Quantum Games: Ball Games Without a Ball

Henning Pohl, Christian Holz, Stefanie Reinicke, Emilia Wittmers, Marvin Killing, Konstantin Kaefer, Max Plauth, Tobias Mohr, Stephanie Platz, Philipp Tessenow, and Patrick Baudisch

Hasso Plattner Institute, Potsdam, Germany

ABSTRACT

We present Quantum games, physical games that resemble corresponding real–world sports—except that the ball exists only in the players' imagination. We demonstrate Quantum versions of team handball and air hockey. A computer system keeps score by tracking players using a Microsoft Kinect (air hockey) or a webcam (handball), simulates the physics of the ball, and reports ball interactions and scores back using auditory feedback.

The key element that makes Quantum games playable is a novel type of physics engine that evaluates not one, but samples the set of all plausible ball trajectories in parallel. Before choosing a trajectory to realize, the engine massively increases the probability of outcomes that lead to enjoyable gameplay, such as goal shots, but also successful passes and intercepts that lead to fluid gameflow. The same mechanism allows giving a boost to inexperienced players and implementing power–ups.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation**]: User Interfaces: Input Devices and Strategies, Interaction Styles

General Terms

Design, Human Factors

Keywords

Games, probabilistic, imaginary

1. INTRODUCTION

Augmented reality games, such as *Human Pacman* [3] and *AR Quake* [9] bring the benefits of virtual games, such as game modifiers and power–ups into a physical game. They do so using mobile or head–mounted displays. Unfortunately, as with traditional video games, these games require players to focus on screens rather than onto the physical world and each other.

In this paper, we explore what happens when we leave all displays out and build a game based on the concepts of *spatial, non-visual interaction* inspired by screenless mobile devices (*imaginary interfaces* [4]).

2. A QUANTUM GAME

Figure 1a shows two players playing Quantum Air Hockey, a quantum game that is played on a regular table. The game resembles the corresponding real–world game, Air Hockey, except that the puck exists only in the players' imagination. The overhead Kinect camera [11] tracks the game by observing players' arms and hands on the table. A loudspeaker provides players with updates on the state of the game, such as "Blue shoots … intercepted by Red".

The main reason for making the puck imaginary is that this enable gaming functionality that can be achieved in virtual games, but not in physical games, in particular power-ups and player balancing, thus enabling players of different skill levels to play together. In games with a physical ball or puck, there is a single clear reality. Quantum games, in contrast, do not have this obvious reality. This allows Quantum games to make choices of outcomes, allowing them to implement more interesting game play.

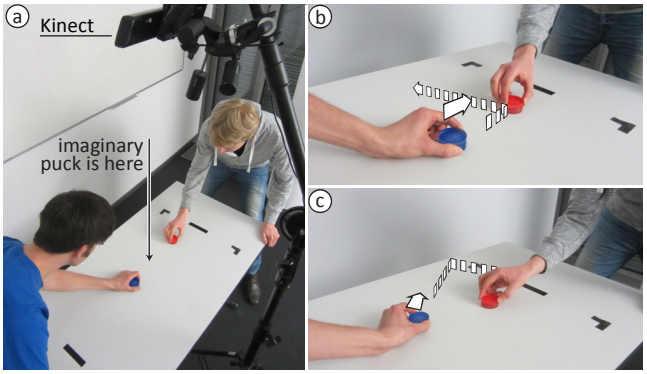

Figure 1: (a) In this game of *Quantum Air Hockey,* the left player is trying to score. (b) He is shooting the imaginary puck at Red's goal, but his shot is blocked. (c) In a second attempt, Blue plays the puck via the left wall and scores. A *Kinect* camera mounted above tracks players' hands and mallets.

The game starts when the puck drops at a known spot in Blue's half. This is communicated by an audio announcement proclaiming that the puck was dropped in front of Blue's goal. (b) Blue shoots the puck towards Red's goal. Red blocks the ball and (c) Blue shoots again, this time via the wall, avoiding the mallet red had placed in front of his goal—goal!

The maybe obvious question is how this game can be playable. How can players successfully hit a puck that they cannot see? The secret is the particular type of physics engine which we created for quantum games.

Figure 2 shows a debug view (this view is never seen by the players—there is no visual feedback in Quantum games) of what happened under the hood during the previous play. We see that what presumably is a single puck is represented as a collection of a puck *particles* (here we use 1000), each of which represents a

location where the puck *may* be located with 1/1000 probability. Accordingly, the Quantum physics engine computes trajectories for each particle independently.

