First Implementation of the Virtual Reality Dynamic Anatomy Tool

By

YOHAN BAILLOT

MS Electrical Engineering, Montpellier, France, 1996

Graduate Committee

Dr. Jannick Rolland, CREOL and School of Optics,
joint appointment School of Computer Science.
Dr. Michael Moshell, School of Computer Science.
Dr. Rebecca Parsons, School of Computer Science.
Dr. Niels Da Vitoria Lobo, School of Computer Science.
Dr. Kurt Lin, Mechanical and Aerospace Engineering department,
Institute for Simulation and Training.

Submitted in partial fulfillment of the requirements
for the degree Master of Science
in the School of Computer Science
in the College of Arts and Sciences
at the University of Central Florida

Fall Term
1998

ABSTRACT

This document summarizes one and half year of research and development in the frame of a larger 5-year research project proposed and led by Dr. Rolland: the Virtual Reality Dynamic Anatomy tool (VRDA). Several researchers from different disciplines including Computer Sciences, Biomedical Engineering, and Psychology participate in this project. The project will lead to the development of a tool allowing a user to see a virtual model of the inside of an anatomical joint superimposed on the corresponding joint of a subject. As the user will manipulate the joint of the subject and move around, the virtual model of the inside of the joint will move accordingly. Because of the merging of real and virtual information as well as the interactivity provided by the system, it is expected that this tool will help students form better mental models of dynamic inner anatomy.

This research provides a framework for the first implementation of the VRDA tool and discusses more specifically the modeling, visualization, and registration issues. The sections of this document are organized as follows. We first introduce in section 1 the problems to solve and set our assumptions. Then in section 2, we review current methods and commercially available software. In section 3, we state our approach, propose some implementations and new techniques, and give experimental results as well. Finally, we conclude with the contributions of this research in section 4 and give some line of sight for future work followed by a bibliography.

ACKNOWLEDGMENTS

I want to thanks my parents that gave me a hard time to make me study when I was younger because it was a good start to reach the point where I am today. I want to thanks Dr Rolland who gave me the opportunity to stay in the US and influence me to get this degree after I completed some work for her at UNC, where I discovered Virtual Reality. I want to thank Dr. Moshell and Dr Lin, as well as the rest of my committee, for reviewing my work during this research. Finally, I want to thank my few friends among the large number of people I just know to help me during the writing of this document, including my current roommates Anna and Scott. A First award of the National Institute of Health (NIH) (1-R29-LM06322-01A1) currently supports the VRDA research.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

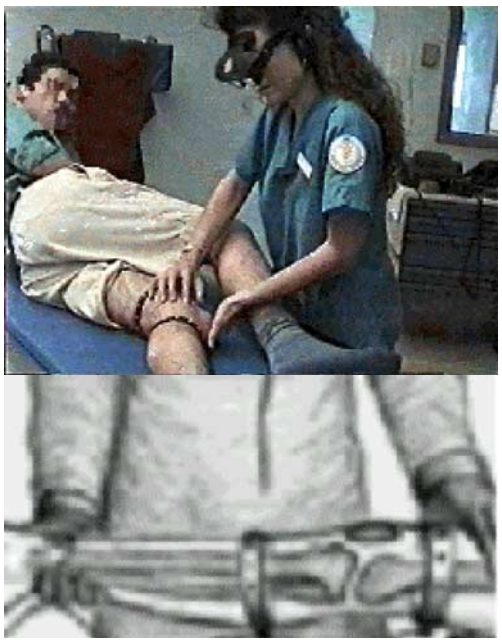

Figure 1: A medical student is manipulating the joint (top) and sees a 3D synthetic model of the joint (bottom)

The Virtual Reality Dynamic Anatomy tool (VRDA) is a device that will allow a user to see the inner components of an anatomical joint, as if the user had 3D X-ray vision (Rolland, 97). As the user manipulates the joint of a subject as illustrated in Figure 1 (top frame), a tracking sensor will measure the attitude of the joint. Using this information, a synthetic image of the inside of the joint is shown to the user on top of the real joint and in the same attitude using a Head-Mounted Display. An illustration of the view the user should have is shown in Figure 1 (bottom frame).

Medical students and professionals need to understand the 3D relationship of internal anatomical structures and the significance of body parts' movement. Current learning processes use still pictures, movies, training simulations, and cadaver dissection labs. On the one hand, these representations usually do not visualize both internal and external structures. On the other hand, even training simulations do not provide the spontaneous feedback that a living human palpation would provide.

The VRDA tool will superimpose internal 3D structures in motion on top of a subject's anatomical joint so that the student will have access to visual as well as tactile cues when learning how the joint behaves. Therefore, we anticipate that this tool will help students to form more accurate mental models of the joint motions in shorter time periods (Wright, 95).

The presented research has been done to create a first implementation of the tool that will allow for further developments. In the following, chapter 1 presents the goals required to realize the tool. Chapter 2 reviews the various methods to model the anatomical joint motions. Chapter 3 presents a new motion modeling method, and chapter 4 shows the validity of the approach on test objects. Chapter 5 presents modeling results on the knee joint. Chapter 6 presents the core simulation application. Chapter 7 describes the Augmented Reality setup used for the VRDA tool, and Chapter 8 describes a novel method to calibrate it.

1 AIMS OF THE RESEARCH

We identified three major goals for the research related to a first implementation of the VRDA tool: modeling, simulation, and superimposition. These parts are now detailed respectively.

1.1 Modeling of the motion of an anatomical joint

The synthetic model represented on the subject joint is an animated set of three-dimensional models representing the components inside the corresponding anatomical joint. Motion modeling is the task that consists of specifying what is the position and orientation of these components as the joint is animated. The modeling is done at discrete points of the animation, called motion steps or key frames.

The components of the model of the joint represent realistically the anatomical parts of the joint. These components are naturally complex in shape, and some of their features hide the contact points. For this reason, specifying manually the relative attitude of the components at each motion step without creating any gap or intersection between the components can be difficult. Moreover, this task becomes tedious when the number of motion steps considered is important.

Additionally, because the synthetic model used must be adapted to the subject joint so that it appears to realistically represent the inside, the model must be scaled. In the future, the actual geometry of the components of the joint of the

subject could be segmented from a detection procedure like X-rays and used to represent the synthetic model. To create systematically a motion model for any anatomical joint efficiently, an automatic method to model the motion of the joint must be used. The main contribution of the presented research is the design of a way to realize this task.

1.2 Simulation of the joint motion

To evaluate the motion model developed on the considered joint, a replay of the modeled motion of the joint must be done. The animation can be smoothly replayed using interpolation between the specified motion steps to obtain a continuous animation. This animation must be examinable on a monitor from any viewpoint in order to assess completely the model.

The subjective impression that the user is looking at a real joint must be realized by using real-time realistic rendering. Real-time rendering is achieved by using appropriate software and hardware resources so that the motion of the joint appears to be happening at the same time that the control producing this motion occurs. Realistic rendering is realized by specifying precisely the material properties of the components and the lighting conditions, and by using quality rendering hardware and software.

1.3 Superimposition of the virtual model on a real joint

Once the model has been assessed on a monitor, the next step in building the VRDA tool is to superimpose the joint on its real counterpart. By integrating synthetic imagery to a real scene, one augments the real world with useful

information. Thus, this technology is called Augmented Reality (AR). Augmented reality has two approaches to superimpose models: optical or video. The optical approach, that we will use, has the advantage eliminates proprioceptor conflicts involved when using a video approach.

To correlate the attitude and location of the synthetic model of the joint with its real counterpart, the attitude and location of the subject joint must be tracked. The rendering software then uses the tracking information to display the model correctly to the user's eyes. By accomplishing this task, the system gives the user the subjective impression that the synthetic model is a three-dimensional virtual object inside the subject's joint.

The precise superimposition of a synthetic object on its real counterpart, known as registration, is still a challenging problem inherent in the tracking hardware (Rolland, 1999) used as well as the calibration of the whole system. Because no ideal technology has been found yet, the tracking hardware has often limited static and dynamic precision, and a limited range of measurement. The multiple technologies found in an Augmented Reality system makes calibration a difficult problem that is still not completely solved today. Moreover, because the variability between humans can be noticeable, the need for user-specific calibrations further adds to the complexity of the problem.

2 CURRENT MODELING APPROACHES

The knee joint is one of the most complex anatomical joints of the human body as far as the motions involved between the components are concerned. Therefore, for this first implementation of the VRDA tool, we chose to make the tool work on this specific joint. More specifically, this research focuses on the bones of the tibiofemoral joint.

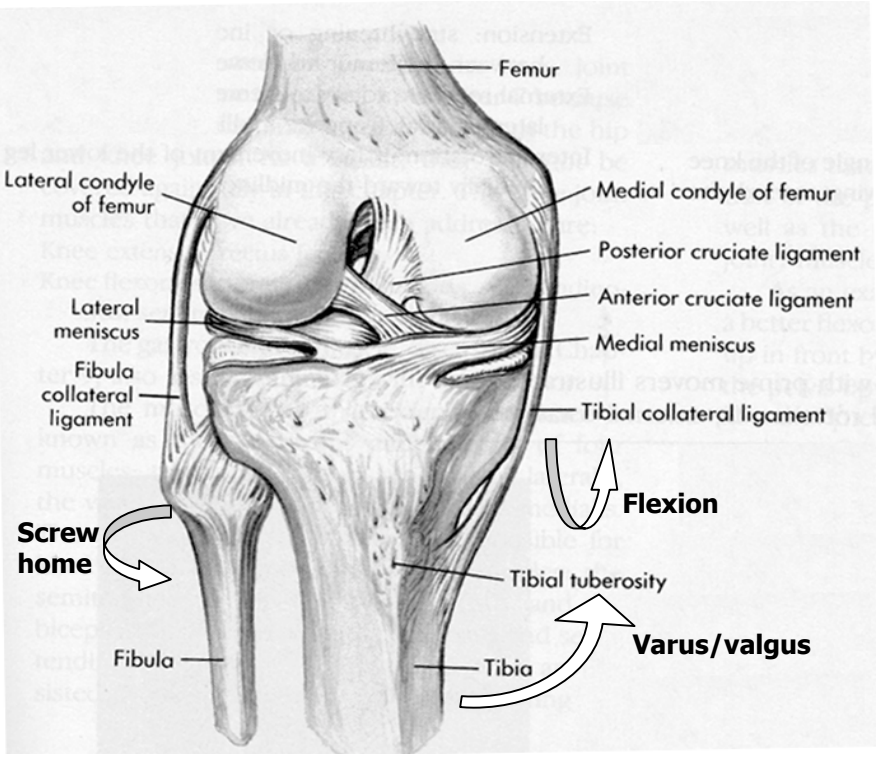

Figure 2: The human knee joint and its components (no patella).

In the next section, a comprehensive description of the knee joint components and motions is given. Then, a review of the modeling methods used in the

biomedical field to model knee joint motion is given. Finally, some software packages used in modeling are identified and their suitability to solve the modeling problem is analyzed.

2.1 The human knee joint

To understand why modeling knee joint motions is challenging, one must first understand how the human knee joint works. This section describes the knee joint components, their functionality, and their mobility.

An illustration of the tibiofemoral joint is shown in a frontal view on Figure 2, with the name of the components and the motions involved. The joint is composed of bones, muscles, tendons, menisci, and ligaments. The bones are the rigid structure of the link and are activated by the muscles attached to them by the tendons. The menisci are lubricating the contact of the link, while the ligaments are ensuring its cohesion.

The bones are the tibia and the femur. The patella is not considered in this first study. The fibula motion does not play a predominant role in the motion of the joint, so it is not considered in this study either for the modeling part. The femur has two sphere-like contact surfaces called the condyles. The medial condyle is closer to the center of the body and is larger than the lateral condyle. The tibia has two corresponding spherical surfaces receiving the femur condyles, which are called the tibial plateaus.

The menisci are divided into the lateral and the medial meniscus. The meniscus is a sponge-like body, however its deformation is not perceptible to the human

eye (Frankel, 97). The menisci are lubricating the contact between the bone surfaces. Moreover, they take some of the load of the body acting at the contact surfaces by increasing the contact surface and distributing the pressure. During the motion of the joint, the menisci slide on the tibial plateaus and wrap around the condyles to naturally fill the eventual gap between the bones surfaces (Thompson, 89; Minami, 91; Bylski-Austrow, 94).

The ligaments are spring-like strings that connect the tibia and the femur at specific points. Their interactions with the shape of the contact surface drive the motion of the bones as the joint is moved (Huiskes, 90). Some of the ligaments are the lateral, medial, and anterior and posterior cruciate ligaments. The ligaments are not modeled in this study because they do not enter in our general modeling scheme. Further studies will attempt to consider them.

The muscles creates the motion of the joint along two degrees of freedom known as the flexion/extension and the varus/valgus, as illustrated on Figure 2 (Nordin, 80). These degrees of freedom condition the relative orientation of the bones along two perpendicular axes. The ligaments and the contact surfaces create constraints limiting and producing the relative positions of the bones, and their orientation along the long axis of the tibia. The varus/valgus has a limited range of motion due to the constraints imposed by the ligaments and the contacting surfaces.

