Bridging Geometry and Semantics for Object Manipulation and Grasping Workshop Paper

Tolga Abacı
VRlab–EPFL
tolga.abaci@epfl.ch

Michela Mortara
IMATI–CNR
mortara@ge.imati.cnr.it

Giuseppe Patanè
IMATI–CNR
patane@ge.imati.cnr.it

Michela Spagnuolo
IMATI–CNR
patane@ge.imati.cnr.it

Frédéric Vexo
VRlab–EPFL
frederic.vexo@epfl.ch

Daniel Thalmann
VRlab–EPFL
daniel.thalmann@epfl.ch

Abstract

In this paper, we present our on-going work towards grasping in an object manipulation context. Our proposal is a novel method that combines a tubular feature classification algorithm, a hand grasp posture generation algorithm and an animation framework for human-object interactions. This method works on objects with tubular or elongated parts, and accepts a number of parameter inputs to control the grasp posture.

Keywords: virtual environments, grasping, shape analysis, smart objects, animation

1 Introduction

Realistic animation of object grasping for an autonomous virtual human is a difficult problem, with many different sides to take into account. The human hand is a complicated articulated structure with 27 bones. Not only the movements of these joints must be calculated, but also the reaching motion of the arm and the body needs to be considered. For real-time performance in a VR system with many agents, fast collision-detection and inverse kinematics algorithms [20] will be necessary in most cases.

The calculation of the hand and body postures is not the only difficulty in grasping: realistic grasping also requires significant input about the semantics of the object. Even if the geometric and physical constraints permit, sometimes an object is simply not grasped “that way”. For example, a door handle must not be grasped from the neck section if the goal is to turn it. A fully-automatic grasping algorithm that only takes the geometry of the object into account cannot always come up with solutions that are satisfactory in this sense. It is evident that the grasping operation is strongly dependent on the artificial intelligence of an autonomous virtual human.

Fortunately, the grasping problem for autonomous virtual humans is easier than its robotics counterpart. Simply put, we do not have to be as accurate and physical constraints are much less of a problem. The main criterion is that the grasp must “look” realistic. In fact, the apparent physical realities of a virtual environment can be very different from those of the real-world,

with very different constraints being imposed. For example, we can imagine a virtual human holding an object that is several times his size and weight in air, while grasping it at a small site on the edge. This does not conflict with the previous examples addressing the reality issue, as for an autonomous virtual human in a virtual setting, this is more a question of what he intends to do with the object (semantics) than the actual physics of grasping.

In this paper, we present our on-going work towards grasping in an object manipulation context. For the solution of this problem we propose a novel method that combines a tubular feature classification algorithm, a hand grasp posture generation algorithm and an animation framework for human-object interactions (with *smart objects* [8]). Our method works on objects with tubular or elongated parts, and accepts a number of parameter inputs to control the grasp posture.

2 Related Work

2.1 Smart Objects

Making objects interaction-capable usually requires solutions to closely-related issues on two fronts: the specification of behavior and its reflection through animation.

On the behavior front, virtual human–object interaction techniques were first specifically addressed in the object specific reasoner (OSR) [12]. The primary aim of this work is to bridge the gap between high-level AI planners and the low-level actions for objects, based on the observation that objects can be categorized with respect to how they are to be manipulated. This works gives little consideration to interaction with more complex objects.

The work on Parameterized Action Representation [1] addresses the issue of natural language processing for virtual human-object interactions. A PAR describes an action by specifying conditions and execution steps. Recently, Vosinakis and Panayiotopoulos have introduced the Task Definition Language [22]. This language supports complex high-level task descriptions through combination of parallel, sequential or conditionally executed built-in functions.

Rule-based behaviors are a popular technique: according to the system state, applicable rules can be selected to evolve the simulation. In addition, state machines are widely used for specifying behaviors as they have useful graphical representation. A good example for a system utilizing both techniques is Improv [17].

Animation of virtual humans can be examined in many categories, depending on the type of action. Generation of realistic human walking motion has been an interesting subject for researchers [5]. Inverse kinematics is also commonly used for creation of reaching motions for articulated structures [2, 20, 23], but it is still difficult to obtain realistic full-body postures without substantial tweaking. On the other hand, database-driven methods [24, 4] cope better with full body postures. These methods are based on capturing motions for reaching inside a discrete and fixed volumetric grid around the actor. The reaching motion for a specific position is obtained through interpolation of the motions assigned to the neighboring cells.

