An Animation System for Imitation of Object Grasping in Virtual Reality

Matthias Weber, Guido Heumer, Heni Ben Amor, and Bernhard Jung

VR and Multimedia Group
TU Bergakademie Freiberg
Freiberg, Germany
matthias.weber|guido.heumer|amor|jung@informatik.tu-freiberg.de

Abstract. Interactive virtual characters are nowadays commonplace in games, animations, and Virtual Reality (VR) applications. However, relatively few work has so far considered the animation of interactive object manipulations performed by virtual humans. In this paper, we first present a hierarchical control architecture incorporating plans, behaviors, and motor programs that enables virtual humans to accurately manipulate scene objects using different grasp types. Furthermore, as second main contribution, we introduce a method by which virtual humans learn to imitate object manipulations performed by human VR users. To this end, movements of the VR user are analyzed and processed into abstract actions. A new data structure called *grasp events* is used for storing information about user interactions with scene objects. High-level plans are generated from grasp events to drive the virtual humans' animation. Due to their high-level representation, recorded manipulations often naturally adapt to new situations without losing plausibility.

1 Introduction

Achieving accurate body movement in virtual humans for interactive object manipulations is still a difficult task. Previous work in this field has led to two different approaches to realistic character animation: data-driven methods, using motion-capture data, and model-driven methods, synthesizing accurate animations at runtime. Usually, the data-driven approaches suffer from limited adaptability to new situations, such as changing environments and different character proportions. Model-driven approaches on the other hand allow for high adaptability but suffer from an increasing number of control parameters. This normally results in a tedious trial-and-error process for finding an accurate set of parameters. For example, when using PID controllers [] for animation, appropriate gain values have to be choosen. Although one could hard-wire such parameters in a model-driven animation system, flexibility against modified objects or differently sized virtual humans would be lost as a consequence.

This paper follows a novel, imitation based approach to animation called *Action Capture* [1] which combines the strengths of data-driven and model-driven techniques. The interactions of a real user with the virtual environment

are recorded, processed and abstracted, such that they can be adapted to new situations. This allows for fast and easy recording of new animations without need for parameter optimization and, at the same time, also enables the flexible reproduction of interactions with scene objects.

While the generic conceptual framework for Action Capture including its relationships to the fields of imitation learning and robotics has been described elsewhere [1], this contribution focusses on the details a specific implementation for imitation of object grasping by virtual humans. For this, a hierarchical control architecture for animation of virtual humans is realized. On the highest level of abstraction plans describe animations as sequences of object interactions. Plans instantiate goal-oriented behaviors to influence body movement. Behaviors use low-level motor programs to directly move the body's limbs. Additionally, for storing information about user interactions with scene objects a new data structure called *grasp events* is introduced. In turn, grasp events are converted into high-level plans in order to drive the virtual human's movement.

This type of approach yields great flexibility when creating animations for object grasping. It is not necessary to build animations by hand any more and animations still work on changing environments. How to model a system following this imitation-based approach will be explained in the following sections.

2 Related Work

In order to enable a virtual human to manipulate other objects, it is of importance to specify different meaningful ways of interaction. An interesting approach to this was introduced by Kallmann and Thalmann [2]. The basic idea of the *smart object* approach is that each object "knows" the way an interaction with it should look like. Additionally, each object stores a number of parameters and other meta-data, that can be accessed and used by the virtual human. This meta-data has to be specified by the system developer when creating the objects and the world. Still, there is the question of how to synthesize an appropriate animation from this data. Early work on animation and control for manipulation and grasping was reported by Sanso and Thalmann [3]. Kuffner and colleagues [4] proposed the use of a path planner and inverse kinematics in order to plan a collision-free trajectory connecting an initial arm position with a goal position. In an extension of this algorithm [5] they were also able to bias the inverse kinematics algorithm towards natural-looking poses using a motion database. In contrast to this, the work of Douville et al. [6] describes a *behavioral* approach to motion synthesis. Here, different objects are grasped using specialized grasping behaviors and an *object specific reasoner*. The latter helps to break down the high-level plans into motions.