While one may think that the knee joint is a simple cylindrical liaison as the knee is flexed, the instant center of rotation of the tibia with respect to the femur is not

fixed in space. If the joint is considered planar and viewed along the sagital plan as shown on Figure 3, the instant center of rotation of the bones is actually describing a well known C-shaped curve (Soudan, 79; Nordin, 80). The center of rotation is not fixed because the shape of the condyles along the sagital plan is elliptical rather than circular.

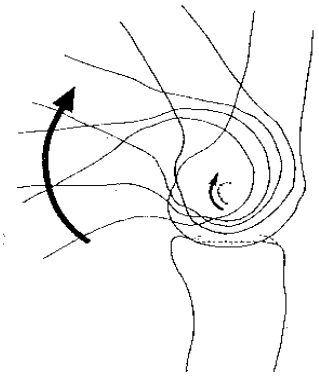

Figure 3: The C-curve described by the instant center of rotation as the knee is flexed (Nordin, 80)

Furthermore, a three-dimensional view of the joint shows the instant axis of rotation of the bones, illustrated in Figure 4. The path described is far more complex than the one that would be obtained in the case of a cylindrical liaison when the joint is flexed. The ligaments are attached at fixed points on the bones and, together with the contact surfaces shapes, form a sophisticated constrained kinematic system whose motion is complex.

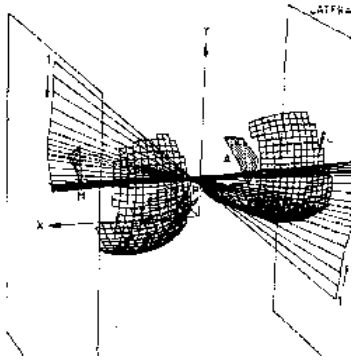

Figure 4: The 3D axis of rotation of the tibia with respect to the femur as the knee is flexed (Blankevoort, 88)

The condyles of the femur do not only slide on the menisci and the tibial plateaus as the knee is flexed, but they also roll. Consequently, the contact points of the condyles on the tibia and the menisci are not fixed and the instant axis of rotation moves.

Additionally, because the femur is asymmetric since one of the condyles is larger than the other, a rotation of the tibia along its main axis known as

screw-home is produced as the knee is flexed (Shiavi, 87; Blankevoort, 90; Conlay; 94). The tibia performs an external rotation due to the longer circumference of the medial condyle. The opposite rotation, internal this time, produced when the knee is extended produces a locking process of the joint ensuring the stability of the joint.

2.2 Modeling methods

In this section, a review of the modeling methods used to model the motion of a knee joint is done. We distinguish two main categories of methods: the methods using some approximation of the knee joint motions to yield simpler models, and the methods considering the exact motion of the joint produced by the dynamic interaction of the components. Each category will be now described.

As stated earlier, the modeling of the motion of the joint consists of specifying the position and orientation of the bones as the joint is manipulated along the degrees of freedom. In the case of the knee joint for example, the position and orientation of the tibia with respect to the femur have to be known as the joint is flexed so that the motion appears to be realistic.

The modeling of the motion of the knee joint using a more or less automatic technique has been a large area of research in the biomedical field but not in the virtual reality field. We found that the problems the researchers wanted to solve are different from ours. We shall review the modeling techniques employed to discuss the reasons why they are not adapted to our needs.

2.2.1 Methods using approximation

A simple way to do motion modeling is to manually specify the position and orientation of the components for any attitude of the joint in the range of motion considered. In the case of the knee for example, one can perform this task for any flexion and varus/valgus angle, so that the bones are in contact realistically. First, this task is tedious since the number of attitudes considered could be important. Second, the change of model used for the bones requires specifying a new motion model and this task can take days or months. Third, the complex shape of the contacting surfaces prevents having a clear view of the contacting surfaces, so it is difficult to verify visually that the modeling is correct. By correct we mean that the motion is smooth and geometrically convincing.

Numerous researchers are using motion curves that represent the trajectory of the components as the range motions on the degrees of freedom are spanned. These curves can be extracted from the literature on previous measurements or can be measured on a subject (Ounpuu, 91). First, measurement techniques are typically not precise because non-invasive measurement is currently not precise and invasive measurements cannot be done on living subject onto whom the VRDA tool is used. Second, invasive measurements on a cadaver are still complicated because access to the bones can only be granted by the removal of components conditioning the motion of the bones. Third, in any case, the motion measured on a joint has to be adapted to the same geometrical model of the joint, which cannot be done at this time because non-invasive measurements are not precise. Fourth, the change of model using this method would require

developing some methods to change the motion curve accordingly. Fifth, measurements found in the literature are usually not accompanied with the geometrical model of the components and the frame of reference necessary to produce the same motion.

Some models are two-dimensional, thus the motion of the components as the knee is flexed is assumed to be planar (Yamaguchi, 89). The technique employed is usually to make a mathematical description of the contour of the tibia and of the femur to find the location of one bone relative to the other as the knee is flexed. A simple way to describe the femur mathematically is to use an ellipse or a spiral to approximate the contour of the condyles (Rehder, 83; Kurosawa, 85; Loch, 92), while the tibial plateaus are fitted to a plan (Delp, 90). Then, because the location of the contact points of the condyles on the tibia is referenced in the literature (Nisell, 85), the motion of one bone with respect to the other can be easily deduced. This type of model assumes that both condyles have the same section, which is not correct, and cannot realistically be employed for a three-dimensional model. Further, some three-dimensional mathematical models have been defined but because a mathematical model is elaborated for a specific joint, the method cannot be extrapolated to any joint (Wismans, 80; Kuiper, 88).

The method that is the closest to our approach uses correction of the motion parameters extracted from experimental data by constraining the femur condyles in exact contact with the tibia. The algorithm uses the distance between control points of B-splines approximating the femur condyles and the tibia (Walker, 88;

Ateshian, 93). This model uses the geometry of the joint and thus can be adapted to any joint. However, it considers only two contacts, which is not realistic. Another problem is that it cannot model the screw-home motion.

2.2.2 Use of constrained rigid body dynamics

Most of the current knee models are typically three-dimensional dynamic models (Wismans, 80; Blankevoort, 88; Blankevoort, 91; Garg, 90; Huiskes, 90). These models are dynamic because the research attempts to determine parameters related to forces or loads. One goal of these models may be be to estimate parameters that cannot be determined in vivo. For example, the reference strain of a ligament has been determined using this method (Van Eijden, 86).

The locations of the components are determined using Newton's laws combined with kinematic constraints known as rigid body dynamics. The first problem with such models is that the implementation of dynamics on a computer is known to be computationally intensive. Further, its stability and correctness is greatly dependent of the initial conditions and the step size employed in the numerical computation. The details of Rigid Body Dynamics are given in Appendix A.

Kinematic constraints are usually determined from literature measurements. For example, the contact point location or the orientation of some components can be used to reduce the degree of complexity of the model. However, generally, these models are only adapted to the joint used and they cannot be used for geometric models of any shape and size (Seedhom, 72, Huiskes, 85).

To our knowledge, there exists no model of this type that has been successfully applied to the knee joint for the complete range of motion of the flexion (Hefzy, 96). In addition, none of the models developed is modeling the screw-home motion.

2.3 Off-the-shelf software packages

We reviewed modeling software that could have helped us create a motion model. This allowed us to determine their suitability to solve our modeling problem. Such packages need to have a collision detection capability to detect intersections the user cannot see because of the complexity of the contacting surfaces. Collision detection algorithms use the geometric specification of the model to detect the intersection of graphics primitives. Additionally, these packages should have a motion curve viewer to compare the created motion curves with the ones found in the literature.

We first investigated the biomedical software SIMM available from Musculographics Inc (Delp, 95). This software was chosen because it has been developed specifically for biomedical modeling of anatomical joints. It provides a graphical interface divided in modules: the viewer and the curve editor are two of them. Given a joint, the user specifies the motion curves of the bones using the curve editor. The viewer is showing polygonal models of the bones animated by the motion curves specified in the modeler. The mouse can be used to change viewpoint.

A configuration file for each component of the joint is used to specify the motions range and the values of some parameters of the components of the joint. For example, these files can be used to describe the muscles and the ligaments attached to the bones in order to compute forces and torques produced at any point of a bone subject to a muscular activity.

Despite the qualities of this software, we decided to not use it because it was not adapted to our need. The viewer is not designed for interactive modeling necessary to specify the motion of bones of arbitrary size and shape rather than using the curve editor. Moreover, no collision detection capability is integrated to help the user in the modeling task.

3D Studio from Kinetix and Power Animator from Alias/Wavefront are famous animation packages working respectively on PC and on SGI. They are the most used software for the synthetic animation industry and include advanced features allowing complex rendering. However, they do not support collision detection according to the respective technical support consulted at this occasion. While the modularity of the packages provide an advantageous way to import plugins that add new capability and that can be created by anybody, we did not try this solution to create a modeling method.

Other packages such as Multigen and AutoCAD as well as other packages found on the web, were all rejected from our review. Some did not have the required capabilities, others required too much time to learn and extend. Others were not

available for evaluation. This review brings us to the design of a new application handling our specific problem, that is the main contribution to this research.

3 A NOVEL ALGORITHM FOR MODELING ANATOMICAL JOINTS MOTIONS

None of the modeling approaches identified in chapter 2 has been found useful to produce a motion model appropriate for the VRDA tool. We thus designed a novel automatic modeling method to model the motion of anatomical joints.

The method relies on the use of an original algorithm to find the stable position and orientation of two rigid bodies in contact. We shall first describe our approach to modeling the motion of anatomical joints and then describe the novel algorithm features in detail.

3.1 Overall approach

The method consists in first dividing the range of motion into motion steps. In the case of the knee for example, the varus/valgus and flexion/extension motions will be divided into small angle slices of one degree. Each attitude of the joint, characterized by a varus/valgus angle and a flexion angle will then be a motion step. At each motion step, the stable position and orientation of the bones of the joint will be searched, considering that the bones are pushed along a direction that we shall call Δ. Once a stable position and orientation is found for a motion step, the position and orientation of the bones will be recorded in a lookup table indexed by the entry angles. This lookup table will then be used during the simulation.

To perform the search at each motion step, an original incremental algorithm detailed in sections 3.2 to 3.7 is used. The algorithm works on two rigid bodies, which means that the geometrical models do not change in shape and size during the process. One is fixed and is called the reference, the other is moving and is called the moving object.

During the use of the VRDA tool, the joint of a model patient is under no load since it is manipulated by a user. The muscles activity as well as the weight of the subject do not play a predominant role in the attitude of the joint. However, the ligaments and the surfaces are still constraining the motion of the components of the joint in a stable position as the knee is flexed. Therefore, it is valid to consider that the ligaments and the contact surfaces produce some kinematic constraints on the joint motion, which is the base of the modeling algorithm proposed.

3.2 Stability search paradigm

The joint motion we want to simulate in the VRDA tool does not require dynamics because no load is considered. However, the reaction forces that are considered in a dynamic approach seem to be appropriate to find the stable attitude of the components. Specifically, our algorithm uses the normals at the contact points that are in the same direction as the reaction forces used in Dynamics. However, the novelty of our algorithm is that it only considers the direction of the reaction forces and nothing else. Consequently, the result of the algorithm is not

dependent upon the initial conditions of position, speed, and of the reaction force amplitudes.

Therefore, the algorithm will always find the local minimum corresponding to the chosen initial position. When applying the algorithm to anatomical joints, the shape of the contacting surfaces and the initial position of the bones before searching the stable position are thus taken in consideration.

Figure 5 shows a flow chart of the working principle of the algorithm. A typical modeling step consists in detecting the collision of the rigid bodies, removing the rigid bodies out of collision to establish exact contact. The contact points and the normals at these contact points are determined. By combining the normals, the algorithm determines the next motion, which can be a rotation, a translation, or the stability of the moving solid. The algorithm moves the solid along Δ when there is no collision. Therefore, upon collision, the translation occurs in a plan perpendicular to Δ and the rotation is produced around Δ.

