An Example-Based Motion Synthesis Technique for Locomotion and Object Manipulation

Andrew W. Feng* Yuyu Xu† Ari Shapiro‡

Institute for Creative Technologies, University of Southern California

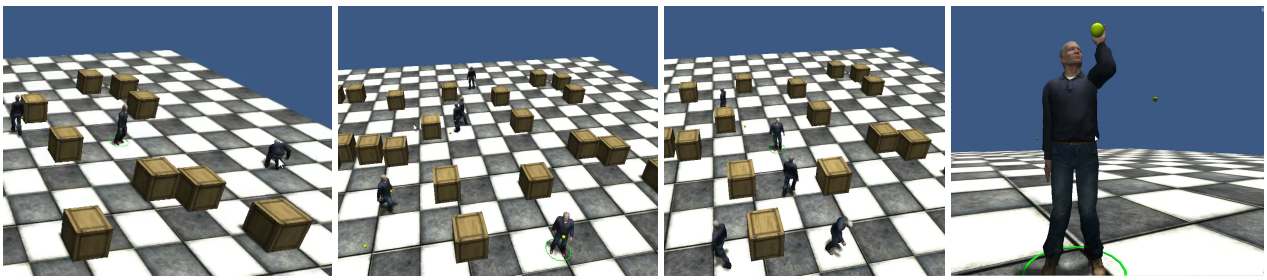


Figure 1: *A virtual character navigates through the environment, avoids obstacles, and then picks up a ball. Our system utilizes example-based locomotion, path finding, grasping, and gaze controllers to complete this reaching task in real-time.*

Abstract

We synthesize natural-looking locomotion, reaching and grasping for a virtual character in order to accomplish a wide range of movement and manipulation tasks in real time. Our virtual characters can move while avoiding obstacles, as well as manipulate arbitrarily shaped objects, regardless of height, location or placement in a virtual environment. Our characters can touch, reach and grasp objects while maintaining a high quality appearance. We demonstrate a system that combines these skills in an interactive setting suitable for interactive games and simulations.

CR Categories: I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism—Animation

Keywords: Character Animation, Object Manipulation, Motion Blending, Virtual Human

1 Introduction

A highly skilled virtual character needs to be able to interact appropriately with its virtual environment. Locomotion has been an active area of research, and the use of locomotion in animation systems is commonplace in both academic circles and in commercial applications, such as video games. Thus skillful walking and running have become mainstays of simulations of virtual characters. Without additional skills, however, a character's interaction with these virtual worlds is limited to navigation, avoiding large obstacles, and crowd-related behaviors. Many simulations and games require characters to interact directly with dynamic objects in their environments, and thus require skills related to reaching, grasping and touching. Object manipulation tasks, however, usually require that the target object be within an arm's length away of a virtual character's body. Such fine movements are usually only present in simulations that run in smaller environments than are typically used for locomotion simulation; on a scale of centimeters, rather than meters. Thus a skillful virtual character who can both locomote as well as manipulate objects must either have a very accurate locomotion system, one that can position a character precisely to the right point and orientation to where the manipulation can occur, or have an object manipulation system that is robust to handle variations in orientation and positions while still synthesizing a natural-looking animation result. In addition to arm movement, a skillful virtual character must be able to touch and grab objects using human-like hand postures. A convincing animated result must also respect the objects' shape and volume, ensuring no penetrations between the object and the hand. Also, in order for a character to generate natural-looking motion, the motion must not only appear human-like by itself, but be appropriate for its context. Thus, any reaching or grabbing motion should incorporate an attention model; people touching other objects often look at the object being touched or grabbed.

We present a highly skilled interactive virtual character by combining locomotion, path finding, object manipulation and gaze. We achieve a natural-looking result by using example-based locomotion and reaching techniques at interactive speeds, thus being suitable for games and other real-time applications. Our characters are capable of touching, grasping, holding and placing objects anywhere in the virtual scene at any elevation within their reach that they are capable of approaching. Note that we attempt to solve the generic object manipulation problem; reaching, grasping and touching arbitrary objects in a dynamic space. This is in contrast to many systems or techniques that are solving specific object manipulation tasks in fixed environments. These other techniques can synthesize motion by replaying motion capture or artistically-driven animations to manipulate specific objects at fixed locations. By contrast, we have no knowledge of the location, shape or placement of the objects. In addition, we preserve the natural look of the synthesized motion through enforcement of constraints that allow the character to grab objects while simultaneously turning their bodies in arbitrary directions, or play animations unrelated to the object manipulation tasks.

*e-mail: feng@ict.usc.edu
†e-mail: yxu@ict.usc.edu
‡e-mail: shapiro@ict.usc.edu

2 Related Work

Character Animation Control Controlling a virtual character to perform various tasks while producing realistic human motions is a challenging problem. Many previous methods have been proposed to address the problem using either kinematics control or physics-based control. Kinematic-based methods make use of existing animation data to produce new animations that satisfy the user constraints. One popular method is to segment the example motions into a set of sub-motions and build a graph structure by computing suitable transition points between sub-motions [Kovar et al. 2008; Lee et al. 2002a]. The animation control can then produce the character motions via suitable graph walk to satisfy user constraints, such as walking along a path. Further extensions are also developed by using motion interpolation or parameterized motions to improve the accuracy and interactivity [Safonova and Hodgins 2007; Heck and Gleicher 2007]. The work by [Safonova et al. 2004] uses existing motions to construct a low dimensional space, and perform optimization in the subspace to produce character motions with desired behaviors. Physics-based characters can produce physically-plausible animations based on the results of physical simulation. Seminal work of Raibert and Hodgins provides a basic control mechanism to generate locomotions through a simplified physical model [Raibert and Hodgins 1991] and basic athletic tasks [Hodgins et al. 1995]. The work in [Yin et al. 2007] also proposes a physics-based controller that can either be constructed manually or trained with motion capture data. Such physically-based methods are neither fast enough, nor accurate enough nor produce natural-looking results for our purposes. Hybrid kinematic-dynamic methods have been developed to exploit the advantages of each technique such as responding to contact [Shapiro et al. 2003], searching motion capture for proper responses to contact [Zordan et al. 2005] or tracking motion capture under simulation with kinemtically driven root joints [Wrotek et al. 2006]. Hybrid approaches can handle high-energy impacts between objects and characters, but aren't of principle importance to our system, whose focus is on navigation, low energy manipulation and attention instead. Alternatively, [Liu 2009] proposes an optimization based method to generate physically plausible hand motions. The work in [Jain et al. 2009] also applies optimization to synthesize physics-based character motions in dynamic environments. However, optimization-based methods such as these cannot typically run at interactive speeds.