Grasping is perhaps the most important and complicated motion that manipulation of objects involves. Robotics techniques can be employed, as in [7] for automatic grasping of geometrical primitives, based on a pre-classification of most used hand configurations for grasping [6]. Planning algorithms for determining collision free paths of articulated arms have also been developed for manipulation tasks [10], and they have been successfully used for interactive generation of reaching and transfer motions [9]. Because of the random nature of this method, complicated motions can be planned, but with high and unpredictable computational cost. A huge literature about motion planning is available, mainly targeting the motion control of different types of robots [11].

2.2 Shape Analysis

Knowledge about the presence of elongated features is relevant in the context of animation for the definition of posture and grasping motion for virtual humans. While tubular or elongated features can be quite easily defined during the design processes, their automatic extraction from unstructured 3D meshes is not a trivial task. Moreover, geometric parameters such as tube axis or section size should be made readily available to the animation tool.

Among the many methods for shape analysis, skeleton extraction techniques are the most suitable for identifying tubular features. Topology-based skeletons, for example, code a given shape by storing the evolution of the level sets of a mapping function defined on its boundary. A geometric skeleton is usually associated to this coding, defined by the barycenters of the contours. The shape is decomposed into parts which can be characterized as protrusion-like features or branching sites of the shape, even if the protrusions can be arbitrarily shaped. The Reeb graph is an example of topology-based skeketons, whose computation has been proposed in the literature using different approaches [19, 21, 3]. Topological graphs usually preserve genus information and therefore could be used to identify tubular parts that define handles, but the location and shape of other elongated feature requires a more specific analysis.

For example, in [13], tubular parts are identified using a sweeping techniques along the arcs of the skeleton which is constructed by joining the edges remaining after an edge collapse process on the whole mesh. These edges are linked in a tree structure, and it is used as a support for the sweeping process where the mesh is intersected by a set of planes and tubes are identified by looking at the geometry of the cross-sections.

Other skeletons, such as the well-known Medial Axis Transformation (MAT), define a structure which could be useful for the identification of tubular features. But the MAT of a 3D object is generally a non-manifold complex, computationally heavy to compute, and sensitive to noise because tiny perturbations may produce a whole new arc. Furthermore, there is not a direct relation between tubular features and specific components of the MAT, especially when the tubes have an arbitrary shape and the cross sections do not exhibit any symmetry.

3 Approach

Our primary goal is to address grasping issues for virtual human object manipulation. For this, we propose the flow comprised of the steps given below:

1. Given an object, the tubular or elongated features of an object are recognized and a list of cross-sections is associated to the features.

2. During smart object design, the designer selects the sections of the extracted features that are relevant for grasping. Additional grasping parameters are specified for each of these sections.

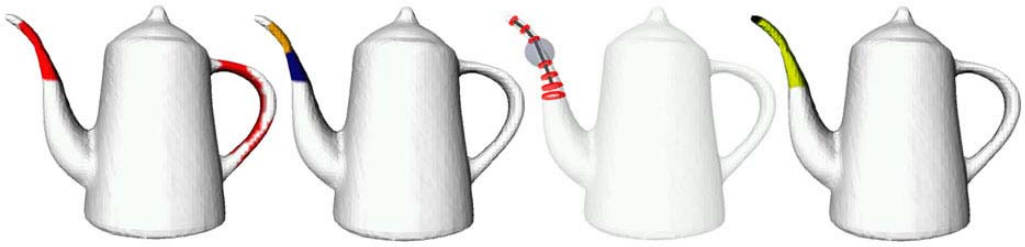
3. At run-time, grasping is performed using the data specified in the smart object, as a part of the object manipulation sequence.

The algorithm to detect tubular features is called *Plumber* and it is a specialized shape classification method for triangle meshes. The algorithm segments a surface into connected components that are either body parts or elongated features, that is, handle-like and protrusion-like features, together with their concave counterparts, i.e. narrow tunnels and wells. The segmentation can be done at single or multi-scale, and produces a shape graph which codes how the tubular components are attached to the main body parts. Moreover, each tubular feature is represented by its skeletal line and an average cross-section radius.

