Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2009)
H. Prautzsch, A. Schmitt, J. Bender, M. Teschner (Editors)

Space Foosball: Coupling Tangible Interfaces with a Real-time Game Physics Engine

Hyunwoo Bang[1,2] Yunsil Heo[2] Jinwook Kim[3] Young J. Kim[4]

[1]School of Mechanical and Aerospace Engineering, Seoul National University, Korea
[2]Everyware Creative Computing Group, Korea
[3]Imaging Media Research Center, Korea Institute of Science and Technology, Korea
[4]Department of Computer Science & Engineering, Ewha Womans University, Korea

savoy@snu.ac.kr *info@everyware.kr* *jwkim@imrc.kist.re.kr* *kimy@ewha.ac.kr*

Abstract

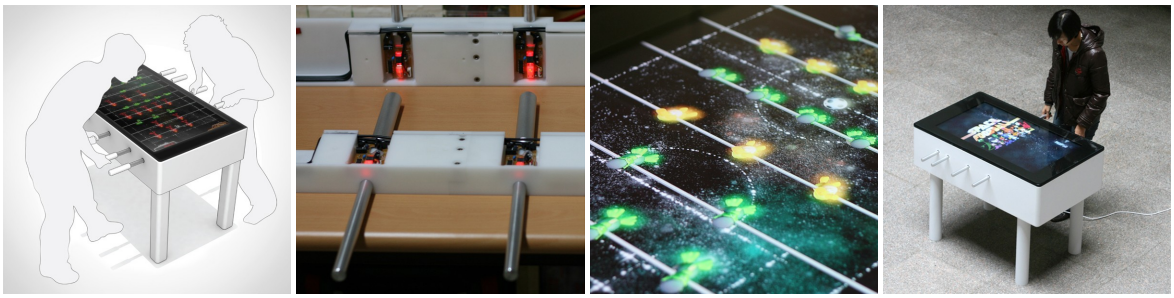
In this paper, we address the problem of designing and implementing low-cost yet effective user interfaces for interactive computer games that heavily use physically-based animation. Due to the nature of a physics-driven gaming setup in our system, we require that the interfaces should mimic the tangibility of real-world interfaces to maximize the playability of the game. Our prototype gaming system, called Space Foosball, is a virtual realization of the real-world foosball in a space-age setting. The biggest challenge to build our system was to design effective and robust interfaces to control the motion of user paddles, which in turn drive the physics simulation of the secondary motion between a soccer ball and the environment, and between a ball and game characters. To meet our tight development budget and schedule, we opted for off-the-shelf optical sensors as a basis of the controlling mechanism. These sensors are low-cost but provided a robust solution to our problem. Another important task to build the Space Foosball was implementing a high-performance game physics engine that suits for simulating the very dynamic foosball environment. To meet this demand, we designed and implemented an in-house physics engine, called Virtual Physics, based on a mathematical formulation of Lie groups. In less than a short period of two months, we successfully built our prototype gaming system which effectively utilizes tangible interfaces while robustly simulating the game physics environment.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Due to a recent commercial success in the gaming industry, providing tangible interfaces to games have become more and more popular. Meanwhile, the explosive development trends on solid state microchip sensor technologies over the past two decades [Dun97, Jan03, Bog03] have made it feasible to mass-produce and integrate these sensor devices into consumer entertainment systems such as Wii from Nintendo [SGR07]. In this paper, we investigate an alternative form of a tangible gaming interface for an interactive video game that heavily relies on physically-based animation. As a proof-of-concept system equipped with tangible interfaces, we choose the game of foosball and morph it into simulated, electronic entertainment.

The foosball is a casual tabletop sports game where more than two people play a miniature-soccer game. The goal of this game is that each player controls soccer-player-like-characters mounted along rotating bars to hit and move a ball into the opponent's scoring area. In our simulated foosball game, called *Space Foosball*, there exist no physical characters, balls or control-bars but only an LCD screen through which players can see 3D virtual characters playing in the world of Space Foosball. Moreover, there are 8 physical rods on each side of the screen with which players can manip-



(a) Game Concept (b) Interfacing HW Modules (c) Game Screen (d) Final System

Figure 1: ***The Space Foosball Game.*** *(a) The Space Foosball is a tangible tabletop implementation of the foosball game. Players can play the virtual foosball game in a simulated soccer-field with intuitive man/machine interfaces. The simulation is performed in a physically-plausible manner by our in-house physics engine, Virtual Physics. (b) Players' manipulation of control paddles is read by optical sensor modules and used as game inputs. (c) A virtual soccer game is rendered on an LCD screen by taking advantage of the programable shaders on GPUs. (d) The Space Foosball system in action.*

ulate virtual soccer players. These rods provide a tangible interface to our gaming system (see Figure 1).

An important gaming characteristics in Space Foosball is that the secondary motion affected by the motion of game characters should mimic what real-world objects would make, *e.g.* a ball bounced off the wall or characters. Thus, a real-time simulation of virtual world is the crucial component of our system. In particular, a fast and reliable performance is strongly desired for the employed physics engine. For this purpose, we have developed our in-house physics engine, called *Virtual Physics*, which is based on an elegant theory on Lie groups and its compact implementation (more details in Section 4).

