Navigation is an ubiquitous user interaction task in Virtual Environments (VEs). Travel is the motor component of navigation and comprehends control of both locomotion and orientation of viewpoint within a VE. Commonly, travel is done via a metaphor that maps user input to changes in the orientation and position parameters. In this work, we present a distinct approach, called Physicam, in which the camera is represented by a rigid body with physical properties such as mass, tensor of inertia, and coefficient of friction; travel happens when forces and/or torques are exerted upon the rigid body, in response to user input. The dynamics simulation involved in this new approach is entirely handled by a physics engine, while a graphics engine manages the rendering. Our aim is twofold: to provide a camera whose travel behavior is easy to program, and; to engage users of a desktop VE in a richer interaction experience capable of raising their level of immersion and, hopefully, improving the navigation process. To demonstrate our proposal, we have built three VEs. Each of these VEs is capable of simulating physical properties and requires users to complete a specific navigation task, namely search, maneuvering, and exploration. We did an informal user study with a group of volunteers who interacted with all three VEs, and used their subjective feedback to evaluate our proposal.

1. Introduction

Game engine technology has increasingly been used as basis for desktop virtual environments. Many virtual reality developers have either designed or repurposed game engines to support the creation of their own VR applications [8, 9].

Depending on the type of simulation involved in a VR application, game engines can be an adequate tool, capable of saving considerable developing time [8]. A typical example of well-suited application is simulations that require keeping track in a consistent manner of users, objects, and locations in complex three-dimensional (3D) worlds.

The visual realism afforded by game engines is all the more valuable in desktop virtual reality systems, since it is the main responsible for the sense of immersion reported by their users [12]. Additionally, the users of engine-driven VR systems may have their virtual experience further enhanced when the simulation presents convincing physically-based behavior. We believe that dynamics simulation, which, say, let users feel as if objects have mass and momentum, or suffer from rotational effects, deepens their engagement, thereby improving the sense of presence experienced [4]. This extra level of realism is available through the use of a set of library modules that simulates Newtonian physics, known as physics engines.

This work introduces a camera model, called Physicam, based on rigid body dynamics. The camera is associated with a rigid body that has specific properties such as mass, linear and angular momenta, and friction coefficients. The user input generates forces and torques that acts upon the camera’s rigid body, which, in turn, moves the camera’s viewpoint throughout
a given environment. We rely on game engine technology to perform both the 3D rendering and the dynamics simulation behind camera control.

Our work has three objectives: i) to confirm by direct experience that game engines can be adapted for VR purposes; ii) to design a camera model that supports a set of travel behaviors, such as walking or wheeled vehicle, and, at the same time, is ‘easy’ to program – i.e. it only requires the tuning of physical parameters to enable the camera to move differently; and iii) to harness dynamics simulation in a way that engages users of a desktop VE in a richer travel experience, possibly raising their sense of presence.

Next we present some of the related work regarding travel control, followed by a background on navigation and physics engine. In the subsequent section, we outline our proposed solution, followed by the description of the experiments with the Physicam. In the final sections we discuss the results, and present the conclusion.

2. Related work

Most of the work related to VR that involves the use of physics has been focusing mainly on how to increase the overall realism of the simulation experience [7]. Little work has specifically employed physics to improve or control navigation, or enhance the sense of presence.

One of the few exceptions is the early work done by Turner et al. [19] on the use of a physically based camera to control travel. The authors proposed an abstract physical model to control the behavior of a virtual constrained camera that reacted to the force data from 3D input devices. Their camera behavior was determined by parameters such as mass, moment of inertia, and friction coefficients, which afforded different types of camera control metaphors. To get a proper camera behavior, firstly it is necessary to interactively fine-tune the set of physical parameters, otherwise the camera could, for instance, move too fast and miss the target destination because of say, a poorly set value for viscous friction. This requirement makes the camera control a difficult task, unless the user has some background on kinematics and rigid body dynamics.

Following a different approach, Wan et al. [20] tried to support travelling by associating artificial force fields to a VE. Their objective was to avoid collision between the camera and the environment. A drawback of this approach is that the repulsion applied to the camera might, sometimes, seen unnatural to the user. This side-effect may break immersion, since there is no visual cue that justify the user’s viewpoint being pushed away. Nonetheless, the authors reported good subjective results when compared to unconstrained travel and travel with collision detection enabled. A similar approach was proposed by Xiao [21] in which a distance field was used to guide a centered flight path of a physically based camera designed to aid in a virtual colonoscopy procedure.