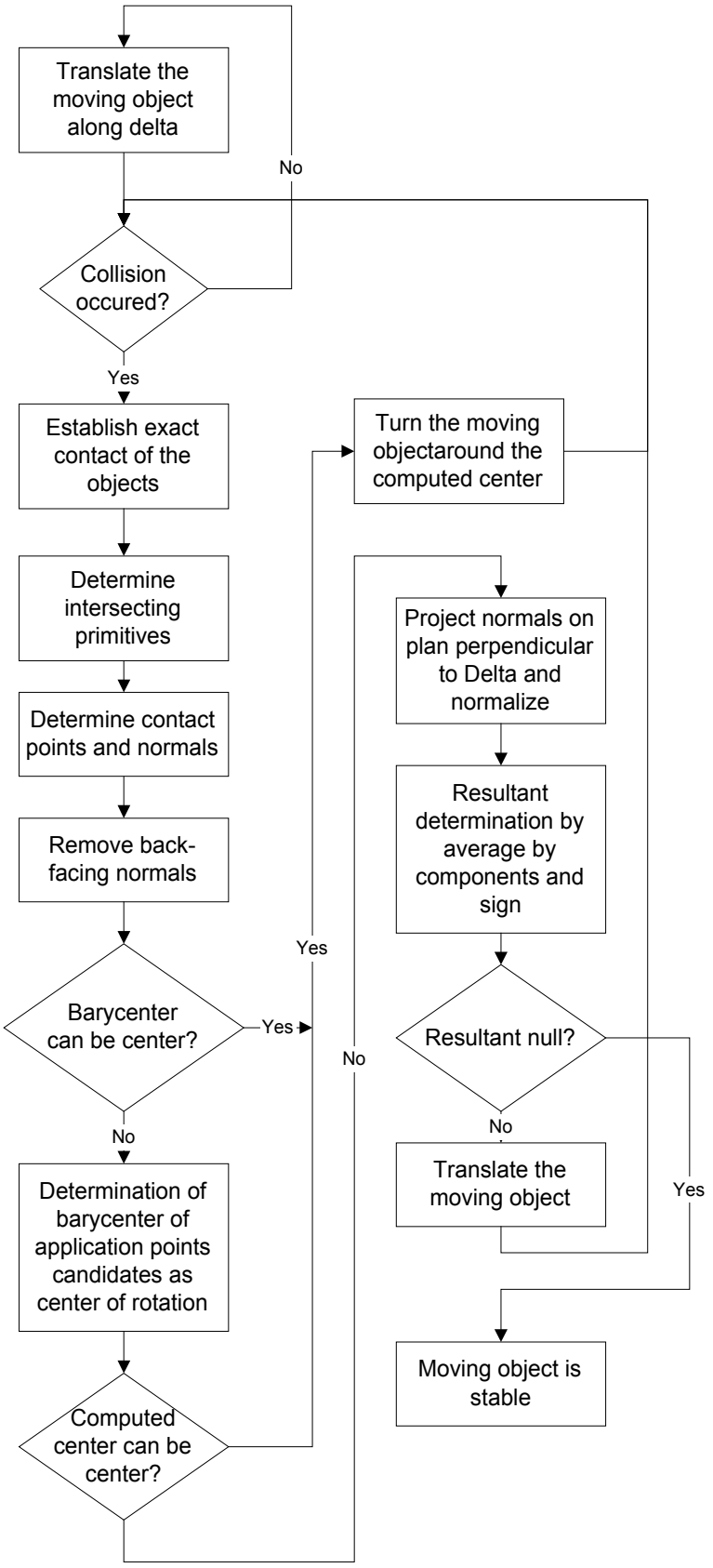

Figure 5: flow chart illustrating the working principle of the algorithm to search a stable relative position and orientation of two rigid bodies.

3.3 Determination of the contact points and normals

We use exact collision detection to determine the graphical primitives (e.g. polygons) of the geometrical models or objects that are intersecting according to the current attitude of the objects.

Collision detection is a subject that has been largely studied and there exist several algorithms adapted to different types of graphical objects such as polygon-based, NURBS-based, rigid, or deformable models. The algorithm we have developed for modeling can employ any of the existing collision algorithms depending upon the type of graphical model considered.

Once the intersections of the objects have been detected by collision detection, exact contact is established between the objects in order to determine the contact points. This task is done by performing a dichotomy on the last motion step and detecting at each step whether or not there is collision.

Because the algorithm produces a translation or a rotation of the moving solid at each motion step, the dichotomy can be done on one of these two motions as well. The dichotomy stops when the step size of the dichotomy is below the user-specified modeling resolution for the translation or the rotation according to the last motion executed before collision..

The colliding primitives at the last step of the dichotomy are memorized to determine the contact points of the objects. The colliding primitives are given as a list of pairs of primitives, with the primitive on the reference solid first and the primitive on the moving solid next.

If the models are described as polygons, the exact contact points can be determined in theory by intersecting the lines going through the sides of one of the intersecting polygons with the surface of the other, and vice versa. If the intersection is a point, the contact is a point. If the intersection is a line, then the contact is a line. If the intersection is at least two lines, then the contact is a surface, and its extent is determined by computing the intersections of the other lines of the polygons on both objects. Then, a contact point should be considered at each vertex of the intersection surface.

Because the dichotomy search has a limited resolution, the contact is not exact and neither is the intersection. One must use a method that is not sensitive to approximate contact. Currently, our algorithm uses the center of the triangle primitives as the contact points. This approximation is valid if the triangles are small enough so that their centers are roughly at the same location than the contact points.

The algorithm uses the contact point as the application point of the normal to the surface at a specific contact point. The normal is determined by interpolation of the normals at each vertex of the polygon. The normals at each vertex are given in the geometrical model.

The normal obtained is oriented and placed as the reaction force that would be produced in the real world. Because our algorithm currently uses the center of the triangles as the application point, we average the normal at the three vertices of the triangle to obtain the normal at the center.

Each intersection of two primitives produces two normals, one on the reference object and one on the moving object. However, only one of them must be considered for each contact point. The choice of taking a contact point on the reference object or on the moving object requires the comprehension of the working principle of the algorithm, so it will be detailed further in section 3.7. A normal orientation is only function of the orientation of the object to which it is attached. However, an application point location is function of both the position and the orientation of the object to which it is attached.

The last step of the procedure is to ignore the normals that are directed in the half plan oriented by the direction of Δ. This can happen when during the last step of the collision a primitive at the back of the reference model is intersected. Because these reaction forces will not be produced in the real world, and because they would contribute to push the solid against each other instead of pushing them apart, they are removed from the list. These normals are detected by computing the dot product of the normals with the vector Δ and verifying that it is not negative.

3.4 Rotation prediction

A torque around a center B can be generated if all the reaction forces directed by the normals produce momenta of same sign around the center. If the signs of the momenta for different normals change, the moving object will not turn because in the real world, the reaction forces would balance each others to produce a null torque.

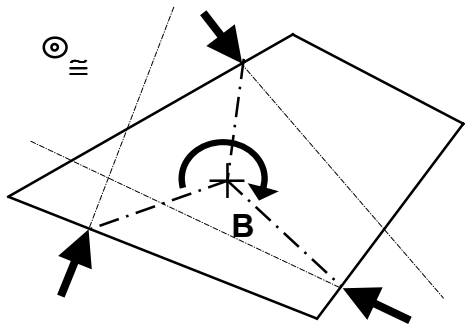

Figure 6: Illustration of a case where the isobarycenter of the application points can be a center of rotation

To compute the momentum produced by a normal around a center, one must compute the cross product of the normal with a vector going from the considered center to the application point of the normal. The algorithm only verifies that the signs of the momenta are the same among all the normals. If it is the case, the axis of rotation is chosen to be parallel to Δ as stated earlier and the axis passes through the considered center B (see Figure 6).

The procedure to determine is the moving solid turns is the following. First, the algorithm tries to take the isobarycenter of all the application points as the center of rotation. If all the normals at the considered contact points produce some momenta of same sign, then the rotation of the solid is performed around the axis parallel to Δ and passing by the isobarycenter. This case is depicted in Figure 6.

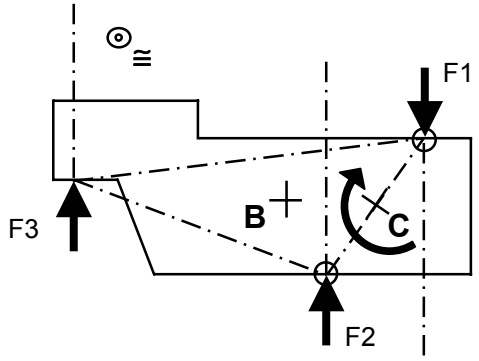

Figure 7: Illustration of the case where the isobarycenter cannot be center of rotation and each application point is evaluated as a possible center of rotation. In this case, only the application points of the normal F1 and F2 can be centers.

If the barycenter cannot be the center of rotation, then each of the application points is taken as a possible center of rotation. The computation of the momentum produced by the normals at the remaining application points is computed to

determine its suitability as a center of rotation. Then, the isobarycenter of the application points that are suitable to be center of rotation is taken as the center of rotation. This case is illustrated Figure 7. As a security step, the algorithm verifies if the computed center is valid using the momentum produced by the normals around this point. In all the tests of the algorithms, we found that the center selected was valid.

In both cases, the amount of rotation is set by the user-specified rotation-modeling step. The direction of rotation is set by the sign of the momenta as in the real world. When a rotation is produced, the algorithm then iterates to the next modeling step defined in section 3.2. The algorithm verifies if a rotation of the solid is possible in a quadratic time of the number of contact points.

3.5 Translation prediction

If none of the rotation tests is satisfied, the algorithm attempts to translate the moving solid along the plan perpendicular to Δ. In order to determine the resulting motion that should be performed to translate the solid toward its stable position, we reviewed several potential solutions solutions.

Each solution consisted in sources vectors extrapolated from the normals and combined using an operation whose result determined the next motion step. Some types of source vectors could have been the normals themselves, the projections of the normals on the plan perpendicular to Δ, the unit vectors directed by the normals, or others. Some types of operation could have been the vector sum or the averaging of the vectors.

The implementation of all the types of source vectors and operations in our modeling software allowed us to verify the applicability and detect the problems of each combination source vector/operation. This explorative study allowed us to establish that the combination of motion vectors using the average by component is the method leading to the most natural motion of the moving object.

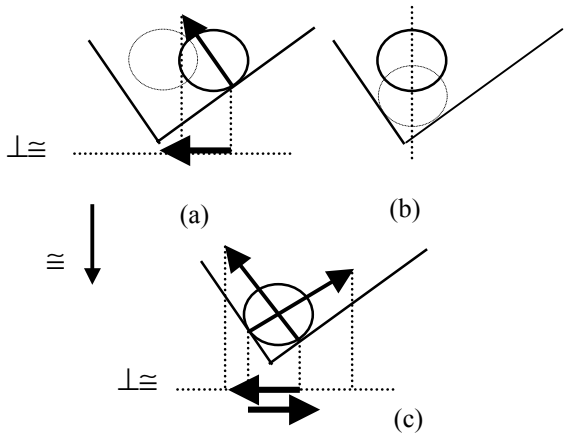

Figure 8: Illustration of the use of the normal to determine the next motion to perform towards the optimal position. (a), (b), (c) represent the sequence of locations that the circle performs. The horizontal arrows are the normalized projection of the normals at the contact points on the plane perpendicular to Δ. In (c) the sum of these vectors is null and the circle is stable.

We define the motion vector corresponding to a normal as the unit vector directed by the projection of the normal on the plan perpendicular to Δ ($\perp\Delta$). The unit normalization makes the influence of each normal independent of the slope of the contact surfaces because motion of the moving solid toward its stable position and orientation should not be influenced by the slope of the contact surfaces.

We then use the average by component method to combine the various motion vectors. The average by component method considers four orthogonal directions on the plan perpendicular to Δ. For example, if this plan is defined by the X and Y-axes, the four directions could be +X, -X, +Y, and –Y. All the motion vectors are projected along these four directions when applicable. Then, the projections

are averaged along each direction to produce four new vectors. Finally, these vectors are added together to produce a resultant vector that sets the direction of motion of the moving object.

Such a combination of the vectors has the particularity to remove the duplicate vectors of same direction and compute the resulting vector in a linear time of the number of application points. The resulting vector is guaranteed to be within a square of side dimension equal to two on the plane perpendicular to Δ. The algorithm multiplies this vector by the modeling step size to obtain the translation that the moving rigid body must perform.

The moving solid is stable when the amplitude of the translation predicted is below the translation-resolution step. The resulting reaction force is in this case almost directly opposite to the direction of motion Δ and thus the moving object should be stable. When a translation is produced, the algorithm iterates to the next modeling step.

Figure 8 illustrates the use of the method in the two dimensional case with a circle falling between two lines. Figure 9 shows how the vectors are combined in the plane perpendicular to Δ to obtain the resultant vector when considering the three dimensional case.

When a planar surface on one of the object is divided in several graphical primitives, several normals oriented in the same direction may be detected. It can be noted that the computed resultant is not the exact motion to perform, but it is a

better approximation than using the vector sum because it removes the duplicate vectors of same direction, as illustrated in Figure 9.

This heuristic approach may generate an invalid motion that can be produced when the computed motion of a moving object is making it move into a surface with which it was in contact.. The correct motion is obtained by applying the cycle detection capability now detailed.