Another important aspect of our gaming system is designing optical sensors that transform the motion of players into that of control bars, thereby controlling the virtual players in Space Foosball. We have tried various mechanisms to improve the players' controllability of bars and finally ended up with an extremely cheap yet quite flexible solution, that is optical sensors taken out of off-the-shelf PC accessories (mice) and tweaked, and each of these sensors costs less than 10 USD.

Main Contributions In this paper, we highlight our experience in developing an affordable, video gaming system that extensively uses physically-based animation, coupled with tangible user interfaces. As a proof-of-concept system, we introduce the Space Foosball game, and believe that this system is an attractive arcade-style game to both gamers and the game industry due to its modest development budget and high playability. Moreover, we also introduce a Lie group-based formulation of the rigid body dynamics which is a very effective mathematical concept to compactly represent dynamics among solids.

Organization The rest of this paper is organized as follows. In Section 2, we briefly survey the previous work relevant to ours. In Section 3, we show the overall design of our system, and discuss how our physics engine is implemented in Section 4. In Section 5, we show experimental results of the Space Foosball game and conclude the paper in Section 6.

2. Previous Work

In this section, we review the previous work in the areas of designing tangible user interfaces for interactive applications and real-time physics engine.

2.1. Tangible Interfaces

Arguably, one of the most successful tangible interfaces may be a pointing device called a mouse. Through tangible interfaces, users can interact with digital information and contents physically. As noted by [IU97, Ish08], tangible user interfaces empower collaboration, learning, and design by using digital technology and at the same time taking advantage of human abilities to grasp and manipulate physical objects and materials. They propose *Tangible Bits*, a visionary concept to bridge the gap between cyberspace and the physical environment by making digital information(bits) tangible.

Due to its intuitive characteristic, a variety of tangible interfaces have been considered as effective input and output devices in many game developments. Curball [KSFS] is a combination of Curling and Bowling games using tangible balls as input devices, where a tangible ball with embedded sensors is thrown physically by an elderly person. The study reports that tangible interfaces help elderly people enjoying digital entertainment contents more easily. In the game *Weathergods*, tangible interfaces represent bridges to simulate the physical pieces of the contents in a digital tabletop game [BVvdH*07]. Also the classical game *Warham-*

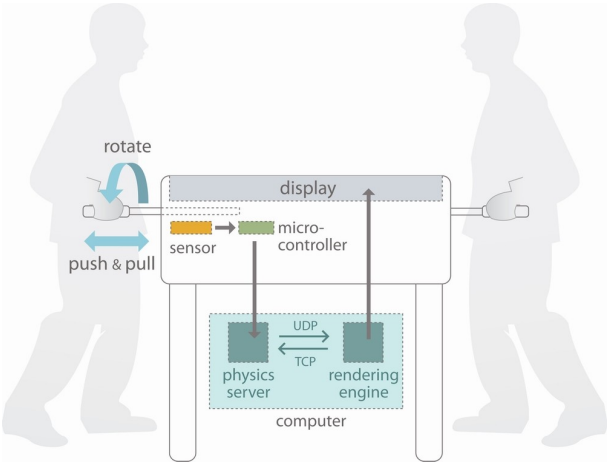


Figure 2: ***The Space Foosball Game System Architecture.*** *Players' manipulation of the paddles are interpreted by optical sensor modules and transmitted to a PC via USB communication. An integrated micro-controller board processes analog signals from the optical sensors and converts it to digital values before sending them to the PC. This digital input is mapped to dynamic internal variables of the Virtual Physics engine. The Virtual Physics engine broadcasts the current configurations of all game objects with UDP communication to which the XNA rendering engine listens so that it can visualize the current state of a scene in the Space Foosball game. Meanwhile, certain game states are sent back to the Virtual Physics engine from the XNA renderer via TCP/IP protocols when needed.*

mer 40,000 has been digitally augmented with virtual information. They try to combine the virtual and the physical world, and the advantages thereof, *i.e.* , the endowment of traditional play environments with virtual information and novel interaction capabilities [HL09]. The Wii Remote, a consumer video game controller from Nintendo's provides a capability of motion sensing through the use of accelerometer and optical sensor technology [SGR07]. Compared to classical gaming controllers, it enables gamers to enjoy the game contents using gesture-based interaction methods.

2.2. Physics Engine

Over the past two decades, tremendous research efforts have been put into the study of simulating the motion of graphical objects using physics laws, in the area of physically-based modeling and simulation [WB01, ESHD05]. Some of these techniques have been relatively matured and adopted for interactive game development. A set of softwares to perform real-time physics simulation is known as physics engine middleware and typical simulation functions included in a physics engine are rigid and articulated-body dynamics, particle dynamics, cloth simulation, vehicle dynamics, etc. Well-known physics engine middleware are Havok, PhysX, Bullet, ODE, endorphin, OpenTissue, etc [Cou09]. Some of these middlewares attempt to offer a diverse set of simulation functions whereas others are more focused on a specific simulation technique; *e.g.* motion synthesis. However, these physics middlewares employ rather proven technologies from the field except for few research prototypes and thus they are not best suited for experimental game designs that this paper aims at. Moreover, the performance of existing physics middlewares is rather slower compared to the state of the art in the fields.

