Real-time Physical Modelling of Character Movements with Microsoft Kinect

Hubert P. H. Shum
School of Computing, Engineering and Information Sciences
Northumbria University
Newcastle, United Kingdom
hubert.shum@northumbria.ac.uk

Edmond S. L. Ho
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong SAR
edmond@comp.hkbu.edu.hk

ABSTRACT

With the advancement of motion tracking hardware such as the Microsoft Kinect, synthesizing human-like characters with real-time captured movements becomes increasingly important. Traditional kinematics and dynamics approaches perform sub-optimally when the captured motion is noisy or even incomplete. In this paper, we proposed a unified framework to control physically simulated characters with live captured motion from Kinect. Our framework can synthesize any posture in a physical environment using external forces and torques computed by a PD controller. The major problem of Kinect is the incompleteness of the captured posture, with some degree of freedom (DOF) missing due to occlusions and noises. We propose to search for a best matched posture from a motion database constructed in a dimensionality reduced space, and substitute the missing DOF to the live captured data. Experimental results show that our method can synthesize realistic character movements from noisy captured motion. The proposed algorithm is computationally efficient and can be applied to a wide variety of interactive virtual reality applications such as motion-based gaming, rehabilitation and sport training.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Virtual reality*

Keywords

Real-time motion synthesis, Microsoft Kinect, physical simulation, virtual reality

1. INTRODUCTION

Synthesizing natural movements for virtual characters has been an active research area in past decades. In particular, using captured human motion to control virtual characters is getting more popular, thanks to the advancement of motion acquisition hardware. However, many kinematics-based control algorithms such as Motion Graph [12] require carefully crafted/cleaned motion. When synthesizing character movements from a stream of real-time captured motions such as those from the Microsoft Kinect, the result is usually sub-optimal due to the noise and incompleteness of the data. Simply replaying the captured motions would result in jittery movements and awkward joint orientations. Furthermore, the synthesized characters cannot react to external forces or interact with physically simulated objects, which degrades the user experience in virtual reality applications.

On the other hand, dynamics controllers that are originally proposed for robotic controls provide alternative methods to synthesis realistic character movements. Locomotion controllers such as [5] can track a series of postures and optimize for dynamically stable movements. The synthesized characters are guaranteed to be physically correct and can response to external perturbations. The major problem is that users have limited control over the resultant movements, since the simulation is strictly governed by physical laws. Some movements such as standing with one leg are very difficult to be dynamically reproduced without accurate information about the body mass and fictional force. Furthermore, motion captured from real-time systems usually contains a lot of noise, which can dramatically weaken the balance of the dynamically simulated character.

In this paper, we propose a unified framework to control characters with real-time captured motion from the Microsoft Kinect. We propose to track an input motion with a physically simulated character. Unlike the robotics simulation algorithms that use joint torques to drive a robot, our method controls the character with external forces and torques computed by a PD controller, which supports the balance of the character and enables it to perform any posture. Our method combines the advantage of kinematics and dynamics simulation. On one hand, it creates smooth and plausible movements that tightly resemble the user performed motions. On the other hand, it allows the simulated character reacting to external forces and interacting with the virtual environment.

The major problem of controlling characters with live captured motion from Kinect, or any capturing systems in general, is the noise and the incompleteness of the motions. During capture, some joint information may be missing because of occlusion or the restricted capturing volume. What is worse is that the number of tracking points in Kinect

VRST'12, December 10–12, 2012, Toronto, Ontario, Canada.

is very limited. Losing one point practically means losing both position and orientation of one joint, making it impossible to control a complete virtual character. To tackle this problem, we propose an efficient method based on Principal Component Analysis (PCA) to search for a similar posture in a motion database and substitute the missing information. This ensures the robustness of the system and relieves the requirement of clean input motion.

Experimental results show that realistic character movements can be synthesized from noisy captured motion. As the characters are physically synthesized, they react to external forces, and can interact with virtual objects. The proposed algorithm is computationally efficient, and can simulate multiple characters in real-time. Apart from motion-based gaming, our system is best to be applied in virtual reality applications such as those involving rehabilitation and sport training.

The rest of the paper is organized as follow. We first review the related works in motion synthesis based on user control signals and physical simulations in Section 2. We then explain the modelling of the physical environment in Section 4. We further explain the construction and usage of the motion database in Section 5. The core of the framework explained in Section 6 is a posture solver that involves both positional controls and rotational controls. Section 7 details how to improve the accuracy of the simulation with motion retargeting. Section 8 presents the experimental results. Finally, in Section 9, we conclude the paper, as well as discuss the limitations and future research directions.