Some modern game engines, such as Unreal Development Kit (UDK) [Epic Games 2011] also provide functionalities such as motion blending and inverse kinematics for motion control. While using similar building blocks, our system differs from these game engines in several ways. First, our system build a *motion parameterization* to translate high level control parameters such as speed or turning rate into suitable blending weights to provide precise motion control. Game engines usually provide a mechanism for motion blending, but require user to provide the weights. Moreover, the IK method adopted in game engines are based on heuristics such as cyclic-coordinate descent (CCD), which is difficult to adjust full body motions. On the other hand, our IK method is based on Jacobian pseudo inverse, which can solve for full body pose that satisfies the constraints while adapting to secondary goals such as a reference pose.

Object Manipulation. Several approaches have been investigated for animating a virtual character to perform various tasks and object interactions [Yamane et al. 2004; Huang et al. 2010; Huang et al. 2011; Aydin and Nakajima 1999]. The early work by [Rijpkema and Girard 1991] propose knowledge-based rules to procedurally generate grasping hand and finger motions. Yamane *et al.*[2004] propose an off-line method that combines motion capture examples and planning algorithm to generate object manipulation animations. Their method is able to generate natural looking and collision free arm motions at the cost of longer processing time. The method in [Huang et al. 2010] applies motor controllers with biomechanical rules to coordinate arm, spine, and leg movements to generate a full-body reaching motion including stepping. The work by Lv *et al.* [2011] also utilizes various reaching strategies from biomechanics. The motion is then generated by optimization in reduced dimension. Our method does not handle step and reach strategies, although it could if a set of set-reaching examples were included. Yahya and coworkers [1999] present a method to generate reaching pose from a pose database. Grasping hand posture is then determined procedurally based on object shapes. Similar to our goal, the work in [Huang et al. 2011] combines locomotion and upper body planner to control the virtual character reaching for a distant target. Their method amortizes computation costs over the whole planned motion to achieve real-time performance. Our method separates the motion blending from path planning, and produces the new motion on the fly in dynamic environments. The overall goal of our system is to build an integrated solution for reaching that integrates gaze, reach, and locomotion planning. In our current implementation, the system can only avoid obstacles in macro scale through path planning, unlike the ones in [Shapiro et al. 2008], [Huang et al. 2011], which can handle more cluttered environments with additional upper body planning. Our system can be run in real time, thus is more suitable for games and simulations.

Motion Blending. Motion blending provides a simple yet powerful method to generate realistic human motions by parameterizing existing motion data. New motions that satisfy a given parameter can be synthesized by interpolating motion data with appropriate weights. The main issue in motion blending is to construct an effective parameter space that is both accurate and efficient. Rose *et al.* [2001] formulates the problem as scatter data interpolation and uses radial basis function (RBF). To improve the blending quality of parameter space, they also propose to generate pseudo examples in regions where the accuracy is poor. Kovar and Gleicher [2004], on the other hand, apply K-nearest neighbors method to produce the blending weights using only examples in close proximity. Their method generates a densely sampled parameter space by randomly inserting pseudo examples. The work in [Huang and Kallmann 2010] takes a different perspective on the problem and solves it with direct optimization. Their method is able to compute the locally optimal blending weights that satisfy the spatial constraints in real-time. In our experiments, we found that the best blending method depends on the context of motions to be parameterized. For locomotion animation, where the mapping from parameter to blend weights is usually linear, it is more important to ensure the weights will vary smoothly when parameter changes. Thus a simple blending scheme based on barycentric coordinates works better to ensure visually pleasing results. On the other hand, a reaching motion forms a non-linear parameter space and requires better precision for grasping. Therefore methods that construct a dense parameter space are preferred for accuracy. We do not pursue the motion graph approaches such as [Kovar et al. 2008] and [Lee et al. 2002a], as they are generally not suitable for large numbers of character in real time, and can lack accuracy. In addition we chose not to use semi-procedural techniques such as [Johansen 2009] which perform accurate path following, but typically cannot achieve the high level of motion quality as can pure example-based methods.

Order	*Controller*	*Comments*
1	World offset	Global orientation and position
2	Idle motion	Underlying idle pose
3	Locomotion	Overrides idle pose during locomotion phases, ignored during idle states
4	Animation	Non-locomotive animations, can encompass entire body or just upper body during locomotion.
5	Reach	Allows reaching and pointing using arms
6	Grab	Hand control for touching, grabbing, and picking up objects
7	Gaze	Looking with eyes, head, shoulders and waist
8	Constraint	Allows constraints that may have been violated due to impact of preceding controllers (i.e. keeping character's hands on a table while turning to look at another object)

Table 1: *Controller stack showing flow of character state during evaluation step.*

3 Overview

Our animation system is designed around a hierarchical, controller-based architecture [Shapiro 2011; Thiebaux et al. 2008]. The state of the character is manipulated by series of controllers, with the output of one passed as the input to another. Each controller can either override, modify or ignore the state of the virtual character. The controllers know the state of the character during the previous step, as well as the state of the character during the current evaluation phase. The controller stack in our system, which controls the state data flow, is listed in Table 1 in the order of execution. Note that this hierarchy implies that the higher numbered controllers are given higher priority, since they can override the results of any lower priority controller. However, the controller stack simultaneously implements a generalization-specialization hierarchy; lower priority controllers typically control a greater number of body parts, while higher-priority controllers typically control fewer. For example, the idle motion controller typically produces motion for the entire body, while the grab and hand controller overrides motion for the hand and fingers.