3. System Design

In this section, we explain the overall framework of our Space Foosball gaming system, both in software and hardware aspects.

3.1. Software Development Framework

Initiated as an academy-oriented project with a tight development schedule of less than two months, at the initial stage of development, we started looking into possible options for the game development framework. There are many inexpensive [Gar09] or even open and free game developing tools [Mic09, JMo09, LWJ09] and technologies available on the web that enable independent game developer groups to create full-blown video games in an agile manner. After assessing the plausibility of various game development frameworks, we chose Microsoft's XNA [Mic09] based on its available resources and community-support, stability, functionality and speed. The XNA game engine is a set of tools with a managed runtime environment that can rapidly facilitate computer game development and management. The underlying philosophy of XNA is to free game designers from writing "repetitive boilerplate code" and bring different aspects of game production into a single system.

Right from the beginning, we intended our system to be highly versatile and extendible so that we could add future gaming components seamlessly. Thus, we wanted to make the components of our system as modular as possible, and separated the main game engine part based on Microsoft XNA, including the main game logics and rendering routines, from the physics simulation part based on Virtual Physics, particularly because Microsoft XNA supports C# and Microsoft DirectX while our game physics engine is written in C++. In the final system, while the physics engine simulates the virtual foosball environment and broadcasts the simulated results of game objects by using interprocess communication protocols at 1KHz update rates, the XNA-based main game engine takes these data and renders the game objects as such in a synchronized manner. Figure 2 shows this software architecture.

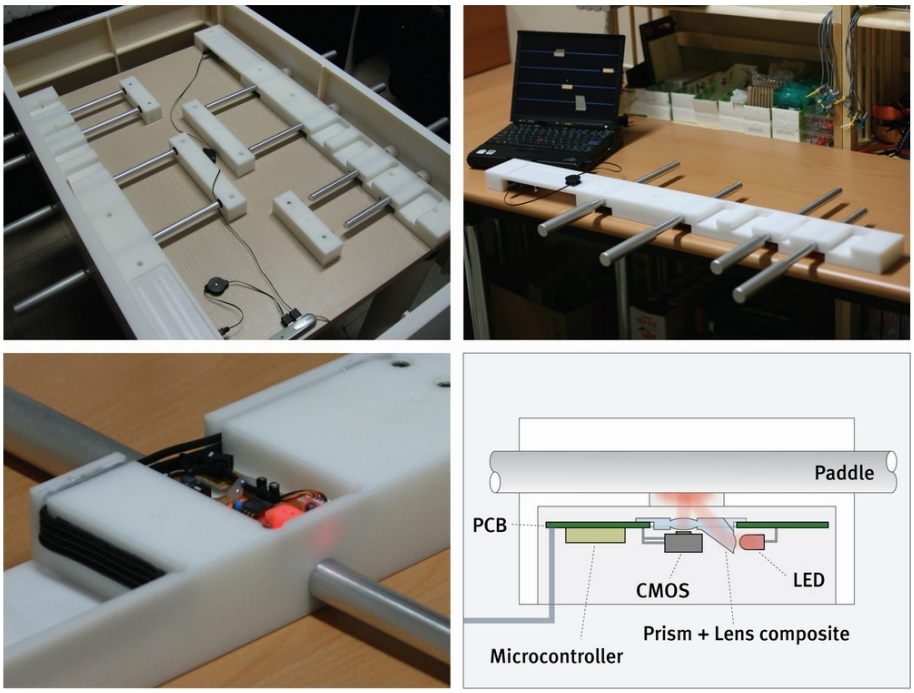


Figure 3: ***The Sensor Modules.*** *An integrated CMOS chip with an LED lightbulb from a PS2 mouse have been modified to work as a sensor module. With an optics composite of prism and lenses, the sensor module detects the relative motion of the controller paddle. As the sensing is non-contact type, it is robust to resist the fierce manipulation of excited players.*

3.2. Hardware Platform

Our system can intensify player's gaming experiences by keeping the tangibility of the real foosball game. So the most challenging task during the development of the Space Foosball system was designing and physically building a robust, tangible interface with a modest budget yet in a short amount of time. To mimic the motion of a paddle of a real foosball table, we built a sensing module that could simultaneously measure the rotational and coaxial motion of a cylindrical rod.

Conventional rotary motion sensors usually require an encoder mounted on the moving body and a decoder on the stationary one to sense their relative motions. But, in our case, as we need to move the paddle along its axis, the rotary decoder should also follow the paddle while being strictly aligned with the rotary encoder. This will introduce some mechanical complexity on the sensing module as the decoder needs to keep its posture. So we had to design a non-contacting sensing mechanism for both the rotational and axial motions of the paddle that do not require any coupling between the paddle and the sensor module.