2. RELATED WORK

Controlling the movement of virtual characters has been a popular research area in last two decades. The existing motion synthesis techniques can be divided into two categories: data-driven approaches and physics-based simulations. We first review the previous work in using live captured body movements of the user for controlling virtual characters. Then, we review researches in dynamics simulations using the proportional-derivative (PD) controller, which is a fast and effective way to simulate human motions.

2.1 Intuitive Control for Virtual Characters

Controlling the full body motion of the virtual characters is a challenging problem as the number of Degrees of Freedom (DOF) of the character is usually high. While techniques such as Motion Graphs [11, 12] have been widely used for controlling characters with pre-captured motions, they control characters with offline captured motions. In this research, we are more interested in algorithms that can deal with live captures.

Previous works try to control the full body motion with low-dimensional signals such as the positions of a few reflective markers [3], the readings of a small number of inertial sensors attached to the upper body [14], and the readings from a few 3D accelerometers attached to the limbs [23, 25]. The motion of the character can be estimated by referencing a motion database according to the low-dimensional signals. However, the resultant quality depends heavily on the size and quality of the database, as only a limited DOF are captured during run-time. With the Microsoft Kinect, it is now possible to capture motion with more DOF that can contribute to the synthesis quality. Shiratori et al. [22] propose to attach Wii controllers on a user's legs and synthesize walking cycle directly by extracting physical parameters from the input signals. However, the algorithm is specific for locomotion, making it difficult to be applied in other kinds of VR applications.

On the other hand, live performance of the user has been used for controlling the virtual character by full body motion capture [10, 13, 17, 21]. While we share similar interests with [13, 17] in controlling virtual characters using the live performance of the user to interact with the objects in the virtual world, our method tackles some of the problems that have not been addressed and explored in the previous works. In [17], the live performance is captured by an optical motion capture system. With the large number of markers used, the positions and orientations of the body segments can be calculated even when some markers are occluded. However, for affordable devices such as the Microsoft Kinect, the number of tracked points is extremely limited. Liu and Zordan [13] try to apply pre-recorded motions in a database with Kinect to simulate more realistic movements. However, it is unclear how the motion database is constructed and how it affects the quality of the resultant motion.

2.2 PD Controller for Physics-based Systems

In physics-based simulations, forward dynamics has been used extensively for gait simulations [19, 26]. One of the popular implementations of these systems is the proportional-derivative (PD) controller. A wide variety of motion can be synthesized with the controller, such as walk-to-run and run-to-walk transition [7, 22], as well as walking with different step length [8]. Usually a finite state machine is used to connect multiple PD controllers and handle the transition between controllers. Apart from locomotion, athletic motions such as cycling and handspring vaulting [9], and somersault motion [18] can be generated. Such systems require a certain amount of manual design to create different types of motions. PD control is also suitable to generate responsive motions such as the falling back motions when one is being pushed [27].

Since generating realistic full body motions require carefully designed dynamics systems, it is proposed to control characters by simply tracking captured human movements, such that the movements can react to external forces [1]. To speed up the simulations, simplified polygons model can be used to represent the characters with high degrees of freedom [16]. In this research, we use PD controller to track a posture that is computed from the captured motion and the best matched motion from a motion database.

3. SYSTEM OVERVIEW

The overview of our proposed method is shown in Figure 1. At every time-step, the posture of the user is captured by Kinect. The streamed postures contain only limited number of joints and are usually incomplete due to occlusion. We propose to search for a matching posture in a motion database to provide a reasonable estimation on the missing information. The Kinect posture, database posture and optional environment constraints are fed into a physical simulator as positional and rotational constraints. The simulator then solves for the final posture and simulates the resultant scene.

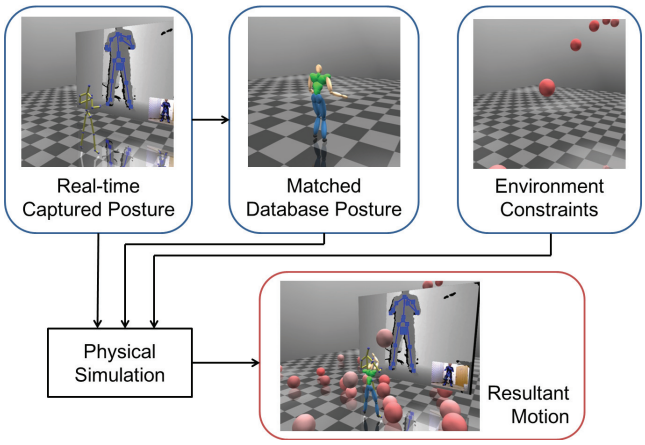


Figure 1: The overview of our proposed framework.

