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

Thomas Y. Yeh† Glenn Reinman† Sanjay J. Patel‡ Petros Faloutsos†
†Computer Science Department, UCLA, {tomyeh, reinman, pfal}@cs.ucla.edu
‡AGEIA Technologies, sjp@ageia.com

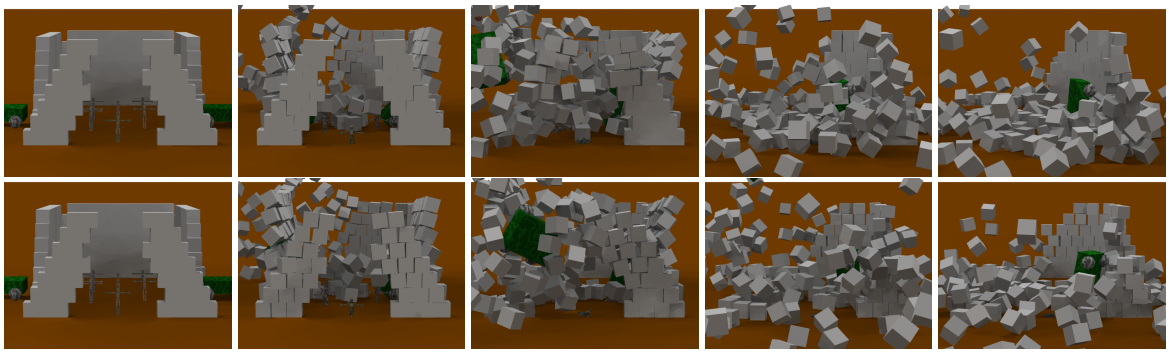

Figure 1: *Snapshots of two simulation runs with the same initial conditions. The simulation results shown on top is the baseline, and the bottom row is simulation computed with 7-bit mantissa floating-point computation in* Narrowphase *and* LCP. *The results are different but both are visually correct.*

Abstract

The error tolerance of human perception offers a range of opportunities to trade numerical accuracy for performance in physics-based simulation. However, most previous approaches either focus exclusively on understanding the tolerance of the human visual system or burden the application developer with case-specific implementations. In this paper, 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 two techniques. The first is the hardware optimization technique of precision reduction which reduces the size of floating point units (FPUs), allowing more of them to occupy the same silicon area. The increased number of FPUs can significantly improve the performance of future physics accelerators. A key benefit of our approach is that it is transparent to the application developer. The second is the software optimization of choosing the largest timestep for simulation.

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

Keywords: physics-simulation, perceptual errors

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, making it 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 to improve the performance of PBA is an active area of research both in terms of software techniques and hardware accelerators. Commercial solutions, such as PhysX [AGEIA], GPUs [Havok Inc.], and the Cell [Hofstee 2005], have just started to address this problem.

But while parallelism can help PBA achieve a real-time frame rate, there is another avenue to help improve PBA performance which also has the potential to reduce the hardware required to exploit parallelism: *perceptual error tolerance*. 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 both 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 [Reitsma and Pollard 2003a; 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

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

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 paper, we address the challenging problem of leveraging perceptual error tolerances to improve the performance of real-time physics simulation in interactive entertainment. The main challenge is the need to establish a set of error metrics that can 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 specific hardware solutions.

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 hardware precision reduction and software time-step tuning to exploit error tolerance to improve PBA performance.

2 Background

In this section, we present a brief overview of physics simulation, identify its main computational phases, and categorize possible errors. We then review the latest results in perceptual error metrics and their relation to physical simulation.

2.1 Computational Phases of a typical Physics Engine

Physics simulation requires the numerical solution of a discrete approximation 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; Havok Inc.]. Time is discretized with a fixed or adaptive time-step. The time-step is one of the most important parameters of the simulation and largely defines the accuracy of the simulation. For interactive applications the time step needs to be in the range of 0.01 to 0.03 simulated seconds or higher. The errors and the stability issues associated with the time-step are orthogonal to the first part of this study. The time-step is a high-level parameter that should be controlled by the developer of an application. Our work identifies both low-level trade-offs that are part of a hardware solution and as such are transparent to an application developer as well as high-level trade-offs to assist the programmer. As we demonstate later, the errors that our low-level approach introduces in the simulation results are no greater that those introduced by the discretization alone.

Physics simulation can be described by the dataflow of computational phases shown in Figure 2. Cloth and fluid simulation are special cases of this pipeline. Below 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 *Broadphase* 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.

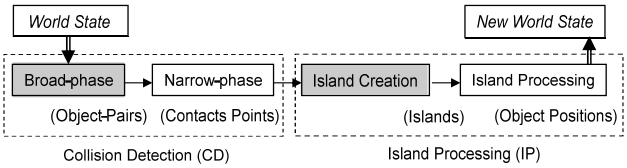

Figure 2: *Physics Engine Flow. All phases are serialized with respect to each other, but* **unshaded** *stages can exploit parallelism within the stage.*

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. This phase contains no floating-point operations, so it is excluded from this study.