We tried various sensing methods including two sets of rotary encoders and decoders linked together with a friction ball (sensor module taken out from an old ball-mouse), capacitive sensors by the Hall effects [RSPM01] and optical sensors with two-dimensional imaging modules. After repetitive trials and errors, we finally ended up with an optical sensor module, taken out from Logitech's optical mouse [Log09] for its robustness, easiness of control (PS2 interface), fast response, sensing resolution (up to 25 microns) and low cost. Optical mice work by using an opto-electronic sensor to take successive pictures of the surface on which the mice operate. Powerful special-purpose image-processing chips are embedded in the mouse itself which enabled the mouse to detect relative motion on a wide variety of surfaces, translating the relative motion of the mouse and eliminating the need for a special mouse-pad. Optical mice illuminate a surface that they track over, using an LED or a laser diode. Changes between one frame and the next are processed by an image processor embedded in the chip and translated into a movement on the two axes using an optical flow estimation algorithm [Hor81, BB95]. Figure 3 shows the internal structure of our sensing modules.

As we can see from Figure 3, the rotational and axial motion of the cylindrical rod is translated into an x and y motion of the mouse by the optical sensor and is transmitted to the micro-controller [Atm09]. The micro-controller unit gathers sensing information from all 8 paddles and packs them into a communication packet, then sends to the PC via USB communication. The developed sensing modules had been integrated into a PC with an Intel Core 2 Quad CPU at 2.4 GHz and 2GB system memory with NVidia GeForce 9500GT,

and a 42 inches LCD panel. All these hardwares were assembled in a wooden frame which later has been covered with ABS plastic housing and a heat-treated top glass panel (see Figure 1). The ABS plastic housing had been fabricated by a numerically-controlled mock-up prototyping machine.

4. Physics Engine

Simulating the dynamics of a rigid body system implies computing acceleration of rigid bodies given external forces applied to the bodies, or vice versa. Hence the representation of dynamic states and properties including acceleration, force, and mass is crucial to derive the underlying, governing equations and to develop efficient solvers for them. In the case of 3-dimensional problems, it is common to consider linear and angular quantities of the rigid body such as a position and an orientation as a pair of 3-dimensional vectors, which results in the corresponding pairs of equations, known as the Newton-Euler equation.

However recent advances in computational dynamics based on modern differential kinematic geometry provide more efficient treatments of the linear and angular components of dynamics states in a unified manner. In this section, we introduce the basic concepts of the underlying theory, which is very little known to the game physics community, and explain how it can open new avenues to computing the kinematics and dynamics of rigid body systems efficiently as well as implementing them in an object-oriented manner. Our in-house physics engine, Virtual Physics, is written based on these theories.

4.1. Geometric Formulation of the Rigid Body Dynamics

Newton's first law of physics for a particle system is

$$f = m\dot{v}, \tag{1}$$

where $f, v \in \mathbb{R}^3$ and $m \in \mathbb{R}$ represent force, velocity and mass of the particle, respectively. In the case of a rigid body, the relationship between a torque and an angular acceleration also needs to be accounted for as:

$$\tau = I\dot{\omega} + \omega \times I\omega, \tag{2}$$

where $\tau, \omega \in \mathbb{R}^3$ and $I \in \mathbb{R}^{3\times 3}$ represent the torque, the angular velocity and the inertia tensor, respectively. The concepts from modern differential geometry unify the above equations in a more compact way as follows:

$$F = J\dot{V} + ad^*_V JV, \tag{3}$$

where $F \in dse(3)$ and $V \in se(3)$ represent the generalized force and velocity of a rigid body. The generalized velocity combines the linear and angular velocities into a unified quantity. Similarly, the generalized force represents the linear force and the torque simultaneously. The generalized inertia tensor J relates the generalized acceleration and the generalized force in Eq.(3). The dual adjoint mapping ad^* plays a similar operation over the generalized velocity and force like the cross product between 3-dimensional vectors. The details on mathematical backgrounds of the theory are beyond the scope of this paper, and the readers are recommended to refer to the work by [Spi78, MLS94, PBP95, PP99].

Eq.(3) can be also extended to a recursive forward dynamics of articulated rigid body system. We refer to the appendix A for a description of the algorithm.

4.2. Dynamics Computations

Since the generalized velocity, for example, unifies linear and angular velocities, it can be represented as a single 6-dimensional vector by concatenating two 3-dimensional vectors, each representing the linear and angular velocities. Similarly, the generalized inertia tensor can be represented as a 6×6 positive definite matrix and the dual adjoint operation can be also represented as a 6×6 matrix. Therefore Eq.(3) can be regarded as a matrix-vector equation. Note that this interpretation is similar to the spatial dynamics by [Fea87].

Even though the formulations based on vector and matrix calculations can still utilize the results from vector and matrix calculus, the formulations related to kinematics and dynamics can be represented in its own language, particularly from a modern differential geometric viewpoint. The formulations based on such a geometric language are not only elegant but also compact and thus computationally efficient. For example, the dual adjoint operation is defined as:

$$ad^*_V F = (\tau \times \omega + f \times v\ ,\ f \times \omega), \tag{4}$$

where $w, v, \tau, f \in \mathbb{R}^3$ and $V = (w, v), F = (\tau, f)$ represent the generalized velocity and force.

