constraint reordering is a well-studied technique that improves the stability of the numerical constraint solver, and it is employed by commercial products such as AGEIA ‖. For our purposes, it provides an objective way to establish which physical errors are acceptable when we develop the error metrics in Section 4.

The first two categories are the basis for perceptually-based approaches such as ours. Specifically, we investigate how we can leverage the perceptual error tolerance of human observers without introducing catastrophic errors in the simulation.

The next section reviews the relevant literature in the area of perceptual error tolerance.

2.3 Perceptual Believability

O'Sullivan et al. [2004] is a state-of-the-art survey report on the field of perceptual adaptive techniques proposed in the graphics community. There are six main categories of such techniques: interactive graphics, image fidelity, animation, virtual environments, and visualization and nonphotorealistic rendering. Our article focuses on the animation category. Given the comprehensive coverage of this prior survey paper, we only present prior work most

related to our article and point the reader to O'Sullivan et al. [2004] for additional information.

Barzel et al. [1996] is credited with the introduction of the plausible simulation concept, and Chenney and Forsyth [2000] built upon this idea to develop a scheme for sampling plausible solutions. O'Sullivan et al. [2003] is a recent paper upon which we base most of our perceptual *metrics*, but the *thresholds* presented for these metrics are not useful for evaluating complex situations as shown later. For the metrics examined in this article, the authors experimentally arrive at thresholds for high probability of user believability. Then, a probability function is developed to capture the effects of different metrics. The study in this prior work uses only simple scenarios with 2 colliding objects.

Harrison et al. [2004] is a study on the visual tolerance of lengthening or shortening of human limbs due to constraint errors produced by PBA. We derive the threshold for constraint error in Table II from this paper.

Reitsma and Pollard [2003] is a study on the visual tolerance of ballistic motion for character animation. Errors in horizontal velocity were found to be more detectable than vertical velocity. Also, added accelerations were easier to detect than added deceleration.

We examined the techniques of fuzzy value prediction and fast estimation with error control to leverage error tolerance in Yeh et al. [2006]. As an initial attempt at quantifying error, we showed the absolute difference in object position, object orientation, and constraint error against the baseline simulation. Our current study provides the error measuring methodology and maximum tolerable error lacking in our previous study.

In general, prior work has focused on simple scenarios in isolation (involving 2 colliding objects, a human jumping, human arm/foot movement, etc.). Isolated special cases allow us to see the effect of instantaneous phenomena, such as collisions, over time. In addition, they allow a priori knowledge of the correct motion which serves as the baseline for exact error comparisons.

Complex cases do not offer that luxury. In complex cases such as multiple simultaneous collisions, errors become difficult to detect and may cancel out. We are the first to address this challenging problem and provide a methodology to estimate the perceptual error tolerance of physical simulation corresponding to a complex, game-like scenario.

2.4 Simulation Believability

Seugling and Rolin [2006, Chapter 4] compares three physics engines (ODE [Smith], Newton [], and Novodex [AGEIA]) by conducting performance tests. These tests involved friction, gyroscopic forces, bounce, constraints, accuracy, scalability, stability, and energy conservation. All tests show significant differences between the three engines, and the engine choice produces different simulation results with the same initial conditions. Even without any error-injection, there is no single *correct* simulation for real-time PBA in games as the algorithms are optimized for speed rather than accuracy.

3. METHODOLOGY

One major challenge in exploring the trade-off between accuracy and performance in PBA is coming up with the metrics and the methodology to evaluate believability. Since some of these metrics are relative (i.e., the resultant velocity of an object involved in a particular collision), there must be a reasonable standard for comparing these metrics. In this section, we detail the set of numerical metrics we have assembled to gauge believability, along with a technique to fairly compare worlds which may have substantially diverged.

3.1 Experimental Setup

To represent in-game scenarios, we construct a complex test case which includes stacked, articulated, and fast objects shown in Figure 1. The scene is composed of a building enclosed on all four sides by brick walls with one opening. The wall sections framing the opening are unstable. Ten humans with anthropomorphic dimension, mass, and joints are stationed within the enclosed area. A cannon shoots fast (88 m/s) cannonballs at the building, and two cars collide into opposing walls. Assuming time starts at 0 sec, one cannonball is shot every 0.04 sec. until 0.4 sec. The cars are accelerated to roughly 100 miles/hr (44 m/sec) at time 0.12 to crash into the walls. No forces are injected after 0.4 sec. Because we want to measure the maximum and average errors, we target the time period with the most interaction (the first 55 frames).

The described methodology has also been applied to three other scenarios with varying complexity.

(1) 2 spheres colliding ([O'Sullivan et al. 2003], video);
(2) 4 spheres and a chain of square objects (Figure 3, video);
(3) Complex scenario without humans (not shown).

Our physics engine is a modified implementation of the publicly available Open Dynamics Engine (ODE) version 0.7 [Smith]. ODE follows a constraint-based approach for modeling articulated figures, similar to Baraff [1997] and AGEIA ‖, and it is designed for efficiency rather than accuracy. Like most commercial solutions it uses an iterative constraint solver. Our experiments use a conservative timestep of 0.01 sec. and 20 solver iterations as recommended by the ODE user-base. This is a conservative estimate of the required timestep, and in Section 5.1 we show that this can be reduced to 60 FPS.

3.2 Error Sampling Methodology

To evaluate the numerical error tolerance of PBA, we inject errors at a per-instruction granularity. We only inject errors into Floating-Point (FP) add, subtract, and multiply instructions, as these make up the majority of FP operations for this workload.

Our error injection technique is fairly general, and should be representative of a range of possible scenarios where error could occur. At a high level, we change the output of FP computations by some varying amount. This could reflect changes from an imprecise ALU, an algorithm that cuts corners, or a poorly synchronized set of ODE threads. The goal is to show how believable the simulation is for a particular magnitude of allowed error.