Based on the above system, the reach action can be realized by a series of four different controllers : locomotion, grasp, gaze, and constraint. The locomotion controller generates the path and the corresponding animation to help the character navigate toward the reach target. Once the target is within the grasping distance of the character, the grasp controller generates a full-body motion to have the character pick up the target object. The gaze controller is also augmented to enhance the realism by producing eye and head movements toward the target. Finally, the constraint controller is applied to ensure that the motions from the aforementioned stages are combined together without violating the target goal and other user specified constraints.

Notation A characters pose $P = \{\tau, \Theta\}$ is determined by the root position τ and a set of joint angles $\Theta = \{\theta_1, \theta_2, \cdots\}$. We define $M^i = \{{P_{t_0}}^i, \cdots {P_{t_n}}^i\}$ as the i-th example motion, where $(P_t)^i$ is the pose state at time t. We further define $(M^i(ts \rightarrow te) = (P^i_{t_s} \rightarrow P^i_{t_e})$ as the subset of motion M^i in the time interval t_s to t_e. Note that since a motion may be played backward, it is not necessary that $t_e > t_s$.

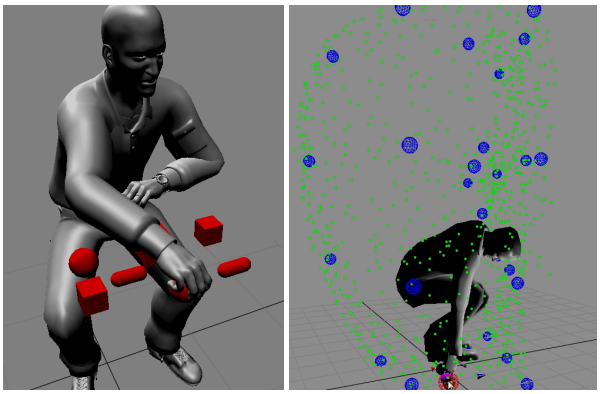


Figure 2: *(Left) Reaching and grabbing during sitting. Notice that the pinky of the right hand did not collide with the target object, and thus was allowed to blend into the grabbing pose, whereas the other fingers collided with the target object, and remain at the collision surface. (Right) Example-based reaching for right side of body. The blue spheres represent the original examples, while the green dots are examples interpolated from the original examples. Note that the character engages his entire body during the reaching motion, based on the example motions given.*

4 Locomotion

4.1 Parameterized Animation

The character locomotion is generated by blending a set of example motions. We use 19 different example animations of character for the motion interpolation. Each example animation M^i contains two walking or running cycles of character either moving straight, moving sideway, or turning around at various speed. We also manually segment each example motion M^i into a set of sub-motions $\{M^i(t_1 \rightarrow t_2), \cdots M^i(t_{n-1} \rightarrow t_n)\}$, where $t_k(k = 1 \cdots n)$ indicates the time stamp at k-th foot placement. This segmentation prevents the blending of motions over different foot placement cycles, and thus reduces the foot sliding artifacts at run-time. These animations are then parameterized in three dimensions using forward velocity v_f, turning rate ω, and sideways velocity v_s. The parameter values can be automatically extracted from the example motions, such as determining the average forward velocity of a motion. Thus each motion clip represents a parameter coordinate $\mathbf{p^i} = ({v_f}^i, \omega^i, {v_s}^i)$ in the three dimensional space, and new motion with novel parameters can be synthesized by interpolating the appropriate motions. There are several previous methods for constructing a parameter space for interpolation, including radial basis function [Rose et al. 2001], K-nearest neighbors [Kovar and Gleicher 2004], or direct optimization [Huang and Kallmann 2010]. Since the locomotion parameters such as velocity or turning rates can be linearly mapped to the blending weights, we choose to construct the parameter space using tetrahedrons. Given a set of example motions $M^i(i = 1..n_e)$ we generate the tetrahedralization $V = (T_1, T_2, \cdots T_v)$ to connect the parameter points $\mathbf{p^i}(i = 1 \ldots n_e)$. Here $T_j = (\mathbf{p^{j_1}}, \mathbf{p^{j_2}}, \mathbf{p^{j_3}}, \mathbf{p^{j_4}})$ represents a tetrahedron connecting four example parameter points. While there exists automatic method for generating the tetrahedralization [Si], its main application is for geometry reconstruction from large point sets. Since our locomotion examples consist only

less than 20 parameter points, we choose to manually define the tetrahedrons.