The *Smart Objects* paradigm is based on extending objects (shapes) with additional information on their semantics. Its focus is on autonomous virtual humans within virtual environments. The semantic information that a smart object carries is mainly about the “behavior” of the object when an interaction occurs between the object and a virtual human. By behavior, we mean the changes in the appearance and state of an object as a result of the interaction (i.e. a virtual human opening a door).

In our approach, the smart objects control the manipulation sequences. Grasping is perhaps the most important part of a manipulation sequence, but it is not alone. A full sequence can consist of walking and reaching to the object, looking at it, grasping it multiple times, and keeping the hands constrained to the object while it is moving. Therefore, the smart objects are required to provide a full manipulation sequence, putting the grasping action into the proper context.

The manual specification of the grasp parameters in the second step makes the approach semi-automatic. While we can attempt to derive these parameters automatically, it is very difficult to do so only based on the geometri-



(a) (b) (c) (d)

Figure 1: *Plumber* method: (a) identification of limb vertices, (b) extraction of their connected components and medial loop, (c) iteration, (d) tube and a cap (black) found at this scale.

cal properties of the object. To determine which tubular sections of complex object are of relevance to grasping, we need additional input on how the object is to be manipulated. For example, a teapot containing hot tea to be poured into a cup should normally be grasped by the handle, not the neck. The current state of the art in artificial intelligence does not offer a general, working solution for this problem yet, so our practical solution is to make the teapot a smart object and specify the required semantic information during its design. As our approach uses smart objects for simulating manipulation, the grasping parameters will be stored together with other attributes of the object, which are also specified in the design phase.

It is possible to generate the grasp postures (execute the third step) before run-time. This can be accomplished by calculating a fixed grasping posture, and storing it in the smart object and simply making the hand assume the stored posture during run-time. While this approach results in simpler and faster run-time execution, we have chosen not to take this route for a number of reasons. Firstly, a fully pre-computed grasping posture is dependent on the hand for which it was computed. Different virtual humans can have different hand sizes, therefore the resulting grasp will not be accurate enough. In addition, as we will explain later, certain grasping parameters are specified as ranges, which are then searched to find a satisfactory solution for the particular virtual human configuration at the time of grasping. This introduces a degree of variation into the manipulation sequences, which is hard to achieve with fixed pre-computed grasping postures.

4 The Plumber method

The *Plumber* method analyses the shape of an object by studying how the intersection of spheres centered at the mesh vertices evolve while the sphere radius changes. For example, for a thin limb, the curve of intersection between the mesh and a sphere will be simply connected for a small radius and then will rapidly split into two components when the radius increases and becomes greater than the tube size. While a detailed description of the shape analysis technique which uses intersecting sphere and of the *Plumber* method can be found in [15, 16], we will summarize here the main properties of *Plumber* and describe how the geometric parameters are associated to elongated features.

First of all, *Plumber* can identify tubular features whose section and axis can be arbitrarily shaped, and the size of the tube is kept as a constraint during the identification process. Moreover, since the shape is analysed using a set of spheres of increasing radius, the recognition follows a multi-resolution schema.

Chosen a sphere of radius R, *Plumber* performs the following steps:

1. identify *seed-tube* regions; these regions will produce one intersection area with the sphere, with two boundary curves of intersection (see Figure 1(a));

2. shrink each of the two selected intersection curves along the surface to the medial-loop, whose points are nearly equidistant from the two border loops (see Figure 1(b));

3. expand-back the medial-loop by sweeping the extent of the shape in both directions. More precisely, at each iteration we place a

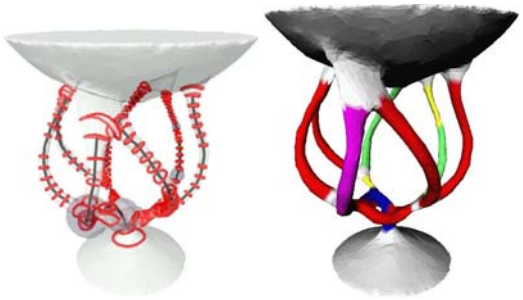
sphere of radius R in the barycentre of the new medial loops. If the intersection between the sphere and the surface generates two loops, mesh vertices inside the sphere are marked as visited;

4. the procedure is iterated in both directions until:

   - no more loops are found, or more than one loop is found on not-visited regions;
   - the new loop lies on triangles that are already part of another tube, or the length of the new loop exceeds a predefined threshold.

5. the tube skeleton is extracted by joining the loops' barycentres.