To achieve this generality, we randomly determine the amount of error injected at each instruction, but vary the absolute magnitude of allowed error for different runs. This allowed error bound is expressed as a maximum percentage change from the correct value, in either the positive or negative direction. For example, an error bound of 1% would mean that the correct value of an FP computation could change by any amount in the range from -1% to 1%.

A random percentage, less than the preselected max and min, is applied to the result to compute the absolute injected error. By using random error injection, we avoid biasing of injected errors. For each configuration, we average the results from 100 different simulations (each with a different random seed) to ensure that our results converge. We have verified that 100 simulations are enough to converge by comparing results with only 50 simulations; these

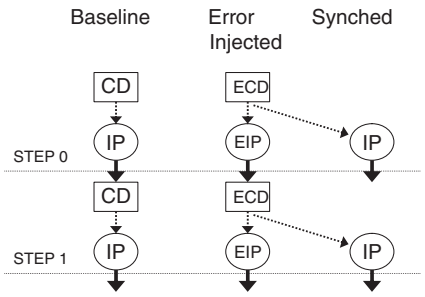

Fig. 4. Simulation worlds. *CD* = Collision Detection. IP = Island Processing. E = Error-injected.

results are identical. We evaluate the error tolerance of the entire application and each phase individually.

Both error injection and precision reduction are done by replacing every floating-point add, subtract, and multiply operation by a function call which simulates either random error injection or precision reduction. This is done by modifying the source code of ODE. ODE uses its own implementation of the solvers, and the errors are introduced in every single floating-point add, subtract, and multiply.

3.3 Error Metrics

Now that we have a way of injecting error, we want to determine when the behavior of a simulation with error is still believable through numerical analysis. Many of the metrics we propose are relative values, and therefore we need to have reasonable comparison points for these metrics. However, it is not sufficient to simply compare a simulation that has errors injected with a simulation without any error. Small, but perceptually tolerable differences can result in large behavioral differences, as shown in Figure 3.

To address this, we make use of three simulation worlds as shown in Figure 4: *Baseline*, *Error-injected*, and *Synched*. All worlds are created with the same initial state, and the same set of injected forces (cannonball shooting or cars speeding up) are applied to all worlds. *Error-injected* refers to the error-injected world, where random errors within the preselected range are injected for every FP $+/-/*$ instruction. *Baseline* refers to the deterministic simulation with no error injection.

Finally, we have the *Synched* world, a world where the state of every object and contact is copied from the error-injected world after each simulation step's collision detection. The island processing computation of *Synched* contains no error injection, so it is using the collisions detected by *Error-injected* but is performing correct island processing. The reason for synching after, instead of before, collision detection is that both gap and penetration already provide information specific to the effects of errors in collision detection. The Synched world is created to isolate the effects of errors in island Processing.

We use the following seven numerical metrics:

—*Energy Difference*. Difference in total energy between *baseline* and *error-injected* worlds: due to energy conservation, the total energy in these two worlds should match.

—*Penetration Depth*. Distance from the object's surface to the contact point created by collision detection. This is measured within the simulation world.

—*Constraint Violation*. Distance between object position and where object is supposed to be based on statically defined joints (car's suspension or human limbs).

—*Linear Velocity Magnitude*. Difference in linear velocity magnitude for the same object between *Error-Injected* and *Synched* worlds.

—*Angular Velocity Magnitude*. Difference in angular velocity magnitude for the same object between *Error-Injected* and *Synched* worlds.

—*Linear Velocity Angle*. Angle between linear velocity vectors of the same object inside *Error-Injected* and *Synched* worlds.

—*Gap Distance*. Distance between two objects that are found to be colliding, but are not actually touching.

We can measure gap, penetration, and constraint errors directly in the *Error-injected* world, but we still use *Baseline* here to normalize these metrics. If penetration is equally large in the *Baseline* world and *Error-injected* world, then our injected error has not made things worse.

The aforesaid error metrics capture both globally conserved quantities, such as total energy, and instantaneous per-object quantities such as positions and velocities. The metrics do not include momentum because most simulators for computer games trade off momentum conservation for stability [Seugling and Rolin 2006].

4. NUMERICAL ERROR TOLERANCE

In this section, we explore the use of our error metrics in a complex game scene with a large number of objects. We inject error into this scene for different ODE phases. The response from these metrics determines how much accuracy can be traded for performance.

Before delving into the details, we briefly articulate the potential outcome of error injection in different ODE phases. Because Broad-phase is a first-pass filter on potentially colliding object-pairs, it can create functional errors by omitting actually colliding pairs. Since Narrow-phase does not see the omitted pairs, the omissions can lead to missed collisions, increased penetration (if collision is detected later), and, in the worst case, tunneling (collision never detected). On the other hand, poor Broad-phase filtering can degrade performance by sending Narrow-phase more object-pairs to process.

Errors in Narrow-phase may lead to missed collisions or different contact points. The results are similar to omission by Broad-phase, but can also include errors in the angular component of linear velocity due to errors in contact points. Also, additional object-pairs from poor Broad-phase filtering could be mistakenly identified by Narrow-phase as colliding if errors are injected in Narrow-phase.

Because Island Processing sets up the constraint equations to be solved, errors here can drastically alter the motion of objects, causing movement without applied force. Errors inside the LCP solver alter the applied force due to collisions. Therefore, the resulting momentum of two colliding objects may be severely increased or dampened. Since the LCP algorithm is designed to self-correct algorithmic errors by iterating multiple times, LCP should be more error tolerant than Island Processing.

While our study focuses on rigid body simulation, we qualitatively argue that perceptual error tolerance can be exploited similarly in particle, fluid, and cloth simulation. We leave the empirical evaluation for future work.