Island Processing. For each island, given the applied forces and torques, the engine computes the resulting accelerations and integrates them to compute the new position and velocity of each object. 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 non-trivial 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

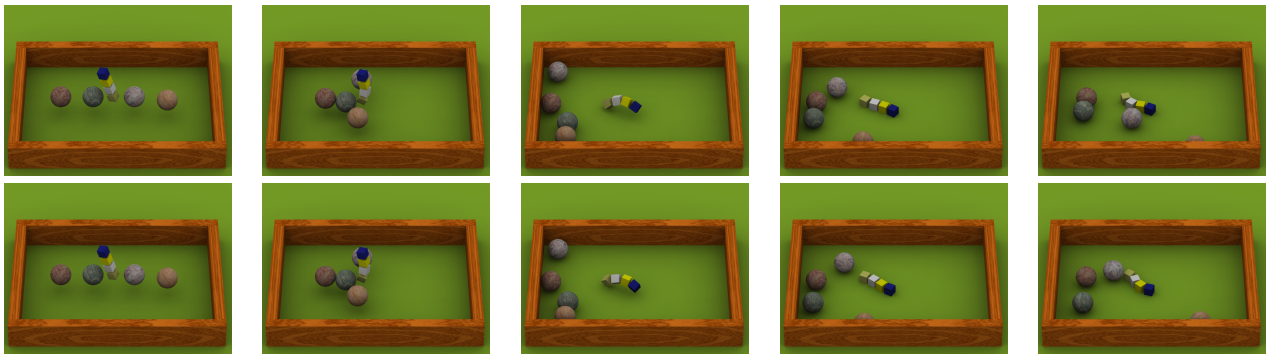

Figure 3: *Snapshots of two simulation runs with the same initial conditions and different constraint ordering. The results are different but both are visually correct.*

considered visible but bounded. All errors that lead to simulation blow-up are considered catastrophic.

An interesting example that demonstrates imperceptible 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.[2] 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 impoves the stability of the numerical constraint solver and it is employed by commercial products such as [AGEIA]. For our purposes though, it provides an objective way to establish what 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 2004 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 non-photorealistic rendering. Our paper will focus on the animation category. Given the comprehensive coverage of this prior survey paper, we will only present prior work most related to our paper 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. For the metrics examined in this paper, 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. This paper only uses 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 physics simulation. We derive the threshold for constraint error from this paper.

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

[Yeh et al. 2006] examines the techniques of fuzzy value prediction and fast estimation with error control to leverage error tolerance in interactive entertainment applications. However, this prior work does not evaluate how to measure tolerable error nor deal with maximum tolerable error. It shows absolute differences in terms of position and orientation of objects and constraints against baseline simulation without error. In fact, our work argues against such methods of measuring error using absolute position and orientation differences as demonstrated by Figure 3.

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 apriori knowledge of the correct motion which serves as the baseline for exact error comparisons. Complex cases do not offer that luxury. On the other hand, in complex cases, such as multiple simultaneous collisions, errors become difficult to detect and, may in fact 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

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

[2]The complete animation of this experiment is included in the accompanying video.

3 Methodology

One major challenge in exploring the tradeoff between accuracy and performance in PBA is coming up with a set of metrics that will evaluate believability. Furthermore, 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 one complex test-case which includes stacking, highly articulated objects, and highspeed objects shown in Figure 1. The scene is composed of a building enclosed on all four sides by brickwalls with one opening. The wall sections framing the opening is unstable. Ten humans with anthropormorphic dimension, mass, and joints are stationed within the enclosed area. A cannon shoots highspeed (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 seconds until 0.4 seconds. The cars are accelerated to roughly 100 miles/hr (44 m/s) at time 0.12 to crash into the walls. No forces are injected after 0.4 seconds. 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; 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 time-step of 0.01 seconds and 20 solver iterations as recommended by the ODE user-base.

3.2 Error Sampling Methodology

To evaluate the numerical error tolerance of physics simulation, we will 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 will be fairly general, and should be representative of a range of possible scenarios where error could occur. At a high level, we will be changing 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%.

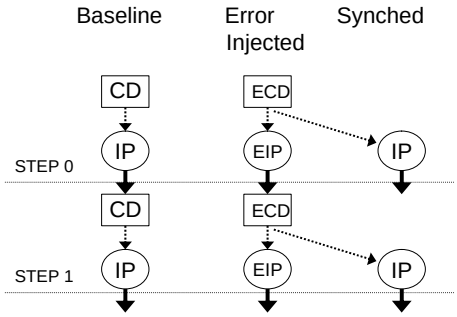

Figure 4: *Simulation Worlds. CD = Collision Detection. IP = Island Processing. E = Error-injected*

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 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 be able to determine when the behavior of a simulation with error is still believable through numerical analysis. Many of the metrics we will 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 error injected into it with a simulation without any error. Small, but perceptually tolerable difference 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 (cannon ball 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.