As shown in Figure 2, tubular features are recognized at different scales and their geometric description is computed also in case of interacting features. For the purpose of extracting grasping sites for a virtual human, like handles for instance, the radius value can be set with respect to hand anthropometric measures.



(a) (b)

Figure 2: Tubular features recognized by *Plumber* on a complex model: (a) tube axis and loops, (b) tubes colored with respect to their scale.

After the location of seed tubular regions and the computation of the medial loop, the tubes are recovered by expanding the loop by controlled procedure which, at each step, extends the center-line and at the same time ensures that the surface is tubular around it. For the expansion process, intersecting spheres are used again, but centred on the tube axis. A first medial sphere is drawn, whose centre p is the barycentre of the medial loop, and whose radius is R. If $M \cap S(p, R)$ does not have two boundary components, the growing stops and the candidate tube is discarded. Otherwise, a new sphere with the same radius is centred in the barycentre of the two intersection loops; the process is then split into two parts, trying to grow the tube in both directions. Now we focus on the sphere moving in one of the two directions, since the other case is symmetric.

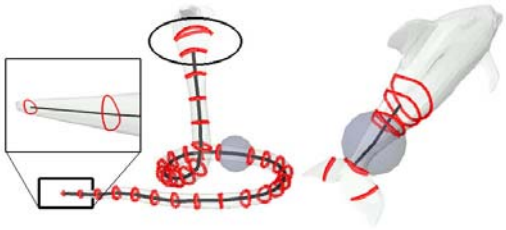
At each iteration, the sphere rolls to the barycentre of the next loop, and the triangles laying completely or partially inside the sphere are marked as belonging to that tube. Then, the intersection between the sphere in the new position and the mesh is again computed, taking into account only the intersection curves through non visited triangles (all the spheres except the medial one have always a "backward" loop, passing on the already marked triangles). During the loop expansion, the following cases may arise:

- no intersection curves are found. This is the case of a tubular protrusion terminating in a tip; visited triangles locate a *cap* (see Figure 3(a), in the square);
- the intersection curve consists of one loop (see Figure 3(a)). If its length is less than a pre-defined threshold, the size of the tube section is not varying too much; the loop becomes a new cross section and its barycentre contributes to the skeleton as a new node. Otherwise (see Figure 3(b), in the oval), the growth stops.
- the intersection counts two, or more loops; that is, a bifurcation occurs (see Figure 3(b)). The growing of the tube in this direction stops, and the last visited triangles are unmarked.

Finally, the barycentres of the medial loops are joined to define the tube skeleton.

5 Smart Objects

In essence, smart objects provide not only the geometric information necessary for drawing them on the screen, but also semantic information useful for manipulation purposes. We have



(a) (b)

Figure 3: (a) No new loop is found on the snake tail (in the box), and a loop discarded after the length check on the head (in the oval). (b) A branching occurs on the dolphin tail.

built a framework for real-time animation of virtual human – object manipulation sequences. This framework provides smart objects capabilities and is composed of the following components:

- A design tool that incorporates the definition of semantic information in the process of object design.
- An XML-based specification for virtual objects, including appearance, animation and interaction aspects.
- An extended scene-graph structure that enables storage and query of semantic information at run-time.
- An event-based mechanism and scripting functionality for controlling and coordinating animation of objects and virtual humans.

Attributes are the primary means of specifying information on how a virtual human manipulates its environment. They convey various kinds of information (e.g. where and how to approach for manipulating the object or to position the hands in order to grasp it), animation sequences (e.g. a door opening) and general, non-geometric information associated with the object (e.g. weight or material properties). The semantic information in the smart object is used by the virtual characters to perform actions on/with the object, e.g. grasping, moving it, operating it (e.g. a machine or an elevator).

We have integrated the attribute definition process into 3D Studio MAX, a popular software package for design and visualization of virtual environments. Using our plug-in, attribute sets can be created as the geometry of the environment is designed. This has the advantage of providing a consistent working space for designers.