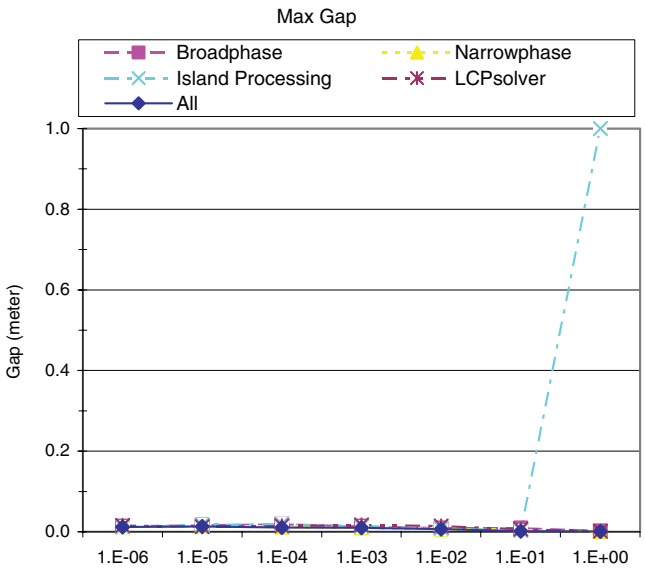

Fig. 5. Gap for Error-injection. X-axis shows the maximum error to value ratio for injected errors. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.

4.1 Numerical Error Analysis

For this initial error analysis, we performed a series of experiments where an increasing degree of error is injected into different phases of ODE. Figures 5, 6, 7, and 8 demonstrate each metric's maximal error that results from error injection. Following the error injection methodology of Section 3, we progressively increased the maximum possible error in order-of-magnitude steps from 0.0001% to 100% of the correct value, labeling this range 1.E-6 to 1.E+00 along the x-axis. We show results for injecting error into each of the four ODE phases alone, and then an *All* result when injecting error into all phases of ODE.

The serial phases, Broad-phase and Island Processing, exhibit the the highest and lowest per-phase error tolerance, respectively. Only Broad-phase does not result in simulation blow-up as increasingly large errors are injected. Island Processing is the most sensitive individual phase to error. The highly parallel phases of Narrow-phase and LCP show similar average sensitivity to error. We make use of the per-phase requirements when trading accuracy for performance.

As shown by Figures 5, 6, 7, and 8, most of the metrics are strongly correlated. Most metrics show a distinct flat portion and a knee where the difference starts to increase rapidly. As these errors pile up, the energy in the system grows with higher velocities, deeper penetrations, and higher constraint violations. The exceptions are gap distance and linear velocity angle. Gap distance remains consistently small. One reason for this is the filtering that is done in collision detection. For a gap error to occur, Broad-Phase would need to allow two objects which are not actually colliding to get to Narrow-Phase, and Narrow-Phase would need to mistakenly find that these objects touch one another. Gap errors are more relevant to manually constructed scenarios that have been used in prior work. A penetration error is much easier to generate; only one of the two phases needs to incorrectly detect that two objects are not colliding.

The angle of linear velocity does not correlate with other metrics either; in fact, the measured error actually decreases with more error. The problem with this metric is that colliding objects with even very small velocities can have drastically different angles of deflection depending on the error injected in any one of the four phases. From the determination of contact points to the eventual solution of resultant velocity, errors in angle of deflection can truly propagate. However, we observe that these maximal errors in the angle of linear velocity are extremely rare and mainly occur in the bricks composing our wall that are seeing extremely small amounts of jitter. This error typically lasts only a single frame and is not readily visible.

While these maximal error values are interesting, the average error in our numerical metrics and the standard deviation of errors are both extremely small (data not shown). This shows that most objects in the simulation are behaving similarly to the baseline. Only the percentage change in magnitude of linear velocity has a significant average error. This is because magnitude change on extremely small velocities can result in significant percentage change.

4.2 Acceptable Error Threshold

Now that we have examined the response of physics engines to injected error using our error metrics, the question still remains as to how much error we can tolerate and keep the simulation believable. Consider Figures 5, 6, 7, and 8 where we show the maximum value of each metric for a given level of error. Instead of using fixed thresholds to evaluate tolerable error, we argue for finding the knee in the curve where simulation differences start to diverge towards instability. The average error is extremely low. The majority of errors result in imperceptible differences in behavior. Of the remaining errors, many are simply transient errors lasting for a frame or two. We are most concerned with the visible, persistent outliers that can eventually result in catastrophic errors. For many of the maximal error curves, there is a clear point where the slope of the curve drastically increases; these are points of catastrophic errors that are not tolerable by the system, as confirmed by a visual inspection of the results.

Table I summarizes the maximum % error tolerated by each computation phase (using 100 samples), based on finding the earliest knee where the simulation blows-up over all error metric curves. It is interesting that All is more error tolerant than only injecting errors in Island Processing. The reason for this behavior is that injecting errors into more operations with All is similar to taking more samples of a random variable. More samples lead to a converging mean which in our case is zero.

To further support the choices made in Table I, we consider four approaches: (1) confirming our thresholds visually, (2) comparing our errors to a very simple scenario with clear error thresholds [O'Sullivan et al. 2003], (3) comparing the magnitude of our observed error to constraint reordering, and (4) examining the effect of the timestep on this error.

4.2.1 *Threshold Evaluation* 1. First we visually investigate the differences in our thresholds. The initial pass of our visual inspection involved watching each error-injected scene in real time to see how believable the behavior looked, including the presence of jitter, unrealistic deflections, etc. This highly subjective test confirmed our thresholds, and only experiments with error rates above our thresholds had clear visual errors, such as bricks moving on their own, human figures flying apart, and other chaotic developments.