We use the following seven numerical metrics:

- Energy Difference: difference in total energy between *baseline* and *error-injected* worlds – due to energy conservation, these should match.
- Penetration Depth: distance from the object's surface to the contact point created by collision detection.
- 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 – perhaps the time-step is too small.

The above 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. Furthermore, the approximate collision resolution models often do not conserve momentum either.

4 Numerical Error Tolerance

In this section, we will explore the use of our error metrics in a complex game scene with a large number of objects. We will inject error into this scene for different ODE phases, and measure the response from these metrics to determine how far we can go when trading accuracy for performance.

Before delving into the details, we briefly articulate the potential outcome of error injection in different ODE phases. Because *Broadphase* is a first-pass filter on potentially colliding object-pairs, it can only create functional errors by ommiting actually colliding pairs. Since *Narrowphase* will not see the ommitted 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 *Broadphase* filtering can degrade performance by sending *Narrowphase* more object-pairs to process. Errors in *Narrowphase* may lead to missed collisions or different contact points. The results are similar to omission by *Broadphase*, but can also include errors in the angular component of linear velocity due to errors in contact points. 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* will 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* will most likely 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. Rigid body simulation in some sense has tighter requirements for accuracy than other types of real-time physical simulation. Unexpected behaviors (interpentration, unexpected increases in velocity, etc) are more often visually apparent because rigid bodies are often used to model larger objects, which usually affect gameplay. In contrast, errors in a particle simulation are more likely to be tolerated by the viewer because the artifacts themselves are smaller phenomena. We conjecture that fluid and cloth simulation (particularly those that are particle-based) are likely to exhibit similar numerical tolerance, as the visual representations (such as the fluid surface or cloth mesh) represents a down sampling of the underlying physical model. That is, many particles can represent a volume of fluid, but fewer actually represent the surface. We leave the empirical evaluation for future work.

4.1 Numerical Error Analysis

For this initial error analysis, we ran a series of experiments where we injected an increasing degree of error into different phases of ODE. Figure 5 demonstrates the maximal error that results from error injection on our seven perceptual metrics. 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, *Broadphase* and *Island Processing*, exhibit the the highest and lowest per phase error tolerance respectively. Only *Broadphase* 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 *Narrowphase* and *LCP* show similar average sentivity to error. We will make use of these per-phase sensitivity differences when trading accuracy for performance.

As shown by the graphs in Figure 5, 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 as there are higher velocities, deeper penetrations, and more 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, broadphase would need to allow two objects which are not actually colliding to get to narrowphase, and narrowphase would need to mistakenly find that these objects were actually not in contact with one another. 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.