During run-time, given a desired forward velocity v_f', turning rate ω', and sideways velocity v_s', we can obtain a new parameter point $\mathbf{p} = \{v_f', \omega', v_s'\}$. We can use the new parameter point $\mathbf{p}'$ to search for a tetrahedron T_j in V that encloses $\mathbf{p}'$. Based on the fact that only one tetrahedron may contain $\mathbf{p}'$ and the barycentric coordinates are all positive when $\mathbf{p}'$ is inside a tetrahedron, we can obtain T_j by computing and testing the barycentric coordinates of $\mathbf{p}'$ for each tetrahedron in V. Once we obtain T_j, the example motions $(M^{j_1}, M^{j_2}, M^{j_3}, M^{j_4})$ associated with T_j will be selected for motion blending. We also compute the blending weights $\mathbf{w}$ based on the barycentric coordinates of $\mathbf{p}'$ in T_j. A resulting motion M' is then a concatenation of $\{M'(t_1 \rightarrow t_2), \cdots M'(t_{k-1} \rightarrow t_k)\}$, where

$$M'_{t_k \rightarrow t_{k+1}} = \sum_{n=1}^{4} w_{j_n} {M^{j_n}}_{t_k \rightarrow t_{k+1}}$$

is the sub-motion blending of example motions $(M^{j_1}, M^{j_2}, M^{j_3}, M^{j_4})$ from T_j. Figure 3 shows the locomotion parameter space and the resulting locomotion produced through motion blending.

We have found that the above example-based blending method produces a realistic result, although the quality of the resulting animation depends highly on the example motions. This locomotion engine does not use IK, and relies entirely on motion data blending to produce a smooth animation of the character moving at different speed and turning rates. This continuous control over movement parameters therefore connects nicely with the path finding components for steering the character.

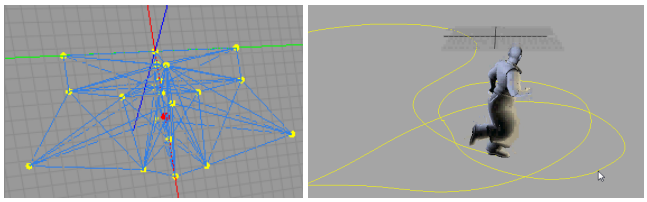


Figure 3: *(Left) Visualization of the example-based locomotion. Yellow dots on the red axis represent forward motions (walking, jogging, running, etc.) Yellow dots along the green axis represent turning motions. Yellow dots along the blue axis represent strafing, or sideways motions. (Right) Example-based locomotion and path.*

4.2 Path Finding

The above example-based locomotion produces natural looking animation based on the input parameters, and a user may change those parameters in real-time to help the character maneuver around. However, it is more intuitive for the user to simply provide a target destination and have the system figure a collision free path toward the goal in a dynamic environment. This is where a path finding system comes in handy to guide the character through obstacles. In our implementation, we use SteerSuite [Singh et al. 2009] as our steering system to handle path finding. Given a destination, the steering system computes a new path toward the target on the fly to avoid any obstacles along the way and continuously feed the movement parameters to example-based locomotion. The separation of locomotion from path finding simplifies the design and also allows for the development of each area separately. This enables us to switch to different steering algorithms without affecting other components. However, this separation also has a consequence; without knowing the limits of locomotion capabilities, the path planner may require movements that are sometimes unrealistically fast, and sometimes visually unappealing. For example, the path planner might decide to suddenly switch direction to avoid an obstacle faster than the locomotion system can realistically move the character, thus causing a discrepancy between the planned path and the actual path. This has been brought up in [Singh et al. 2010], but has not yet been fully explored. We also noticed that many path planners are tailored to handle large scale movement, such as traversing long distances around large objects, and very few path planners handle intricate movement in small areas and indoors. Improvement on both the capability of example-based animation and the steering algorithm to handle this type of environments would be an interesting direction for future investigation.

5 Grasp

Our system utilizes an example-based approach for grasping. The goal is to produce a natural looking motion for the character to pick up or put down the target object. We separate this task by synthesizing and combining two different motions – one that moves the arm near the target, and the other that controls the fingers to enclose the target object. This separation helps reduce the number of example motions needed to model different behaviors, such as pick up, put down, or simply touch the object.

5.1 Arm and Body Motion

The arm motion is generated by interpolating the motion examples. We use 32 different example motions to allow a character to point his arm to most objects within arms-length. Each arm uses 16 example motions from four elevations (high, mid-high, mid-low, low). In practice, motions created for one hand can be mirrored to the other hand in order to reduce the number of examples needed. The arm movements for a typical reach action can be separated into three different stages – move toward the target, hand on the target, return from the target. Thus in our system, each example motion M^i with time duration ${t_e}^i$ will also consist of the same stages. Specifically, a motion M^i starts with the character reaching for the target at time $t = 0$, putting his hand on the target at time $t = {t_a}^i$ and then returning to the rest pose at ${t_e}^i$. We segment an reach motion into these stages so we can handle the motion blending properly at each stage.

Similar to the example-based locomotion, we then parameterize the example motions in a three dimension space, using the apex position $\mathbf{p}$ of the end effector as the parameter. However, since the hand position can not be mapped linearly to the blending weights, it requires more care to build an accurate parameter space. A straightforward method to improve the accuracy of interpolation is to add more motion examples, but it would also be impractical to produce thousands of motions to fill up the parameter space. Therefore we adapt the method in Kovar et al [Kovar and Gleicher 2004] to build the parameter space using K-nearest neighbor (KNN) and pseudo-examples. Pseudo-examples are generated by randomly sampling the blend weight space to fill out the gap between motion examples. Once we have a dense set of examples, we build a k-D tree data structure for fast proximity query at run-time.

The run-time stage produces the reaching motion by blending the nearby examples. Given a new reach target $\mathbf{p_r}$, the k closest examples $(M_{n_1}, \cdots M_{n_k})$ near $\mathbf{p_r}$ can be efficiently found from the k-D tree. An reach motion $\tilde{M}(0 \rightarrow t_a)$ that moves the arm to the target can then be generated using motion blending :

$$\tilde{M}(0 \rightarrow t_a) = \sum_{j=1}^{k} w_{n_j} M_{n_j}(0 \rightarrow {t_a}^{n_j})$$

(A) (B)

Figure 4: *Examples of applying inverse kinematics in grasping motion. With only motion blending, the resulting motion may not precisely reach the target, as shown in (A). Accurate reaching results are generated in (B) by combining IK with blended motion.*

, where w_{n_j} is defined using the normalized inverse distance between $\mathbf{p_r}$ and $\mathbf{p_{n_j}}$ in the parameter space. Once the arm has reached the target, the returning motion $\tilde{M}_{t_a \to t_e}$ can be synthesized in a similar manner.