Figure 2a: As the puck is dropped into the game, it instantly breaks down into a collection of puck particles. (b) While Blue is getting ready to shoot, the particles spread out—we describe this mechanism in detail later. (c) Blue takes his shot and Blue's mallet collides with a larger number of particles. This triggers the shot. (d) Each particle follows a different trajectory; the engine thus needs to decide which one of them to realize. It computes the trajectories of *all particles* at once, here visualized as lines, and groups particles by common fate, such as "goal via the left wall" or "intercepted by opposing mallet" etc. It counts the particles for each fate and uses these counts to computes probabilities for each possible outcome. (d) The Quantum game engine modifies probabilities, to make desirable outcomes more likely, here the intercept. (e) The engine draws an event from the resulting probability distribution and determines that Red has blocked the ball (Figure 1b) and plays back a collision sound. (e) The blocked particles spread out at the new location. (f) Blue takes another shot, this time via the wall (as in Figure 1c) and scores the goal (g).

We will look at the underlying algorithms in more detail in the "Algorithms" Section.

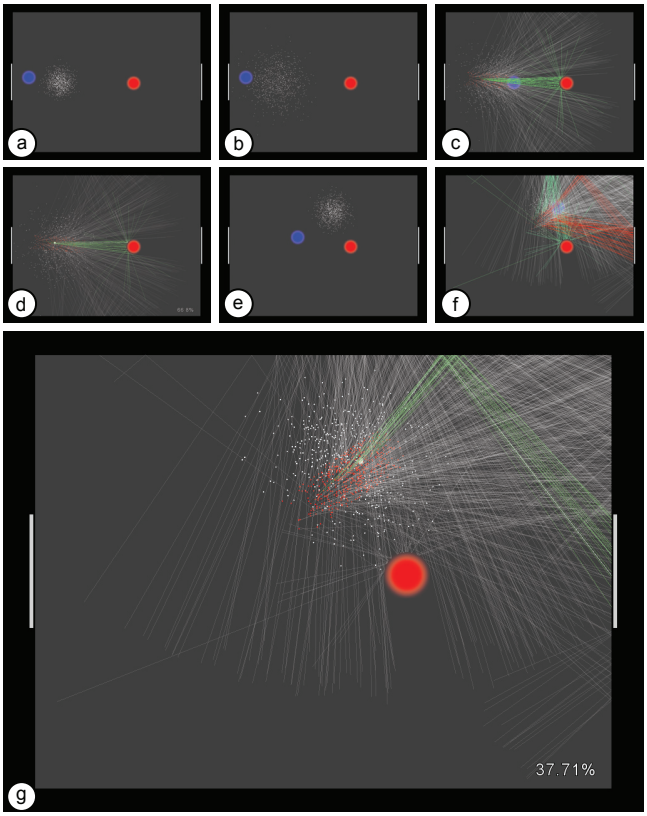

Figure 2: A debug view illustrating how the system processed the game actions we just saw in Figure 2

2.1 Extension to whole-body physical games

Quantum air hockey is just one possible representative of a Quantum game. Figure 3a shows *Quantum team handball*, an example for a whole body Quantum game. Again, the game is tracked from above.

3. CONTRIBUTION

The main contribution of Quantum games is the idea to play with an invisible or "imaginary" ball and how to resolve the apparent absurdity using a probabilistic physics engine. Because of this special type of physics engine, Quantum games create interesting game play not despite, but *because* of the high level of uncertainty resulting from the lack of visual feedback.

While we think of Quantum games primarily as games, we can also look at them as imaginary interfaces [4]. In this case, our contribution is that we explore how to create *shared* imaginary interfaces, i.e., spatial, non–visual interfaces that multiple users are engaged in simultaneously.

Figure 3: (a) In this game of Quantum team handball, the orange player is trying to pass the ball to his pink teammate. (b) The underlying game engine represents the ball as a set of particles each representing one plausible ball trajectory.

4. DESIGN RATIONALE

Quantum games are designed to achieve two design goals: *game of skill* and *gameflow*. While these are achieved by a wide range of traditional *visual* games, the lack of visual feedback requires us to take a different approach.

4.1 Quantum Games are Games of Skill

In order to require skill, players need to understand how to act in order to win. Traditional games achieve this by hinting players using visual feedback. Quantum games, however, cannot offer such feedback.