4.2.2 *Threshold Evaluation* 2. Second, we constructed a similar simulation as the experiment used in O'Sullivan et al. [2003] (but with ODE) to generate the thresholds for perceptual metrics. This scenario places two spheres on a 2D plane: one is stationary

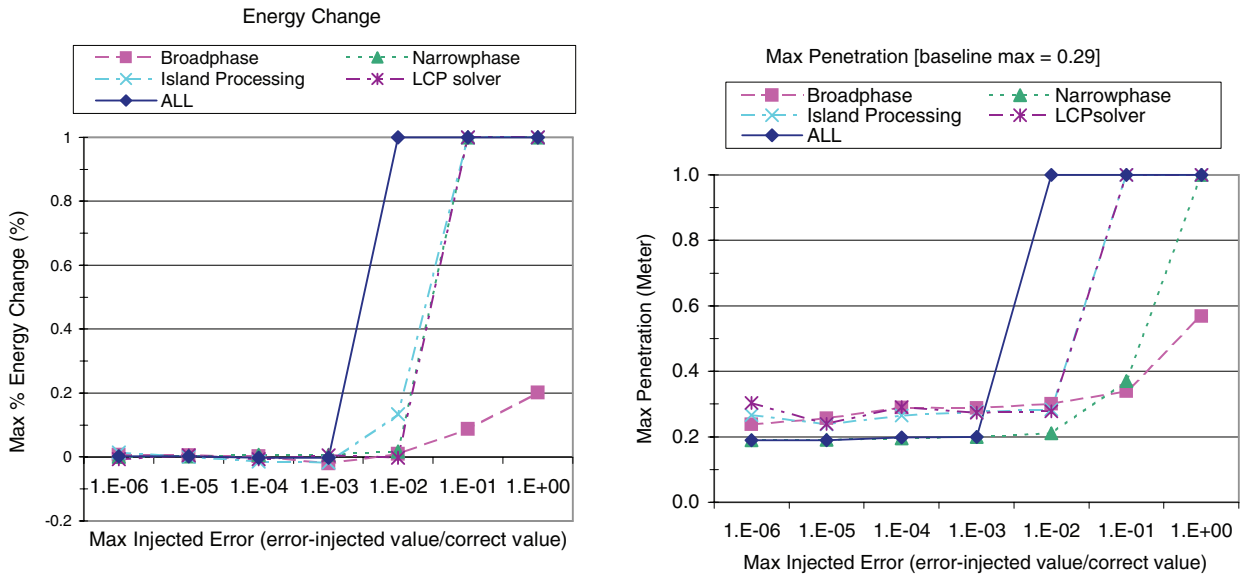

Fig. 6. Energy and penetration data for Error-injection. X-axis shows the maximum possible injected error. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.

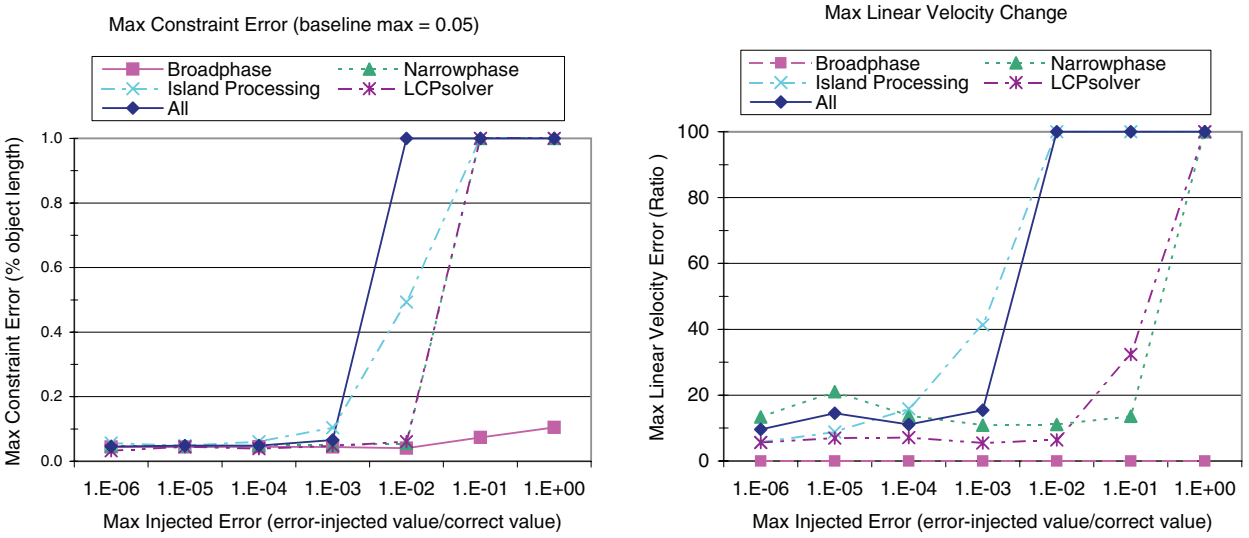

Fig. 7. Constraint violation and linear velocity data for Error-injection. X-axis shows the maximum possible injected error. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.

and the other has an initial velocity that results in a collision with the stationary sphere. No gravity is used, and the spheres are placed two meters above the ground plane.

We injected errors into this simple scenario by using the error bounds from Table I. The error bounds that we experimentally found in the previous section show no perceptible error, as according to thresholds from O'Sullivan et al. [2003], for this simple example. The first column of Table II shows the perceptual metric values for 0.1% error injection in all phases, and the rightmost column shows thresholds from prior work [O'Sullivan et al. 2003]. For the additional metrics we introduce, we mark them as Not Available (*NA*) for the simple threshold column.