Figure 4: Object manipulation with grasping

The animation of virtual humans is handled by "actions". Actions provide a higher level view of animation tasks. For example, the *look* action requires a vector as a parameter and keeps the virtual human looking at this position while it is active. The *walk* action takes a vector as a parameter, which is used as the target of the walk. The *reach* action takes a hand posture and a matrix as parameters. The hand of the virtual human is brought to the position and orientation specified by the matrix by using inverse kinematics. Once the hand is at the target, it assumes the given posture for grasping.

```
Human.WalkTo(Wheel.FrontPosition)
WaitUntilEvent(WalkReached(Human))
Human.Reach(Wheel.LeftHand)
Wheel.StartAnim(Wheel.Turn)
Repeat
  Event = WaitAndReceiveEvent()
  If Event.Is(AttribChanged(Wheel.LeftHand)
    Human.NewReachTarget(Wheel.LeftHand)
Until Event.Is(AnimFinished(Wheel.Turn)
Wheel.Turned = True
```

Figure 5: Sample manipulation pseudo-script

Scripts and events are used for flexible high-level control and coordination of animation elements. Consider the example in Figure 4 from a training application, where a virtual human needs to manipulate a machine. In this particular case, the action to be performed is turning a wheel for adjustment. The sequence of movements that the human should make and the

changes in the state of the machinery in response is described by a script. Such a script, in a simplified pseudo form, is given in Figure 5.

Grasping Extension Usually, designers define the grasping hand postures for a smart object manually during the design phase. This is tedious and results in manipulation sequences that are always executed exactly in the same way. Also, the results are satisfactory only for the fixed dimensions. We propose to modify this process, reducing it to specification of a few grasping parameters relevant to the grasping algorithm presented in this paper.

During the design phase, the designer is presented with the Plumber output, and first identifies the tubular regions of the object that are relevant to grasping. These exist as sets of (approximated) cylinders that are connected in a chain configuration. For each such region, the designer then defines the following parameters:

- *Wrist position/orientation* relative to the tubular section. Both can be specified as either fixed or a range of values.
- *Touch tolerance*, essentially specifying how much a finger can "sink" into the object. This value sets the threshold in the capsule intersection algorithm.
- *Thumb configuration* can be specified as closed or on-the-side. If specified as closed, the grasping algorithm will try to make the thumb encircle the section to be grasped, just like the other fingers. If specified as on-the-side, the algorithm will try to make the thumb touch one of the tubes, in a parallel orientation.
- *Finger spread* specifies the angle in between each of the four fingers, effectively defining how much the fingers will be spread.
- *Finger selection* specifies which fingers will be involved in the grasp.

These parameters are stored in the object description file, together with all the other attributes. There can be multiple sets of parameters per region.

6 Grasping

6.1 Collision Detection

Our real-time grasping algorithm is based on approximating the parts of a tubular section and the finger segments with *capsules*. A capsule (or capped cylinder) is the set of points at a fixed distance from a line segment. Two capsules intersect if and only if the distance between capsule line segments is smaller or equal to the sum of the capsule radii.

Given a finger segment and a tubular region, we first find out which part of the tubular region is most likely to intersect with the finger segment. We accomplish this by intersecting the finger plane with each tube center line segment. We define the finger plane as the plane perpendicular to the axis of rotation of the distal finger joints. It is dependent on the finger spread parameter. We then run the capsule intersection test to determine whether the tube and the finger segment intersect.

To determine whether two capsules intersect, we need to compute the minimum distance between points on two capsule line segments. The parametric equations by the line segments are given by $\vec{L}_0(s) = \vec{B}_0 + s\vec{M}_0$ for $s \in [0,1]$, and $\vec{L}_1(t) = \vec{B}_1 + t\vec{M}_1$ for $t \in [0,1]$. The squared distance function for any two points on the line segments is $Q(s,t) = |\vec{L}_0(s) - \vec{L}_1(t)|^2$ for $(s,t) \in [0,1]^2$. The function is quadratic in s and t, and given by

$$Q(s,t) = as^2 + 2bst + ct^2 + 2ds + 2et + f,$$

where $a = \vec{M}_0 \cdot \vec{M}_0, b = -\vec{M}_0 \cdot \vec{M}_1, c = \vec{M}_1 \cdot \vec{M}_1, d = \vec{M}_0 \cdot (\vec{B}_0 - \vec{B}_1), e = -\vec{M}_1 \cdot (\vec{B}_0 - \vec{B}_1)$, and $f = (\vec{B}_0 - \vec{B}_1) \cdot (\vec{B}_0 - \vec{B}_1)$.