In many applications it is desirable to have a virtual human imitate the behavior of a real person while at the same time being subject to the physics of his own virtual environment. While this approach preserves the plausibility of the animation, it also demands for special algorithms which can abstract actions from raw motion capture data. Such algorithms have long been studied in the

robotics and Artificial Intelligence literature (AI); a survey on this topic was published by Jung et al. [1]. Pollard and Zordan [7] present an algorithm, which extracts a set of parameters from motion capture data. The parameters are in turn fed into a controller used for performing active and passive grasping.

The system presented in this paper combines several of the above ideas with a hierarchical architecture for animation synthesis. In contrast to previous work, the motion capture data is not only used for parametrizing particular controllers or behaviors. Instead, the motion is analyzed, abstracted and transformed into an intermediate representation called *grasp event*. Grasp events hold information about the interaction with scene objects and thus allow the recorded action to be performed in different environments and settings.

3 Animation Synthesis

This section presents a hierarchical control architecture for interactive animation of virtual humans, with a particular focus on animating object manipulation tasks. The architecture is currently implemented as an extension of the VR system *Avango* [8]. In principle, however, it is independent of any particular VR software. At its core, the framework consists of hierarchically organized modules at three distinct levels: plans, behaviors, and motor programs.

Fig. 1. Control Architecture

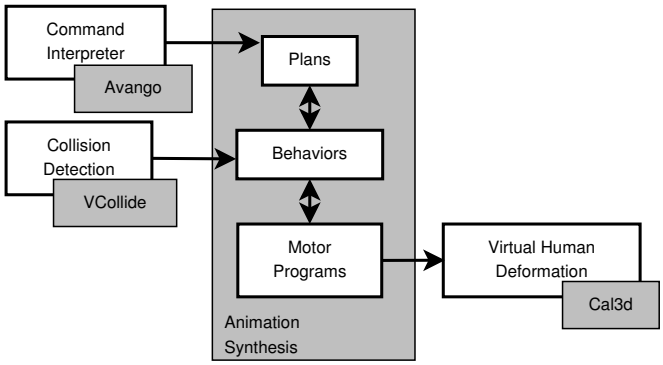


Fig. 1 gives an overview of the control architecture. In general, the control flow for animation synthesis is as follows: High-level plans are abstract descriptions of action sequences. The selection and execution of plans can be triggered by issuing an appropriate command from Avango's integrated Scheme interpreter. Execution of a plan will instantiate several behaviors which in turn start one or more motor programs to control the basic movements of the virtual human, i.e. changing bone rotations of the virtual character's skeleton. Behaviors

are notified about collisions occurring between body parts and scene objects, e.g. when grasping an object. They can decide if it is necessary to stop movement when collisions occur. Behaviors query for stopped motor programs and plans query for finished behaviors, leading to the instantiation of new behaviors as described in a plan until the plan ends.

Motor Programs

On the lowest level of the animation synthesis hierarchy, motor programs control the joint values of a virtual human's bones. The specification of a motor program defines different parameters, e.g. begin and end time. Additionally a scope has to be defined, i.e. the set of bones influenced by its execution in the kinematic chain of the skeleton. Motor programs do not directly set joint values, but instead modify motor states. In general, a motor state represents an arbitrary state of a bone, i.e. position and orientation, during the movement of the virtual human at a certain time. After a motor program has calculated a new state, the state is passed to an arbiter component which directly sets the joint values. As many motor programs may run in parallel, conflicts may occur when calculating motor states referring to the same bone. Conflicts are resolved by the arbiter through motor state priorities, which were set by associated motor programs. This way resolution of conflicting bone movements occurs on a fine-granular motor state level rather than on the complete scope of the motor program or on the higher level of behaviors. To support different types of bone movement, several types of programs are supported, which currently fall into two categories. One type of motor programs interpolates motor states from current states to specified goal states, defining the end-posture for a part of the virtual human. The second type uses inverse kinematics to let a kinematic chain of arbitrary length reach a certain position.