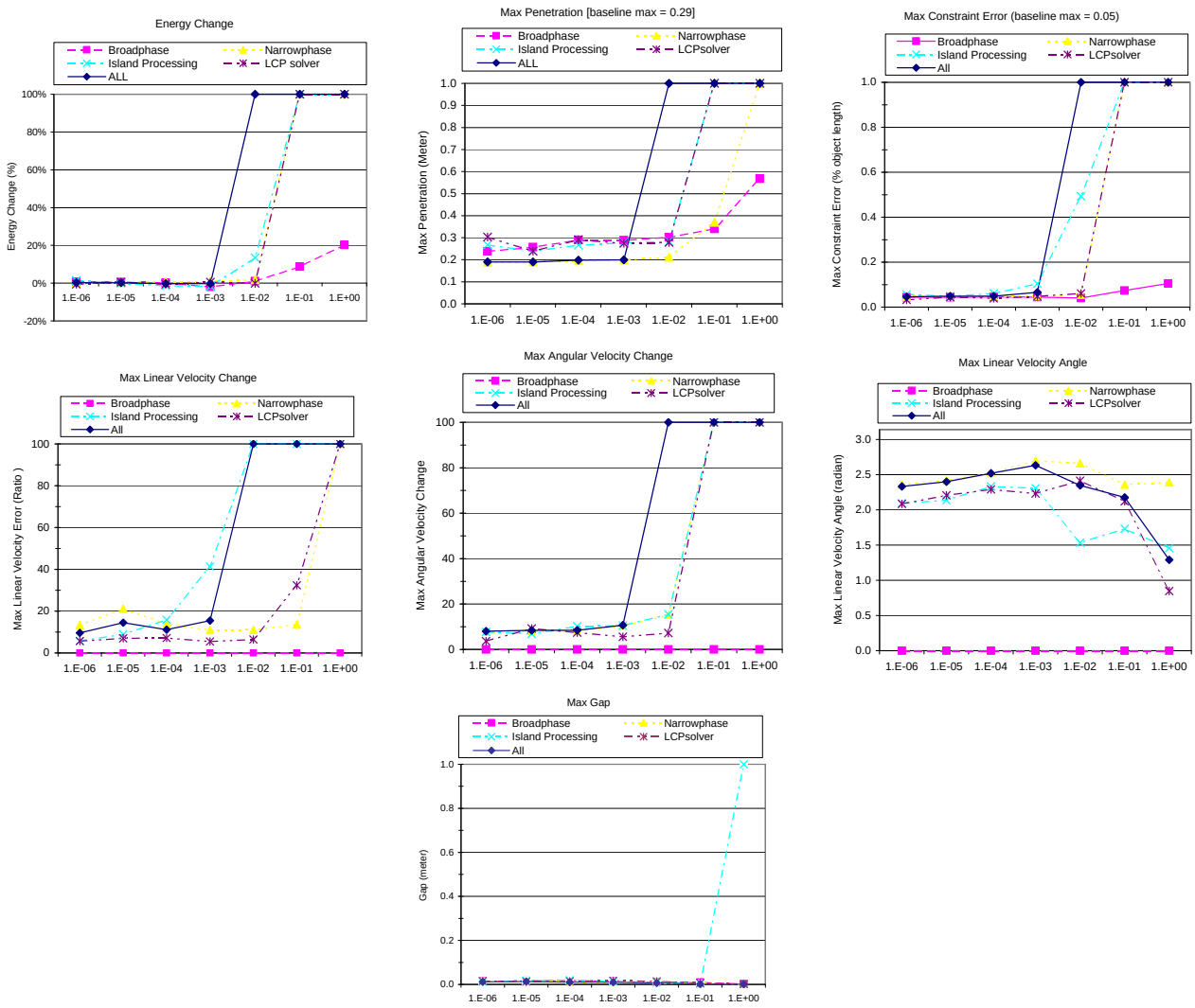

Figure 5: *Perceptual Metrics Data for Error-Injection. X-axis shows the maximum possible injected error. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.*

While these maximal error values are interesting, as shown in figure 6, the average error in our numerical metrics and the standard deviation of errors are both extremely small. 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 results in significant percentage change.

4.2 Acceptable Error Threshold

Now that we have examined the response of PBA to injected error using our error metrics, the question still remains as to how much error we can actually tolerate and keep the simulation believable. Consider figure 5 where we show the maximum value of each metric for a given level of error. Instead of using fixed thresholds to evaluate simulation fidelity and find the largest amount of 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, figure 6, with the majority of errors resulting 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.

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

Table 1: *Max Error Tolerated for Each Computation Phase*

Table 1 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 sam-

Penetration	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.0010%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.0100%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.1000%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1.0000%	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	inf	inf
10.0000%	0.00	0.01	0.00	0.02	inf	inf	inf	inf	inf	inf
100.0000%	0.00	0.02	inf	inf	inf	inf	inf	inf	inf	inf

Constraint	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.0010%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.0100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.1000%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
1.0000%	0%	0%	0%	0%	1%	3%	0%	0%	inf	inf
10.0000%	0%	0%	inf	inf	inf	inf	inf	inf	inf	inf
100.0000%	0%	0%	inf	inf	inf	inf	inf	inf	inf	inf

Linear Vel	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0%	0%	3%	34%	0%	8%	0%	9%	1%	18%
0.0010%	0%	0%	10%	79%	1%	14%	0%	10%	3%	38%
0.0100%	0%	0%	3%	34%	7%	36%	1%	11%	4%	25%
0.1000%	0%	0%	2%	20%	91%	245%	1%	9%	11%	49%
1.0000%	0%	0%	3%	26%	826%	2427%	1%	13%	10%	185%
10.0000%	0%	0%	12%	53%	5335%	42312%	7%	49%	inf	inf
100.0000%	0%	0%	inf	inf	inf	inf	inf	inf	inf	inf

Angular Vel	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0.00	0.00	0.00	0.09	0.00	0.09	0.00	0.04	0.00	0.09
0.0010%	0.00	0.00	0.00	0.09	0.00	0.07	0.00	0.11	0.00	0.09
0.0100%	0.00	0.00	0.00	0.09	0.00	0.11	0.00	0.09	0.00	0.10
0.1000%	0.00	0.00	0.00	0.12	0.00	0.11	0.00	0.06	0.01	0.12
1.0000%	0.00	0.00	0.01	0.18	0.04	0.23	0.00	0.08	inf	inf
10.0000%	0.00	0.00	inf	inf	inf	inf	inf	inf	inf	inf
100.0000%	0.00	0.00	inf	inf	inf	inf	inf	inf	inf	inf

Linear Vel Angle	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0.00	0.00	0.01	0.07	0.00	0.04	0.00	0.05	0.01	0.06
0.0010%	0.00	0.00	0.01	0.07	0.00	0.05	0.00	0.05	0.01	0.07
0.0100%	0.00	0.00	0.01	0.08	0.01	0.06	0.01	0.06	0.01	0.08
0.1000%	0.00	0.00	0.01	0.10	0.01	0.05	0.01	0.06	0.02	0.09
1.0000%	0.00	0.00	0.02	0.11	0.03	0.04	0.01	0.07	0.03	0.09
10.0000%	0.00	0.00	0.02	0.10	0.03	0.07	0.03	0.10	0.04	0.09
100.0000%	0.00	0.00	0.01	0.06	0.04	0.05	0.04	0.05	0.03	0.06

Gap	Broadphase		Narrowphase		Island Processing		LCPsolver		ALL	
Max Error	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev	Avg	Stddev
0.0001%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.0010%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.0100%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.1000%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1.0000%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10.0000%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
100.0000%	0.000	0.000	0.000	0.000	inf	inf	0.000	0.000	0.000	0.000

Figure 6: *Average and Standard Deviation of Error-Injection*

ples of a random variable. More samples lead to a converging mean which in our case is zero.