The goal is to minimize $Q(s,t)$ over the unit square $[0,1]^2$. Q is a continuously differentiable function, therefore the minimum occurs either at an interior point of the square where its gradient is equal to $(0,0)$ or at a point on the boundary of the square. [18] includes further details on how this minimization is performed.

For grasping, we need to determine whether the finger segment "touches" the object, therefore the test method described above is not adequate since it merely reports intersections. Therefore, we introduce the touch tolerance into

the capsule collision test inequality as a tolerance value. Let R_{sum} be the sum of the capsule radii, D_{min} the minimum distance between the capsule line segments, and ϵ the touch tolerance. We can distinguish between three cases:

- $D_{min} > R_{sum}$: The finger segment does not touch the object and it is outside the object.
- $R_{sum} \geq D_{min} > (R_{sum} - \epsilon)$: The finger segment touches the object.
- $(R_{sum} - \epsilon) \geq D_{min}$: The finger segment is inside the object.

In fact, the touch tolerance value implies a relaxed suggestion on how much the capsules can sink into each other. This, in turn, can create the impression of a tighter or looser grasp on the object. This is an advantage of using the capsule intersection test for the collision detection calculations.

Even though the choice of (uncapped) cylinder as the collision detection primitive comes into mind, we have decided not to use it. The main reason is that the intersection test for cylinders is a fairly expensive one (e.g. [18] uses the method of separating axes). Furthermore, a capsule gives a nice approximation of a finger segment that includes the finger tip. Another choice for the collision detection primitive would be the box, but we do not use it since the results would be too coarse. We need to create postures where the fingers encircle the tubular sections, which is not possible to do satisfactorily with box-based collision detection.

6.2 Posture search

The final grasp posture is computed by executing a dichotomy search (similar to the one in [14]) in the configuration space of the hand. This space is defined by the range of wrist position and orientation plus the ranges of orientation of the finger joints. Fortunately, its dimensions can be reduced thanks to the anatomy of the hand:

- The metacarpophalangeal (MCP) joints are biaxial joints, with two degrees of freedom.
- The distal interphalangeal (DIP) and proximal interphalangeal (PIP) joints are uniaxial (hinge type) joints, with only one degree of freedom around the lateral axis.
- We can assume that the DIP joint angle is a function of the PIP joint angle, further reducing the dimensions.

There are also optional reductions that can be made, to make the search faster in object-specific cases. The finger spread parameter can be fixed, resulting in reduction of the degrees of freedom for the MCP joints from two to one. Also, in case they are not needed for the grasp, some fingers may be omitted from the search, fixing their posture to a predefined one.

At each step during the search, we generate a hand posture to be tested, which obeys the joint limits. Then, the collision detection algorithm described above is invoked for the posture. The search continues until one of the following:

- A posture that fulfills all the constraints is found. This posture is returned as the final grasp posture, to be used during the object interaction sequence.
- The maximum number of postures that can be tested is reached. If this happens, we assume that the virtual human cannot grasp the object.

In most cases where a valid grasping posture exist, the search will terminate relatively quickly, thanks to existence of the touch tolerance value. In those cases where a grasp posture cannot be found, the most likely course of action is for the designer to modify the design of object to relax the grasp parameters, to increase the chance of finding a grasp posture. This is a consequence of the tradeoff between a fully-automatic grasping method with less control or a semi-automatic method like ours with more control over how the grasping takes place.

The reason why we have chosen to compute the final grasp posture by searching instead of analytical methods is that it provides a practical means to satisfy all the constraints and still remain flexible. Not only the joint limits impose constraints, but there are also dependencies between the joint angle values, primarily between

the DIP and PIP joints. Searching provides an easy solution to these problems, within reasonable computational demands.

An alternative to searching could be inverse kinematics, but use of that method requires specification of the finger touch points (end effector positions) on the object. This is not practical and it may actually be better to ask the designer to design the whole hand posture instead. In fact, we have observed that the actual finger tip positions are not critical as long as a valid grasping posture is generated.

7 Conclusion

We have described our grasping framework, which brings together a tubular feature classification algorithm, a hand grasp posture generation algorithm and an animation framework for human-object interactions. This unique combination is capable of handling grasping tasks within the proper context of virtual human object manipulation. This is very important since how an object is to be grasped depends strongly on how it is be used. Most existing works ignore this aspect.