Perceptible errors can be detected as the errors are increased by an order of magnitude. The thresholds are conservative enough to flag catastrophic errors such as tunneling, large penetration, and movement without collision.

4.2.3 *Threshold Evaluation* 3. However, when applying the same method to a complicated game-like scenario, the thresholds from prior work become far too conservative. Even the authors of O'Sullivan et al. [2003] point out that thresholds for simple scenarios may not generalize to more complex animations. In a chaotic environment with many collisions, it has been shown that human

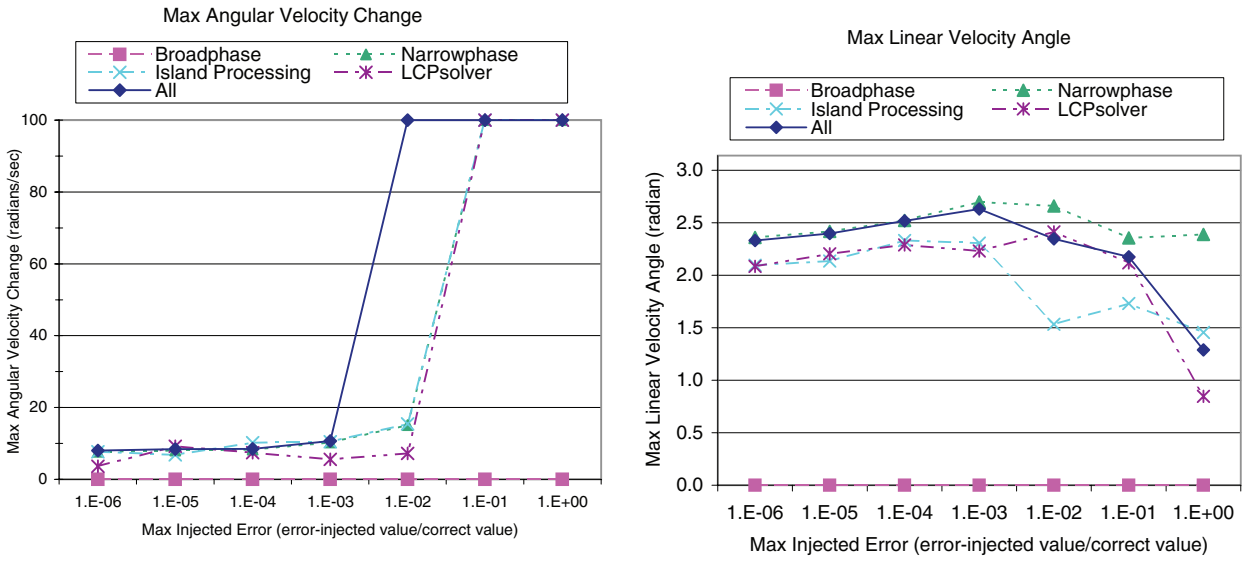

Fig. 8. Angular velocity and linear velocity angle data for Error-injection. X-axis shows the maximum possible injected error. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.

Table I. Max Error Tolerated for Each Computation Phase

Error Tolerance	Broadphase	Narrowphase	Island Processing	LCP	All
(100 Samples)	[1%]	[1%]	[0.01%]	[1%]	[0.1%]

Table II. Perceptual Metric Data for Simple Scenario

Perceptual Metrics	Simple Scenario Error Injection All Phases 0.1%	Simple Scenario Threshold
Energy (% change)	1.5	NA
Penetration (meters)	0.17	NA
Constraint Error (ratio)	0.00	[0.03,0.2]
Linear Vel (ratio)	0.13	[−0.4,+0.5]
Angular Vel (radians/sec)	0.00	[−20,+60]
Linear Vel Angle (radians)	0.005	[−0.87,+1.05]
Gap (meters)	0.000	0.001

perceptual acuity becomes even worse, particularly in areas of the simulation that are not the current focal point.

When we enable reordering of constraints in ODE, most of our perceptual metrics exceed the thresholds from O'Sullivan et al. [2003], compared to a run without reordering. There is no particular ordering which generates an absolutely correct simulation when using an iterative solver such as the one used in ODE. Changes in the order in which constraints are solved can result in simulation differences. The ordering present in the deterministic, baseline simulation is arbitrarily determined by the order in which objects were created during initialization. The same exact set of initial conditions with a different initialization order results in constraint reordering relative to the original baseline.

To understand this inherent variance in the ODE engine, we have again colored objects with errors and analyzed each frame of our simulation. The variance seen when enabling/disabling reordering results in errors that are either imperceptible or transient. Based on this analysis, we use the magnitude of difference generated by reordering as a comparison point for the errors we have experimentally found tolerable in Table I. Our goal is to leverage a similar level of variance as what is observed for random reordering when measuring the impact of error in PBA.

Table III. Perceptual Metric Data for Complex Scenario

Perceptual Metrics	Error Injection All Phases 0.1%	Random Reordering	Baseline
Energy (% change)	−0.23%	−1%	NA
Penetration (meters)	0.20	0.25	0.29
Constraint Error (ratio)	0.07	0.05	0.05
Linear Vel (ratio)	15.4	5.27	NA
Angular Vel (radians/sec)	10.7	4.48	NA
Linear Vel Angle (radians)	2.63	1.72	NA
Gap (meters)	0.01	0.01	0.01

Table III compares the maximum errors in our perceptual metrics for error injection or reordering as compared to the baseline simulation of a complex scenario. We injected errors into this complex scenario by using the error bounds from Table I.

For baseline simulation, only absolute metrics (i.e., those that require no comparison point like gap and penetration) are shown, and relative metrics that would ordinarily be compared against the baseline itself (i.e., linear velocity) are marked *NA*.