3.1 Contributions

We have two major contributions in this paper:

- We propose a unified framework, which combines positional controls and rotational controls, to model character movements in a physically simulated environment based on real-time captured motions. On one hand, simulated characters can resemble any posture produced by the users similar to kinematics motion synthesis. On the other hand, they response to external forces and are capable of interacting with virtual objects as in dynamics simulations.

- Motions captured from real-time system such as the Microsoft Kinect usually do not provide the necessary degree of freedom (DOF) required to control a virtual character. This is because of the limited number of tracking joints and the occluded/noisy joints that cannot be captured. We propose to substitute missing DOF from a motion database by searching for the best matching posture. To enhance the computational efficiency, we implement the database in a reduced space using Principal Component Analysis (PCA).

4. PHYSICAL WORLD MODELLING

In this section, we describe how we model the characters and the environment in the simulator.

We use the Open Dynamics Engine [24] to simulate the physical world. Each character is represented by 19 body segments and 20 joints according to the Kinect skeleton definition. Since Kinect does not enforce the bone length, we set the size and the mass of each segment according to [2]. Segments are modelled with capsules for efficient collision detection, and the joints are modelled with ball joints which indicating that each segment has 3 degrees of freedom in rotation. Although it is possible to model some specific joints such as the elbows and the knees with hinge joints to enhance the realism of the characters, the system will need to solve an inverse kinematic problem to calculate the orientation of their respective parent joints (i.e. the shoulder and the hips), which increases the computational cost. The trade off between the enhancement of realism and computational efficiency shall be adjusted according to the availability of computational resources.

Environment objects in our systems are modelled with primitive shapes. We create an infinity large plane in the ODE world as the floor plane, which provides supporting force to the characters and the objects. Gravity is implemented such that when no control force is applied, the characters and objects fall onto the ground naturally.

5. REFERENCE MOTION SELECTION

In this section, we present the process of selecting a reference posture that is the most similar to the posture captured from Kinect. We will explain the proposed method to construct the motion database, enhance the computational cost by dimensionality reduction techniques and select the best matching posture.

5.1 Motion Database Construction

Here, we explain how we compose our motion database.

We create the motion database by motions captured from a commercial optical motion capture system. We retarget the captured motions to the character definition explained in Section 4 using commercial software. We also remove the global rotation along the vertical axis and the global 3D translation for normalization. Each posture is represented by a set of joint positions (P_d), as well as a set of joint rotations along the joint axes (Θ_d). To enhance the efficiency of run-time database searches, we calculate the sum of squared differences of joint positions, and remove similar poses if the differences is smaller than a predefined threshold.

The motions that should be contained in the database depend on the target application. The idea is that we compose a database with the motions that the users are expected to perform. In our implementation, our database includes boxing motions, sword fighting motions, walking motions, as well as general exercising motions. The unfiltered database contains roughly 1500 postures, which is then filtered into around 300. Our database is relatively compact due to the scope of the target motion. However, if the application requires the user to perform a large variety of motion, such as dancing in different style, a larger database will be needed.

5.2 Motion Comparison

Here, we explain how we apply PCA to reduce the computational cost of database searches with the Kinect input.

During run-time, we obtain a set of joint positions (P_k) from Kinect, and retrieve the best match poses from the database (P_d). While we can consider the P_k and P_d as two point clouds and conduct motion comparison [11], this costs computational power unnecessarily, as there exists an intrinsic redundancy among the joints in the postures. We proposed to reduce the dimensionality of the motions by PCA to improve the performance of database queries. In other words, we reduce P_d into a reduced representation P_{dr}. During run-time, the projection matrix calculated by PCA is used to reduce the Kinect joint positions P_k into P_{kr}. We then conduct a search to find the most similar posture between P_{dr} and P_{kr}, by considering their sum of squared differences. Although more dedicated algorithms such as k-d tree can further speed up the searching process, we find that brute force searches in the reduced space is fast enough for real-time applications. In our implementation, P_d has 60 dimensions and P_{dr} has 7. Please refer to Section 8.4 for the analysis on deciding the optimal dimension of the reduced space.

In the situation when a joint is missing in the data obtained from Kinect due to occlusions or the restricted capture volume, we assume the joint position to be the mean value of that joint from all posture in the database, and compose P_{kr} accordingly. Empirically, we find that this assumption works well in database queries even when a few joints are missing.

6. POSTURE SOLVER

In this section, we explain how we combine the Kinect posture, database posture and optional environment information into a set of constraints. We separately control the target positions and the target rotations along the joint axes, and solve for the final posture with the physical simulator. At the end of this section, we discuss why our hybrid controller is superior to a pure positional/rotational controller.

6.1 Positional Constraints

Here, we explain how we generate positional constraints by combining the Kinect posture and the database posture. Then, we present the optional constraints that can be obtained from the environment. Finally, we give details on how to calculate the control forces to satisfy the positional constraints.