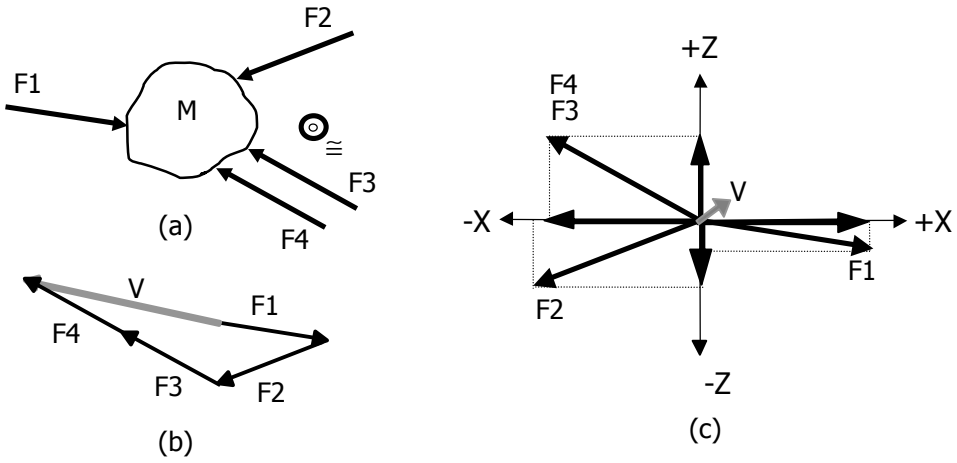

Figure 9: (a) A moving object M viewed along the direction Δ is submitted to multiple forces (Fx) that do not produce a torque. (b) The vector sum leads to a resultant vector (V) that is too large due to the duplicate vectors F3 and F4. (c) The average by component method allows the removal of duplicate vectors of same direction.

3.6 Detection of cycle

The procedure of detection of cycle verifies after each step of the stability algorithm defined in section 3.2 if the current relative attitude of the objects is similar to a previous one at the precision of the modeling resolutions. The attitude of an object is similar to a previous one if the difference in angle is less than the

angular resolution and the difference in position are less than the translation resolution.

If any such condition occurs, a cycle is detected and the modeling step size corresponding to the last motion, which is either a translation or a rotation, is divided in half. The modeling step size of the translation is reset to its original value when a torque is produced. The modeling step of the rotation is reset to its original value when a translation is produced.

The detection of cycle first forces the convergence of the algorithm given a translation step size and an initial position. In effect, imagine a circle between two inclined surfaces forming a sink as illustrated in Figure 8. The circle ends up exactly at the bottom of the sink only for some discrete initial positions and step sizes, else it will indefinitely oscillate at the bottom of the sink between the two surfaces. With the detection of oscillation, the translation step size will be reduced in half at each oscillation to finally reach a value inferior to the translation resolution, where the moving solid will be considered stable.

Similarly, the cycle detection also fixes the problem occurring when the algorithm does not produce a torque when one should be produced. Imagine a bar falling between two pyramidal surfaces as illustrated in Figure 8. Because of the initial position and modeling step size, the bar can oscillate in rotation this time between two surfaces. Therefore, the algorithm does not detect the production of a torque because there is no detection of a center of rotation if there is only one

application point. By using the oscillation detection, we can make the bar finally touch the two surfaces and produce a torque.

The final contribution of the detection of cycle during the translation is to compensate for the non-zero resultant motion obtained using the average by component method. In this case, an invalid motion due to forces that are directly or quasi opposites can be produced: the moving object goes in collision with a surface of the reference object that it already touches. Because of the detection of cycle, the collision with the surface will make the moving object retry the same motion with a translation step size divided in half at each step of the algorithm. Finally, the translation step size will be under the translation resolution and the object will be correctly considered stable.

3.7 Curvature and contact points issues

We pointed out earlier that each of the normals and application points must be taken either on the reference object or on the moving object. This section explains how the choice must be done. Imagine a pyramidal cap oriented along the axis Δ and pushed along Δ in a pyramidal sink as illustrated Figure 10. The top frame shows a top and 3D views of the objects. On the bottom, two schematic views are shown with the normals at different considered contact points. The white vectors are the normals extracted from the sink while the gray vectors are the normals extracted from the cap. The cap has to turn around Δ due to a torque and to translate along Δ to find a natural stable position.

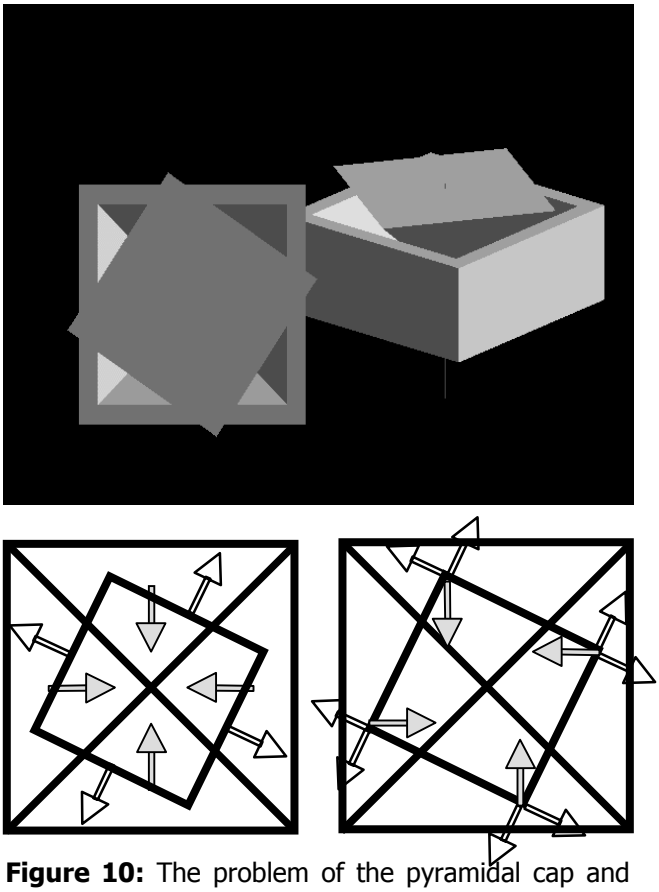

Figure 10: The problem of the pyramidal cap and sink. The top frame represents a top view and a 3D view of the objects. The bottom frames are schematic views with the normals extracted from the cap (white) and from the sink (gray). The left frame considers the normals at the center of the triangles, while the right frame consider the exact contact point.

Now, suppose that the normals are taken at the triangles centers and each face of the object is made of exactly one triangle. This case is illustrated in the bottom left frame in Figure 10. In this case, we note that no torque can be produced since all the normals are going through the center. Because the edges of the cap that touch the surfaces of the sink are sharp and discontinuous, two surfaces would be detected to be in collision on the cap at each contact point.

By applying the normals at the exact contact points rather than at the triangles' centers, as illustrated in the bottom right frame in Figure 10, a torque can be detected if the normals are taken on the pyramidal sink. In effect, we can note that because the surfaces of the cap are discontinuous, i.e. sharp, at the contact points, the two surfaces are in collision and two opposite torques are produced at each point. In the real world, because the surfaces are always continuous only one reaction force corresponding to the normal we used at each contact point would be produced exactly opposite to the one on the sink. However, such a

normal cannot be computed using polygonal models, therefore we should not take the normals on the object having a sharp contact point.

These observations showed that the algorithm must apply the normals at the exact contact points. Furthermore, at each of these contact points the algorithm must take the normals to the surface of least curvature if the models are described using discontinuous surfaces such as polygons. The problem of finding which surface has the less curvature can be solved using a connectivity graph that contains an edge between every pair of surfaces that share a common side. By computing the dot product of the normal at the contact surface with the normals of the adjacent surfaces, one can estimate a value proportional to the curvature of the object.

If the models are described using continuous geometry such as patches for example, choosing the curvature is not a problem since on each object the normals will be the same and will be unique for a given contact point. In the discontinuous case, if the normals are taken on the object with the least curvature at each contact point, the algorithm will then be able to determine the correct motion in the general case.

3.8 Contributions of the algorithm

The described algorithm searches the stable relative position and orientation around Δ of two rigid objects when pushing them against each other along a defined axis Δ (Baillot, 99). The algorithm produces a stair-like motion when the

solid is moving toward its stability because the translation along Δ and perpendicularly to Δ are treated separately.

The complexity of the algorithm proposed is largely reduced compared to a method using the Rigid Body Dynamics because it does not involve integration. Additionally, it does not use force, speed, and acceleration amplitudes, thus it is robust given it is quasi independent of the initial conditions. A large range of modeling step sizes and initial relative positions and orientations of the object make the algorithm converge to the same result. In fact, we postulate that if the initial position of the moving object is such that in reality the two objects would find a stable position, our algorithm could be shown to converge as well.

In addition, because the design of the algorithm is based on the real geometry of the object, the algorithm can be employed on any model size and shape. This algorithm could be employed for animation purpose by removing the showing of the step where the objects are not in contact, thus replacing the stair-like motion by a smooth motion.

In the case of the modeling of an anatomical joint, when the algorithm finds a stable position, a motion step is then completed. The position and orientation of the objects are recorded in a lookup table indexed by the entry angles spanning the degrees of freedom. Then, the modeling algorithm iterates to the next motion step by changing the attitude of the reference object using the entry angles of the joint. Finally, the algorithm is applied on the new motion step and when all the motion steps have been considered, the modeling is completed.

4 IMPLEMENTATION AND VALIDATION OF THE ALGORITHM

In this chapter, we describe the implementation of the proposed algorithm detailed in Chapter 3, as well as the validation of its behavior on test objects.

4.1 Algorithm implementation

A SGI Onyx VTX workstation equipped with two 150 MHz processors was used for the implementation of the algorithm. The application executes both the graphical and the modeling processes in parallel. The graphical process displays the geometrical models whose position and orientation are computed by the modeling process. The modeling process initializes the application by loading the geometrical models, reading the modeling parameters, and creating the graphical process.

First, the modeling process loads the geometric models. The objects' geometry is specified using a file format referred to as Vertices-Faces (VF) and is given the extension "obj". The VF format is divided into three parts. The first part describes the number of vertices and faces that compose the polygonal model. Secondly, the three dimensional coordinates of the models' vertices are listed sequentially in order corresponding to their index numbers. Finally, each polygonal face is enumerated by specifying the number of vertices followed by the listing of the index numbers. Examples can be found on the provided floppy disk in the directory *modeler/models.*

At the time the models are loaded, the polygons are divided into triangles using a recursive algorithm. Each recursion utilizes the first three vertices of the polygon to construct a triangle, which is removed to form a new polygon. When the polygon is completely divided, the algorithm is terminated. This algorithm is not optimal because it does not attempt to achieve the homogeneous subdivision of each polygon. A homogeneous subdivision would theoretically divide the model into fewer triangles and therefore increase the modeling and rendering speeds.

The second phase of the initialization consists of reading a configuration file specified in the command line calling the application. The configuration file allows the user to change the controlling parameters of the application instead of hard-coding them, thus avoiding frequent recompilation. The configuration file is a text file containing a keyword in each line followed by values characterizing the parameter designated by the keyword. The parsing is kept simple and no error check is done. However, the application displays the interpreted parameters' values for debugging. Examples of typical configuration files are included on the floppy disk in the directory *modeler/config*. The keywords used in the configuration files are detailed in Appendix B.

The initialization of the application finishes with the creation of the graphical process. The modeling and graphical processes are self-assigned to the first and second processor, respectively. Both of these processes communicate through a shared data structure that stores common parameters such as the three-dimensional geometry of the objects.

The graphical process renders the geometrical models in their current attitude during the modeling. The source code consists of a conventional rendering loop written in OpenGL executing the display lists created in the initialization phase. Some functions displaying the axis of the models as well as the normals' direction have been implemented for debugging.

The modeling process runs the algorithm described in chapter 3. To perform the detection of collision between objects, the collision detection library RAPID, designed at the University of North Carolina at Chapel Hill (UNC-CH) was used. RAPID is a freely available package written in C that determines efficiently the triangles of two rigid bodies that are intersecting (Ponamgi, 95; Gottschalk, 96). This library is adapted to the graphical models we use since it works on rigid models specified as triangles. The library has two calling functions. One asks for the position and orientation of the models and return the triangles that are intersecting in both models if there is intersection. Another quicker function can be used to only determine whether the models are intersecting.

The modeling process currently considers the Δ-axis is along the Y-axis. Therefore, the projection of a normal on the plan perpendicular to Δ to form the motion vector, can be performed by simply taking the X and Z coordinates. The dichotomy used to determine exact contact is simplified for the same reason.

Using the described implementation, the modeling algorithm uses the geometry of the contacting surfaces to determine the stable position along the X and Z axis of the moving object and its orientation around the Y-axis. For more details, the

source code of the application can be found in the directory *modeler* on the provided disk.

4.2 Algorithm validation

In order to validate the translation and orientation component of the algorithm, we used several test objects to verify the algorithm behavior. The Y-axis, vertical in OpenGL, is taken as the Δ-axis to suggest the gravity force exerted on the moving object. The verification of the correctness of the algorithm was done by verifying if the moving object was naturally “falling” in the fixed reference object. Because the results of pushing an object down or up toward another should produce the same result, we tried for each test to invert the reference and the moving objects to verify the invariance of the algorithm

In the following, we first show the results of the translation because this feature was less complex to verify than the rotation. In effect, for the rotation to be produced, the object has first to translate along Δ, so the verification of this feature requires that the translation has been verified first.