In the video-game context, a mass-spring-damper system was utilized to smooth the motion of an exocentric-based virtual camera [17]. A similar idea was later applied to control a flying egocentric camera in 3D geovirtual environments [2]. Also, a game engine helped improving navigation with spatial analysis purposes in outdoor environments within the context of a geographic information system [14].

In general terms, the lessons learned from this review were: i) there have been considerably little use of dynamics to model virtual camera control, perhaps because of the inherent cost associated with the required physics calculation; ii) the concept of a physically based camera model is interesting as long as the control of such a camera is done through metaphors that hide its physical parameters from the end user; and iii) an increasingly number of scientific applications in data visualization, virtual and augmented reality, for example, have been using game engine technology as the basis for their VEs.

3. Background

In this section we briefly describe the concepts of navigation in 3D environments, and game engines, which underpin our proposal, detailed in the next section.

3.1. Navigation in 3D environments

We have chosen to focus on navigation because, in terms of usability of virtual environments, it is considered a major component and one of the most dominant user action in 3D environments. Navigation often is defined by its components, goal, and travel
metaphor [6]. A navigation technique can be further characterized by whether constraints are applied to motion, and the existence of mechanisms to aid travel [3].

Navigation is usually studied as a process composed of cognitive and virtual locomotion aspects, respectively known as wayfinding and travelling [15]. Wayfinding encompasses the determination of the user’s present location and viewpoint within the ve (sometimes called orientation), as well as choosing where the user wants to go and defining a path through the environment to the desired destination. Travel entails the mechanics of the actual movement of the viewpoint from one location to another in response to the user input.

Navigation is an ubiquitous task in many virtual environments and is usually classified in three categories, according to its goal [1]: maneuvering, a common task found in computer-aided design applications, when designers want to examine a modelled object; search, characterized by users wanting to move to a specific target location, as it is the case, for instance, in fire escape simulations, and; exploration, related to the case where users explore a ve without a pre-defined destination, as in a virtual museum or historic building walkthrough.

A travel metaphor is an abstraction that represents a mapping between user input (received via interaction devices) and operations that orient and move the viewpoint from one location to another [15]. For instance, a common metaphor for desktop ve is the virtual control of viewpoint, in which a virtual device is used to change viewpoint; often such a virtual device associates the awsd keys to viewpoint translation and mouse displacements to viewpoint orientation.

Regardless of the travel metaphor used in a virtual environment, its design shares common goals: it should be natural and intuitive, and prevent users from getting lost. By natural and intuitive we mean that user interfaces should aim at abstracting the set of travel commands available to the users, while, at the same time, they should allow viewpoint modification, hopefully in a way that blends in the application context [6]. The other mentioned goal is to avoid or minimize the problem of user disorientation. Several approaches exist that attempt to reduce the user ability to become lost during navigation, such as the inclusion of artificial landmarks [10], use of maps and compass [13], and guiding agents [3], just to name a few. Less invasive solutions, such as constrained travelling, collision detection, and speed of travel control, are more effective when applied in conjunction [1].

3.2. Computer game engines

Game engines can be thought of as a middleware formed by a collection of modules that abstract most of the lowlevel implementation details, and aggregate several functionalities required in a 3d real-time computer game. Their main objective is to reduce the development effort via code reuse. Recently, scientific community has become aware of the game engine potential for serious applications [18, 8].

The benefits of using an engine are various. For instance, engines can speed up the development process, provide 3d rendering adapted to a multi-user network support with content synchronization, make use of high-fidelity hardware accelerated graphics, and support platform-independence and decoupling from third-party application programming interfaces. Trenholme and Smith also add that game engines are robust and extensively tested (both in terms of usability and performance), run on off-the-shelf systems, are easily disseminated via on-line communities, and enable content extension by means of editing/supporting tools [18].

In this work we were interested in reusing a graphics engine for presenting 3d visually realist worlds, and a physics engine for both the simulation of the ve’s dynamic behavior and the control of camera motion in such a ve. A graphics engine is a more common concept to ve developers as an alternative to ve development toolkits [16]. A physics engine, less frequently found in ve development, can be defined as a group of library code that provides a way of abstracting believable physical behavior, such as rigid body dynamics, cloth or ragdoll simulation. Physics engines enable a programmer to specify, for instance, ballistic behavior in a way that it is independent of the type of projectile being fired [11].