At each frame, we obtain a set of joint positions P_k from Kinect. Since P_k may be incomplete due to the run-time tracking error, we substitute the missing joint positions from the matched database posture P_d that is found as described in the previous section, and create the resultant set of positions P_e. Notice that P_e is an invalid posture in which segment lengths are not maintained, as it is created by simply replacing missing joints with those in P_d. Thus, we cannot directly force the character to perform P_e. Instead, we consider P_e as a set of positional constraints defined for every joint, and implement a tracking system to track the constraints.

We can include optional constraints based on the requirement of the application or the environment. For example, we can change the target location of a hand in a reaching motion such that the character can touch a virtual object. In such a case, we replace some DOF in the positional constraints defined in P_e with a calculated position. Notice that we only need to adjust the subset of joints that are explicitly related. For example, adjusting the positional constraint of the hand in a reaching motion results in the corresponding movement of the lower arm and the upper arm. The major advantage of using positions, instead of rotations, to represent the constraints here is that they are more trivial to human understanding and facilitate easier motion editing.

With the positional constraints expressed in P_e, we calculate the control force for each joint, and drive the character to fit into all constraints. In each time step, the control force is calculated by a PD controller:

$$F = K_e(p_{target} - p_{current}) + K_d(p'_{target} - p'_{current}) \quad (1)$$

where p_{target} is the target position of a joint defined by P_e, $p_{current}$ is the current position of the joint, p'_{target} and $p'_{current}$ are the respective derivatives, K_e is the elasticity gain and K_d is the damping gain. A high K_e can improve the responsiveness of the character, while a high K_d produce more stable movements. We manually tune the smallest possible K_e and K_d, as a system with high control forces is usually not stable, and use the pair to control all joints. Furthermore, the magnitude of the resultant force F is bounded by a predefined value to avoid unexpected high control force while the target values are very different from the current ones.

Notice that positional constraints implemented with control forces are not guaranteed to be met in the final result. If we wish to ensure meeting a specific constraint, we have to further define a virtual ball joint to connect the joint with a constraint position, which can either be on another object or in the empty space. This allows the constrained joint to rotate around the target position, but not moving away from it, even when forces are applied. Notice that if too many virtual joints are created, there may be incompatibility among them, such as fixing the feet on the floor while requiring the hand to move to an unreachable position. In such a situation, the posture solver will fail and the segment lengths will no longer be maintained.

In our implementation, we use virtual joints to fix the supporting feet on the floor. At the moment when the constraint positions of the feet touch the floor, we use virtual joints to fix it until the constraint positions move away from the floor. Furthermore, we use virtual joints to emulate the effect when a character holds an object such as a mace by fixing the hand with the object. When solving for the final posture, ODE maintains virtual joints as hard constraints, and coordinates other available DOF to achieve the desired movement. Empirically, we found that applying virtual joints on end effectors produces plausible results.

6.2 Rotational Constraints

Here, we explain the rotational constraints that control the rotation of joints along the joint axes.

To control a full character, apart from the positional constraints explained in the previous section, we also need the joint orientations. We define the rotation along the joint axes obtained from Kinect as Θ_k. Due to the limited number of tracking points in Kinect, Θ_k is incomplete whenever one or more point is not tracked.

Since the matched database posture P_d is similar to the Kinect one P_k, it is reasonable to assume that the corresponding rotational information, Θ_d, is similar to Θ_k as well. We substitute the missing DOF of Θ_k with those in Θ_d, and create the resultant set of rotation along the joint axes, Θ_e.

We formulate the control torque in ODE for each joint with a PD controller:

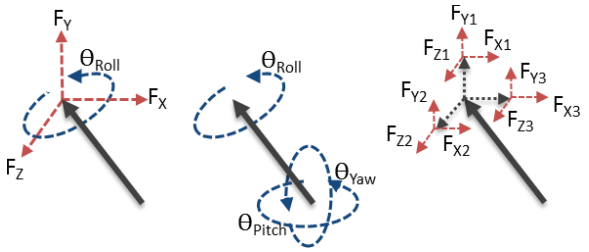
$$T = K_\epsilon(\theta_{target} - \theta_{current}) + K_\delta(\theta'_{target} - \theta'_{current}) \quad (2)$$

where θ_{target} is the rotation of a joint along the joint axis defined in Θ_e, $\theta_{current}$ is the current rotation, θ'_{target} and $\Theta'_{current}$ are the respective derivative, K_ϵ and K_δ are the hand tuned elasticity gain and damping gain. Similar to the force calculation, the torque T is bounded by a predefined value.

6.3 Physical Simulation

Here, we describe how ODE handles the control forces and torques to simulate the final posture.