To further support the choice made in Table 1, we consider four approaches: confirming our thresholds visually, comparing our errors to a very simple scenario with clear error thresholds [O'Sullivan et al. 2003], comparing the magnitude of our observed error to constraint reordering, and examining the effect of the time-step 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 – bricks moving on their own, human figures flying apart, and other chaotic developments.

Second, we specifically flagged errors that exceeded the thresholds from [O'Sullivan et al. 2003]. By using their thresholds, we could filter out errors that were certainly imperceptible. Objects that were not flagged were made translucent, and objects that had been flagged as having an error were colored according to the type of error. We analyzed this behavior by slowly walking frame by frame through the simulation. The errors we saw for error injection at or below the thresholds of table 1 were transient – typically lasting only a single frame – and we would not have been able to distinguish them were it not for the coloring of objects and our pausing at each frame to inspect the object positions. [3]

4.2.2 Threshold Evaluation 2

We recreated 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 2-D plane: one is stationary 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. When evaluating this simple scenario, the thresholds shown in the rightmost column of table 2, extrapolated from prior work [O'Sullivan et al. 2003; Harrison et al. 2004], are sufficient. They are conservative enough to flag catastrophic errors such as tunneling, large penetration, and movement without collision.

However, when applying this method to a complicated game-like scenario, these thresholds 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 perceptual acuity will become even worse - particularly in areas of the simulation that are not the current focal point.

But since these metrics work for this simple example, we will use the error rates from table 1 to inject error into this simple example. Using the thresholds from [O'Sullivan et al. 2003], we were unable to find any errors that exceeded these thresholds – the error rates that we experimentally found in the previous section show no perceptable error for this simple example.

4.2.3 Threshold Evaluation 3

Interestingly, when we enable reordering of constraints in ODE, most of our perceptual metrics will exceed the thresholds from [O'Sullivan et al. 2003], compared to a run without reordering. There is no particular ordering which will generate an absolutely correct simulation when using an iterative solver such as ODE's quickstep. 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 will result in constraint reordering relative to the original baseline.

Therefore, 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 will use the magnitude of difference generated by reordering as a comparison point for the errors we have experimentally found tolerable in table 1. 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 2 shows the maximum errors in our perceptual metrics when enabling reordering versus a baseline simulation of a complex scene without any reordering. As a further reference point, we also include the perceptual thresholds from the simple simulation described in [O'Sullivan et al. 2003]. For the simple simulation, some of our metrics were not used, and we mark these as not available (*NA*). For baseline simulation, only absolute metrics (i.e. those that require no comparison point like gap and penetration) are shown,

[3] One such experiment is shown in the accompanying video.

and relative metrics that would ordinarily be compared against the baseline itself (i.e. linear velocity) are also marked *NA*.

Perceptual Metrics	Random Reordering	Baseline	Simple Threshold
Energy (% change)	-1%	NA	NA
Penetration (meters)	0.25	0.29	NA
Constraint Error (ratio)	0.05	0.05	0.03/0.2
Linear Vel (ratio)	5.27	NA	-0.4/+0.5
Angular Vel (radians/sec)	4.48	NA	-20/+60
Linear Vel Angle (radians)	1.72	NA	-0.87/+1.05
Gap (meters)	0.01	0.01	0.001

Table 2: *Perceptual Metric Data for Random Reordering, Baseline, and Simple Simulations*

The first thing to notice from these results is that the magnitude of maximal error for a complex scene (reordering and baseline) can be much more than the simple scenario. 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 time step, but by observing the amount of penetration from a baseline at a given time step, we can ensure that it does not get any worse from introduced error.

The sheer magnitude of the errors from reordering relative to the very conservative thresholds from prior work [O'Sullivan et al. 2003] demonstrates that these thresholds are not useful as a means of guiding the tradeoff between accuracy and performance in large, complex simulations. Furthermore, the similarity in magnitude of the errors we found to be tolerable and the errors from reordering establish credibility for the results in table 1.

4.2.4 Threshold Evaluation 4

Some metrics, such as penetration and gap, are dependent on the time-step of the simulation – if objects are moving rapidly enough and the time-step is too large, large penetrations can occur even without injecting any error. To demonstrate the impact of the time-step on our error metrics, we reduce the time step to 0.001. The maximum penetration and gap at this time-step when injecting a maximum 0.1% error into all phases of PBA reduce to 0.007 meters and 0.005 meters respectively. The same metrics for a run without any error injection reduce to 0.000 meters and 0.009 meters respectively. Both metrics see a comparable reduction when shrinking the time-step, which demonstrates that the larger magnitude penetration errors in figure 5 are a function of the time-step and not the error injected.