A physics engine works in the following manner. The content designer creates rigid bodies, and sets initial values for their physical
parameters, such as position and orientation, mass, velocity (both linear and angular), and inertia tensor. Later, each rigid body is associated to polygonal objects, in an operation that resembles graphics material (e.g. textures) being assigned to objects.

During the application execution, the rigid bodies state, determined by individual position, orientation, velocity, acceleration and forces being applied upon, serves as input to the integrator — an engine’s component that numerically integrates differential equations corresponding to physical laws. Next, the integrator yields new position/orientation to update the current rigid bodies state. These newly calculated values, in turn, are passed downstream to modules that deal with contacts (generation, resting contact, and friction), and collision processing (i.e. detection, interpenetration, and response). Figure 1 illustrates these steps.

Figure 1. Schematics of the dataflow through a physics engine. The application runs the loop composed of steps 1 to 4 prior to rendering a frame.

In addition to the advantages of using a game engine, described earlier in this section, we may add that the current physics engines (commercial or free) are capable of supporting a physically-based general purpose realistic simulation. This means that it is possible to achieve the same level of realism in a simulation, be it based on rigid body or, say, fluid dynamics. Indeed, the development of complex physical simulation without the aid of a physics engine can be a rather tedious and complex process, whose end result might be imprecise/unstable and hard to maintain or expand. Finally, developers can treat the use of a physics engine as a ‘black box’, i.e. they do not need to know its inner workings (specially the dreaded math behind it); the knowledge of the engine’s application programing interface will suffice.

4. Model based techniques

We believe that virtual reality system developers should draw from the video-game industry’s large experience and borrow some of their clever ideas and valuable insights. To keep game players engaged and compensate for the lack of immersive displays, computer-based games have invested in realistic visual effects, better sound and a physically responsive environment.

Our motivation, therefore, was to take advantage of these experiences and design a camera model to perform travel based on rigid body dynamics. By camera model we mean an abstraction that maps input commands to camera commands, thereby controlling the user’s viewpoint motion and orientation through a virtual environment. By travel based on rigid body dynamics we mean that the viewpoint is associated with physical parameters, such as mass, position, and linear/angular momenta.

In our proposal, called Physicam, we let a physics engine in charge of the dynamics simulation behind the camera control. In this model, certain forces and torques are assigned to the user input, and the camera has typical rigid body properties. Consequently, the user or environment generated forces alter the camera position and orientation.

To illustrate our proposal we compare the ‘traditional’ way navigation is handled, as shown in the diagram of Figure 2a, with a modified version that accommodates the Physicam, as shown in Figure 2b. In both pictures the travel metaphor layer is responsible for providing a mapping between the data from input devices (e.g. mouse coordinates, key strokes) and the camera’s parameters (orientation and position). Subsequently, the camera model translates these requests into actual camera actions within the ve.

The extra layer present in Figure 2b represents the dynamic system that controls the physical simulation of the rigid body entity associated with the virtual camera. In this new scheme the data from the travel metaphor are interpreted by the physically-based model as forces and torques applied to the the rigid body attached to the camera, thereby performing travel within the ve. Notice an arrow coming out of the virtual environment layer into the physically-based
model — it represents collision detection data or environmental forces that might be acting upon the camera’s rigid body, like gravity or wind gusts. This new input, in turn, may trigger camera motion (i.e. data flow downwards again) and cause force data to be sent upwards to the haptic rendering pipeline. Consequently, our proposed scheme naturally accommodates the association of haptic feedback to the camera control, which is known to improve presence.