4.2.1 Test of the translation feature

To test the translation feature, we used objects that guaranteed no possible torque and we verified that the algorithm indeed did not detect any torque. We tested that a sphere was correctly directed at the center of a dish-like object when it is dropped anywhere above the dish. The choice of these objects is because a condyle resembles a sphere and a tibial plateau resembles a dish.

Three frames taken during the modeling of a ball falling in a dish can be seen in Figure 11.

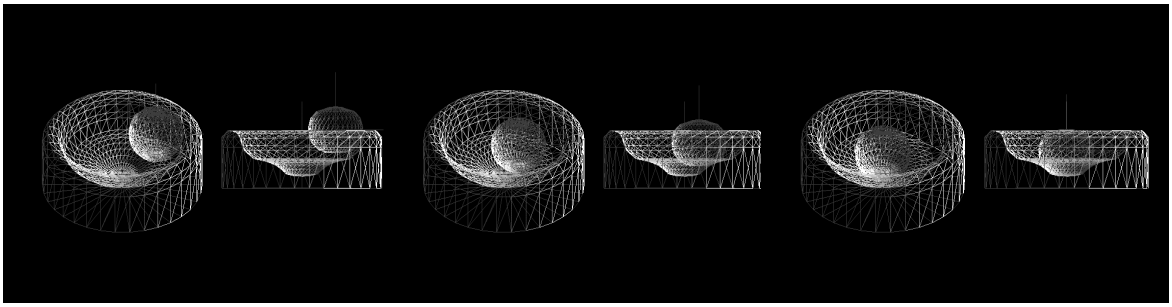

Figure 11: The example of the modeling of a ball falling in a dish. Here three frames can be seen during the modeling with a 3D view and a side view of the objects on each frame. It can be seen that the ball converges correctly toward the center of the dish.

We varied the initial position of the sphere so that the vertical axis of the sphere was anywhere above the dish. We varied the relative size of the sphere with respect to the dish between two extreme values. The largest was such that the two objects had the same size and the smallest was such that the sphere was smaller than a triangle of the dish. We varied the relative diameters of the sphere along the three axes to analyze the effect of non-proportional scaling. We last varied the modeling step size up to value reaching the order of magnitude of the size of the objects.

We also used several types of dish object sections including a section where the sphere had to go along a non-monotone slope to reach its final position. We tested the peg-in-a-hole problem by leaving a hole at the bottom of the dish and verifying that the moving object was going through this hole when it reached the bottom of the dish.

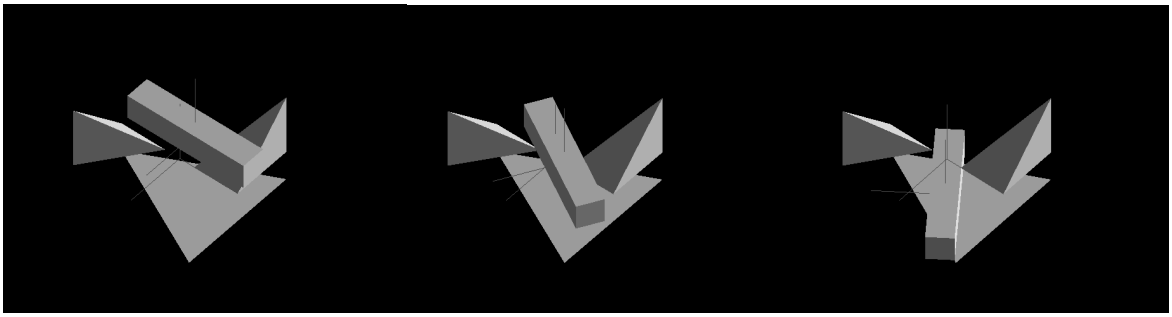

Figure 12: The example of the modeling of a bar falling between two pyramids forming a sink. The bar must rotate as result of torque to fall on the bottom surface. This motion is correctly observed.

For all the variation of position, size, scaling, and proportion, the sphere was converging at the center of the dish as long as its vertical axis was initially placed above the dish-like object. We noted that by keeping the translation resolution at least ten times smaller that the translation-modeling step as a secure ratio ensured avoiding the algorithm to converge in locations where it should not due to computation errors. The final position was only slightly changed according to the translation resolution specified by the user.

4.2.2 Test of the rotation feature

The rotation feature was designed to compute the relative orientation of the bones of the knee joint under the kinematic constraints imposed by the contact surfaces that produce the screw-home angle in the case of the knee joint. To test this component, we attempted to produce a torque on an object falling into another. Toward this goal, two types of pair of objects were used: the *torque* and the *top* configurations.

The *torque* configuration is composed of a bar falling in a sink formed of two pyramids so that when the bar goes down, the pyramids produce a torque making the bar rotate. Such a configuration is depicted in Figure 12.

We tested the correct convergence of the algorithm on this pair of object when we varied the step size, the proportions, or the initial position and orientation of the objects. For each test we performed, we verified that the center of rotation was correctly computed by modifying the initial position of the bar along its length. One must note that the torque produced when the center of the bar is not centered at the center of both pyramids is not realistic because the object should in reality fall on one side. We do not desire this behavior here because we need to constrain the rotation around an axis parallel to Δ for the modeling of the knee joint.

The top configuration was composed of a four faces pyramidal sink receiving a pyramidal cap of the same shape, as seen in Figure 13. The pyramidal cap had to turn and translate to the bottom of the sink to be stable. We chose to report this example because it made us discover the problem of not choosing the exact contact point as well as the smallest curvature as described in section 3.7.

When both objects had a relative orientation around Δ, we found that the moving object was not rotated as it was directed toward the reference object.

This problem was solved by dividing the triangular surfaces of the sink in two, and taking the normals from the sink object. Considering these two solutions together was sufficient for the algorithm to judge if the cap should turn clockwise

or counter-clockwise. However, as we saw in section 3.7, a more general approach will be needed to include any case. For this example of objects and after the described modifications have been done, we used several modeling step sizes, initial positions and orientations, as well as different relative sizes of the objects.

For both type of objects considered to test the rotation feature, we noted that it was secure to keep the modeling rotation step for the orientation at least ten times larger than the rotation resolution. In effect, such a ratio was in general wise to avoid that a stable orientation to be detected when it should not. Under this condition, the moving solid always correctly landed in a natural stable position and orientation.

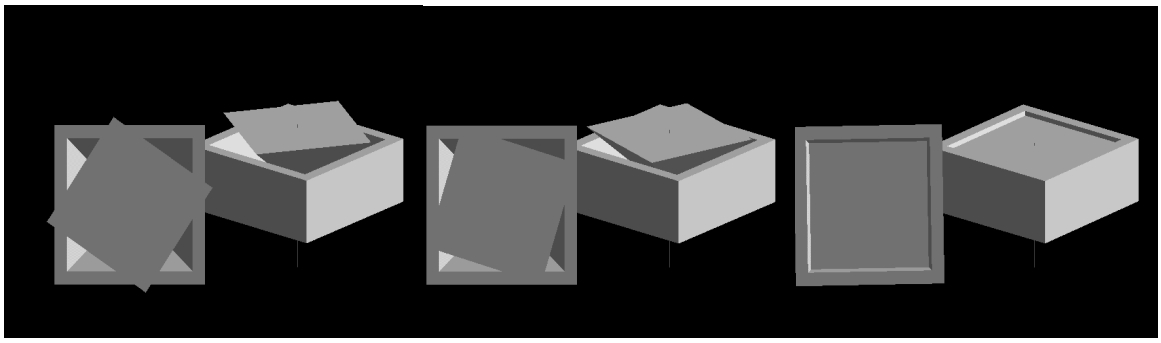

Figure 13: The example of the modeling of a pyramidal cap going in a pyramidal sink with a relative orientation of the objects around Δ. In this case, the correct location of the contact point was approximated by dividing the triangles forming the sink surface in two. The cap can be seen to converge correctly on the pyramidal sink by performing translations and rotations.

5 APPLICATION OF THE ALGORITHM TO THE KNEE JOINT

In this section, we first describe the assumptions we made on the motion model of anatomical joints. Next, we present the geometrical knee model we employed. We then explain the modification of the modeling application we conducted to perform the modeling of the knee joint motion. Finally, we discuss our modeling results on the geometrical model considered.

5.1 Modeling assumptions

As stated earlier, there are no loads applied to the components of the knee joint during the use of the VRDA tool. The components are essentially constrained to stay together.

We consider that the ligaments are producing forces of infinite amplitude rather than force of finite amplitude such as produced by spring-like forces. We can thus consider that the ligaments are producing kinematic constraints on the models, which is in accordance with our algorithm.

Further, we assume that the resultant of the forces produced by the ligaments has a constant direction that we call Δ. While we consider this direction constant during the whole motion modeling process this direction can be modified without lost of generality of our algorithm once this direction is known in more detail.

We currently consider the knee model to be composed of two rigid bodies: the *femur* and the *tibmen*. The *tibmen* is composed of the tibia and the menisci. We consider that the menisci are rigid on the whole range of motion. The fibula is not included in the modeling because it is not in contact with the femur on the range of motion of the knee.

This assumption allows us to produce a first motion model before quantitative data on the exact deformation of the menisci are available. Once these data are available, the menisci will be deformed accordingly at each motion step. During a same motion step, they will then be considered rigid. Because the menisci are considering rigid, they can been seen as kinematic constraints, a condition required for the algorithm to work.

Because the menisci shape has not been modified, a gap can be created between a condyle and the corresponding tibial plateau when the varus/valgus is spanned. Because a gap will not be created in the real case given that the menisci would fill it, we chose to only model the flexion/extension of the joint such that both condyles always touch the tibia. However, we still have to perform the modeling for any varus/valgus angle to find the instant axis of flexion for which both condyles touch.

We divide the motion modeling into motion steps for the overall range of motions. The step size conditions the accuracy desired between motions step during the simulation. At each motion step, the angles putting the joint in a given attitude are kept constant. For the knee, the degrees of freedom are the varus/valgus and

flexion angles. These angles constrain the relative orientation attitude of the femur and the tibia for a given motion step.

In the current implementation of the algorithm, we consider that the normals are taken on the reference objects. This means that the reference object must be the bone of the joint, which has the least curvature at each contact point.

In the case of the knee joint, the femur and the tibia have shapes that have similar radii of curvature at the contact point so no problem can be encountered. However, the sides of the menisci are sharp and have a curvature a lot larger than the curvature of the condyles at the contact points. Therefore, we chose the femur to be the reference object and the tibmen to be the moving object.

5.2 Generic 3D model of the knee bones

Figure14: The knee joint distributed by Viewpoint Inc. and rendered using Open Inventor.

The geometric model we used is a three-dimensional generic model of the bones of the human knee joint distributed by Viewpoint Inc. The model was digitized from a human male knee joint. The model comes in different formats including Inventor 2.0 and Object. The complete model rendered using Inventor is shown in Figure 14. For

the modeling part, we chose to use the Object format, which is easier to parse and to manipulate.

The model is a high-resolution model of the inner components of a human knee joint in extension. The components of the model are the muscles, the bones, the menisci, the tendons, and the ligaments. The model has a length of about twenty centimeters along its long axis.

The model is described as polygons. The frame of reference attached to all the components in this attitude is the same and is located roughly at the center of the joint. The Y-axis is directed along the length of the joint. The X-axis is going from the medial to the external part of the joint, i.e. from the left to the right of the left joint of a subject seen from the front. The Z-axis is going horizontally from the front to the rear of the joint. Using OpenGL and in the absence of rotations, the joint is seen vertically and from the front.

The Object file format is a text format describing all the components of a model in three files: the coordinate file, the normal file, and the element file. The coordinate file contains the coordinates of the vertices of all the components of the model. The normal file contains the surface normals for each of the vertices. The element file contains the polygon description by specifying for each face the element to which the face belongs, followed by the enumeration of the indices of the vertices that compose that face.

We transformed this format into the ASCII format because the ASCII format uses exactly one file for each component. Because each file describes one

component, it is then possible to load only the components that interest us, such as the bones for example.

Toward this goal, we developed a C converter *ObjToAsc* placed on the floppy provided. This converter extracts the components of the model from the Object file, counts their polygons, and creates an ASCII file for each component. At the same time, a normal file is also produced for rendering purpose if such data are found in the Object format.

Below is the listing of the output produced by the converter for the components we need for the simulation out of the 18 composing the entire model. The *Medmen* and *Latmen* components are the Medial and Lateral menisci, respectively. The listing also shows the number of triangles generated using the triangulation technique described earlier.

Components	**Polygons**	**Triangles**
Tibia	1484	1746
Patella	308	378
Femur	1300	1516
Medmen	570	665
Latmen	640	742

5.3 Modeling application modifications

The modeling application has been modified so that the graphical process constructs display lists out of the triangles of the loaded models. Display lists are

used in OpenGL to accelerate the execution of sequence of instructions whose parameters are constant. By recording the sequence of instructions in a structure in memory, OpenGL can execute them faster. The triangles of a rigid body have fixed coordinates in the frame of reference of the rigid body. Therefore, the rendering of thousands of triangles to render a model is the type of task that is a good candidate to be recorded as a display list.