The first thing to notice from these results is that the magnitude of maximal error for a complex scene can be much more than the simple scenario data shown in Table II. The second thing to notice is that despite some large variances in velocity and angle of deflection, energy is still relatively unaffected. This indicates that these errors are not catastrophic, and do not cause the simulation to blow-up. It is also interesting to notice the magnitude of penetration errors. Penetration can be controlled by using a smaller timestep, but by observing the amount of penetration from a baseline at a given timestep, we can ensure that it does not get any worse from introducing errors.

The magnitude of the errors from reordering demonstrates that the thresholds from O'Sullivan et al. [2003] are not useful as a means of guiding the trade-off between accuracy and performance

in large, complex simulations. Furthermore, the similarity in magnitude between the errors of error injection, which we found to be tolerable, and the errors from reordering establishes credibility for the results in Table I.

4.2.4 *Threshold Evaluation* 4. Some metrics, such as penetration and gap, are dependent on the timestep of the simulation; if objects are moving rapidly enough and the timestep is too large, large penetrations can occur even without injecting any error. To demonstrate the impact of the timestep on our error metrics, we reduce the timestep to 0.001. The maximum penetration and gap at this timestep for a simulation without any error injection reduce to less than 0.001 meters and 0.009 meters, respectively. Both metrics see a comparable reduction when shrinking the timestep, which demonstrates that the larger magnitude penetration errors in Figure 6 are a function of the timestep and not the error injected.

4.2.5 *Believability Prediction*. As described in Section 4.1, most metrics are strongly correlated. As higher velocities, deeper penetrations, and higher constraint violations are observed, the simulation energy of the system grows accordingly. Given this behavior, we conclude that the difference in total energy is a reliable predictor of believable physical simulation in interactive entertainment applications. Human subject studies utilizing real-world applications may be used to further evaluate this conclusions. In the next section, we utilize this finding to evaluate four different methods of trading accuracy for performance.

5. CASE STUDIES

In this section, we present four case studies to make use of the perceptual believability methodology presented this article to trade off accuracy for performance. The first two, simulation timestep and iteration count, deal with the tuning of physics engine parameters. Fast Estimation with Error Control (FEEC) is a software optimization proposed in Yeh et al. [2006], and precision reduction is a hardware optimization.

5.1 Simulation Timestep

As described in Section 2, the simulation timestep largely defines the accuracy of simulation. In this case study, we apply a similar methodology to study the effects of scaling the timestep. We restrict the data shown here to the max % energy difference as it has been shown to be the main indication of simulation stability. We evaluate timesteps corresponding to frame rates of between 15 to 60 Frames Per Second (FPS). The baseline for energy comparison is the energy data using 60 FPS or 0.0167 sec. per frame. All simulations performed use 20 iterations for the LCP solver.

As shown on Figure 9, the simulation for our test scenario begins to stabilize at 34 FPS or a timestep of 0.0294 sec. per frame. While the energy data for 30 FPS is acceptable, it is within the region of instability between 15 FPS and 33 FPS. For a gaming application, it may be appropriate to select a timestep of 0.0167 sec. per frame (60 FPS) for two reasons: (1) to avoid instability during game play with different user input and (2) to synchronize the timing of rendering and physics. Although the appropriate timestep may depend on details of the exact scenario, our methodology can be leverage to tune the timestep for optimal performance.

5.2 Iteration Count

In addition to the simulation timestep, the number of iterations used within the constraint solver is another important parameter affecting simulation accuracy. In this case study, we evaluate the effect of scaling from 1 to 30 iterations for a given timestep. The baseline energy data is from simulation using 60 FPS and 20 iterations.

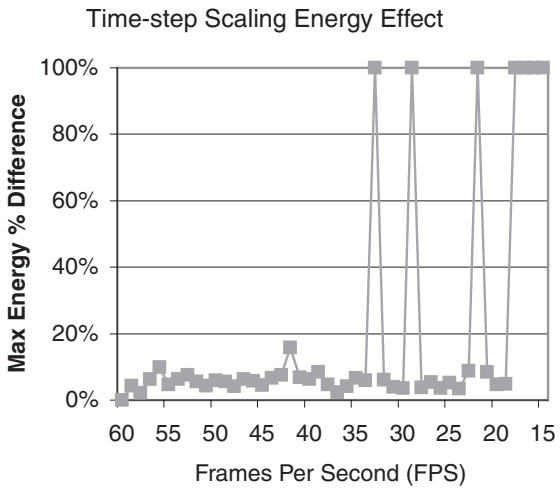

Fig. 9. Effect on energy with timestep scaling.

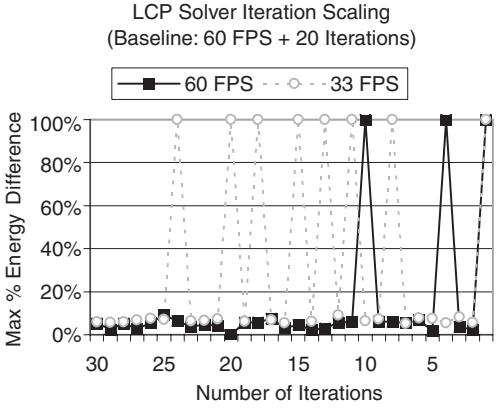

Fig. 10. Effect on energy with iteration scaling.

In Figure 10, we present detailed data on 2 points representative of the entire FPS range from the timestep case study (60 FPS and 33 FPS). The 60 FPS curve represents the stable points in Figure 9. As shown, simulation remains stable from 30 to 11 iterations. Additional reduction in iteration count causes simulation blow-up. This confirms that the suggested default of 20 iterations for ODE's LCP solver is a conservative choice. The next case study evaluates a performance optimization that further reduces the iteration count.