Another parameter of B is the position vector \( \mathbf{r} \), which is updated at every simulation frame according to the Equation 1, which corresponds to the Newton’s Second Law of motion. For this update, it is necessary to know the sum vector corresponding to all forces acting on \( B \) in a time frame \( t \).

\[
\mathbf{F}(t) = m \mathbf{a} = m \dot{\mathbf{v}} = m \ddot{\mathbf{r}}. \tag{1}
\]

The numerical integration of the second-order differential equation in \( \mathbf{r} \) (Equation 1) can be done by solving a system of two first-order differential equations: \( \dot{\mathbf{r}} = \mathbf{v} \) and \( \dot{\mathbf{v}} = \mathbf{F} = m \mathbf{a} \). This system may be represented by the vector \( \mathbf{S}(t) = [\mathbf{r} \mathbf{v}]^T \), which is regarded as the system state and needs to be stored for each rigid body (Equation 2).

\[
\frac{d\mathbf{S}}{dt} = \frac{d}{dt} \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \frac{\mathbf{F} \cdot \mathbf{v}}{m} \end{bmatrix} \tag{2}
\]

Solving Equation 2 yields the increment \( \mathbf{r} \) that needs to be added to the \( B \)'s current position \( \mathbf{r} \) every time frame (the elapsed time since last update). Notice that all the calculations described in this section are done by the physics engine.

4.2. Rigid body constraints

A physics engine should also provide mechanisms to apply constraints to rigid bodies in a simulation. Joints implement such mechanisms, limiting the degrees of freedom of a rigid body in relation to a reference frame, or among a group of rigid body entities. The most common types of joint are hinge, allowing only a restricted rotation angle between two objects; pivot, allowing bodies to spin and twist around one another; screw, allowing two objects to simultaneously move along and rotate around each other, and; up-vector, utilized to keep a body’s orientation fixed in relation to a global reference frame. In our work, for instance, a pair of up-vector joints keep the camera’s rigid body from rolling over when a force is applied upon it.

4.3. Travel Metaphors

The Physicam supports all three navigation goals (cf. Section 3.1), as long as the physical parameters are set accordingly. For instance,
if one wants to exam a virtual object (maneuvering), we set the angular acceleration of $B$ so that the camera rotates slowly, thereby enabling slow and more precise movements to support close inspection. Conversely, if one wants to cruise long distances we need to increase the force mapped to, say, a forward command.

The manipulation method for travel adopted in our model is the virtual control, which can be constrained or not [15]. In the case studies developed so far, we have employed the “walk” metaphor — movement is allowed only on the horizontal $XZ$-plane, assuming a right-handed coordinate system ($Y$ is vertical). This metaphor is useful, for example, when simulating an architectural walkthrough. Other metaphors are possible, such as driving a “wheeled vehicle”, which might be useful if one needs to travel long distances in massive ves, or even a “submarine motion” simulation (somewhat similar to a constrained “fly” metaphor).

The “walk” metaphor in the scene is rendered in egocentric view, and allows yaw and pitch control. Translation is accomplished by adding forces along $X$ and $Z$. If the camera rests atop a virtual object or the ground, it is possible to add a small amount of force in the $Y$ direction, as to simulate a jump. However, if the camera does no touch any surface only the gravity and/or buoyancy forces are considered — this prevents user from flying around or jumping right out of water in an unrealistic fashion.

The test for contact is done by shooting a set of line segments of length $L$ downwards and testing whether any of them intercept a virtual surface; $L$ determines the maximum height a camera might reach in a jump. If the slope of the contact surface is lower than a threshold, we reckon the camera to be “on” the contact surface, thus any forces (triggered by the user) in the $Y$ direction must be considered (see Figure 3).

The final camera constraint affects pitch, preventing viewpoint to tilt upside down. The use of joints (up-vector) allows the camera to translate, say, forwards while the viewpoint faces a different direction. Table 1 summarizes the mapping of input commands to camera actions that characterizes the “walk” metaphor. Mouse displacements are mapped to torques applied on $B$ and a joint blocks roll.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>w or up</td>
<td>Add force in the $+Z_c$ direction</td>
</tr>
<tr>
<td>s or down</td>
<td>Add force in the $-Z_c$ direction</td>
</tr>
<tr>
<td>a or ←</td>
<td>Add force in the $+X_c$ direction</td>
</tr>
<tr>
<td>d or →</td>
<td>Add force in the $-X_c$ direction</td>
</tr>
<tr>
<td>r or Page Up</td>
<td>Add force in the $+Y_c$ direction</td>
</tr>
<tr>
<td>f or Page Down</td>
<td>Add force in the $-Y_c$ direction</td>
</tr>
</tbody>
</table>

Table 1. Camera actions associated with input commands. $(X_c; Y_c; Z_c)$ denotes the origin of the camera’s coordinate system.