Behaviors

Mid-level behaviors are used to accomplish certain sub-actions of object manipulations, like closing a hand or reaching for an object. To realize movements of the virtual human, they select and instantiate motor programs, that work directly on the kinematic chain of the virtual human. Behaviors are parametrized, allowing them to reactively adapt to changing virtual scenes. E.g. moving objects can be reached by only setting the object's name as a parameter and letting the behavior calculate where the object is currently located. There are currently four types of behaviors to accomplish different tasks: Open a hand, close a hand to grasp an object, reach a certain point or object, and move joints to a given end or rest posture. These behaviors can be adjusted with parameters like begin and end time or the object to interact with.

Plans

On the highest level, plans describe complete manipulation tasks, e.g. grasping an object, in a declarative fashion. They consist of plan steps, which can be

executed either sequentially or in parallel. Plans are described in an XML-based language. Fig. 2 shows a simple example of such a plan description. Instantiating the plan `grasp-object` with the object `Ball-1`, the grasp port `ball-port`, the hand `RightHand` and the grasp type `spherical` as parameters results in selecting the plan out of a plan database and refining the plan by substituting variables in the plan with the parameters (variables in this example are `$OBJECT`, `$PORT`, `$HAND` and `$GRASPTYPE` respectively). It is not necessary for the triggering command to specify all parameters of a plan: For example, if just the target object of the grasp is specified, then a suitable `$PORT`, `$HAND` and `$GRASPTYPE` will be automatically added, e.g. using default values or object annotations (see below for a description). In the example, first a parallel set of plan steps is instantiated for reaching the grasp port `ball-port` on the object `Ball-1` and for opening the hand. The types (`type`), the name of the virtual character affected (`vh_name`), and several parameters (`param`) can be specified for such behaviors. After reaching the object with an open and correctly oriented hand, the hand is closed (behavior type `GraspClose`) using a possibly given or calculated grasp type, which is selected out of a grasp taxonomy. When the object is grasped, the plan finishes.

Fig. 2. Example for a plan: grasping an object

```
<plan>
  <name>grasp-object</name>
  <parallel>
    <behavior>
      <type>Reach</type>
      <vh_name>Body</vh_name>
      <param name="object">$OBJECT</param>
      <param name="grasp-port">$PORT</param>
    </behavior>
    <behavior>
      <type>GraspOpen</type>
    </behavior>
  </parallel>
  <behavior>
    <type>GraspClose</type>
    <param name="grasp-type">$GRASPTYPE</param>
  </behavior>
</plan>
```

Annotated Objects

To support movement generation, semantically *annotated objects* are used in behaviors to, e.g. find a proper place on a chosen object to grasp. Grasp ports

define, where on an object grasping is possible. Additionally, they describe the shape of the object at that place so it is possible to choose an appropriate hand shape for grasping. Annotated objects are akin to the concept of *smart objects* in literature (e.g. Kallman and Thalmann [2]), differing in the way that we do not explicitly describe behaviors in the objects. Annotated objects are represented in XML structures; see Fig. 3 as an example. In this example the object type `Ball` is defined. Besides a specified scale factor (`scale`), one grasp port with a spherical form (`type="SPHERICAL"`) is defined. The grasp port has the name `ball-port`, a certain position (`position`) and a radius (`radius`). Finally, the graphical model to load for the annotated object is specified.