What makes Quantum games playable is that players know how to play *before* they start. Quantum games accomplish this by *mimicking* an existing sport. Even though Quantum games' inner workings are probabilistic, Quantum games are designed to afford a "Newtonian interpretation", i.e., they suggest the conceptual model that users interact with a *single*, *physical* (yet invisible) ball. This metaphor not only allows players to transfer their knowledge of the rules of play, but also leverages their previous experience with real–world physics, such as ball ballistics.

To convey the metaphor, (1) Quantum games' auditory feedback conveys only discreet, Newtonian events. Internal properties, such as probabilities remain hidden from the user at all times. (2) To maintain the illusion of a Newtonian game, Quantum games sample only game moves that are *plausible* under a Newtonian interpretation.

Playing a Quantum game therefore requires the same skills as the sport that inspired it. In order to successfully *anticipate* a shot, a soccer goalie has to (1) read the opposing player's body language, (2) imagine ball trajectories, and (3) quickly move to the extrapolated location. Quantum games require the same skills.

4.2 Quantum Games Maintain Gameflow

If our goal were only to require skill, a single–ball design would do. However, there would not be much gameflow, because players tend to have little success when interacting with an invisible ball. The Quantum engine resolves this by shuffling probabilities around, so as to increase the likelihood of moves that make for good gameflow, that keep the ball in the game, and allow for longer and more complex moves. Adjusting probabilities does not affect consistency with the Newtonian interpretation—it may cause the engine to realize *unlikely* moves, but never *implausible* moves.

5. Benefits

Quantum games combine properties of different types of games. Similar to physical sports, Quantum games players interact with each other directly, without an intermediary screen. The design also eliminates the need for projectors, which allows applying the concept to potentially very large installations.

Similar to computer games, Quantum games enable game mechanics that bend or break traditional physics, such as wormholes and power–ups. We can also make players of different experience levels compatible using a handicap system. Similar to computer games, the game is "safe" in that there is no physical ball flying around in the living room.

6. Quantum Games are a Family of Games

Not all physical ball games make equally good Quantum games. Since Quantum games tweak the probabilities of different interpretations of the same action, a game has to offer moves that offer such multiple outcomes. At first glance, this is the case for essentially all ball games: The choice between a player hitting/catching the ball *or not* exists in baseball, volleyball, and dozens of other sports, down to golf. All these sports can, in theory, be implemented as Quantum sports, but not all of them benefit equally.

What distinguishes truly suitable games is that they offer situations in which players (ideally from opposite teams) *compete* for the ball. When a soccer player, for example, passes the ball towards the goal, players from team A *and from team B* are trying to get to it. This type of move provides the Quantum engine with true choice, because it allows rewarding skillful play without breaking gameflow: if the players on team A play poorly, the engine can penalize them by *favoring the players on team B*, rather than letting the ball go out.

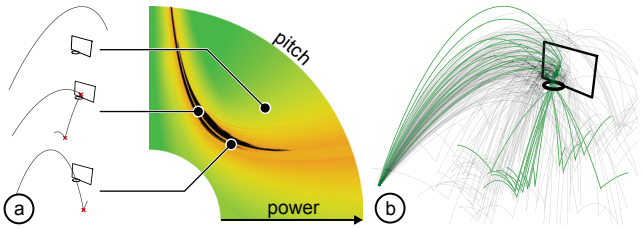

Figure 4: Probability distribution in Quantum basketball

This way, gameflow is preserved and the Quantum engine has simultaneously realized both of its main objectives: "game of skill" and "gameflow". This is different from the "hit/catch the ball *or out*" pattern mentioned earlier, where penalizing poor play lets the ball go out, thus breaking the gameflow.

The more often the "A vs. B (vs. out)" pattern occurs in a game, the more the game benefits from the Quantum engine. Examples include a wide range of sports, including soccer, football, basketball; the latter we have started to implement (Figure 4).

7. Related Work

Probabilistic methods have been applied to a range of interactions, such as touch events and gestures [10]. Chenney et al. use Markov chains to generate plausible–looking animations that satisfy a certain set of constraints [2]. Twigg and James allow users to create animations by first generating many sequences and then eliminating all those that do not meet a given constraint [12].

The concept of introducing virtual game objects into a physical scene has been explored by mixed reality and pervasive games [6]. *Airhockey Over a Distance* locally re–enacts the shots of an opponent located at a remote site [7]. Players of *AR²Hockey* use physical mallets to shoot a puck that is only visible on head–mounted displays [8]. Jebara et al. compute the current best shot in pool billiard and display its trajectories on a head–mounted display [5]. Unlike these games, the ball in *Quantum games* is never shown, which allows it to choose from a wider set of outcomes without leading to a disconnect with reality.