5 Precision Reduction

Now that we can evaluate our perceptual metrics for complex scenarios, we apply our findings to the hardware-optimization technique of precision reduction.

5.1 Prior work

The IEEE 754 standard [Goldberg. 1991] defines the single precision floating-point format as shown in figure 7. The first bit is the sign bit, the next eight bits are the exponent bits, and the next 23 bits are the mantissa (fraction) bits. There is an implicit 1 at the beginning of the mantissa, so logically there are 24 mantissa bits. When reducing a X number of mantissa bits, we remove the least-significant X bits. There will never be a case where all information is lost since there is always an implicit 1.

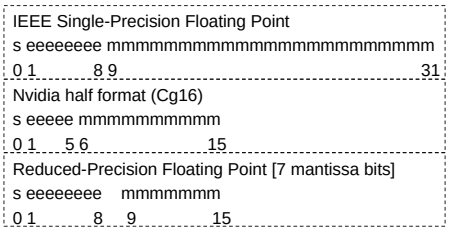

Figure 7: *Floating-point Representation Formats (s = sign, e = exponent, and m = mantissa)*

For graphics processing, Nvidia has proposed the 16-bit "half" format (Cg16) [Mark et al. 2003] as shown in Figure 7. There is one sign bit, and both the exponent and mantissa bits are reduced. The Cg16 format has been used for several embedded applications [Etiemble et al. 2005; Lacassagne et al. 2005; Eilert et al. 2004] outside of graphics. Reducing the exponent bits reduces the range of representable values. While this is tolerable for media applications, physics simulation is highly sensitive and quickly explodes based on our simulations.

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

Prior work [Goddeke et al. 2006] has also compared the use of double precision solvers with native, emulated, and mixed-precision double precision support. They conclude that a mixed precision approach gains performance by saving area while maintaining accuracy. Our work further pushes the boundary of this area by reducing precision below that of IEEE single-precision configured at a per-phase granularity.

5.2 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 use function calls instead of overloading arithmetic operations with a new class. This enables us to separately tune the precision of code segments (such as computation-phase) instead of the entire application. As the data will show, there is a significant difference between the two granularities. 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 will focus on mantissa bit reduction because preliminary analysis of exponent bit reductions shows high sensitivity (no tolerance for even a single bit of reduction). A single exponent bit reduction can cause up to orders of magnitude errors being injected. If we follow the IEEE convention of biased exponents, using 7 instead of 8 bits reduces the exponent range from [-126,127] to [-63,64]. This can result in large % errors as compared to the small % errors injected due to precision

reduction.

5.3 Per Phase Precision Analysis

The following section analyzes the precision reduction tolerance of each phase as well as *Narrowphase + LCP*. The reason for examining this combination of phases instead of all phases is because of their highly parallel nature, described in section 2.4. The goal of precision reduction is to reduce the die area of FPUs – and the fine-grain parallelism in these two phases would be exploited by small, relatively simple cores with FPUs in physics accelerators such as PhysX or in GPUs. The FPUs in such simple cores would occupy a considerable amount of overall core area.

Broadphase and *Island Processing* do not have vast amounts of fine-grain parallelism, and would therefore need a smaller number of more complex cores to ensure high performance. Therefore it is unlikely that the FPU would occupy a sufficiently large fraction of the complex core to make precision reduction worthwhile. However, we still present results here for all four phases for completeness.

Based on the error tolerance shown in Table 1, 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 1, we present a range of possible minimum mantissa bits in table 3 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.

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 3: *Numerically-derived Min Mantissa Precision Tolerated for Each Computation Phase*

The top left graph of Figure 8 shows the actual simulation error generated from precision reduction with rounding. This data corresponds to the numerical analysis of $2^{-(X+1)}$.

Following the same methodology we used in the section 4, we summarize the per-phase minimum precision required in Table 4 based on data in Figure 8. When comparing Table 4 and Table 3, 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.

We also studied the effects of truncation instead of rounding when reducing precision (data not shown due to space constraints). Truncation further optimizes area by removing the adder used for rounding. The metrics point to a minimum of 7 bits for the mantissa when executing both *Narrowphase* and *LCP* with reduced precision. The perceptual believability of 7-bit mantissa simulation is supported by the fact that the average and standard deviation are well below the very conservative thresholds of [O'Sullivan et al. 2003; Harrison et al. 2004].

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

Table 4: *Simulation-based Min Mantissa Precision Tolerated for Each Computation Phase*

5.4 Hardware Area and Performance Improvement

For commercial physics accelerators such as PhysX and GPUs, simple cores make up the bulk of the die area. In 90nm process technology, a simple processor core's area (assuming a $3mm^2$ core) would have the following breakdown: 30% memory area ($0.9mm^2$), 50% logic and flops ($1.5mm^2$), 10% register file ($0.3mm^2$), and 10% overhead ($0.3mm^2$).