Fig. 3. Example for an annotated object: A Ball

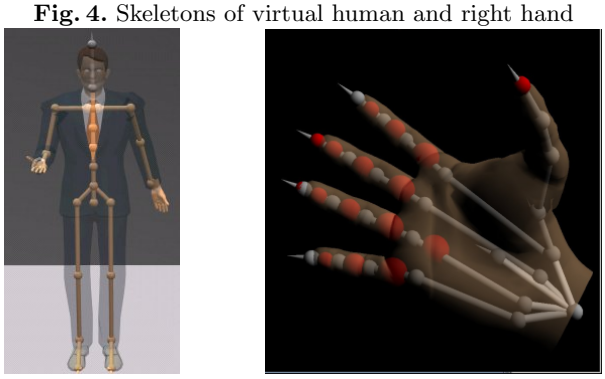
```
<annotated-object>
  <name>Ball</name>
  <scale>0.00595 0.00595 0.00595</scale>
  <grasp-port type="SPHERICAL">
    <name>ball-port</name>
    <position>0.0 0.0 0.0</position>
    <radius>5.775</radius>
  </grasp-port>
  <model>models/iv/ball2.iv</model>
</annotated-object>
```

Virtual Human Body and Hand Model

In skeletal animation, a virtual character's body is represented by a mesh and a skeleton composed of bones. In our system, the virtual humans' skeleton structures are based on the joint hierarchy of the H-Anim standard (`http://www.hanim.org`). Every bone is represented with a name and its orientation and translation relative to it's parent bone. Several virtual characters can populate a scene, acting independently of each other. Additionally, virtual characters can be composed of several independent articulated models. We use this method to integrate different models for body, right and left hand. In this way, the modelling of the virtual human's body is separated from the modelling of its hands. Thus, the hands, which are more complex to model, can be easily re-used with different bodies. Fig. 4 shows the skeletons of a virtual human and of the right hand.

Since the focus of our animations is on object manipulation, particular effort has been put into a detailed virtual hand model. Just as the virtual character's body, the hand model is a skinned mesh, deformed by an underlying skeleton structure. Each finger consists of three segments/bones, with the proximal joints having two degrees of freedom (flexion and pivot) and the others only having

Fig. 4. Skeletons of virtual human and right hand



one (flexion). Additionally, joint angle constraints are enforced, to permit natural looking movements only.

For interaction and collision detection with scene objects, the hand model has been fitted with spherical collision sensors (see Fig. 4). Currently, there is one sensor in the palm, one sensor in each finger segment and an additional one at each finger tip. These sensors fire collision events, as soon as they touch a scene object. This information is used by motor programs and behaviors to adjust the animation in turn to avoid intersection of the hand with the objects.[1]

Collision detection is currently implemented using VCollide [9]. For the visual part, to realize skeletal animation and mesh deformation according to the bone movements, the character animation library Cal3d (`http://cal3d.sourceforge.net`) is used.

4 Imitating Grasp Actions

In the previous section, the process of an autonomous movement generation for virtual character animation was described. Movement is triggered by instructing the virtual character to execute specific plans. In this section, we introduce an alternative method in which the virtual character learns to *imitate* object manipulations performed by a human VR user.

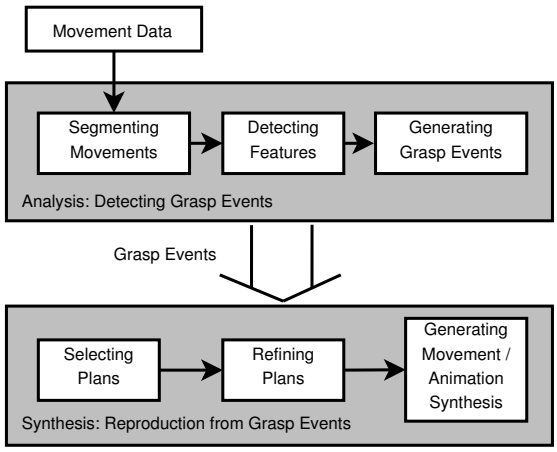
To this end, the interactions of the VR user with scene objects are analyzed and transformed to plans, that can be executed by virtual characters using the animation synthesis architecture described above. Animations are thus generated by recording and analyzing movement data from virtual reality input devices.