The above motion blending scheme can effectively reproduce a typical reach motion that start from a rest pose and then back to the original pose. However, there are cases when the character may reach for one target, then switch to another without returning to the original pose. This type of motion is outside of the capabilities of the original example motions. To generate such motion, we approximate it with pose interpolation. Specifically, given the current target position $\mathbf{p_c}$ and a new target $\mathbf{p_n}$, we compute the poses P_c and P_n respectively using the aforementioned blending method. A new reach motion is then generated by interpolating P_c and P_n. In our experiments, a linear interpolation results in passable but not high quality reaching animation between pose targets. Making use of the example motions to improve interpolation results is an open problem, and is left as the future work.

5.2 Full Body Inverse Kinematics

The above method generates precise arm motions that reach the goal when target position falls within the span of example motions. For the areas not covered by the example motions, the resulting motions may not be accurate enough to grasp the target. Therefore we apply inverse kinematics (IK) based on Jacobian pseudo-inverse to deform the blended motion for more accurate grasping. Given a desired end effector coordinate $\mathbf{x_r}$ and reference joint angles $\tilde{\Theta}$ from the previous section, the IK method solves for the joint angles increment $\Delta\Theta$. The resulting pose will have its end effector constrained at the target position while satisfying the pose constraint $\tilde{\Theta}$ as much as possible. A Jacobian matrix J is a linear mapping that transforms the differential changes in joint angles $\Delta\Theta$ to the end effector coordinates $\Delta\mathbf{x}$. Given current joint configurations $\Theta_c = \{\theta_{c1}, \cdots \theta_{cn}\}$, the (i, j) entries in matrix J are computed as partial derivative $\frac{\delta \mathbf{x_i}(\Theta_c)}{\delta \theta_j}$ between the j-th component of joint angles and i-th component of end effector coordinates. Thus the linearized relationship between $\Delta\Theta$ and Δx at Θ_c is approximated as $\Delta x = J(\Theta_c)\Delta\Theta$. During each IK update, the joint angle increments $\Delta\Theta$ is computed as :

$$\Delta\Theta = J^{+}\Delta\mathbf{x} + (I - J^{+}J)\Delta\mathbf{z}$$

where $J^{+} = J^{T}(JJ^{T})^{-1}$ is the pseudo-inverse of J, $\Delta\mathbf{x}$ is the offset from current end effector coordinates to target coordinates $\mathbf{x_r}$, and $\Delta\mathbf{z}$ is the desired joint angle increments toward target pose $\mathbf{z} = \tilde{\Theta}$.

The above IK method deforms a static input pose to satisfy the end effector constraint. However, when the input is a motion, natively applying the IK method at each time step causes the end effector to move abruptly toward the target. This is because the IK method is formulated to satisfy a static end effector constraint, and we have no control over how fast the results will converge to the desired target coordinates. Moreover, since IK is recomputed at each time step, we need to ensure the resulting joint angles Θ will not change abruptly and cause discontinuous motions. As we can see from the above formulation, the joint angle increments $\Delta\Theta$ depends linearly on $\Delta\mathbf{x}$ and $\Delta\mathbf{z}$. Thus if we can limit the magnitude of $\Delta\mathbf{x}$ and $\Delta\mathbf{z}$ during each time step, we can avoid a large $\Delta\Theta$ that causes discontinuity. For $\Delta\mathbf{z} = \Delta\tilde{\Theta}$, we limit the magnitude of $\Delta\tilde{\Theta}$ to be less than 5° during each update step. This prevents IK to suddenly change the current pose due to a large difference between current pose and desired pose. For $\Delta\mathbf{x}$, we handle it in a similar manner by smoothly moving $\mathbf{x_r}$;instead of setting a final target coordinates $\mathbf{x_r}$ and applying the same constraint throughout a reaching animation, we need to infer a time varying $\mathbf{x_r}(t)$ as the target constraint for each time step. Given a reach motion $\tilde{M}(0 \to t_a)$, we can obtain the pose $\tilde{P}(t_a)$ at $t = t_a$ and the difference $\Delta\mathbf{p} = \mathbf{p_r} - \tilde{\mathbf{p}}(t_a)$ from its end effector $\tilde{\mathbf{p}}(t_a)$ to the desired target $\mathbf{p_r}$. We can then compute $\mathbf{x_r}(t_i)$ at time step t_i based on the current time step and $\Delta\mathbf{p}$:

$$\mathbf{x_r}(t_i) = \alpha(t_i)\Delta\mathbf{p} + (1 - \alpha(t_i))(\mathbf{x_r}(t_{i-1}) - \tilde{\mathbf{p}}(t_{i-1}))$$

, where $\alpha(t_i) = \frac{t_i - t_{i-1}}{t_a - t_{i-1}}$, $\tilde{\mathbf{p}}(t)$ is the end effector coordinates from blended motion at t, and $\mathbf{x_r}(t)$ is the end effector constraint with $\mathbf{x_r}(0) = \tilde{\mathbf{p}}(0)$. The above formula moves the constraint toward $\mathbf{p_r}$ as $t \to t_a$ to produce a reaching animation that satisfy the constraint exactly at $t = t_a$. Since both the target constraint $\mathbf{x_r}$ and target pose $\mathbf{z} = \tilde{\Theta}$ are changing smoothly, the reaching motion is also smooth without artifacts. Moreover, the blending weights can vary at each time step, and therefore the target position can be changed on the fly even when the reaching is in process. Figure 4 shows the comparison of resulting poses with and without IK. By deforming the blended pose with IK, we obtain more accurate hand placement.