Reducing the mantissa from 24 (23 + implicit 1) bits to 8 (7 + implicit 1) bits reduces the logic cost from $1.5mm^2$ to about $0.5mm^2$. The precision-reduced core is 33% smaller than the IEEE single-precision core. This estimate is based on results from a high-level synthesis tool tuned for generating area-efficient floating point datapaths.

This 33% reduction in core area can be leveraged to increase the number of cores in the same silicon area, reduce the overall silicon area required for the cores, add performance-enhancing reduced-precision SIMD hardware to each core, reduce energy by turning off logic to handle unnecessary bits, or increase core clock frequency, just to name a few options. A hybrid chip design that has support for low-precision FP (at a higher throughput) and single-precision FP (at a lower throughput) is a good way to exploit perceptual tolerance because the developer can choose between high-throughput and high-accuracy. Components like the LCP solver and Narrowphase (which dominate the run time of PBA) would greatly benefit from the higher throughput, and as we have shown, they are very tolerant of reduced accuracy.

While the exact precision reduction tolerance may vary across different physics engines, this study shows the potential for leveraging precision reduction for hardware optimizations such as area reduction, SIMD hardware, energy reduction, or increase core clock frequency.

6 Simulation Time-Step

As described in Section 2.4, the simulation time-step largely defines the accuracy of simulation. In this section, we apply a similar methodology to study the effects of scaling the time-step. We restrict the data shown here to % energy difference as it has been shown to be the main indication of simulation stability. Because time advances in different granularities for these simulations, user input needs to be injected at a common granularity where all simulations see the inputs at the exact same time. The % energy difference also needs to be computed at a common interval for all time-steps. Because of this restriction, we show results for time-steps of 0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.015, 0.02, 0.03, and 0.06. The common interval used is 0.06 second.

As shown on Figure 9, the maximum time-step (out of the ones simulated) for the scenario we tested is 0.03 second. Although the appropriate time-step may depend on details of the exact scenario, this methodology can be leverage to tune this physics engine parameter for optimal performance.