For each joint, we apply the control forces calculated in Equation 1 to control the 3D translation, and the control torques calculated in Equation 2 to control the rotation along the joint axis, as shown in Figure 2(a). The physical simulation engine ODE maintains the segment length



(a) (b) (c)

Figure 2: Different control schemes for a joint, with the red arrows representing positional control, and the blue arrows representing rotational control. (a) The proposed control scheme based on 3D positional control and rotational control along the joint axis. (b) Pure rotational control. (c) Pure positional control by sampling extra control points with fixed offsets.

and segment connectivity defined by joints while applying the control forces and torques. The resultant posture is the equilibrium state of the character and it represents the posture that can satisfy most of the constraints.

In case a character is being pushed or hit, extra control force is applied to simulate the impact. Since the segment length is fixed, when a force is applied to a joint, the rest of the joints will be dragged to such a direction. The advantage of using PD controller for tracking the live performance in physical simulation is that the virtual character can adapt to unexpected collisions in the environment while trying to track the target posture given by the user. As a result, realistic movements of the virtual character responding to a dynamic environment can be generated.

Although it is possible to obtain all 3 dimensions of rotation and use Equation 2 to control the overall system as shown in Figure 2(b), we found that the result is sub-optimal. This is mainly because the actual segment movement after rotating the joint depends on the length of the joint, while human is sensitive to positional movements. Thus, to obtain the best result, K_ϵ and K_δ have to be tuned individually for each joint, as opposed to using the same value of gains for all joints in our system. Furthermore, positional based requirements for specific joints, such as fixing the supporting feet on the floor, will require solving an inverse kinematic problem, which requires extra computational effort.

Similarly, it is possible to sample multiple control points based on fixed offsets from a joint to represent its rotation along the joint axis as shown in Figure 2(c). Then, we can use Equation 1 to model all constraints. However, this increases the complexity of the system unnecessarily, and is likely to result in unstable simulations in the physical engine due to the large number of constraints. In our system, we construct a unified framework taking into account both controlling forces and rotational torques along the joint axes, with the latter being a supporting element. We find that it works efficiently and the gains in the equations can be tuned with ease.

7. MOTION RETARGETING

In this section, we explain how we retarget the Kinect posture into a standard size to further improve the performance of the system. While previous works [4, 6] can retarget motion for characters of different sizes, our retargetting algorithm is significantly simpler and efficient. This is because (1) the characters that we consider have the same joint hierarchy definition, and (2) our posture solver provide support for inverse kinematic, thus we do not need to solve it during the retargetting stage.

Since Kinect does not maintain the bone lengths of the tracked character, users with different body size have different character dimensions. In addition, the bone lengths may change during a capture session when the joint positions are not accurately recognized. To improve the stability of the simulation, we retarget the Kinect posture into the character dimensions we defined in ODE such that database matching (Section 5) and constraints definition (Section 6) can be more accurate.

Without loss of generality, for a Kinect joint i with position P_k^i and its parent joint j with position P_k^j, we calculate the normalized directional vector:

$$d_k^i = \frac{P_k^i - P_k^j}{|P_k^i - P_k^j|} \tag{3}$$

The retargeted position of the joint i is calculated as:

$$P_{k'}^i = P_{k'}^j + d_k^i \times L(i,j) \tag{4}$$

where $P_{k'}^j$ is retargeted position of the parent joint j, $L(i,j)$ indicates the bone length between joint i and j designed for the virtual character. Since the retargeted position of a joint depends on its retargeted parent joint, this process have to be started with the joint in the top level of the body structure hierarchy.

8. EXPERIMENTAL RESULTS

In this section, the results generated using our method are presented. We first show the motions created by our method in different scenarios. Next, we evaluate the database matching accuracy. The readers are referred to the attached video for the results.

All experiments ran in real-time on a computer using a single thread of an Intel i7-2600K processor. The proposed method is implemented on Windows with Visual C++, and Microsoft Kinect SDK [15] version 1.5 is used to obtain the live captured motion stream.

8.1 Interacting with Boxes

In the first experiment, the character was controlled by the user to lift and carry a bulky box as shown in Figure 3(a). Notice that as our character was physically simulated, the hand of the character did not penetrate the boxes upon contact, even when the raw Kinect posture illustrated by the yellow skeleton did. Also notice that although the Kinect posture had significant size difference with the simulated character, the resultant motion appears natural due to the motion retargeting process.

In the second experiment, a large number of boxes were added to the virtual environment at random locations. The virtual character was controlled by the user to interact with the boxes by body movements such as punching, kicking and walking, as shown in Figure 3(b). The masses of the boxes were set to be small to magnify the effect of impacts.