The 33 FPS curve represents the unstable points in Figure 9. While certain iteration counts produce acceptable energy data, simulation with 33 FPS is unstable even with over 30 iterations (data not shown). This behavior suggests that iteration count scaling cannot be used to compensate for the errors produced from a small timestep.

5.3 Fast Estimation with Error Control

Based on the results of Yeh et al. [2006] and our observation with simulation energy, error propagation can quickly lead to simulation

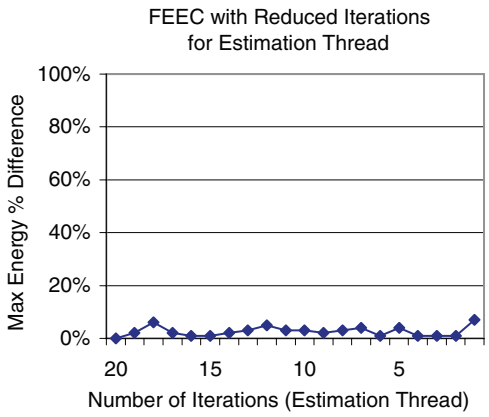

Fig. 11. Effect on energy with FEEC.

instability. This case study examines the energy data of the Fast Estimation with Error Control (FEEC) optimization as presented in Yeh et al. [2006]. FEEC is an optimization technique to trade accuracy for performance. It works by creating two logical threads of execution: (A) a precise thread and (B) an estimation thread. The precise thread produces slow but accurate results, same as the baseline simulation, by simulating with conservative parameters such as the ones used in the baseline of previous case studies (60 FPS and 20 iterations). The previous frame's precise result is fed to the inputs of both the precise thread and estimation thread at the beginning of each frame. The *estimation* thread returns earlier than the *precise* thread and allows other components of the application such as rendering or artificial intelligence to start consuming the estimated results earlier.

There are many estimation methods, but we focus on reducing the number of iterations as described in Yeh et al. [2006]. FEEC effectively reduces the application's critical path by generating fast and usable results for dependent components. At the same time, simulation stability is achieved by correcting all errors when precise results are used at the beginning of each frame. The main cost of FEEC is the increase in hardware utilization.

The prior work which proposed FEEC [Yeh et al. 2006] examined only differences in constraint violations, position, and orientation. We utilize our new methodology of using energy to evaluate FEEC's perceptual quality.

In Figure 11, we present the energy data of using FEEC to scale down the number of iterations for the *estimation* thread. As shown by the data, the energy of the estimation thread is stable from 20 iterations down to 1 iteration. This energy behavior further supports the findings of Yeh et al. [2006].

5.4 Precision Reduction

In this final case study, we apply our findings to the hardware optimization technique of precision reduction.

5.4.1 *Prior Work.* The IEEE 754 standard [Goldberg 1991] defines the single precision floating-point format as shown in Figure 12. When reducing a X number of mantissa bits, we remove the least-significant X bits. There is never a case where all information is lost since there is always an implicit 1.

Our methodology for precision reduction follows two prior papers [Samani et al. 1993; Fang et al. 2002]. Both prior works emulate variable-precision floating-point arithmetic by using new C++ classes.

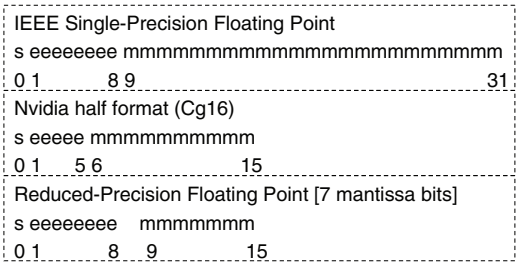

Fig. 12. Floating-point representation formats (s = sign, e = exponent, and m = mantissa).

Table IV. Numerically Derived Mantissa Precision Required in Each Computation Phase

Mantissa Bits Derived	Broadphase	Narrowphase	Island Processing	LCP
[round, truncate]	[5-6, 6-7]	[5-6, 6-7]	[12-13, 13-14]	[5-6, 6-7]

Table V. Simulation-Based Mantissa Precision Requirement in Each Computation Phase

Mantissa Bits Simulated	Broadphase	Narrowphase	Island Processing	LCP	Narrow + LCP
[round, truncate]	[3, 4]	[4, 7]	[7, 8]	[4, 6]	[5, 7]

5.4.2 *Methodology.* We apply precision reduction first to both input values, then to the result of operating on these precision-reduced inputs. This allows for more accurate modeling than Samani et al. [1993] and is comparable to Fang et al. [2002]. Two rounding modes are supported (round to nearest and truncation). We support the most frequently executed FP operations for physics processing which have dedicated hardware support: $+$, $-$, and *. Denormal handling is unchanged, so denormal values are not impacted by precision reduction.

Our precision reduction analysis focuses on mantissa bit reduction because preliminary analysis of exponent bit reductions shows low tolerance (not even a single bit of reduction is allowed). A single exponent bit reduction can cause up to orders of magnitude errors being injected.

5.4.3 *Per-Phase Precision Analysis.* The goal of precision reduction is to reduce the size of floating-point hardware (FPUs) on processors. The cumulative area taken by a large number of FPUs in processors occupies a large percentage of total processor area. The fine-grain parallelism in the computation phases of Narrow-phase and LCP can be exploited more effectively with reduced-precision FPUs in physics accelerators or GPUs.

Based on the error tolerance shown in Table I, we can numerically estimate the minimum number of mantissa bits for each phase. When using the IEEE single-precision format with an implicit 1, the maximum numerical error from using an *X-bit* mantissa with rounding is $2^{-(X+1)}$ and with truncation is 2^{-X}. Rounding allows for both positive and negative errors while truncation only allows negative errors.

Since base 2 numbers do not neatly map to the base 10 values shown in Table I, we present a range of possible minimum mantissa bits in Table IV for rounding and truncation. Now that we have an estimate on how far we can take precision reduction, we evaluate the actual simulation results to confirm our estimation.