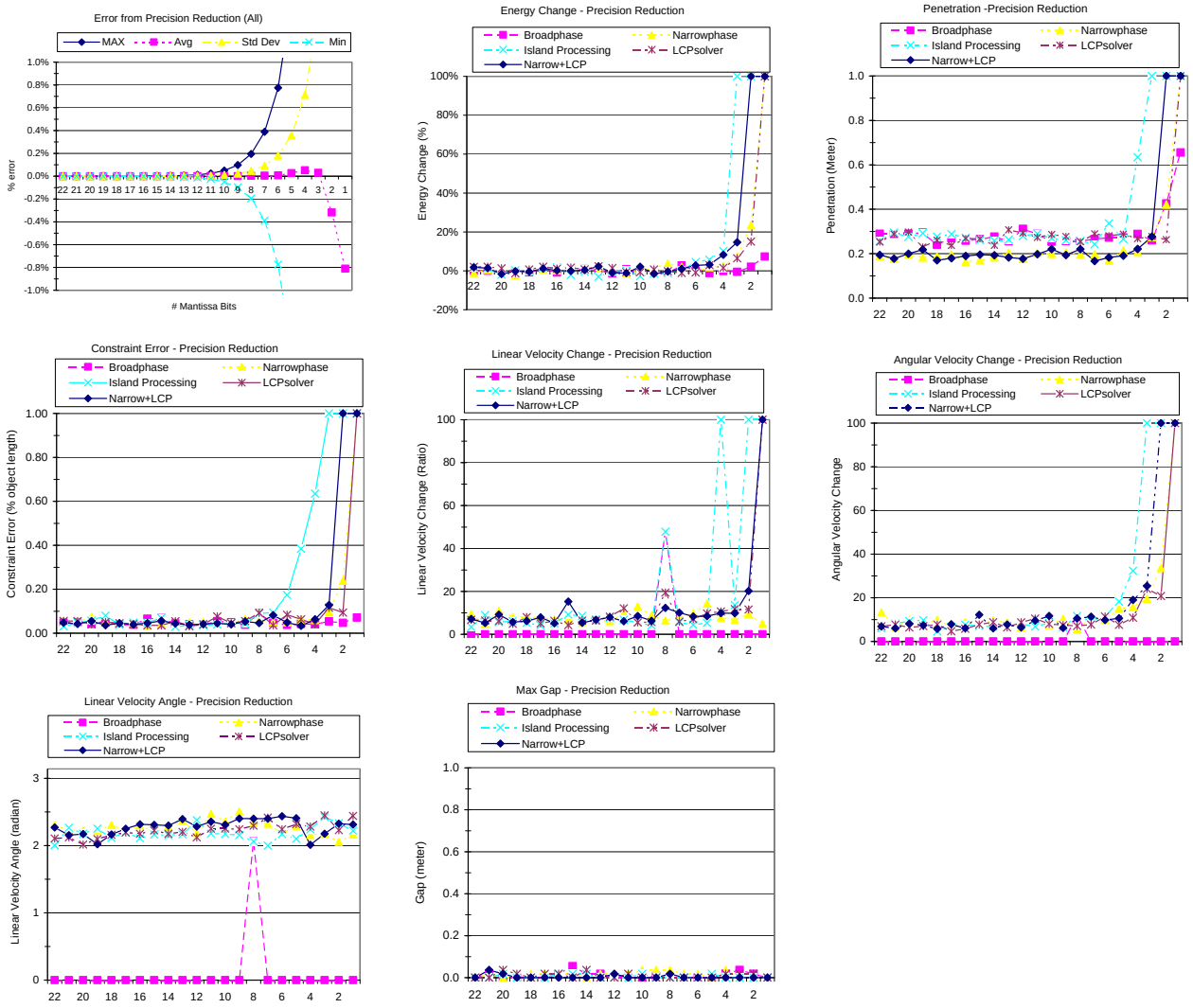

Figure 8: *Perceptual Metrics Data for Precision Reduction. X-axis shows the number of mantissa bits used. Note: Extremely large numbers and infinity are converted to the max value of each Y-axis scale for better visualization.*

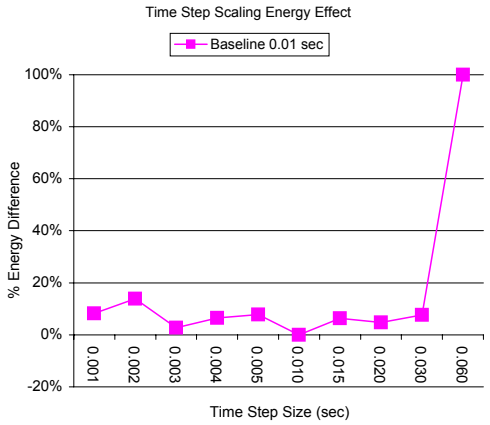

Figure 9: *Effect on Energy with Time-Step Scaling*

7 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 examined two ways of exploiting this error tolerance: reducing precision and tuning time-step.

References

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

BARAFF, D. 1997. *Physically Based Modeling: Principals and Practice*. SIGGRAPH Online Course Notes.

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

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

CHENNEY, S., AND FORSYTH, D. 2000. Sampling plausible solutions to multi-body constraint problems. In *ACM SIGGRAPH*.

EILERT, J., EHLIAR, A., AND LIU, D. 2004. Using low precision floating point numbers to reduce memory cost for mp3 decoding. In *IEEE 6th Workshop on Multimedia Signal Processing*.

ETIEMBLE, D., BOUAZIZ, S., AND LACASSAGNE, L. 2005. Customizing 16-bit floating point instructions on a nios ii processor for fpga image and media processing. In *3rd Workshop on Embedded Systems for Real-Time Multimedia*.

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

GODDEKE, D., STRZODKA, R., AND TUREK, S. 2006. Performance and accuracy of hardware-oriented native-, emulated-, and mixed-precision solvers in fem simulations. In *International Journal of Parallel, Emergent and Distributed Systems*.

GOLDBERG., D. 1991. What every computer scientist should know about floating-point arithmetic. In *ACM Computing Surveys (CSUR)*.

HARRISON, J., RENSINK, R. A., AND VAN DE PANNE, M. 2004. Obscuring length changes during animated motion. In *ACM Transactions on Graphics, SIGGRAPH 2004*.

HAVOK INC. Havokfx.

HOFSTEE, P. 2005. Power efficient architecture and the cell processor. In *HPCA11*.

LACASSAGNE, L., ETIEMBLE, D., AND OULD KABLIA, S. 2005. 16-bit floating point instructions for embedded multimedia applications. In *Seventh International Workshop on Computer Architecture for Machine Perception*.

MARK, W., GLANVILLE, R., AKELEY, K., AND KILGARD, M. 2003. Cg: a system for programming graphics hardware in a c-like language. In *ACM SIGGRAPH*.

MCDONNELL, R., DOBBYN, S., COLLINS, S., AND O'SULLIVAN, C. 2006. Perceptual evaluation of lod clothing for virtual humans. In *SCA '06: Proceedings of the 2006 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*.

O'SULLIVAN, C., AND DINGLIANA, J. 2001. Collisions and perception. In *ACM Transactions on Graphics*.

O'SULLIVAN, C., DINGLIANA, J., GIANG, T., AND KAISER, M. 2003. Evaluating the visual fidelity of physically based animations. In *ACM Transactions on Graphics, SIGGRAPH 2003*.

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

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

REITSMA, P. S. A., AND POLLARD, N. S. 2003. Perceptual metrics for character animation: sensitivity to errors in ballistic motion. *ACM Transactions on Graphics 22*, 3, 537–542.

REITSMA, P., AND POLLARD, N. 2003. Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. In *ACM Transactions on Graphics*.

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 System*.

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

SMITH, R. Open dynamics engine.

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