[1] A further use of the hand model is to support the interactions of human VR users with the virtual world, particularly for grasp analysis (see Sec. 4). Here, the user hand is projected into the virtual world to provide visual feedback to the user as well as to find contact points with scene objects during grasps. In this case, joint angles are deducted from data glove sensor input and then mapped directly to the joints of the virtual hand.

Such devices could be, but are not limited to, optical position trackers and data gloves. The recorded data is processed to calculate certain features to describe a grasp. These features are hierarchically organized from low level features – describing low level data like the joint values of a human skeleton – to high level features, where the grasp is only described by a category belonging to a grasp taxonomy[2]. Low level features need no calculation when movement of a real human is analyzed. High level features achieve high abstraction of human movement by just providing grasp types and objects to grasp. Such a high abstraction makes it possible to adapt virtual human movement to changing scenes.

Figure 5 gives an overview of the whole process of imitating grasp actions. The two parts, movement analysis with the focus on grasp event detection and movement synthesis for the reproduction of actions, will now be described.

Fig. 5. Imitation process



4.1 Analyzing User Actions for Grasp Events

Analyzing movement data involves the following steps: segmentation of the hand movement, mapping of glove sensor values to joint rotation values and calculating grasp features out of the posture of the hand and the grasped object. This process is subject of other work and is not the focus of this paper. But the results are used to imitate human movement, especially in the domain of grasping objects.

A novel representation form is used to store the results: Grasp events. They contain grasp features and a time stamp when the grasp happened with respect

[2] Grasp taxonomies classify grasps into several categories, describing the features of a grasp, e.g. Cutkosky [10] or Schlesinger [11]

to the time when the manipulation sequence began. Low level features are: joint values, hand position, contact points with the object and the object as well as information about where the object was grasped using grasp ports. High level features denote the resulting category of a grasp w.r.t. a certain taxonomy. Grasp events can be stored in XML files for the purpose of later playback. An example for the structure of such an XML file is shown in Fig. 6.

Fig. 6. An XML example for grasp events

```
<event-sequence>
  <event timestamp="3.895" type="grasp">
   <low-level>
    <joint-angle joint-id="r_index1">19.8306 -0.0865252 0.678805 0.729203
    </joint-angle>
    <contact-point joint-id="sensor_r_index1">
      <object>Ball-1</object>
      <pos>0.00730081 -0.0734563 0.0135953</pos>
    </contact-point>
    <object-ids> Ball-1 </object-ids>
    <hand-transform>0.0205884 0.211408 -0.97718 0
                    0.0805939 0.973855 0.212386 0
                    0.996533 -0.0831275 0.00301189 0
                    -0.1502 -0.626599 0.917001 1
    </hand-transform>
    <hand-side> right </hand-side>
   </low-level>
   <high-level>
    <taxonomy>schlesinger</taxonomy>
    <category>spherical</category>
   </high-level>
  </event>
  ...
</event-sequence>
```

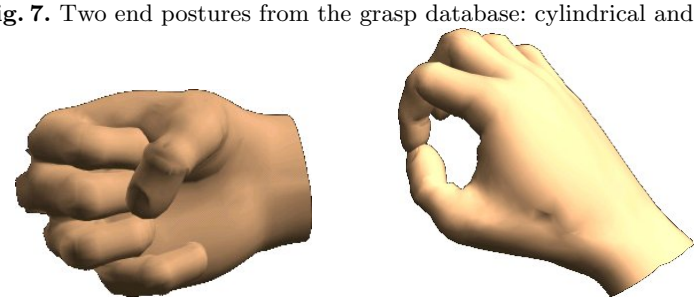
4.2 Reproducing Actions from Grasp Events

To animate the virtual character and reproduce the recorded sequence of grasp events, the events have to be converted into executable plans. This is done in two steps (see the synthesis part of Fig. 5): First, a plan is selected, based on the received grasp events. As a simple example, this could be the plan `grasp-object` whenever events of type `grasp` are received. After that, the selected plan is refined by means of the features contained in the grasp events. Refinement is done by filling unspecified parameters of the plan with reasonable values. This can be done in a very simple way by providing the last value that was received

for a parameter. Another conceivable possibility would be to use the information provided by grasp events and the actions that already took place for computing new parameters.