The algorithm process has been modified to execute the search for the stable position and orientation of the moving object at each motion step. The recording of the stable relative position and orientation of the bones was conducted in a lookup table that will be employed for the simulation phase. The lookup table contains the three orientations and the three translations to apply to the moving object to recover the modeled configuration of the joint.

To avoid computing at each motion step a new valid initial position for convergence of the stability algorithm, the last stable position is taken as the next initial position. Moreover, at each motion step, the reference object is oriented along the degrees of freedom. The advantage of orienting the reference instead of the moving object is that the normals and their application points are rotated at each motion step instead of at each modeling step.

Finally, because we only want to model the flexion of the joint the application was able to determine and record in a file at each motion step whether or not both condyles were touching.

5.4 Results of the modeling of the knee joint and improvements

An illustration of the modeling of the knee joint model using the application is depicted in Figure 15. The *femur* is the reference object and is oriented according the flexion and varus/valgus angles. The tibmen (i.e. tibia and menisci) is the moving object and is manipulated by the stability algorithm.

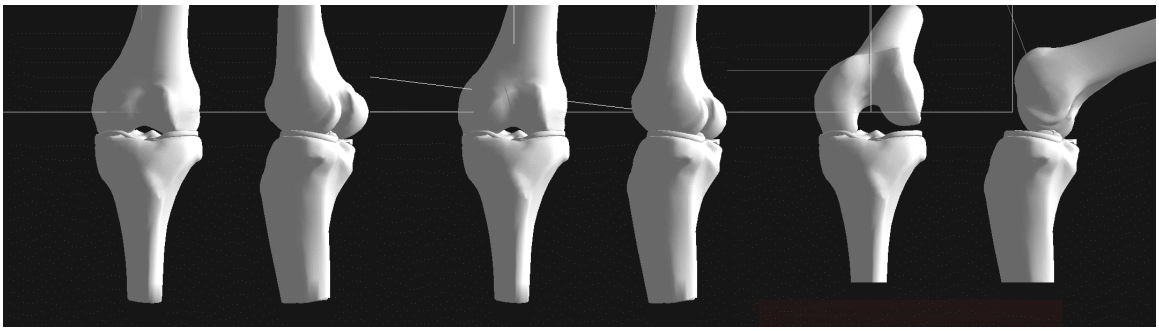

Figure 15: Three frames taken during the modeling of the knee joint motion. It can be noted that the femur is rotated to produce the varus/valgus, while the tibia is pushed against it to find a stable position

Because of the convenient orientation of the axes of our knee model, we span the varus/valgus angle by rotating around the Z-axis and span the flexion angle by rotating around the X-axis. Using these axes, we guarantee that the angle of zero varus/valgus (i.e. defined as both condyles touching) will be found for any flexion angle.

We used some orientations in the range zero to -90° for the flexion angle and some orientations in the range +10° to -10° for the varus/valgus. We then divided each range in step of 1°. We generally used a modeling step ten times larger than the associated resolution.

The resolution of the human eye is one arc minute in the foveal vision, and the viewing distance of the model in our application is typically 0.5 meter. The maximum resolution in translation resolvable by the human eye at 0.5 meter is thus about 0.145 mm. We then set 0.1 mm as the translation resolution and 1 mm as the linear modeling step.

To obtain the corresponding resolution for the rotation, we take into account that the tibia is enclosed in a circle of about 3 cm in radius as the screw-home angle is changed. Therefore, an arc of length 0.145-mm must be produced by an angle of $1.45E^{-4}$ / $3E^{-2}$ rad or 0.27°. We then set the orientation resolution to 0.1°, and the modeling rotation to 1°.

When the modeling was performed, the curve of the translation produced on the Z-axis as a function of the flexion angle showed some jittering of the joint in some places that were also observed during simulation. We understand that these discontinuities are due to the use of a polygonal model discontinuous by definition. We think that the polygons roughness prevents the motion of the tibia during a range of flexion angles, and at a certain angle the motion is possible because one stopping edge is favorably oriented.

Each position or orientation parameter can be extracted from the lookup table and represented as a discrete surface in a space defined by the value of the parameter (e.g. Z-axis) as a function of the flexion and varus/valgus angles. Such a surface is shown in Figure 16 on the left frame.

We smoothed the discontinuities of the motion surface using polynomial surface regression to achieve a least squares fit of the surface for each parameter. By using a polynomial of degree six to smooth the two translations and the rotation obtained during the modeling, we achieved a smooth motion model without perceptible collision. As an example of typical results after smoothing, the smoothing of a surface curve is shown in Figure 16 in the right frame.

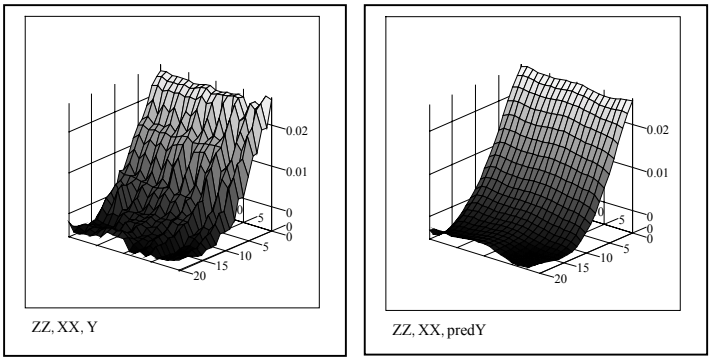

Figure 16: Left frame shows a motion curve described in function of the flexion and varus/valgus angle. The right frame shows the motion curve once it has been smoothed

Because we chose to simulate the knee with a zero varus/valgus angle, a new lookup table, which can be accessed by the flexion angle, was built. We designed a converter to this effect. The converter uses the file recorded during the modeling, which specifies for each motion step whether each condyle touches the corresponding tibial plateau. The converter then scans all the varus/valgus angles for a given flexion angle in the lookup table. Finally, it takes as the zero varus/valgus angle where both condyles touch, the average between the first and the last varus/valgus angles.

Using the described implementation and considerations we produced a smooth motion model of the knee joint without collision or gap created between the geometrical models. The joint motion is smooth on the range of motion, and both condyles are touching. Moreover, screw-home appears to be created.

6 SIMULATION

In this chapter, we first describe how we improve the geometrical knee model so that it looks realistic when it is displayed to the user' eyes. Then, the core simulation application is detailed and simulation results are given.

6.1 Processing of the model using Open Inventor

To create the simulation, we chose to use some APIs that allow using a model with attributes. In effect, OpenGL does not have any dedicated model, so a loader must be designed for each type of file and the attributes of the models must be hard coded. For a review of the basics on the types of models and rendering attributes, the reader should refers to Appendix C.

The knee model we want to obtain will have to be textured, smoothly shaded, and lighten by a light similar to the lighting condition during the use of the VRDA tool. To modify our knee model in Inventor 2.0 format, we are using Open Inventor. Open Inventor is an API built on top of OpenGL coming with some tools useful to manipulate files in Inventor format.

The Inventor format describes a scene graph organized as a hierarchy of geometry components and attributes arranged in a tree as shown in Figure 17. Any attribute modify the geometry of some components of the models that are on the same level, at its right, as well as on the child levels placed under it. For example, a scaling attribute can act on the scaling of the tibia if the tibia

component is placed on the right or on the child level in the case of the knee. The figure shows *gview* a GUI that is used to view and modify the scene graph of a model as well as the model itself.

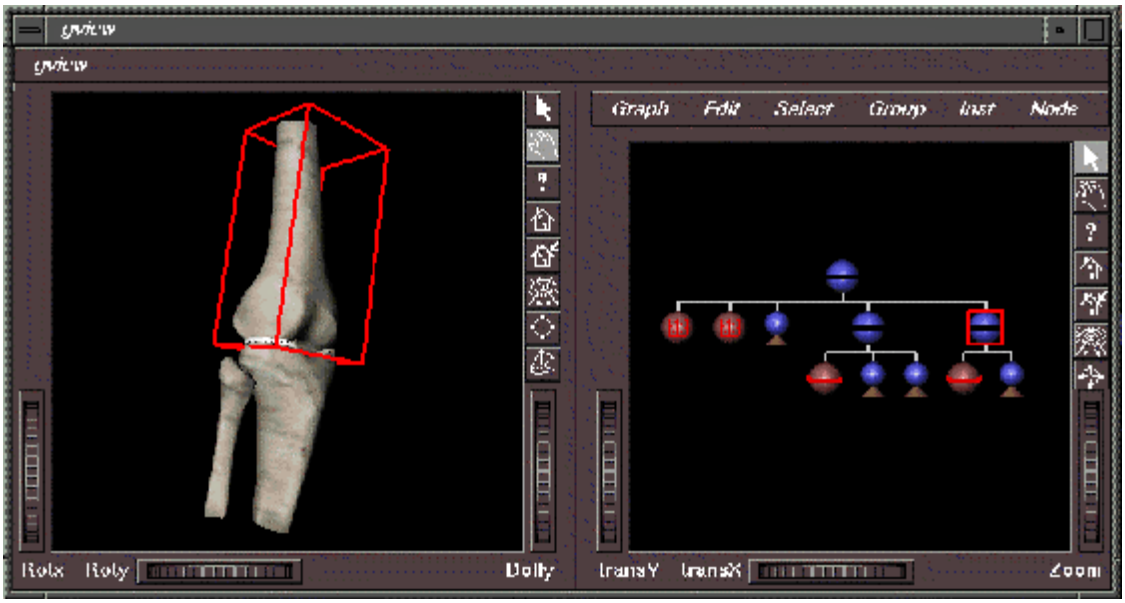

Figure 17: The Open Inventor tool *gview.* On the left of the application, the user can see the model corresponding to the scene graph on the right. The model can be manipulated using the mouse

The use of the Inventor format is convenient because it provides us with tools allowing the visual manipulation of a more intuitive data structure than the hard coding of equivalent functions. Moreover, because it is a famous format it can be converted into or from any famous format as DXF (Alias/Wavefront), PFB (Perfomer Binary), Object (3DStudio), FLIGHT (MultiGen), or others.

We were interested in the bones of the model and thus we used *gview* to separate the files in several components. We discarded the muscles, the patella and the ligaments because they were not considered at this time in our synthetic

knee joint model. We kept the tibia, the fibula, the menisci, and the femur and created two components that we previously modeled, the femur and the *tibmen*.

The *tibmen* was composed of the tibia, the menisci, and the fibula for the simulation. We removed redundant attributes from the *tibmen* scene graph due to the concatenation. We then added texturing, scaling, transparencies shading, and lighting condition attributes.

A marble texture fine-tuned for similarity with a bone texture was used for the bones. The texture was saved as a RGB and loaded with the model. The scaling was setup so that the unit of measure of the model was the meter, which is the unit of OpenGL. Smooth Phong shading was specified. The crease angle of the model was set to $\prod$ and the model was specified to be convex for guaranteed smoothing of the surface of the joint. A transparency of 0.4 was applied to the menisci to be able to see through the menisci and see if the condyles were touching.

The model was then optimized using *ivfix* that transformed the Inventor ASCII format into an Inventor binary format that is optimized for the rendering hardware. Then, the models were converted into Performer Binary format using *pfconv* since we use Performer to render our model as described in the next section.

6.2 Performer application

In this section, we describe how we used Performer as well as the previously built geometrical model of the knee joint to create a virtual model of the bones of a generic knee joint.

6.2.1 Simulation Implementation

In order to get the best performance from our rendering hardware, we used a C library built on top of OpenGL and developed by Silicon Graphics (SGI). This library is a C++ API that has C wrappers to C++, and which allows the coding of real-time simulations using the rendering hardware at its limits. The API provides a scene graph similar to Inventor.

The library contains numerous loaders that allow building a scene graph out of files in any kind of format. Performer includes culling and level of detail capabilities that traverse the scene graph and attempt to remove the most polygons it can to lower the rendering load, based on what the user sees. Finally, performer is running several processes performing the application, the culling, and the draw functions.

These processes are distributed among all the processors and the user can tune the workload among the processors. Thus, Performer is attractive to use even if the language needs an advanced understanding of the graphical processing as well as OpenGL.

The application process is the part of the code that the user can modify freely to program the behavior of the application. The Performer developer option includes many sample codes that allow starting without having to understand the complex intrinsic of the Performer philosophy.

Because the X system calls are cumbersome to implement to control the interaction with the mouse for example, we used a sample program demonstrating the use of the mouse as a trackball in a Performer application: *trackball.c*. For further review of the structure of the code, the reader can look at *vrda.c* whose listing is given on the floppy in the directory *simulation* on the floppy disk.

The control of the object using a trackball allows translating and rotating the object as desired in an easy way. The SGI mouse has three buttons that make this control even easier. The way that these controls are programmed in the trackball sample code allows translating the object on the X and Z-axes of the word referential by holding the left button and moving the mouse. The middle button is used to rotate the object around its origin. The right button allows moving farther or closer by translating the object along the Y-axis.