The method has the advantage that it can work with relatively complex objects, where manual approximation with simple geometrical primitives may not be possible or practical. Furthermore, the method supports many intuitive parameters for controlling the grasping posture, such as the finger spread or the thumb configuration. Since the grasp parameters are specified as ranges, it is possible to generate a different posture each time a virtual human attempts to grasp an object, depending on the current configuration of the virtual human. This introduces a new degree of variety into the virtual environment, which, to the best of our knowledge, was not possible with the previous approaches.

Since Plumber is oriented to the recognition of tubular features, objects without tubular or elongated protrusions cannot be characterized by any grasping oriented annotation. Fortunately, most objects of interest for grasping actually have elongated features which plumber is able to recognize. We approximate these tubular features with capsules (capped cylinders), for efficient collision and touch detection. While this approximation is appropriate for finger segments and mostly cylindrical object parts, it may not be adequate for conical object parts.

Acknowledgements

This work has been supported by the Swiss Federal Office for Education and Science and the European Union in the framework of the European IST-Networks of Excellence AIM@SHAPE.

References

[1] Norman Badler, Rama Bindiganavale, Jan Allbeck, William Schuler, Liwei Zhao, and Martha Palmer. Parameterized action representation for virtual human agents. In *Embodied Conversational Agents*, pages 256–284, Cambridge, MA, 2000. MIT Press.

[2] Paolo Baerlocher. *Inverse kinematics Techniques for the Interactive Posture Control of Articulated Figures*. PhD thesis, Swiss Federal Institute of Technology - EPFL, 2001.

[3] S. Biasotti, S. Marini, M. Mortara, and G. Patanè. An overview on properties and efficacy of topological graphs in shape modelling. In *Shape Modeling International*, pages 10–15, 2003.

[4] B. Bindiganavale and N. I. Badler. Motion abstraction and mapping with spatial constraints. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments (CAPTECH-98)*, volume 1537 of *LNAI*, pages 70–82, Berlin, November 26–27 1998. Springer.

[5] Ronan Boulic, Branislav Ulicny, and Daniel Thalmann. Versatile walk engine. *Journal of Game Development*, 1(1), 2004.

[6] M. Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269–279, 1989.

[7] Z. Huang, R. Boulic, and Daniel Thalmann. A multi-sensor approach for grasping and 3-D interaction. In *Computer Graphics International '95*, June 1995.

[8] Marcelo Kallmann. *Object Interaction in Real-Time Virtual Environments*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2001.

[9] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. In *Proceedings of Eurographics 2003*, pages 313–322, Granada, Spain, 2003.

[10] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 395–408. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[11] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. ISBN 0-7923-9192-2.

[12] Libby Levison. Connecting planning and acting via object–specific reasoning, 1996.

[13] Xuetao Li, Tong Wing Toon, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In *Symposium on Interactive 3D graphics*, pages 35–42, 2001.

[14] Nadia Magnenat-Thalmann, Richard Laperriere, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88*, pages 26–33, June 1988.

[15] M. Mortara, G. Patanè, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Blowing bubbles for the multi-scale analysis and decomposition of triangle meshes. *Algorithmica, Special Issues on Shape Algorithms*, 38(2):227–248, 2004.

[16] M. Mortara, G. Patanè, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In *Ninth ACM Symposium on Solid Modeling and Applications SM'04*, pages 339–344, 2004.

[17] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. *Computer Graphics*, 30(Annual Conference Series):205–216, 1996.

[18] David H. Eberly Philip J. Schneider. *Geometric Tools for Computer Graphics*. Morgan Kaufmann, 2002.

[19] Y. Shinagawa, T.L. Kunii, A.G. Belayev, and T. Tsukioka. Shape modeling and shape analysis based on singularities. *International Journal of Shape Modeling*, 2(1):85–102, 1996.

[20] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graph. Models Image Process.*, 62(5):353–388, 2000.

[21] Anne Verroust and Francis Lazarus. Extracting skeletal curves from 3d scattered data. *The Visual Computer*, 16(1):15–25, 2000.

[22] Spyros Vosinakis and Themis Panayiotopoulos. A task definition language for virtual agents. *Journal of WSCG*, 11(1):512–519, 2003.

[23] Xuguang Wang and Jean Pierre Verriest. A geometric algorithm to predict the arm reach posture for computer-aided ergonomic evaluation. *The Journal of Visualization and Computer Animation*, 9(1):33–47, January–March 1998.

[24] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, November/December 1997.