When only high level features, i.e. grasp taxonomy categories, are used to specify a grasp, a grasp database is used to look up the grasp type and retrieve features useful to generate a similar grasp by the virtual human. For now such useful features include the start and end postures of the grasp. In addition to this, finger groups are used for describing similar behaving fingers. Fingers in a finger group stop whenever one of the fingers stops, caused e.g. by collision detection. As an example the extreme end postures of two grasp categories, cylindrical and tip are shown in Fig. 7. Unfilled parameters in a selected plan are then refined by using these features.

Fig. 7. Two end postures from the grasp database: cylindrical and tip



By execution of the refined plan, the virtual human performs the appropriate movement, using the framework described in Section 4. Behaviors, parametrized with the values computed by the refinement step, are instantiated, which use adequate motor programs to control virtual human movement.

5 Example

The whole process of movement generation by instruction will now be explained by considering the example of grasping a ball, as shown in Fig. 8. Grasps performed by a user are recorded and processed to generate a grasp event containing a spherical grasp for the ball `Ball-1` with timestamp 3.895.

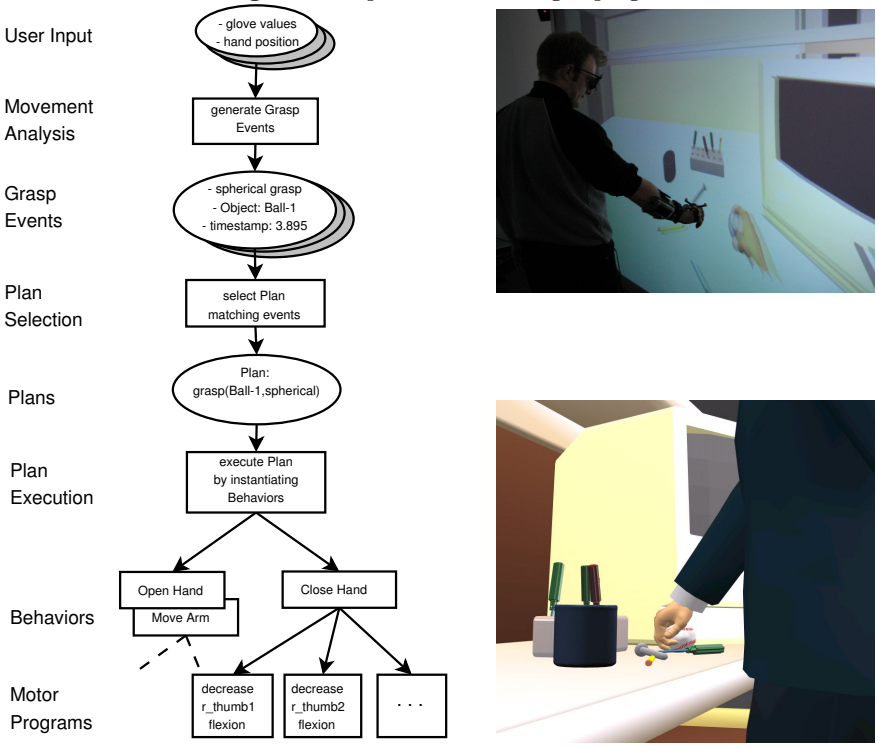
To grasp the ball `Ball-1`, an appropriate plan is selected and parametrized to convert the features into parameters of the plan. In this plan, parallel behaviors for reaching the grasp port of the object and for opening the hand are executed. After finishing these behaviors, a behavior for closing the hand, i.e. grabbing the object, is executed.