5. Results

In order to evaluate the objectives set out in Section 1, three virtual environments were designed, each requiring users to perform a task characterizing one of the three navigation purposes: exploration, maneuvering, and search. All three desktop ves were enhanced with physical properties, such as external forces (gravity, wind, buoyancy), collision, stairs, ramps, pushable obstacles, and so forth. We wanted to try most of the physical properties supported by Physicam, observe the effects of the interaction between the camera’s rigid body and the environment, and investigate any eventual influence on navigation tasks by means of an informal user study. Figure 4 shows a few screenshots taken from the virtual buildings used in these case studies.

The gravity simulation is present in all three ves. Gravity is specially useful to ensure that the viewpoint motion is constrained to the XZ-plane. It is approximated by the Equation 3

![Figure 3. Illustration of the contact test to determine whether we consider any vertical forces applied to the camera. In these pictures the ellipse represents the Physicam, $F$ is one of the camera’s test line segments, $\vec{n}$ is the normal of the contact surface being tested, and the threshold is $T = 35^\circ$. In (a) the gradient of the contact surface is greater than the threshold $T$, thus we ignore vertical forces triggered by the user, while in (b) the gradient is smaller than $T$, hence we consider any vertical forces that may be acting upon the body.](image-url)
Besides external forces like gravity and buoyancy, there are other physical properties that can be set for objects in the environment. These properties are:

- Geometric shape ($s$).
- Mass ($m$).
- Position ($\mathbf{r}$), and orientation ($\mathbf{\Omega}$).
- Linear velocity ($\mathbf{v}$), linear acceleration ($\mathbf{a}$), angular velocity ($\mathbf{\omega}$), and angular acceleration ($\mathbf{\alpha}$).
- Inertia tensor ($\mathbf{I}$).
- Coefficients of static friction ($\mu_s$), kinetic friction ($\mu_k$), elasticity ($\epsilon$), and softness ($k$).

### 5.1. Materials

A total of 16 participants (15 men and 1 woman) took part in our pilot study. Their mean age was 19.3 years ($sd = 1.4$). Participants navigated using mouse and keyboard in a VE presented in a 19" display screen, running on a Intel Core 2 Duo T7250 (2.2Ghz), with 2GB of Ram and a NVidia Geforce 8400GS. The ogre3d graphics engine\(^1\) handled the graphics rendering, while the Newton Game Dynamics\(^2\) controlled the physical simulation for the camera model and the environment. The ogreNewt\(^3\) integrated the physics engine with ogre3d.

#### 5.1.1. Data collected

Before the actual sessions, participants were asked to complete a general background information questionnaire aimed at gathering information on previous user experience with virtual reality systems and first-person style games. After each session, participants were asked to fill in a questionnaire reporting their subjective experiences with the camera, regarding user interface, scene realism, camera control, and immersion. Finally, after finishing the walkthrough in building #3 (cf. Section 5.4) participants were asked to identify the floor plan of the building they have just visited among a set of four options.

#### 5.2. Building #1: maneuvering navigation

The first VE was comprised of a two-floor building connected by a flight of stairs, several scattered boxes having physical properties (i.e. they could be pushed away by direct collision with the Physicam) and a ramp (on the second floor) leading to the exit (see Figures 4a and 4b).

---

1 http://www.ogre3d.org/ (last access 13/Feb/09)
2 http://www.newtodynamics.com/ (last access 13/Feb/09)
3 http://www.walaber.com (last access 13/Feb/09)
The task for this session was to follow a target object (a sphere) that moved at a constant speed through the building up to exit, as soon as any key was pressed – this characterized maneuvering navigation. Participants were asked to follow the target and keep themselves within a certain distance range from the target. Real time feedback indicating whether they were complying with the request was given in the following manner: if the target became green it meant that participant was within range, otherwise the target became red.

At the end, after pursuing the sphere, users had to inspect the faces of a five-sided polyhedron to learn a target symbol. The target symbol, however, would only reveal itself if the viewpoint was in perfect alignment with the center of the face, at a required distance (visual cues were provided). To enable this fine control over viewpoint motion and orientation, the users had to switch the set of physical parameters (angular acceleration, torques, and forces) associated with long-range motion to a suitable set that supported slow and more accurate movements.

5.2.1. Overcoming obstacles
The adopted geometric shape s, a prolate spheroid, is ideal to accommodate an humanoid avatar in its interior and supports obstacle transposition.