We modified the initialization of the code to load the models from the command line and to create a scene graph at running time using any type of 3D model file format. At run time, the simulation application load the lookup table generated by the modeling application. The lookup table is loaded in an array of six

parameters: three rotations around the X-, Y-, and Z-axes and three translations along the same axes.

The X- and Z-axes of rotation were intended to simulate respectively the flexion and varus/valgus angles of the knee and were used to rotate the *femur* in order to keep the *tibmen* vertical during the modeling process. In effect, rotating the femur to produce the flexion and varus/valgus angles brings many simplifications in the computation as explained in the result section of the modeling.

The frame of reference in Performer is oriented differently than in OpenGL. In OpenGL, the frame is such that the X-axis is going right, the Y-axis is going up and the Z-axis is going toward the viewer. The Performer axis is such that the X-axis is going right, the Y-axis is going away from the viewer, and the Z-axis is going up. OpenGL and Performer use of different types of referential. Therefore, we also have to perform the flexion and varus/valgus respectively around the X- and the opposite of the Y-axis in Performer to obtain the same motion than in OpenGL.

Still in the initialization phase of the code, a scene graph is created to animate the knee as needed. Performer has two types of coordinate systems on which objects can be attached: the Dynamic Coordinate Systems (DCS) that can be manipulated during the application and the Static Coordinate Systems (SCS) that are setup once for all in the scene.

When using the Performer loader, one must attach the object to one or the other, depending whether the object must further move or not. The femur is then

attached to a DCS, and the tibia is attached to two consecutive DCSs. The reason why we attached two DCS to the tibia is due to the modeling strategy we adopted.

In effect, we have to apply two rotations to the tibia first to align it to the Δ axis at the considered attitude of the knee. Then, we need to translate the frame of reference so that the Y-axis is along Δ, and then rotate around Δ to simulate the screw-home angle. Because we have the sequence rotation-translation-rotation again, we must divide the motion using two coordinate systems: the first coordinate system moving with respect to the scene coordinate system is to simulate the flexion and the varus/valgus angles. The second coordinate system is to set the screw-home angle and the position of the tibia computed by the modeling algorithm in this new oriented frame of reference.

6.2.2 Simulation results

The use of the Performer binary format allows loading the models in a short time into the application at running time using the dedicated loader for the Performer binary format. The model appears accordingly to the attributes that have been specified using Inventor.

Because we only want to playback the generated modeling motion of the joint, we only move at computed discrete angles, that is using increments of one degree. We correct the joint motion with a polynomial of degree six as explained before in the algorithm implementation section. Thus, an interpolation method

could be based on a polynomial of degree six as well in order to obtain parameters at any angles. Because, the tracking of the flexion is currently not implemented, we did not use interpolation at this time.

Using the animation at one-degree intervals, the motion appears to be smooth and the condyles were touching during the whole range of flexion of the joint considered. Using the application described, the model could be rendered and examined in real time on a monitor from any viewpoint using the trackball manipulation. Figure 18 shows several frames taken during the animation of the joint from several viewpoints. A videotape is also available where both the results of the animation and the inner working of the algorithm are demonstrated.

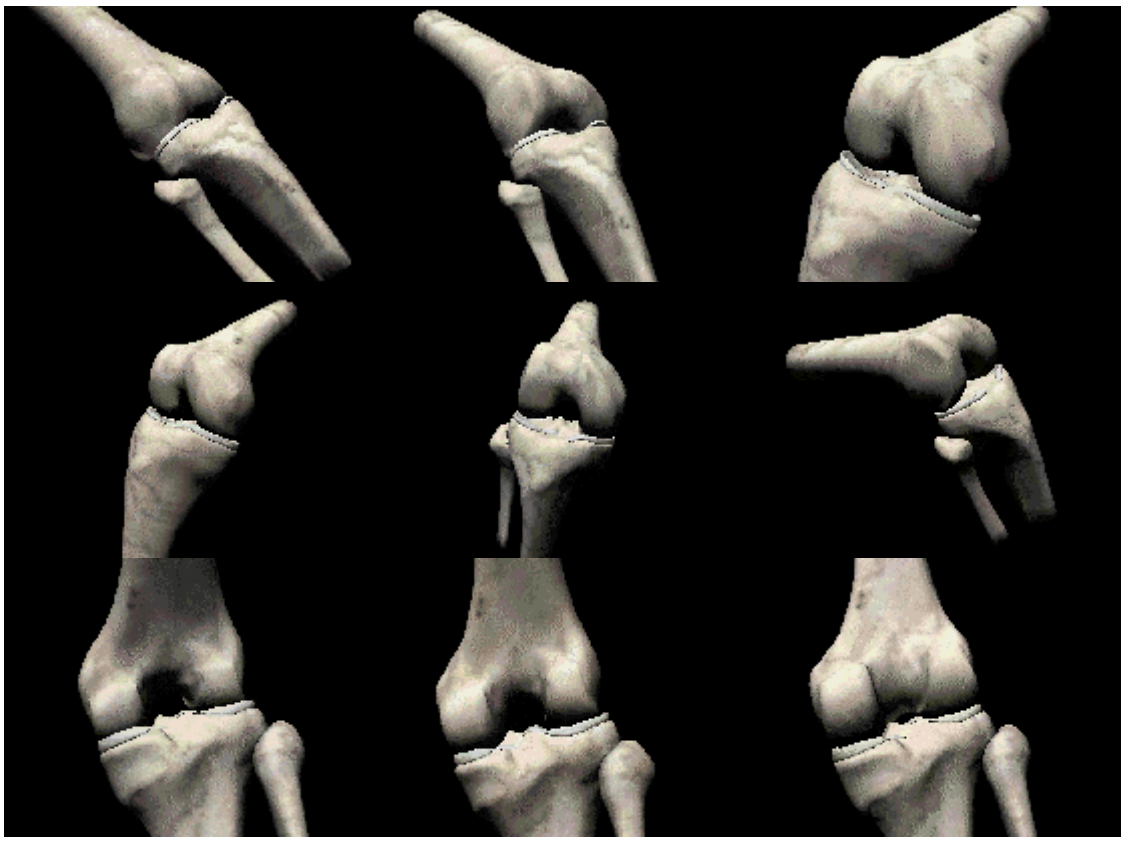

Figure 18: Some frames taken from the resulting rendered version of the knee joint.

7 AUGMENTED REALITY VISUALIZATION

In this chapter, we shall describe the setup used to perform the visualization of the synthetic model (the bones) on the real world (a knee joint) using augmented reality technology. The setup can be divided in three parts: the visualization device, the control devices, and the tracking devices. The software application used for the visualization was improved from the one developed in chapter 6. Therefore, the functionalities of the old application such as the trackball control of the virtual scene were still usable.

7.1 The visualization device

The visualization technology used in our project is called augmented reality because it augments the user view of the real world by superimposing virtual information on real objects. Furthermore in our case, we aim to superimpose virtual anatomy on its real counterpart. This process is known as registration.

The visualization device used to create augmented reality is called a see-through stereoscopic display. The term see-trough refers to the fact that the user can see the real world through the display device so that the virtual world and real world superimposition can be done. The term stereoscopic refers to the fact that the user can see in 3D by presenting to an image to each eye (i.e. stereoscopic pair).

One of the displays available in our lab is a Head-Mounted Display (HMD) called I-glasses from Virtual-IO. This display constitutes a portable HMD for simple use, but does provide Inter-Pupilliary Distance (IPD) like most of the commercially

available HMDs. However, the IPD-tuning is necessary to display virtual objects at a correct depth. Moreover, the quality of the colors and the resolution (320x240 pixels) are poor.

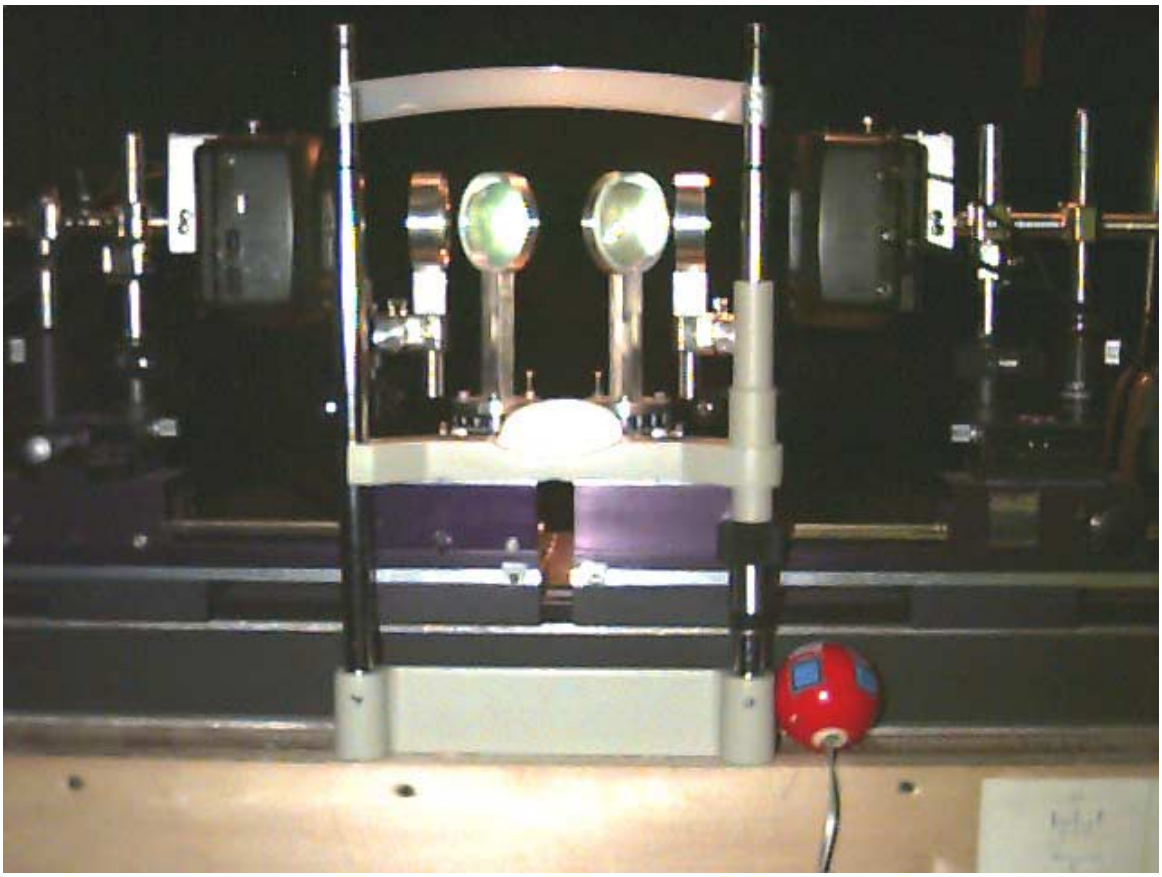

Figure 19: The bench-prototype see-trough stereoscopic display used in our laboratory. The TVs, lenses, and beamsplitters can be seen on the pictures. The user installs his/her head on the chin and front head rest areas (in white). The display can be physically translated on the left and on the right to provide the user with parallax cues.

An important research effort in our lab over the last 4 years has been to conduct experiments on human depth perception in virtual environments. A bench-prototype see-through stereoscopic display has been built for this purpose and we chose to use it for the first implementation of the VRDA tool. The bench is not

portable as a HMD and thus does not allow a large working area but has the advantage to be calibrated and tunable in IPD in opposition to most commercial systems.

The display is installed on an optical table 3 meters long and 1.5 meter wide. Figure 19 shows a close-up of the display itself. The display is mounted on a horizontal rail perpendicularly to the line of sight of the user, so that the user can translate the display left and right in front of the scene. This translation provides a parallax cue of the scene in front of the user so we called this ramp the head-motion parallax stage, also referred as the parallax stage.

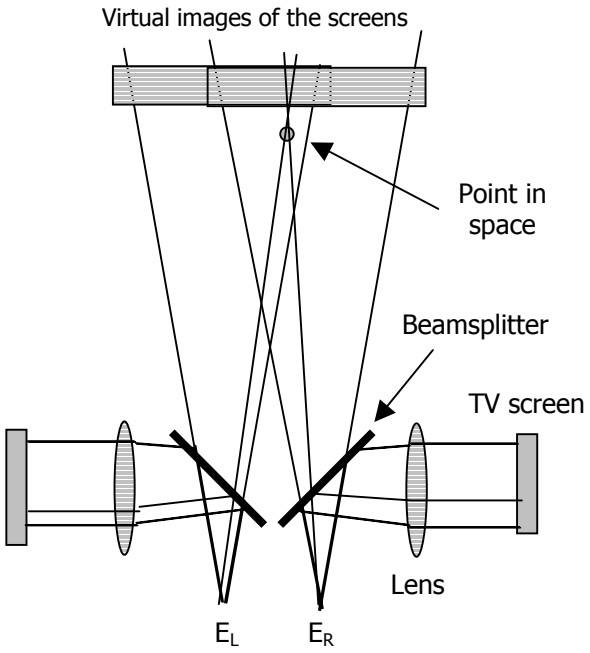

Figure 20: Schema of a stereoscopic see-through head-mounted display. E_L and E_R are the locations of the user's eyes.