For reaching, the behavior will start a motor program that moves the virtual human's hand to the object's grasp port position using inverse kinematics. The scene information database is queried for information about the grasped object.

In this example, this would be an annotated object of type `ball` with a spherical grasp port for `Ball-1`.

Opening and closing of the hand will start motor programs that interpolate joint values to given goal motor states. These are goal joint values for opening and closing the hand with certain joint values that describe an opened or closed hand respectively. When closing the hand, collision sensors are used to stop the movement of bones when fingers get contact with the annotated object.

Fig. 8. Example: Imitation of grasping a ball



6 Conclusion

We have presented a system for instructing virtual humans for the synthesis of virtual human-object interactions. A hierarchical three-level approach is used to generate accurate object manipulations from high-level commands. The system automatically instantiates behaviors and appropriate motor programs to control the virtual human.

Using this framework, grasps performed by a real human in a virtual environment can be imitated. For this, grasps are recorded, analyzed for grasp features,

and stored as grasp events. The recorded grasp events are used to select and refine appropriate plans, which in turn trigger lower level behaviors. In this way, interactions between the human and the environment can be recorded and imitated. Due to their high-level representation, recorded manipulations often naturally adapt to new situations without losing plausibility.

In on-going work we are developing a more elaborate plan generator to create high level plans out of sequences of grasp events. With such plans it will further be possible to create a plan database containing several highly abstract representations of object manipulations in virtual environments. Additionally, we want to improve the refinement step for selecting plans. Missing parameters could, for instance, be computed based on predictors acquired through machine learning techniques. Learning could further be used for improved reproduction of different movement styles exhibited by different VR users.

References

1. Jung, B., Ben Amor, H., Heumer, G., Weber, M.: From Motion Capture to Action Capture: A Review of Imitation Learning Techniques and their Application to VR-based Character Animation. In: Proceedings ACM VRST 2006, Cyprus. (2006)
2. Kallmann, M., Thalmann, D.: Direct 3D Interaction with Smart Objects. In: Proceedings ACM VRST 99, London. (1999)
3. Sanso, R.M., Thalmann, D.: A Hand Control and Automatic Grasping System for Synthetic Actors. Computer Graphics Forum **13**(3) (1994) 167–177
4. Kuffner, J., Latombe, J.: Interactive Manipulation Planning for Animated Characters. In: Proceedings of Pacific Graphics. (2000)
5. Yamane, K., Kuffner, J.J., Hodgins, J.K.: Synthesizing Animations of Human Manipulation Tasks. ACM Trans. Graph. **23**(3) (2004) 532–539
6. Douville, B., Levison, L., Badler, N.: Task-level Object Grasping for Simulated Agents. Presence **5**(4) (1996) 416–430
7. Pollard, N., Zordan, V.B.: Physically Based Grasping Control from Example. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. (2005)
8. Tramberend, H.: Avocado: A Distributed Virtual Reality Framework. In: Proceedings of IEEE Virtual Reality. (1999) 14
9. Hudson, T., Lin, M., Cohen, J., Gottschalk, S., Manocha, D.: V-COLLIDE: Accelerated Collision Detection for VRML. In: Proceedings of VRML'97. (1997)
10. Cutkosky, M.: On grasp choice, grasp models and the design of hands for manufacturing tasks. IEEE Trans. on Robotics and Automation **5**(3) (1989)
11. Schlesinger, G.: Der Mechanische Aufbau der Künstlichen Glieder. In Borchardt, M., et al., eds.: Ersatzglieder und Arbeitshilfen für Kriegsbeschädigte und Unfallverletzte. Springer-Verlag: Berlin, Germany (1919) 321–661