8. Quantum Games System Architecture

As illustrated by Figure 5, a Quantum game system consists of three main components. A tracking system tracks all physical objects, i.e., the players and the playfield, including boundaries, goals/baskets, and power–up respawn locations. It reports these to the Quantum engine. The Quantum engine maintains the spatial model of the scene and computes plausible interactions using its probabilistic model.

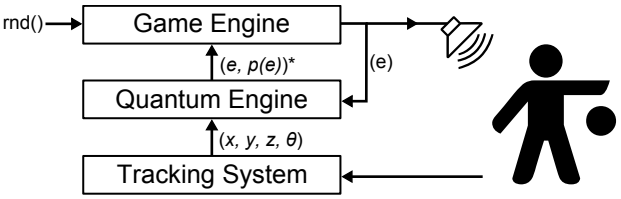

Figure 5: Quantum games system

Whenever an event happens the Quantum engine reports probabilities to the *game engine*, which processes probabilities by applying modifiers, such as handicaps and power–ups. It then draws an event from the resulting probability distribution and returns the decision, which it conveys to the players using auditory feedback. It also informs the Quantum engine about the event so as to allow it to update its model. The game engine itself, however, has no notion of space—all it does it manipulate probabilities.

We now discuss these components in additional detail.

8.1 Tracking System

Implementing Quantum games generally requires determining the position and orientation of all physical objects on the play field, functionality that can, for example, be provided by a motion capture system. Any specific Quantum game, however, typically requires only a subset of these parameters.

8.1.1 Using Kinect For Ad–hoc Quantum Air Hockey

Using a Kinect camera for tracking, Quantum air hockey allows for ad-hoc gaming on regular tables. To start the game, players put the playfield markers on the table, position a Kinect camera above the playfield, and initialize the game. The game runs off a laptop, which also provides the auditory feedback. As shown in Figure 6, our system processes Kinect images as follows. (a) Upon start, our system extracts the playfield markers from the

color image and (b) captures the static depth background. During playtime, our system subtracts from (c) the raw image the stored background, thus creating (d) a mask of players' arms (as in [13]). (e) We obtain connected components in the mask image, extract the 20–pixel environment around the point that is farthest from the respective edge of the playfield minus a constant offset (red dot in Figure 6e). We stop processing if this point is too far removed from the table, such that hovering hands produce no input to the game. Assuming that mallets are attached to players' hands, we derive their final locations using a high-gain Kalman filter on the 2D hand locations and rectify coordinates according to the distortion of the playfield as shown in Figure 6f. We forward the final coordinates to the Quantum Game.

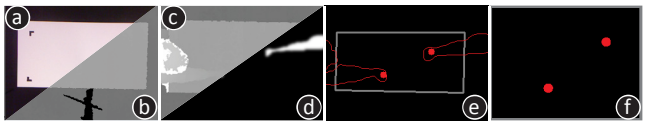

Figure 6: Our Kinect-based tracking captures the (a) playfield and (b) the static depth background upon startup. (c) During runtime, subtracting the background results in a mask of players' arms (d), from which we infer the locations of players' hands and thus mallets (e). (f) Smoothed mallet locations rectified according to the playfield markers.

An earlier prototype of Quantum air hockey used Microsoft Surface to track mallets. Although MS Surface reliably identifies and precisely tracks players' mallets, it does not allow for ad–hoc playing and variable–size playfield setups unlike Kinect.

8.1.2 Color Tracking for Large–volume Games

To allow Quantum games to be played in a substantially larger volume, such as required for playing Quantum team handball as shown in Figure 3, our system tracks players' colored hats from an overhead perspective. Quantum team handball is implemented in 2D and assumes that the ball is always floating at chest height. For inferring player input, our system detects the colored hats in the camera image and tracks players' motions using *Camshift* [1] To detect when players throw a ball, we equip players' hands with accelerometers, which we found to work better than trying to extract hands from camera images.

For future work on tracking games that have a third dimension, such as Quantum basketball, we plan on doing full motion capturing using multiple Kinect cameras.

8.2 Algorithm of the Quantum Engine

The Quantum engine represents the ball (or puck etc.) as a probability distribution over space. Our current implementation uses a discreet representation, i.e., a set of ball particles. For Quantum air hockey, 100 particles work well; our engine handles up to 1000 particles in real time, allowing us to detect events of 0.1% probability with a 1-1/e = 63% certainty.