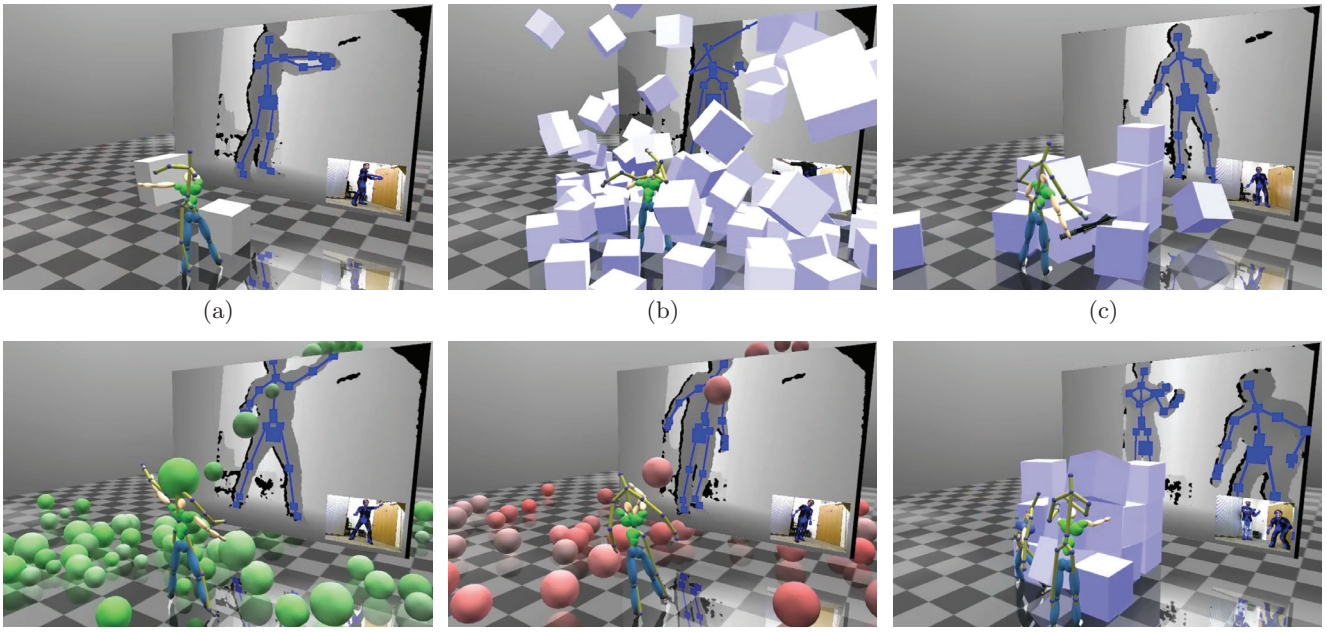


Figure 3: (a) The user controls the virtual character to grab and carry a box. (b) The virtual character interacts with a large number of boxes in the environment. (c) A simulated mace is attached to the hand of the character with a virtual joint. (d) The user controls the virtual character to interact will the balls. (e) Dodging motion is synthesized automatically by positional constraints. (f) Two characters are controlled by two users simultaneously.

In the third experiment, a physically simulated mace was connected to the hand of the character with a virtual joint as described in Section 6.1. The movement of the simulated mace was computed based on the motion of the character's hand, as well as the collision with other objects. The user controlled the character to interact with the boxes with the mace as shown in Figure 3(c). Notice that since the mace had a mass itself, it affected the hand movement of the character, which was similar to a human swinging a heavy object in the real world.

8.2 Interacting with Balls

In the first experiment, a large number of balls with random sizes shot towards the virtual character. The user tried to control the movements of the virtual character to interact with the balls as shown in Figure 3(d). Notice that when the balls hit the virtual character, the effect of external impacts applied onto the body segments was synthesized naturally.

We further show an example of automatically controlling a subset of body segments subject to selected external events. Specifically, we enabled the character to dodge an incoming object by raising its arms. To simulate such an effect, we set the target positions of the forearms around the head as positional constraints when the balls arrive. Figure 4(a) and (b) show the motions synthesized without and with the automatic dodging feature respectively using a non-user controlled T-pose, in which the arms of the character were extended outwards.

We applied the automatic dodging feature for a user controlled character as shown in Figure 3(e). By enabling this feature, the virtual character can automatically react to the balls shooting towards its head. This feature is particularly useful when the users do not want to control all the body segments by himself/herself, as well as synthesizing the reflective motions of human to increase the realism of the virtual character.