5.3 Hand Motion

The motion blending with inverse kinematics enables the character to point his hand at a precise location in space. For a full grasping action, the fingers also need to be animated to enclose the target object. Since target objects can vary in both shapes and sizes, a character may need to place his hand at different position and orientation to naturally grasp the object. For example, a spherical object can be picked up with any hand orientation comfortable for the character, while a long, narrow object will cause the hand to reorient so that the long axis is parallel to the palm, allowing the fingers and thumb to close around the axis. Thus it is impractical to create an example animation for each of them. To simplify the problem, we divide the object shapes into a few categories, and use heuristic that determines the orientation of the hand that is needed to grab that shape.

Hand Orientation We use three types of shapes to represent a typical object – sphere, box, and capsule. A sphere represents an object that can be naturally grasped in any orientation, while boxes and capsules represent objects that need to be grasped in specific orientations. Let $\tilde{P}(t_a)$ be the natural hand pose from blending, $\mathbf{p_h}(t_a)$ be the hand position, and $\mathbf{q_h}(t_a)$ be the facing direction of the backhand from $\tilde{P}(t_a)$. We devised the the following heuristics to infer the hand orientation :

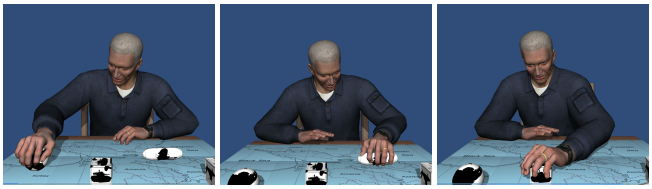


Figure 5: *Examples of grasping objects with various shapes. Our system determines hand orientations and finger poses based on object shapes and collision detections.*

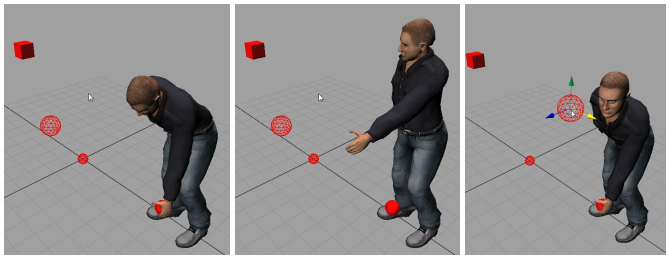
- Sphere : We compute the new hand position as $\mathbf{p}'_{\mathbf{h}}(t_a) = \mathbf{c}+\mathbf{q}_{\mathbf{h}}(t_a)r$, where $\mathbf{c}$ is center of the sphere and r is the radius. This moves the hand outside the sphere while maintaining the same orientation.
- Box : The new hand orientation $\mathbf{q}'_{\mathbf{h}}(t_a) = \mathbf{q}_{\mathbf{n_i}}$ is the normal vector of the i-th face on the box, where the dot-product $\mathbf{q}_{\mathbf{n_i}} \bullet \mathbf{q}_{\mathbf{h}}(t_a)$ is the maximum among all six faces. With this new orientation, we also compute the new hand position $\mathbf{p}'_{\mathbf{h}}(t_a) = \mathbf{c} + \mathbf{q}_{\mathbf{n_i}} r_i$ to move the hand outside the box, where $\mathbf{c}$ is center of the box and r_i is the distance from $\mathbf{c}$ to the i-th face.
- Capsule : A capsule represents a thin and long object that should be picked up from the side. We define $\mathbf{q}_{\mathbf{c}}$ as the long axis of capsule, and the new orientation is computed as $\mathbf{q}'_{\mathbf{h}}(t_a) = \mathbf{q}_{\mathbf{c}} \times (\mathbf{q}_{\mathbf{h}}(t_a) \times \mathbf{q}_{\mathbf{c}})$. Once we have new orientation $\mathbf{q}'_{\mathbf{h}}$ to align the hand with long axis, the new hand position can be computed as $\mathbf{p}'_{\mathbf{h}}(t_a) = \mathbf{c} + \mathbf{q}'_{\mathbf{h}} r$ where $\mathbf{c}$ is center of the capsule and r is its radius.

Since the IK results also depend on hand positions and orientations, our system also feedbacks the new $\mathbf{p}'_{\mathbf{h}}$ and $\mathbf{q}'_{\mathbf{h}}$ back to the IK as constraints to produce a reaching motion with correct hand placement.

Hand Pose Interpolation Once we have reoriented the hand, the hand motion can be generated to have fingers enclose the target object. The hand motion is generated by interpolating between a rest pose and a pinched pose. We use two poses P_r and P_g to represent the hand configuration when the hand is relaxed and when the hand is fully closed. To produce a hand motion, we gradually interpolate P_r toward P_g and detect any collisions between each finger segment and the target object. We treat each finger segment as a capsule to allow fast collision test. Whenever a collision occurs, the collided finger segment and its parents are locked and removed from pose interpolation. This ensure that the hand motion results in a tight grasp without penetration.

6 Gaze

Gaze is an important aspect for human motion. It provides the intent from the character and improves the realism of the overall motion. Although the gaze does not directly affect a grasping motion, it adds human-like behavior showing the intent and focus when interacting with the object. The synthesize reaching and grabbing motion produces a more convincing result when appropriately timed gazing is added to the motion. In our system, we use the method from [Thiebaux et al. 2009] to synthesize the gaze motion. Given a target object, the gaze controller procedurally rotates a hierarchy of joints, including the eyes, neck, shoulders and waist to move the characters eye beams toward the target. Incorporating the upper body movements in gaze controller provide a larger coverage of potential target than using only neck and eyes. To integrate the gaze into grasping motion, we initiate the gaze controller to look at the object before reaching is started, and maintain the gaze through the grasping motion until the object is no longer the target.