Note that the same operation of Eq.(4) in a matrix form is represented as a 6×6 matrix of:

$$ad^*_V F = \begin{bmatrix} -[w] & -[v] \\ 0 & -[w] \end{bmatrix} \begin{bmatrix} \tau \\ f \end{bmatrix}, \tag{5}$$

where $[w]$ and $[v]$ are 3×3 skew symmetric matrices from $w, v \in \mathbb{R}^3$. One can clearly see that computing the matrix-based equation in Eq.(5) requires redundant operations of building skew symmetric matrices and a multiplication with a zero matrix whereas the one using Eq.(4) does not.

4.3. Data Structures

Motivated by the results from the previous section, we design several, efficient geometric data structures to implement the geometric formulations of the kinematics and dynamics of a rigid body system.

First we construct an `SE3` class to represent the elements

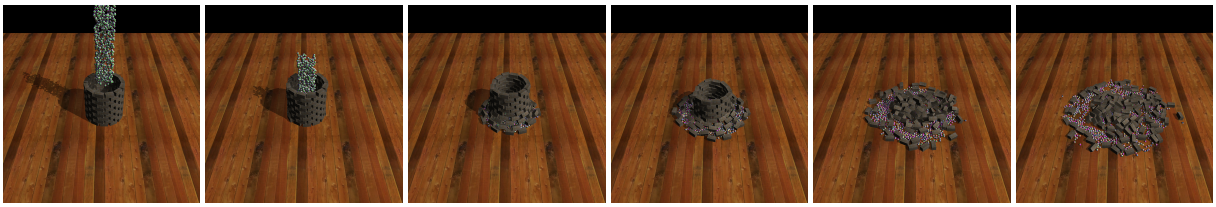


Figure 4: ***Realtime simulation of 1120 rigid bodies.***

of the special Euclidean group $SE(3)$. The `SE3` class consists of variables to describe the position and orientation of a homogeneous transformation matrix and its induced operators. For the operators on `SE3` class, it is sufficient to consider only multiplication and substitution operators since there is no natural definition of addition and subtraction rules on the Lie group $SE(3)$. Hence it is not necessary to overload the addition or subtraction operators. Also there are several functions related to $SE(3)$ class such as inversion, exponential and logarithmic functions, that need to be implemented.

Compared to the 4×4 matrix representation for homogeneous transformation, the proposed `SE3` class consumes less memory (i.e. 12 floating point values) and does not require irrelevant operations, for instance, such as an addition operation, which might be otherwise to be defined if a general matrix representation has been employed. Moreover the special Euclidean group is not a linear vector space. Hence representing it as a matrix without any additional information may result in ambiguous, sometimes incorrect descriptions, and will eventually degrade a computational efficiency.

To implement a generalized velocity, we design `se3` class as an array of six scalars because $se(3)$ is the Lie algebra of $SE(3)$ and therefore is a vector space. A `dse3` class is constructed for a generalized force similarly.

Finally we define `Inertia` and `AInertia` classes to represent a generalized inertia tensor and an articulated inertia tensor. A naive representation of the generalized inertia tensor will be to use a 6×6 positive definite matrix. However, from a computational perspective, it is inefficient to represent the generalized inertia tensor as an arbitrary 6×6 matrix, since the number of independent elements of a generalized inertia tensor is only ten: six elements corresponding to the inertial moments, three elements corresponding to the offset vector, and the remaining element corresponding to the mass. We therefore design `Inertia` class as a triple of these elements. A product operator between `Inertia` class and an `se3` class element returns a `dse3` class element that describes the generalized momentum, equivalent to the product of the generalized inertia tensor and the generalized velocity. The `Inertia` class supports coordinate transformations via a method function `Xform` which transforms the generalized inertia tensor according to a change in the reference frame.

Unfortunately `Inertia` lacks in a memory allocation to be applicable to the well-known, articulated inertia method by [Fea87]. The structure of an articulated inertia tensor should be represented as a 6×6 symmetric matrix. Hence we design the `AInertia` class to be a 6×6 symmetric matrix, and provide a type casting operator that casts an `Inertia` class instance to an `AInertia` class instance. For the full list of defined operators, we refer to the appendix B.

	Computational Operation	Math Operation
1	`dse3` ← `AInertia` * `se3`	$\eta = \hat{J}_i S_i$
2	`AInertia` ← `dse3` * `dse3`	$\Gamma = \eta \otimes \eta$
3	`scalar` ← `se3` * `dse3`	$\Omega = \langle S_i, \eta \rangle$
4	`AInertia` ← `scalar` * `AInertia`	$\Theta = \Gamma / \Omega$
5	`AInertia` ← `AInertia-AInertia`	$\Phi = \hat{J}_i - \Theta$
6	`AInertia` ← `AInertia.Xform(SE3)`	$\Psi = Ad^{*}_{f^{-1}_{i-1,i}} \Phi Ad_{f^{-1}_{i-1,i}}$
7	`AInertia` ← `Inertia` + `AInertia`	$\hat{J}_{i-1} = J_{i-1} + \Psi$