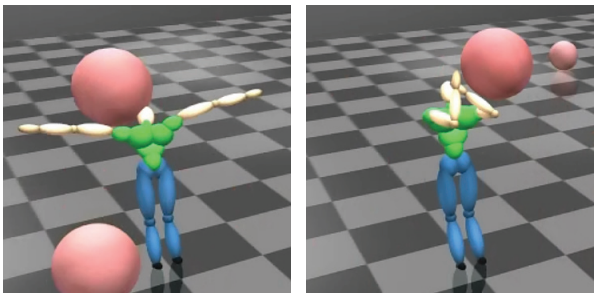


Figure 4: Motion synthesized (a) without the auto dodging feature. (b) with the auto dodging feature.

8.3 Multi-character interactions

In this experiment, two users interacted indirectly with each other in the virtual world by throwing boxes to each other (Figure 3(f)). Notice that with a single Kinect, it is difficult to capture close interaction due to occlusion. More research is needed to enhance the performance for tracking multiple interacting users.

8.4 Performance Evaluation

Here, we evaluate the effectiveness of using PCA to reduce dimensionality of the postures and the impact on the quality

of posture matching. We also discuss the computation cost of the system.

We record 600 frames of motion performed by a user with Kinect. Then, we reduce the dimensionality of the database motions and the Kinect motion to a specific value between 1 and 60, and perform motion matching. Finally, we compute the different between the retrieved database posture and the Kinect posture in the full dimensional space. The posture errors in a frame f is computed by:

$$e(f) = \sum_{i=1}^{60} (P_k^i - P_d^i)^2 \quad (5)$$

where 60 is the total number of joints in the full dimensional space, i is the joint index, P_k^i and P_d^i are the 3D locations of the i-th joint in the Kinect and selected database posture in the full space respectively. The percentage of error is calculated by:

$$\frac{\sum_{f=1}^{n} e_r(f) - \sum_{f=1}^{n} e_{60}(f)}{\sum_{f=1}^{n} e_{60}(f)} \times 100\% \quad (6)$$

where f is the frame number, $n = 600$ is the total number of frames, e_r is the posture error found using r dimensions matching with Equation 5, e_{60} is the posture error with the full 60 dimensions.

The results are plotted as the red line in Figure 5. It shows that when the dimensionality is significantly reduced, there is a high error rate as expected. However, the rate drops below 3% when the dimensionality is reduced to 7. Since then it decreases slowly when the dimensionality increases. Based on the results, we decide that reducing the dimensionality of the motion search space to 7 can balance the trade-off between efficiency and accuracy. The finding agrees with the work from Safonova et al. [20].

The average computational time required for searching the database and synthesizing the motion for a single character is plotted as the blue line in Figure 5. When using 7 dimensions, the time required is roughly 0.65 ms, in which 0.10 ms is for database search and 0.55 ms is for motion synthesize. The time for motion synthesize does not change significantly for different degree of freedom. When comparing to the full dimension search, 24% of computational cost is saved if 7 dimensions is used.

Notice that performance mentioned above does not consider the time required to obtain data from Kinect, as well as to render the outcome. With everything included, the computational cost is roughly 8.5 ms per frame when using 7 dimensions. This means that the system can run 4 times faster than real-time, although the frame rate is limited by Kinect in practice.

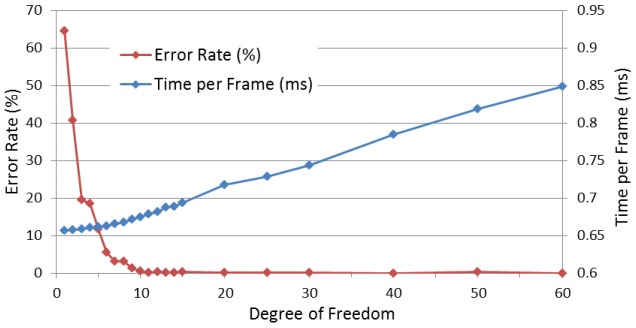


Figure 5: The results of posture matching error and computational cost with respect to the corresponding dimensionality of the postures.

9. CONCLUSIONS AND DISCUSSIONS

In this paper, we proposed a unified framework to track live captured motions from Kinect using physically simulated characters. Our framework can synthesize any posture efficiently using external forces and torques calculated from a PD controller. To overcome the problem of missing DOF of the postures captured from Kinect, we further proposed to select an appropriate posture from a dimensionality reduced motion database to estimate the required information. We demonstrated that the proposed PCA-based motion selection approach is computationally efficient while keeping the matching error under a reasonable level. Experimental results show that our method robustly synthesized the user controlled characters that interact with virtual objects in a physically simulated environment, even when the motions captured from Kinect were very noisy.

While Kinect is used as the motion capture system in this paper, our proposed framework is general and can be applied with different hardware. For example, in traditional optical motion capture system, it is very common that some track points are missing due to tracking error. Our system can substitute the missing DOF from a matched posture and synthesize stable character movements in real-time. Similarly, the framework can be applied to synthesize full body movement from the Wiimote controllers, although further research is required to effectively find a matching posture from the extremely limited DOF provided from these controllers.