By utilizing the methodology of Section 4.2, we summarize the per-phase minimum precision required in Table V based on energy

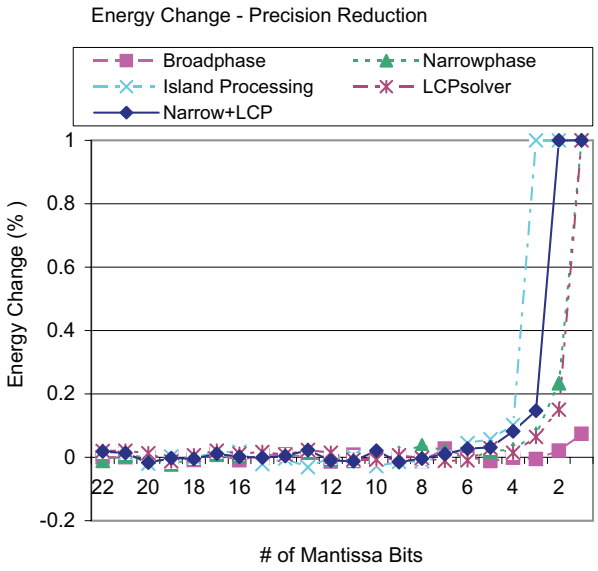

Fig. 13. Energy data for precision reduction using rounding. X-axis shows the number of mantissa bits used. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.

change data in Figure 13. When comparing Table IV and Table V, we see that the actual simulation is more tolerant than the stricter numerically derived thresholds. This gives further confidence in our numerical error tolerance thresholds.

While the exact precision reduction tolerance may vary across different physics engines, this study shows the potential for leveraging precision reduction for hardware optimizations.

6. CONCLUSION

We have addressed the challenging problem of identifying the maximum error tolerance of physical simulation as it applies to interactive entertainment applications. We have proposed a set of numerical metrics to gauge the believability of a simulation, explored the maximal amount of generalized error injection for a complex physical scenario, and proposed the use of maximum % energy difference (as compared to an accepted baseline) to evaluate the perceptual quality of the simulation. We then investigated four different approaches to trading accuracy for performance based on our findings. For error-sensitive applications utilizing PBA such as medical simulations, more detailed examination of perceptual metrics may be required. Future work will extend the proposed methodology for error-sensitive applications.

REFERENCES

AGEIA. Physx product overview. www.ageia.com.

BARAFF, D. 1997. Physically based modeling: Principals and practice. In *Proceedings of the SIGGRAPH Online Course Notes*.

BARZEL, R., HUGHES, J., AND WOOD, D. 1996. Plausible motion simulation for computer graphics animation. In *Proceedings of the Computer Animation and Simulation*.

CARLSON, D. AND HODGINS, J. 1997. Simulation levels of detail for real-time animation. In *Proceedings of the Graphics Interface Conference*.

CHENNEY, S. AND FORSYTH, D. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interface Techniques*.

FANG, F., CHEN, T., AND RUTENBAR, R. 2002. Floating-Point bit-width optimization for low-power signal processing applications. In *Proceedings of the International Conference on Acoustic, Speech, and Signal Processing*.

GOLDBERG, D. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv. 23*, 1, 5–48.

HARRSION, J., RENSINK, R. A., AND VAN DE PANNE, M. 2004. Obscuring length changes during animated motion. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*.

HAVOK. Havokfx. www.havok.com.

HOFSTEE, P. 2005. Power efficient architecture and the cell processor. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*. 258–262.

MCDONNELL, R., DOBBYN, S., COLLINS, S., AND O'SULLIVAN, C. 2006. Perceptual evaluation of lod clothing for virtual humans. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 117–126.

MULLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2006. Position based dynamics. In *Proceedings of the 3rd Workshop in Virtual Reality Interactions and Physical Simulation*.

NEWTON. Newton game dynamics. www.newtondynamics.com.

O'SULLIVAN, C. AND DINGLIANA, J. 2001. Collisions and perception. *ACM Trans. Graph. 20*, 3, 151–168.

O'SULLIVAN, C., DINGLIANA, J., GIANG, T., AND KAISER, M. 2003. Evaluating the visual fidelity of physically based animations. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*.

O'SULLIVAN, C., HOWLETT, S., MCDONNELL, R., MORVAN, Y., AND O'CONOR, K. 2004. Perceptually adaptive graphics. *Eurographics State of the Art Report*.

PROFFIT, D. 2006. Viewing animations: What people see and understand and what they don't. Keynote address. In *Proceedings of the ACM SIGGRAPH/Eurographics Symosium on Computer Animation*.

REITSMA, P. AND POLLARD, N. 2003. Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH Papers*. 537–542.

SAMANI, D. M., ELLINGER, J., POWERS, E. J., AND SWARTZLANDER, E. E. J. 1993. Simulation of variable precision ieee floating point using c++ and its application in digital signal processor design. In *Proceedings of the 36th Midwest Symposium on Circuits and Systems*.

SEUGLING, A. AND ROLIN, M. 2006. Evaluation of physics engines and implementation of a physics module ina 3d-authoring tool. Master's thesis, UMEA University.

SMITH, R. Open dynamics engine. www.ode.org.

YEH, T. Y., FALOUTSOS, P., AND REINMAN, G. 2006. Enabling real-time physics simulation in future interactive entertainment. In *Proceedings of the ACM SIGGRAPH Video Game Symposium*. 71–81.

YEH, T. Y., FALOUTSOS, P., PATEL, S., AND REINMAN, G. 2007. Parallax: An architecture for real-time physics. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*. 232–243.

Received May 2007; revised August 2009; accepted August 2009