Table 1: ***Operation Flow to Compute Eq.(6).***

```
SE3 f[n]; Inertia I[n]; AIneria J[n];
se3 S[n]; dse3 eta;

eta = J[i] * S[i];
J[i-1] = I[i-1]
+(J[i]-(eta*eta)/(S[i]*eta)).Xform(Inv(f[i]));
```

Table 2: ***Code Snippet to Implement Eq.(6).***

4.4. Examples and Comparisons

Consider the following equation corresponding to one step in the articulated inertia formulation of recursive dynamics in [Kim02]:

$$\hat{J}_{i-1} = J_{i-1} + Ad^{*}_{f^{-1}_{i-1,i}} \left(\hat{J}_i - \frac{\hat{J}_i S_i \otimes \hat{J}_i S_i}{\langle S_i, \hat{J}_i S_i \rangle} \right) Ad_{f^{-1}_{i-1,i}}. \qquad (6)$$

Table 1 shows an equivalent, step by step operation flow of Eq.(6). One possible implementation of Eq.(6) is given in Table 2. Note that the symbols Γ, Ω, Θ and Ψ used in the mathematical operations field can be implicitly generated in a computer language implementation, even though they are not explicitly declared.

Figure 5: ***Actual Game Play Shot.***

Now, the following is an equation that performs a similar operation like Eq.(6), but using the spatial operators of Featherstone's [Fea87, Mir96].

$$I_{i-1}^A = I_{i-1} + {}_iX_{i-1}^T \left(I_i^A - \frac{I_i^A S_i S_i^T I_i^A}{S_i^T I_i^A S_i} \right) {}_iX_{i-1}, \qquad (7)$$

where I_i and I_i^A are an inertia tensor and a spatial articulated inertia, corresponding to the general and articulated inertia tensor. Also S_i is an element of spatial vector corresponding to $se(3)$ and ${}_iX_{i-1}$ is a 6 by 6 spatial transformation matrix corresponding to the adjoint mapping represented in a matrix form. At a glimpse, Eq.(7) looks similar to Eq.(6). However an important difference between the two equations is that Eq.(7) is derived in a matrix form while our equation in Eq.(6) is in a purely geometric form and thus it better captures the geometric nature of the underlying dynamics equation.

Table 3 shows an operation count comparison between the geometric computation and the matrix-based computation. As shown in the table, the geometric computations are more efficient than the general matrix-based approach. To evaluate Eq.(6), the matrix-based computation requires 643 multiplications and 485 additions while the geometric computation needs only 360 multiplications and 269 additions.

	geometric computation		matrix based computation	
	×	+	×	+
`Ad(SE3, se3)`	24	18	63	48
`ad(se3, se3)`	18	12	36	30
`Inertia * se3`	24	18	36	30
`Inertia.Xform()`	101	66	459	378
`AInertia.Xform()`	294	210	459	378

Table 3: ***Operation Count Comparison.***

4.5. The Virtual Physics Engine

Based on the geometric formulations and computations described in the previous sections, we developed a prototype physics engine, *Virtual Physics* which aims at solving rigid and articulated rigid body dynamics of complex scenes in real-time. Key features of Virtual Physics include an efficient mathematical formulation based upon Lie group theory, a flexible software architecture allowing developers to define arbitrary types of joints and constraints, efficient collision detection and resolution algorithms, robust numerical integration of governing differential equations. Figure 4 shows an example of a stable simulation of stacking and collision among 1120 rigid bodies in real-time(>60fps). This test runs on a single-threaded Intel Core 2 Duo 2.4GHz CPU.

5. Results

One of the merits of providing tangible interfaces to a virtual environment is that player's gaming experience has no more to be restricted in the real world even the players still interact with physical (real) objects. While developing a prototype game, we tried to maximize this merit and chose the cosmic space as a background scene, a symbolic representation of the extension of the physical boundary of the LCD screen to the infinity. Thus we appropriately dubbed the game as Space Foosball. As the soccer field is a virtual one, players

can also choose their virtual team characters based on their specialties. For example, a tall and thin character is good at offense but poor at defense while small and fat character is poor at offense but excellent at defense on the contrary. To maximize visual and sound effects, we intentionally exaggerated the bouncing motion of soccer balls when they hit characters or the ground, and add realistic sounds and colorful particle effects. We also scaled up the ball size when it goes up in the air to give players a sense of height cues. The scored goals are automatically counted and displayed on the screen after a goal ceremony splash screen.

5.1. Integrating Tangible Interfaces into Physics Engines

Figure 5 shows several screen shots from actual game play. The control paddles are also used as natural, input devices during the initial, logo screen and the character selection screen. Players can start the game by pushing the paddle and choose between different characters by spinning it. To synchronize physical manipulation of the paddle with a virtual motion of the game characters, we had to calibrate the sensor modules to set proper weighting functions while converting raw sensor signals into input values for the physics simulation engine. By curve-fitting the actual values of rotational and axial motions with raw sensor signals, we find precise coefficients for mapping the input values to configurations of the paddles. The sensor could sense a linear velocity of up to 50 m/s with 400 samples/cm resolution. The final system has gone under a public beta test with questionnaires collected after game play.