While we show the effectiveness of the proposed method, there are some limitations. First of all, the database posture matching process becomes inaccurate when the number of joints recognized by Kinect is dropped significantly. This usually happen when capturing the whole body rotation of the user. As a common limitation of any single camera motion capture system, when the user is not facing the camera, the captured motion becomes inaccurate since a large portion of body segments are occluded. One possible solution is incorporating multiple cameras in a capturing session.

The accuracy of the database searches also depends on the content of the database. If the user performed motion is very different from those in the database, the returned posture may not match the input motion well. However, we observe that most applications have their own set of target objectives, which can help to identify the target motion that should be included in the database. In the future, we are interested in a more sophisticated motion completion scheme such as taking into account the temporal coherency of the joint locations. Another interesting direction is to integrate non-vision based motion capture devices, such as inertial sensors, into our proposed method to improve the robustness of the framework.

10. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments and suggestions. We also thank Mr. Chris Blythe for his help in producing the demonstration video and revising the paper.

11. REFERENCES

[1] Y. Abe and J. Popović. Interactive animation of dynamic manipulation. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 195–204, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[2] H. G. Armstrong. Anthropometry and mass distribution for human analogues. volume 1. military male aviators, 1988.

[3] J. Chai and J. K. Hodgins. Performance animation from low-dimensional control signals. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 686–696, New York, NY, USA, 2005. ACM.

[4] K.-J. Choi and H.-S. Ko. On-line motion retargetting. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, PG '99, pages 32–, Washington, DC, USA, 1999. IEEE Computer Society.

[5] S. Coros, P. Beaudoin, and M. van de Panne. Generalized biped walking control. *ACM Trans. Graph.*, 29(4):130:1–130:9, July 2010.

[6] M. Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[7] J. Hodgins. Biped gait transitions. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2092–2097 vol.3, 9-11 Apr 1991.

[8] J. Hodgins and M. Raibert. Adjusting step length for rough terrain locomotion. *Robotics and Automation, IEEE Transactions on*, 7(3):289–298, Jun 1991.

[9] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1995. ACM.

[10] S. Ishigaki, T. White, V. B. Zordan, and C. K. Liu. Performance-based control interface for character animation. *ACM Trans. Graph.*, 28(3):61:1–61:8, July 2009.

[11] L. Kovar, M. Gleicher, and F. H. Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, 2002.

[12] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[13] C. Liu and V. Zordan. Natural user interface for physics-based character animation. In J. Allbeck and P. Faloutsos, editors, *Motion in Games*, volume 7060 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin Heidelberg, 2011.

[14] H. Liu, X. Wei, J. Chai, I. Ha, and T. Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 133–140, New York, NY, USA, 2011. ACM.

[15] Microsoft Corporation. Kinect for windows SDK programming guide version 1.5. 2012.

[16] H. Mitake, K. Asano, T. Aoki, S. Marc, M. Sato, and S. Hasegawa. Physics-driven multi dimensional keyframe animation for artist-directable interactive character. *Computer Graphics Forum*, 28(2):279–287, 2009.

[17] N. Nguyen, N. Wheatland, D. Brown, B. Parise, C. K. Liu, and V. Zordan. Performance capture with physical interaction. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 189–195, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[18] R. Playter and M. Raibert. Control of a biped somersault in 3d. *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, 1:582–589, 7-10 Jul 1992.

[19] M. H. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph.*, 25(4):349–358, 1991.

[20] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 514–521, New York, NY, USA, 2004. ACM.

[21] H. J. Shin, J. Lee, S. Y. Shin, and M. Gleicher. Computer puppetry: An importance-based approach. *ACM Trans. Graph.*, 20(2):67–94, Apr. 2001.

[22] T. Shiratori and J. K. Hodgins. Accelerometer-based user interfaces for the control of a physically simulated character. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–9, New York, NY, USA, 2008. ACM.

[23] H. P. H. Shum, T. Komura, and S. Takagi. Fast accelerometer-based motion recognition with a dual buffer framework. *The International Journal of Virtual Reality*, 10(3):17–24, September 2011.

[24] R. Smith. Open dynamics engine, 2008. http://www.ode.org/.

[25] J. Tautges, A. Zinke, B. Krüger, J. Baumann, A. Weber, T. Helten, M. Müller, H.-P. Seidel, and B. Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Trans. Graph.*, 30(3):18:1–18:12, May 2011.

[26] M. Van De Panne. Parameterized gait synthesis. *Computer Graphics and Applications, IEEE*, 16(2):40–49, Mar 1996.

[27] V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96, New York, NY, USA, 2002. ACM.