(A) (B) (C)

Figure 6: *Constraint controller helps maintain correct poses when several controllers are working together. A character is picking up the object in (A), and the action is disrupted by the gaze controller in (B). By adding a constraint controller, the character is able to engage in full body gaze while maintaining the same grasping pose.*

The procedural gaze model we use tends to focus exactly on the target object. While this gives us precise control to point the eye beams toward the target, it may seem less life-like at times. Therefore we also generates a saccade motion based on the work by [Lee et al. 2002b] or [Deng et al. 2005] to augment the gaze controller. The saccade model uses a statistical eye movement model to generate eye movements over time. Combined with the gaze, it adds randomness in the eye movements and improves the looking motion. In the future, we also hope to develop similar example-based movement models for head and upper body to further increase the quality of gaze motion.

7 Constraint

From the aforementioned methods, we have a series of controllers – locomotion, grasping, and gaze – working together to enable the character to perform the reaching task. Our controller system is designed as a hierarchy that allows each controller to override and modify the character state from the previous controller. Thus we can develop each controller to handle different task and combine them together to form the character motion. However, since a controller may have effects that replace the effects of earlier one, there may be conflict between their individual task. For example, the gaze controller can engage the entire spine during a gaze behavior when orienting a character upper body towards a gaze target. This may disrupt the motion from grasping controller since they both affect the spine joints. Since the gaze is evaluated after the grasping, this causes the hand to move away from the target object and may produce unnatural pose.

We solve this problem by adding a constraint controller at the end of hierarchy evaluation. The core component of constraint controller is the Jacobian-based IK solver from previous section. It takes an input pose $P_r(t)$, and modify it to ensure certain properties such as hand position and orientation are satisfied. Similar to IK stage, it first initializes an output pose $P_{out}(0) = P_r(0)$. Then during each update with time increment δt, it solves for new joint angle increments as $\Delta\Theta(t) = J^+\Delta\mathbf{x}(t)+(I-J^+J)\Delta\mathbf{z}(t)$, where $\mathbf{x}(t)$ is the desired hand position and orientation from grasping controller, and $\mathbf{z}(t) = P_r(t) - P_{out}(t-\delta t)$ is the desired joint angle increments that moves the previous output pose $P_{out}(t-\delta t)$ toward $P_r(t)$. The IK method solves for joint angles that enforce the constraint $\mathbf{x}(t)$ with highest priority, while preserved the configuration from secondary task $P_r(t)$ as much as possible. Therefore the resulting motion is the combination of a reaching motion with exact hand placement and a gaze motion that tries to point toward the target without violating the reaching task. Figure 6 shows the effect of

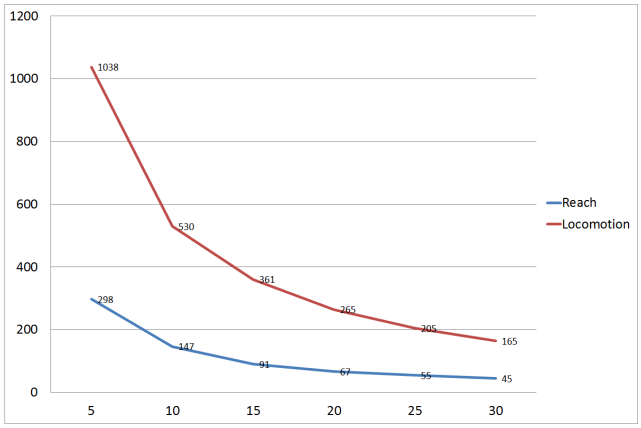


Figure 7: *The plot shows the performance of the system when controlling different number of virtual characters. The horizontal axis is varying number of characters and vertical axis is the frame rate. Both reaching and parameterized locomotion are efficient enough to handle over 20 characters and still maintain the real-time performance.*

constraint controller. The original grasp pose is disrupted by a full body gaze motion and the resulting pose is incorrect and unnatural. After applying the constraint controller to enforce hand placement, both gaze and grasp can be executed without conflicts.

8 Experimental Results

We present several examples to demonstrate the capability of our system. Figure 2 shows the setup of a virtual character for example-based reaching. The motion examples and pseudo examples define a feasible reaching space around the character. The character is able to reach for several different targets around him by utilizing the entire body. The combination of motion blending and inverse kinematics produces precise yet natural reaching motions. In Figure 1, we place several obstacles in the environment and request the virtual character to reach for a distant target. The locomotion controller successfully guides the character toward the target and avoid obstacles along the path before grasping the object. Figure 5 presents another reaching example of a virtual character sitting in the table. Since the sitting character does not utilize lower body movements, this example requires a different set of motion examples. Our system enables the virtual character to manipulate target objects on the fly, and the character is able to grasp objects with different shapes with appropriate hand postures.

In Table 2, we summarize the performance timing for each controller stage. For steering, we report the average timing under typical situations since the performance of steering system may vary when different steering rules are triggered under certain situations. The core components of our system – reaching, locomotion and gaze – consume only less than 2 ms of time during each update. Figure 7 shows the scalability of the reaching and parameterized locomotion stages. These analysis shows that our system is able to handle tens of characters performing both locomotion and manipulation tasks in real-time. [0]

[0]For futher details, our code is available online and can be examined here: http://sourceforge.net/projects/smartbody/develop/

Stage	*Timing (ms)*	*Comments*
Locomotion	0.21	
Steering	0.70	(Average Time)
Reach (Motion Blending)	0.42	($n_k = 4$)
Reach (IK)	0.28	
Gaze	0.07	
Total	1.68	

Table 2: *Performance break down for each controller stage.*

9 Conclusions and Discussion

In this work, we present an character animation system that integrate different controllers to achieve a reaching task. By utilizing path planner, motion blending and inverse kinematics, the system enables a virtual character with skills to maneuver around obstacles and precisely grasp an target object in space. Our system can respond to dynamic environments to generate a new reaching motion in real-time and is therefore suitable for interactive applications like video games.