This simple approach eliminates the need for prior programming that would describe, for instance, how to climb a flight of stairs. When the camera reaches a flight of stairs the net force exerted on the camera’s rigid body (B) makes the camera go up (or down), step by step, hence simulating a climbing behavior. Of course, this ability depends directly on the relative height of each individual step in relation to the height of s. This is true because the system of forces applied to B, i.e. the force triggered by the users (F_u) and the stairs’ collision reaction force (F_s), yields a resulting force (F_r) that pushes B upwardly. Observe that the gravity (F_g) and the normal (N) forces cancel each other out. Figure 5 illustrates this concept.

5.3. Building #2: search navigation
The second ve was a building containing a long corridor leading to a large hall that had, at its center, a pool bounded by two walls. The exit was located on the opposite side of the hall. The side accesses to the pool had a low coefficient of friction, in an attempt to simulate a slippery floor. The task for this session was to find 7 cones distributed throughout the environment, in the least amount of time – this characterized search navigation.

5.3.1. Changing friction and buoyancy effect
Different friction effects were tried in this building, according to Equation 4, where \( \mu (0 \leq \mu \leq 1.0) \) is either the static (\( \mu_s \)) or the kinetic (\( \mu_k \)) coefficient of friction between the material associated with B and the material assigned to the virtual object; and \( F_n \) is the normal force exerted upon the body by the contact surface.

\[
F_{fr} = \mu F_n \tag{4}
\]

We used different values for \( \mu \) to simulate a slippery corridor (\( \mu = 0.15 \)) and a sand access (\( \mu \) close to 1). Through this simple use of friction, it was possible to simulate a walk on different types of surfaces (sand, regular floor, ice/wet floor). The effect was felt by users as the camera slowed down (on sand), moved at normal speed (regular floor), or slipped and even skidded (on icy floor). A pool was also simulated and users were encouraged to jump in, since one of the target cones was located at the bottom. The pool generated buoyancy forces when

---

4 The physical model that simulates the system of forces is an approximation where all forces are applied to the B’s center of mass.
the camera’s rigid body was submerged, according to Equation 5, where \( \rho \) is the density of the liquid, \( V \) is the volume of the submerged body, and \( g \) is the gravitational acceleration.

\[
F_b = \rho V g
\]  
(5)

5.4. Building #3: exploratory navigation

The third virtual environment was a three store building. At the building’s entrance there was a moving walkway, which was capable of transporting the Physicam. The first two floors were connected by a ramp. There was no access from the second to the ground floor, therefore the users had to fall in order to get to the ground floor. On the ground floor there existed a series of three rooms, all interconnected by automatic revolving doors.

The whole idea behind the design of this scenario was to provide an example in which external forces could be applied upon the camera. Specifically, we had gravity, the moving walkway, and the automatic revolving doors – all these environment components could displace/translate the camera.

The task for this virtual environment was to navigate throughout the building, trying to explore and, at the same time, study the shape and disposition of the building – this characterizes exploratory navigation. The participant started at the top floor and had to work her way down to the exit, on the ground floor.

5.4.1. Adding external forces

The principal physical components of this building were: (i) several obstacles that had to be pushed to make way for the camera navigation; (ii) the force of gravity that helped user to work his/her way out; (iii) a moving walkway (realized through the use of coefficient of friction) that could move the camera through a long corridor, had the user chose to; and (iv) an automatic revolving door that could push the camera through to the next room. All these effects might have been more compelling, had the forces generated by the environment been mapped to a haptic device, such as a phantom arm or a force-feedback joystick.

Also, it is possible to set the environment to produce winds gust that could push the camera around, thus simulating a storm.

When the exploration was finished and the participants reached the exit, we asked them to indicate the direction of the building entrance; the angle corresponding to their answer was recorded by the system.

6. Results

From the total of 16 volunteers, 12 reported that they were very satisfied with the visual realism afforded by the dynamics simulation of the virtual environment. The remaining participants classified the visual experience only as acceptable; as expected these 4 individuals were among those who have had over 5 years of intense game experience (i.e. playing nearly every week).

Notably, most of participants who classified their experience as very satisfying added that they were quite impressed with the animation of boxes and obstacles being thrown/- pushed away, as a result of the camera bumping obstacles out of the way – the end result changed depending on both the velocity the camera hit the obstacle and its physical parameters. Another aspect that felt real to participants was the buoyancy effect of the pool in building #2.