The following rules determine all particle behavior.

8.2.1 Initialization

At the beginning of the game a single particle is put into the game in a physically landmarked respawn location.

8.2.2 Creation

If a particle has a probability p above a threshold, it splits into two new particles, each with half that probability. Repeated application of the rule causes the initial particle to spawn the desired number of particles, e.g. 1000.

8.2.3 Brownian Motion

Particles move by a small amount in a random direction with every time step (Figure 7a–b). This simulates how player's knowledge of the ball location tends to blur over time. If, for example, players return from a break they may not remember where the ball was; Brownian motion implements this by making the ball be "everywhere" (Figure 7c).

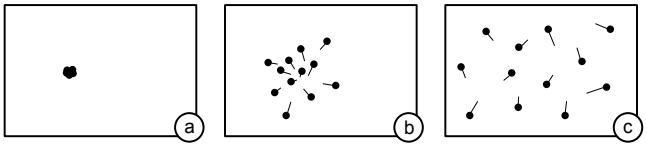

Figure 7: (a-c) Brownian motion moves particles randomly.

8.2.4 Collision

For every frame, the Quantum engine tests whether the *ball* has collided with a physical object, such as a wall, goal, player/mallet. It does so by testing *each particle* for collisions against these physical objects (but not against each other, because the 'existence' of two particles is mutually exclusive), as illustrated by Figure 8a.

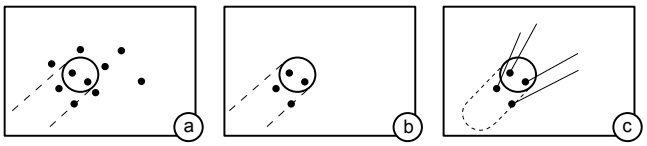

Figure 8: Collision detection: (a) An air hockey mallet collides with particles. (b) When the engine decides a shot took place, all particles that recently collided with the mallet (from the *purgatory*) are kept alive. (c) The left over particles spawn a broad field of shots.

Each particle itself behaves in a Newtonian way. We can therefore compute their collisions using a regular physics engine. We initially tried *PhysX*, *Havoc*, and *Box2D*, but ended up writing a custom one for speed.

Whenever a particle collision is detected, the engine needs to decide whether this means that the actual ball has collided. To do so, the engine tests the collided particle's probability against random. If 200 of 1000 particles collide, for example, the actual ball has collided with p = 200/1000 = 20%. If so, the user is informed using auditory feedback.

8.2.5 Deletion

One way of interpreting what happened is that one particular particle caused the collision of the ball. This would mean that the particle would take on a probability of 1, all other particles take on a probability of 0 and thus can be removed: the position of the ball would now be fully determined; uncertainty has collapsed to a single point and the collision would result in a single beam of particles.

8.2.6 Maintaining Variation

This interpretation, however, is bad for gameflow. The reason is that it reduces the probability distribution more than necessary. The size of probability distribution matters, because more variation provides the Quantum engine with more choice, which in turn increases its chance to find an outcome that leads to good gameflow. While we *have to* remove particles that have become implausible (i.e. that conflict with the state conveyed to the players using auditory feedback), any particle we eliminate beyond this means a loss for variation.

To preserve as many particles as possible, we proceed as follows: (1) The engine picks one particle as the *primary outcome*. This particle determines the auditory feedback. (2) The engine preserves all similar particles, i.e., those that would produce similar feedback.

We implement this mechanism as follows: in every time step, particles that underwent a collision are moved into a ring buffer ('*purgatory*'). If a ball collision takes place, the content of the purgatory is used to generate the shot. If not, particles drop out of the purgatory and are removed. As a side effect, moving a mallet clears out all particles in the covered area. This matches our intuition, because players now know that the ball cannot be there.

Still any collision eliminates particles, causing overall variation to drop and which poses a risk to gameflow. Splitting particles with high probability will restore the intended particle level, but since resulting pairs of particles are identical it does not solve the problem quite yet. We create fresh variation using two mechanisms: (1) Particles floating around on the table in the moment of splitting pick up variation through Brownian motion. (2) Particles that undergo splitting in the moment of collision are given a slightly different direction. This is *plausible* if we think of mallets as having a rough surface [2]. We use a similar mechanism to introduce variation into particles representing a thrown ball.