A player-character was modeled as a set of square boxes in the Virtual Physics world. Hinged at shoulder positions with angular joints, four character consisted in one row aligned at the screen position of the physical controller paddle. The long pipe connecting these four character was omitted in the physics simulation in order not to annoy users but to direct them to focus only on the ball and the game characters (i.e. the game play itself). Physics variables concerning dimensions, mass, friction and gravitation were all tuned to maximize playability not reality.

6. Conclusion

In this paper, we have presented how we design and implement our prototype game system, Space Foosball. The Space Foosball has intuitive tangible interfaces driving a physics simulation that governs the motion of objects in the scene. Our system was built in less than two months with a total budget of less than two thousand USD.

One more functionality we want to add to the next version of our system is a haptic [PT02] response mechanism. For example, players can feel and measure the power of a kick when the ball hits the game character by variable vibration of the paddles. Moreover, we want to add other characters undergoing different types of physics simulation such as articulated motion or deformation. This will increase the complexity of physics simulation, but will add a possibility of extended playability as well as more fun to the game.

Acknowledgements. This work was supported in part by the IT R&D program of MKE/MCST/IITA (2008-F-033-02, Development of Real-time Physics Simulation Engine for e-Entertainment) and the KRF grant (KRF-2007-331-D00400). The authors thank the anonymous reviewers for helpful suggestions to improve the paper.

Appendix A

Consider n numbers of rigid bodies articulated by arbitrary joints. Let V_i be the generalized velocity of body i, F_i the generalized force transmitted from body $i-1$ to body i through joint i. Also, let $f_{i-1,i} = M_i e^{S_i q_i}$ denote the position and orientation of the body i frame relative to the body $i-1$ frame with $M_i \in SE(3), S_i \in se(3)$. Further J_i and $\hat{J}_i$ are the generalized inertia tensor and the articulated inertia tensor of body i. Then recursive forward dynamics algorithm of the articulated rigid body system is as follows:

- Initialization :

$$V_0,\ F_{n+1},\ \tau,\ q,\ \dot{q}$$

- Forward recursion : for i = 1 to n

$$\begin{aligned} f_{i-1,i} &= M_i e^{S_i q_i} \\ V_i &= Ad_{f_{i-1,i}^{-1}} V_{i-1} + S_i \dot{q}_i \\ c_i &= ad_{V_i} S_i \dot{q}_i \end{aligned}$$

- Initialization :

$$\begin{aligned} \hat{J}_n &= J_n \\ b_n &= -ad^*_{V_n} J_n V_n + Ad^*_{f_{n,n+1}^{-1}} F_{n+1} \\ \Omega_n &= \langle S_n, \hat{J}_n S_n \rangle \end{aligned}$$

- Backward recursion : for i = n to 2

$$\begin{aligned} \hat{J}_{i-1} &= J_{i-1} + Ad^*_{f_{i-1,i}^{-1}} \Big(\hat{J}_i - \frac{\hat{J}_i S_i \otimes \hat{J}_i S_i}{\Omega_i} \Big) Ad_{f_{i-1,i}^{-1}} \\ b_{i-1} &= -ad^*_{V_{i-1}} J_{i-1} V_{i-1} + Ad^*_{f_{i-1,i}^{-1}} (\hat{J}_i c_i + b_i) \\ &\quad + \frac{\tau_i - \langle S_i, \hat{J}_i c_i + b_i \rangle}{\Omega_i} Ad^*_{f_{i-1,i}^{-1}} \hat{J}_i S_i \\ \Omega_{i-1} &= \langle S_{i-1}, \hat{J}_{i-1} S_{i-1} \rangle \end{aligned}$$

- Forward recursion : for i = 1 to n

$$\begin{aligned} \ddot{q}_i &= \frac{1}{\Omega_i} \Big(\tau_i - \langle S_i, \hat{J}_i (Ad_{f_{i-1,i}^{-1}} \dot{V}_{i-1} + c_i) + b_i \rangle \Big) \\ \dot{V}_i &= Ad_{f_{i-1,i}^{-1}} \dot{V}_{i-1} + S_i \ddot{q}_i + ad_{V_i} S_i \dot{q}_i \end{aligned}$$

Appendix B

SE3 class operations and functions	
Substitution operation	SE3 ← SE3
Multiplication operation	SE3 ← SE3 * SE3
Inverse mapping	SE3 ← Inv(SE3)
Exponential mapping	SE3 ← Exp(se3)
Log mapping	se3 ← Log(SE3)
Adjoint mapping	se3 ← Ad(SE3, se3)
	dse3 ← dAd(SE3, dse3)

se3 class operations and functions	
Substitution operation	se3 ← se3
Addition operation	se3 ← se3 + se3
Multiplication operation	se3 ← scalar * se3
Reciprocal product	scalar ← se3 * dse3
Adjoint mapping	se3 ← ad(se3, se3)

dse3 class operations and functions	
Substitution operation	dse3 ← dse3
Addition operation	dse3 ← dse3 + dse3
Multiplication operation	dse3 ← scalar * dse3
Reciprocal product	scalar ← dse3 * se3
Adjoint mapping	dse3 ← dad(se3, dse3)