The results of the spatial orientation test (building #3) showed that most of users (87.5%) were able to build a cognitive map of the environment. However, we are aware that the results are not conclusive, since the buildings could hardly be considered complex.

Regarding the viewpoint control, most of the users reported that the Physicam achieved its goal of engaging users in their virtual experience. A group of 5 participants informed that sometimes the movement seemed “unreal” and “artificial”, as though they were “rolling a sphere”. We identified that this undesired effect was caused by a poor set of physical parameters. This created a little delay in halting the camera after users stop pressing the forward key – this effect was caused by the camera’s momentum. We noticed that the users who did not complain about the ‘momentum’ effect realized that to bring the camera to an immediate halt they should press the backward key, which counteracted the forward force, thereby functioning as a type of ‘car breaks’.

Surprisingly, approximately half the participants informed that they would rather use the
unconstrained flying camera without collision detection/response than the ‘walk’ camera with collision processing. We believe that this preference could be explained by their prior experience with the unconstrained flying camera in other desktop vs. From this study we have been able to identify the following advantages in utilizing the Physicam:

- It is possible to simulate several camera motion behaviors by changing the body’s shape, physical properties, or combination thereof. For instance, it is viable to simulate a walking-like behavior, which is known to improve the sensation of walking in vs; or even make a camera climb stairs without having to explicitly code that behaviour.

- The camera behavior programming is simplified – we just need to define the physical properties of a given material, assign it to an object, define how any two material should react to one another and let the physics engine worry about their reaction when the contact occurs.

- The camera is subject to the forces of the environment, such as gravity, wind, and friction. Also, as a side effect, collision detection and response is enforced by the physics engine. This is a natural way of aiding navigation and improving spatial awareness, as reported in [5].

7. Conclusion and Future Work

In this work we aimed at harnessing game engine technology to help us build a virtual environment and perform navigation, an ubiquitous user interaction task. Travel is the physical component of navigation, encompassing both locomotion and orientation control of viewpoint within a virtual environment. Instead of directly altering a set of orientation parameters (e.g. yaw, pitch and roll) and translating the camera position to perform travel, we encapsulate a camera within a rigid body and let travel be governed by forces and torques applied by users. The dynamics simulation involved in this new approach to control camera travel (called Physicam) is entirely handled by a physics engine.

Our motivation for Physicam was twofold: (i) to harness dynamics simulation to increase scene realism, and to perform travel in such a way that resembles the natural movement of a body with mass and inertia; and (ii) to increase the productivity of a ve programmer by offering a flexible camera whose travel behavior was easily configured. We hoped that, by introducing constraints, believable collision response, and enabling the camera to respond to environmental forces, the navigation process would, somehow, be improved.

In order to test our proposal we developed three virtual environments, each of which enhanced with physical properties, such as gravity forces, buoyancy, friction, an collision response. Each ve was designed for a specific navigation task, namely search, maneuvering, and exploration. A total of 16 volunteers took part in our pilot study, giving us valuable subjective feedback. Overall the Physicam received good evaluation regarding visual realism, and quality of camera control. The user response also indicated that it is necessary to fine tune physical parameters in order to get a locomotion behavior adequate to the task at hand. Therefore, a set of parameters (force, mass, friction, etc.) chosen to cover long stretches of a virtual scenery is most likely not appropriate to perform close inspection of an object.

In terms of future work, we shall investigate how to adapt physics concepts to other aspects of 3d interaction, such as object selection and manipulation. For instance, one may want to fire a gravitational gun to select and move objects from on location to another.

Finally, further investigation will be conducted shortly, after adjusting some of the materials and the experiment design. The goal is to test the hypothesis that the use of Physicam may improve the sense of presence and spatial awareness in desktop virtual environments.

8. Acknowledgement

The authors acknowledge financial support from Fundação de Apoio à Pesquisa do Estado do Rio Grande do Norte (fapern/cnpq/PPP, grant number 610031/2006-6), INCT-MACC (cnpq, grant number 573710/2008-2), Conselho Nacional de Desenvolvimento Científico e Tecnológico (cnpq, grant number 308857/2008-9), pci/lncc/mct, and Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (faperj, grant number E-26/103.106/2008).
9. References