In the future, we would hope to add collision avoidance in our grasping controller to handle obstacles during arm movements. We are also interested in improving the locomotion controller for cluttered environments. Specialized motion blending and steering algorithms need to be developed to improve the maneuverability of example-based locomotion and achieve this goal.

Limitations There are several limitations of our system. First, since we apply only kinematic-based methods to synthesize new animations from example motions, the results may not be physically correct and the quality would depend on the input data. Second, we do not handle collision detections during arm movements. How to adjust the motion blending method to avoid obstacles while preserving original motions is not a trivial problem, although the work in [Huang et al. 2011] could be a good starting direction for research. Finally, our method assumes the target object is small and can be grasped with single hand. A different set of heuristics and example motions need to be developed for the virtual character to handle large objects with both hands.

Acknowledgements

Thanks to Ed Fast, Arno Hartholt, Apar Suri, Shridhar Ravikumar, Adam Reilly, Saladin Begum and Jamison Moore on the Integrated Virtual Humans team at ICT for support, development and testing efforts. In addition, thanks to Marcus Thiebaux for his development work on the SmartBody codebase. Thanks to Matt Liewer, Teresa Dey and Nicholas Kelly for art support. Also, thanks to Stacy Marsella for his continuing vision and support.

References

AYDIN, Y., AND NAKAJIMA, M. 1999. Database guided computer animation of human grasping using forward and inverse kinematics. *Computers & Graphics 23*, 1, 145–154.

DENG, Z., LEWIS, J. P., AND NEUMANN, U. 2005. Automated eye motion using texture synthesis. *IEEE Comput. Graph. Appl. 25* (March), 24–30.

EPIC GAMES, 2011. Unreal development kit, http://www.udk.com.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '07, 129–136.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 71–78.

HUANG, Y., AND KALLMANN, M. 2010. Motion parameterization with inverse blending. In *Proceedings of the Third International Conference on Motion In Games*, Springer, Berlin.

HUANG, W., KAPADIA, M., AND TERZOPOULOS, D. 2010. Full-body hybrid motor control for reaching. In *Proceedings of the Third international conference on Motion in games*, Springer-Verlag, Berlin, Heidelberg, MIG'10, 36–47.

HUANG, Y., MAHMUDI, M., AND KALLMANN, M. 2011. Planning humanlike actions in blending spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

JAIN, S., YE, Y., AND LIU, C. K. 2009. Optimization-based interactive motion synthesis. *ACM Transaction on Graphics 28*, 1, 1–10.

JOHANSEN, R. S. 2009. *Automated Semi-Procedural Animation for Character Locomotion*. Master's thesis, Aarhus University, the Netherlands.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, SIGGRAPH '04, 559–568.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2008. Motion graphs. In *ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, SIGGRAPH '08, 51:1–51:10.

LEE, J., CHAI, J., REITSMA, P., HODGINS, J. K., AND POLLARD, N. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 1 (July), 491 – 500.

LEE, S. P., BADLER, J. B., AND BADLER, N. I. 2002. Eyes alive. *ACM Trans. Graph. 21* (July), 637–644.

LIU, C. K. 2009. Dextrous manipulation from a grasping pose. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3.

LV, P., ZHANG, M., XU, M., LI, H., ZHU, P., AND PAN, Z. 2011. Biomechanics-based reaching optimization. *Vis. Comput. 27* (June), 613–621.

RAIBERT, M. H., AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph. 25* (July), 349–358.

RIJPKEMA, H., AND GIRARD, M. 1991. Computer animation of knowledge-based human grasping. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '91, 339–348.

ROSE, C. F., PETER-PIKE, I., SLOAN, J., AND COHEN, M. F. 2001. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum 20*, 239–250.

SAFONOVA, A., AND HODGINS, J. K. 2007. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph. 26* (July).

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph. 23* (August), 514–521.

SHAPIRO, A., PIGHIN, F., AND FALOUTSOS, P. 2003. Hybrid control for interactive character animation. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, PG '03, 455–.

SHAPIRO, A., KALLMANN, M., AND FALOUTSOS, P. 2008. Interactive motion correction and object manipulation. In *ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, SIGGRAPH '08, 57:1–57:8.

SHAPIRO, A. 2011. Building a character animation system. In *MIG*, 98–109.

SI, H. TetGen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator. http://tetgen.berlios.de/.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of the 2nd International Workshop on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG '09, 158–169.

SINGH, S., KAPADIA, M., REINMANN, G., AND FALOUTSOS, P. 2010. On the interface between steering and animation for autonomous characters. In *In Workshop on Crowd Simulation, Computer Animation and Social Agents*.

THIEBAUX, M., MARSELLA, S., MARSHALL, A. N., AND KALLMANN, M. 2008. Smartbody: behavior realization for embodied conversational agents. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS '08, 151–158.

THIEBAUX, M., LANCE, B., AND MARSELLA, S. 2009. Real-time expressive gaze animation for virtual humans. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS '09, 321–328.

WROTEK, P., JENKINS, O. C., AND MCGUIRE, M. 2006. Dynamo: dynamic, data-driven character control with adjustable balance. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM, New York, NY, USA, Sandbox '06, 61–70.

YAMANE, K., KUFFNER, J., AND HODGINS, J. 2004. Synthesizing animations of human manipulation tasks. *ACM TOG 23*, 3, 530–537.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: Simple biped locomotion control. *ACM Trans. Graph. 26*, 3, Article 105.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph. 24* (July), 697–701.