Inertia class operations and functions	
Substitution operation	Inertia ← Inertia
Multiplication operation	dse3 ← Inertia * se3
Transformation	Inertia ← Inertia.Xform(SE3)

AInertia class operations and functions	
Substitution operation	AInertia ← AInertia
	AInertia ← Inertia
Addition operation	AInertia ← AInertia + AInertia
Multiplication operation	AInertia ← scalar * AInertia
	dse3 ← AInertia * se3
Kronecker Product	AInertia ← dse3 * dse3
Transformation	AInertia ← AInertia.Xform(SE3)

References

[Atm09] ATMEL: Atmel micro-controller, 2009. http://www.atmel.com/.

[BB95] BEAUCHEMIN S. S., BARRON J. L.: The computation of optical flow. *ACM Computing Surveys* (1995).

[Bog03] BOGUE R.: MEMS sensors: past, present and future. *Sensor Review* (2003).

[BVvdH*07] BAKKER S., VORSTENBOSCH D., VAN DEN HOVEN E., HOLLEMANS G., BERGMAN T.: Weathergods: tangible interaction in a digital tabletop game. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction* (New York, NY, USA, 2007), ACM, pp. 151–152.

[Cou09] COUMANS E.: *Physics Simulation Forum*, 2009. http://www.bulletphysics.com.

[Dun97] DUNN J. F.: A new digital camera startup busts price/performance standards with CMOS sensor. *Advanced Imaging* (1997).

[ESHD05] ERLEBEN K., SPORRING J., HENRIKSEN K., DOHLMANN H.: *Physics-based animation*. Charles River Media, 2005.

[Fea87] FEATHERSTONE R.: *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.

[Gar09] GARAGEGAMES: Torque, 2009. http://www.garagegames.com/.

[HL09] HINSKE S., LANGHEINRICH M.: W41k: Digitally augmenting traditional game environments. In *Proceedings of Tangible and Embedded Interaction 2009, Cambridge, UK* (Feb. 2009).

[Hor81] HORN B. K. P.: Determining optical flow. *Artificial Intelligence* (1981).

[Ish08] ISHII H.: Tangible bits: beyond pixels. In *Proceedings of the 2nd international conference on Tangible and embedded interaction* (2008), ACM New York, NY, USA.

[IU97] ISHII H., ULLMER B.: Tangible Bits: Towards Seamless Interfaces between People. *Bits and Atoms, CHI 97* (1997), 234–241.

[Jan03] JANESICK J.: Development and applications of high-performance CCD and CMOS imaging arrays. *Annual Review of Nuclear and Particle Science* (2003).

[JMo09] JMONKEY: JMonkey engine, 2009. http://www.jmonkeyengine.com/.

[Kim02] KIM J.: *Geometric Algorithms and Data Structures for Physics based Simulation*. PhD thesis, Seoul National University, 2002.

[KSFS] KERN D., STRINGER M., FITZPATRICK G., SCHMIDT A.: Curball–a prototype tangible game for inter-generational play. In *Proceedings of WETICE*, vol. 6.

[Log09] LOGITECH: PS2 mouse, 2009. http://www.logitech.com/index.cfm/mice_pointers/mice/devices/344&cl=us,en.

[LWJ09] LWJGL: Light Weight Java Game Library, 2009. http://lwjgl.org/.

[Mic09] MICROSOFT: XNA, 2009. http://msdn.microsoft.com/en-us/xna/default.aspx.

[Mir96] MIRTICH B.: *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, UNIVERSITY of CALIFORNIA, 1996.

[MLS94] MURRAY R., LI Z., SASTRY S.: *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[PBP95] PARK F., BOBROW J., PLOEN S.: A Lie Group Formulation of Robot Dynamics. *The International Journal of Robotics Research 14*, 6 (1995), 609.

[PP99] PLOEN S., PARK F.: Coordinate-invariant algorithms for robot dynamics. *Robotics and Automation, IEEE Transactions on 15*, 6 (1999), 1130–1135.

[PT02] PIEKARSKI W., THOMAS B.: ARQuake:the outdoor augmented reality gaming system. *Communications of the ACM* (2002).

[RSPM01] R. S. POPOVIC Z. R., MANIC D.: Integrated Hall-effect magnetic sensors. *Sensors and Actuators A: Physical* (2001).

[SGR07] SHIRAI A., GESLIN E., RICHIR S.: WiiMedia: motion analysis methods and applications using a consumer video game

controller. In *Proceedings of the 2007 ACM SIGGRAPH symposium on Video games* (2007), ACM Press New York, NY, USA, pp. 133–140.

[Spi78] SPIVAK M.: A comprehensive introduction to differential geometry. *Bull. Amer. Math. Soc. 84 (1978), 27-32. DOI: 10.1090/S0002-9904-1978-14399-7 PII: S 2*, 9904 (1978), 14399–7.

[WB01] WITKIN A., BARAFF D.: Physically Based Modeling. *ACM SIGGRAPH Course Notes* (2001).