8.3 Decision Making—The Game Engine

The most important part of our algorithm we already mentioned in the walkthrough section: Before the engine decides on a shot, the engine first computes all alternatives, assesses them, and tweaks these probabilities to increase the chance of desirable outcomes.

This process starts by the Quantum engine computing the future trajectories of *all particles*; in Figure 2, these were illustrated as lines. Each trajectory is characterized by where it ends (in the opponent's goal, the player's own goal, or somewhere on the playfield, etc. we call this *fate*) and how it got there, i.e., directly or via (a sequence of) collisions. We now tweak the probability of each particle by multiplying with a fate–specific factor, such as {(goal, 5), (ownGoal, 0.1), (mallet, 5), (field, 1)}. Based on these tweaked probabilities the outcome is decided.

These last steps (tweaking of probabilities, decision making, auditory feedback) take place in a separate part of the system, the game engine (see Figure 5). The game engine also applies two additional factors to particle probabilities: 1. Handicaps: a player–specific factor that boosts inexperienced players to make games between players of different experience more balanced. 2. Game modifiers, power–ups and penalties: during the duration of the power–up a player's abilities are increased or decreased.

Power–ups can change the probabilities of particles of a certain fate: *Penalty*: the probabilities of particles hitting a player's mallet are reduced. If set to zero, the mallet is essentially disabled, as it cannot interact with the ball anymore. *Attract*: the same process, but probabilities are increased, which helps receiving passes and intercepting the ball. *Goal seeker*: increased the probabilities of particles hitting the opposing goal, etc.

We can also introduce *conditional* penalties: *stun,* for example, imposes a penalty on a mallet, unless it stays put. We can create a wide variety of power–ups this way, such as *slow motion* (player cannot move faster than threshold) or *black ice* (player needs to continue current motion and speed) and so on.

9. Conclusions

In this paper, we presented Quantum games, physical ball games without the ball. Our main contribution is the game concept itself, i.e., the idea to write a game around an "imaginary" ball and how to implement the concept using a probabilistic physics engine. The presented work is work in progress and we are still implementing some of the functionality, tweaking algorithms and implementing additional sports. In the future, we plan to apply the Quantum engine to non–game related interactions.

10. REFERENCES

[1] Bradski, G. Real Time Face and Object Tracking as a Component of a Perceptual User Interface. In *Proc. WACV'98*, 214–219.

[2] Chenney, S. & Forsyth, D. Sampling plausible solutions to multi–body constraint problems. In *Proc. SIGGRAPH'00*, 219–228.

[3] Cheok, A.D., Goh, K.H., Liu, W., et al. Human Pacman: a mobile, wide–area entertainment system based on physical, social, and ubiquitous computing. *Personal and Ubiquitous Computing*, 8(2), 2004, 71–81.

[4] Gustafson, S., Bierwirth, D., and Baudisch, P. Imaginary Interfaces. In *Proc. UIST'10*, 3–12.

[5] Jebara, T., Eyster, C., Weaver, J., Starner, T., and Pentland, A. Stochasticks: augmenting the billiards experience with probabilistic vision and wearable computers. In *Proc. ISWC'97*, 138–145.

[6] Magerkurth, C., Cheok, A.D., Mandryk, R.L., and Nilsen, T. Pervasive Games: Bringing Computer Entertainment Back to the Real World. *Computers in Entertainment*, 3(3), 2005, 1–19.

[7] Mueller, F., Cole, L., O'Brien, S., Walmink, W. Airhockey over a distance. In *Proc. ACE '06*, 70

[8] Ohshima, T., Satoh, K., Yamamoto, H., and Tamura, H. AR2Hockey. A Case Study of Collaborative Augmented Reality. In *Proc. of VRAIS'98*, 268–275.

[9] Piekarski, W. and Thomas, B. ARQuake: The Outdoor Augmented Reality Gaming System. *Communications of the ACM*, 45(1), 2002, 36–38.

[10] Schwarz, J., Mankoff, J., and Hudson, S.E. Monte Carlo Methods for Managing Interactive State, Action and Feedback Under Uncertainty. In *Proc. UIST'11*, 235–244.

[11] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., Fitzgibbon, A. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. UIST '11*, 559-568.

[12] Twigg, C.D. and James, D.L. Many–worlds browsing for control of multibody dynamics. *ACM Transactions on Graphics*, 26(3), 2007, 14:1–14:8.

[13] Wilson, A.D. Using a depth camera as a touch sensor. In *Proc. ITS '10*, 69–72.