The display itself is composed of two arms mounted parallel to the parallax ramp. On each arm, a see-trough optical system is composed of a beam splitter, an imaging lens, and a TV screen as depicted in Figure 20. On each arm, the image displayed on the TV screen is virtually imaged through the lens and then reflected towards the eye by the beamsplitter. Because the beamsplitter is semi-transparent, the real world in front of the optics is seen by the user as well. The distance between the two arms can be adjusted so that the

centers of the images projected to the eyes of the user can be aligned with the centers of the pupils. This capability provides effective interpupilliary distance (IPD) adjustment specific to each user.

The optical components have been precisely centered on the center of the screen and oriented by an optical calibration procedure using a conventional laser alignment technique. The imaging lens has been custom-designed using off-the-shelf single lenses and optical design software. The focal length of the lens has been measured and entered in the software. By placing the display at the focal point of the lens, a virtual image of the screen is projected at the infinity. The nominal distance used in our depth perception experiment is 0.8 meter. We used the optical model to compute the distance of the display to the focal point of the lens so that a virtual image is created at this 0.8 meter. Matching the distance of the real object and of the optical image allows eliminating eyestrains by making the user focus at a unique distance.

When looking at the real world, a user can see in 3D because his/her eyes see two different images and the location of each point in space can be recovered by triangulation. Inversely, the two images a user should see of an object in space can be shown to a user's eyes with the described setup so that a 3D view of the object in space can be simulated. To generate the two images, a model of the object to display is created in the virtual world and the rendering of the object is done from two different viewpoints apart of distance equal to the IPD.

Our graphical workstation directs the graphical output to only one destination. The Multi-Channel Option (MCO) is a device connected to the workstation that can create four RGB outputs by splitting in four parts the framebuffer area typically displayed on the monitor. To create two images for the stereoscopic display, we draw in the upper right quarter to create the image for the left eye, and we draw in the lower right quarter to create the image for the right eye. The original console area is 1280x1024 pixels large so each quarter area is 640x480 pixels large, which correspond to the NTSC format. Each RGB output, giving three signals (Red, Green, and Blue) in one output, is fed to a video scan converter transforming the three signals in one NTSC signal. This signal is then sent to one of the screens of the display using a coaxial cable.

7.2 The control devices

In order to set the visualization parameters, some control devices were setup to provide an interface between the real and virtual worlds. The control devices we considered were a potentiometer and the keyboard.

The potentiometer is connected to a micro-controller board, the Little Giant from Zworld, which handles the control of several devices in the depth experiments. The use of a micro-controller allows to discharge the graphical workstation of the load produced by the interaction controls and to adapt some devices for which the workstation does not have any inputs. The Little Giant is a micro-controller board based on a Z80 microprocessor, which has ROM, RAM, counters, A/D

converters, and other I/O interfaces. The board is also equipped of several serial and parallel ports.

One of the serial ports can be connected to a PC. The PC has an application allowing developing code in Dynamic C, a dedicated C language for the micro-controller allowing the programming of the behavior of the board. At run time, the code is compiled, sent to the micro-controller RAM through the serial link, and then executed by the microprocessor. When the code developed is final, an EPROM can be engraved and the micro-controller can be used to run constantly the same code. The micro-controller then reacts as a dedicated hardware device without the need for a PC.

The second serial port is used to communicate with the workstation. A library of functions has been developed to read and write to the serial port to communicate with the micro-controller via serial communication using special characters. Each character sent corresponds to a request to read the output value of one of the interaction devices. In return, the micro-controller sends the requested reading that can be used by the workstation.

The potentiometer is connected to one of the A/D converter of micro-controller with a precision of ten bits resolution which can be read by a dedicated Dynamic C instruction and sent to the workstation. This setup implements a general-purpose potentiometer that can be used to modify at run time some specific parameters. The parameters to tune are selected using dedicated keys of the

keyboard. The code developed and reported in the last chapter gave us a framework to implement such control of the keyboard inputs.

7.3 The tracking devices

In our virtual environment application, tracking must be performed to measure the position and orientation of both the user head and real counterparts of the graphical objects. Measuring the user's head position and orientation allows modifying the location and orientation of the viewpoint in the virtual world to show the user what he/she should see according to the attitude of his/her head. This action is called head tracking. The tracking of an object in an augmented reality environment, where the user see both the real and virtual world, enables displaying its virtual counterpart with the same attitude in the virtual environment for real/virtual registration. We currently use two different devices to perform the tracking: the parallax stage for head tracking and the Optotrak for object tracking.

The first tracking device, the parallax stage, is the head tracker. The only degree of freedom allowed by the parallax stage is a translation of the user's head to the right or to the left. The stage's cart is attached to the slider of a linear optical encoder of resolution 5 microns. The linear encoder reads a rail engraved every 5 microns with an optical detector as it moved. This reading produce two square electrical signals with a phase difference of 180 degrees. The logic signals are sent to the micro-controller Little Giant described in the previous section.

The micro-controller has been improved to accept the linear encoder signals with a 16 bits counter incrementing or decreasing according to the direction of motion

of the encoder. The micro-controller has been programmed to make the periodic reading of the pins of the IC counter output reflecting the current relative position of the encoder since the last reading. Then the microcontroller reset the counter and update the absolute position of the ramp. This position is itself relative to the initial position of the ramp at startup. This position being relative to the initial position of the stage, the calibration of the parallax stage must be done at run time using a physical stop. Upon request, the position read by the microcontroller is sent through the serial link to the workstation.

To perform the tracking of real objects we employed the Optotrak 3020 from Northern Digital, an optical tracker measuring the spatial position of infrared LEDs, shown in Figure 21. Typically, the principle of operation of an optical tracker is based on the use of multiple camera views of some markers (LEDs). The 3D to 2D mappings of each point in space to each camera's CCD are known. The relative transformations from one camera to another are also known. Given these parameters for each point, one can draw some lines to backproject the points detected on the cameras' CCD into lines crossing in space at the 3D position of the point observed.

Figure 21: The Optotrak 3020 from Northern Digital (left frame) mounted on a stand with its three cameras. A rigid body or tracking probe (right frame) with 6 markers and the control device to fire the LEDs.

The Optotrak measures the static position of a marker (LED) with a typical resolution of 0.01 mm and accuracy of 0.1 mm (Northern Digital, 92). This measure can be made effectively within a range of 2 to 6 meters in depth and in a field of view of 34 degrees horizontally and vertically for each of the three cameras. The processing unit sends a report of the position of each LED through a SCSI link. The measuring frequency is maximal for one marker and its maximal value is 3600 Hz. This value progressively decreases as the number of markers to track increases.

The Optotrak comes with a set of tools for PC to construct a rigid body out of several markers. A rigid body is a structure composed of several markers placed on a rigid object, so that the relative position of the markers is kept constant with respect to a referential attached to the object. The Optotrak system knows the

rigid body's structure and thus is able to determine the position and orientation of the rigid body attached to the object to track.

The computation is done by a dedicated processor built in the tracking system that search the transformations to apply to the rigid body to align it with the measured position of the markers. To compute the attitude of the rigid body, three markers on the object to track must be seen by the three cameras of the tracker so that all six degrees of freedom of the rigid body are set. These transformations are transmitted through the SCSI link to the workstation using them to render the graphics without any further processing. An API on the workstation side reads data at a frame rate approaching 100 Hz for one rigid body, which decreases as the number of rigid body to track increases. The workstation can then render the virtual counterpart of the tracked object in the same attitude in the virtual world so that registration is achieved.

Two problems inherent to optical tracking are the deterioration of the spatial resolution as the object tracked is going further away from the camera, and the possible occlusion of some of the markers. Occlusion is problematic when some of the markers exit the tracking volume or are hidden by the object itself, so that less than three markers are seen by the cameras. These problems can be solved by first keeping the object to track in the tracking volume. Further, the occlusion by the object itself can be prevented by spreading out cleverly the markers all around the object to track. If the distribution of the marker is designed correctly, three markers can be seen from the three cameras for any attitude of the tracked object.

8 A PROPOSED VISUAL CALIBRATION METHOD

A main problem encountered in augmented reality technology is the difficulty to perform the correct registration of real and virtual entities when they represent the same object. The registration errors have static and dynamic causes.

The static errors can be produced simply by the incorrect calibration of the components of the system. Because the components of an augmented reality system are issued from multiple technologies (i.e. electronic, optical, mechanical), the overall calibration of the system is tedious and challenging and often not completely performed. Because the anatomical parameters of the person using the system are part of the correctness of the visualization, as the IPD for example, static errors can be introduced by the user itself if a user-specific calibration is not included. The dynamic errors are produced by the processing time of the system: there is a delay between an action happening in the real world and the corresponding reaction in the virtual world. The technology used to build the system plays an important role in achieving a good calibration.

Tracking, graphical, and optical calibration have all to be performed to achieve the complete static calibration of the system, but most studies have taken in account some of these components (Janin, 93; Holloway, 95). To cope for the lag due to the processing of the information in the dynamic case, prediction techniques have been developed and implemented (Azuma, 95; Welch, 97; Bajura, 95). However, such techniques can only work if the static problem is

solved. Finally, commercial hardware components used to build the virtual environments are still often not adapted to implement successfully a calibration technique. For example, only-magnetic trackers do not provide consistent reading. Another example is that most HMDs do not include IPD adjustment.

The contribution of this research toward the calibration is to first generalize several methods solving the tracking of specific components into a unified method for the complete system that account for human variability. Only the static registration will be considered because dynamic registration is not needed at this time. Further research will complete this study to include the dynamic case. Finally, to solve the inadequacy of some commercial hardware, we used some custom designed hardware developed in previous studies.

We introduce an all-visual calibration method divided in two parts. One part is performed once for all for the system's components. The second part is achieved by interactive adjustment of some parameters of the system by each new user.

8.1 Viewport measurement

The displays used in stereoscopic devices have features of operation not always intuitive. The apparent viewing area is usually smaller than the framebuffer and its center is often offset with respect to the framebuffer center. Figure 22 shows the case of our configuration. Consequently, the image can be cut and not centered, and calibration errors will appear if the viewing area is not correctly specified in software.

In order to determine the part of the framebuffer that is visible, a scanning of the framebuffer must be performed at the pixel level. The whole window area is first considered, and a vertical line of one pixel is moved using the control devices

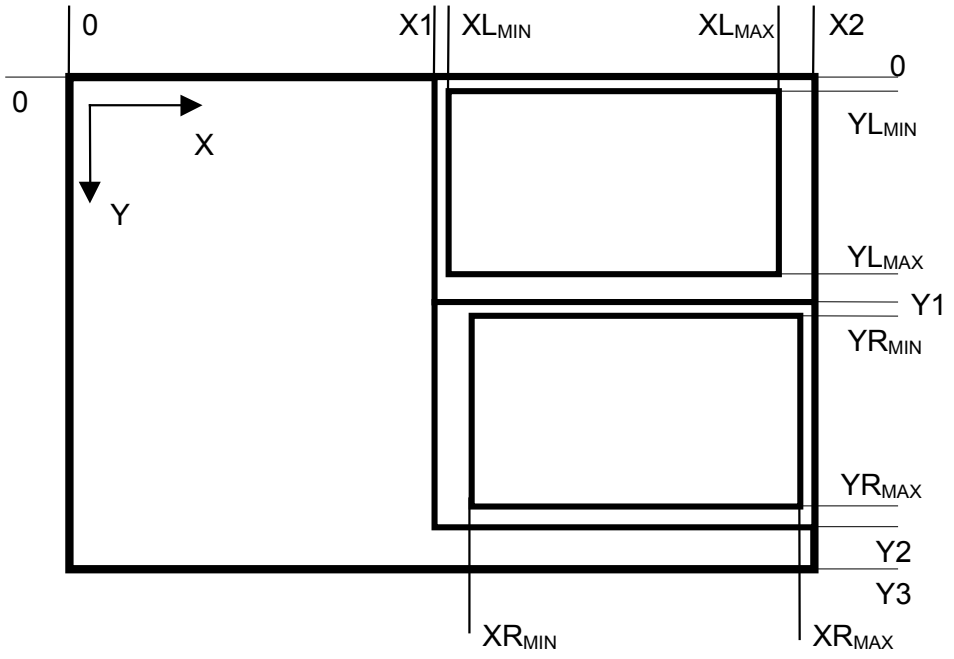

Figure 22: Illustration of typical viewport areas in the frame buffer. Here, in the case of our configuration, the viewports for the left and right are smaller than the rendering windows and their center are not at the same locations than the centers of the windows.

from left to right by interaction of the user. The user notes the coordinates in pixels of XL_{MIN} when the line appears for the first time and XL_{MAX} when it disappears. The same operation is done for the vertical scan. These two operations are performed for both screens. The rendering of the scanning line is done by changing the pixels values directly in the framebuffer. Once this operation is done the viewport area, defined as the area of the frame buffer actually seen, can be specified to the rendering hardware using appropriate instructions. Consequently, the rendering will only happen in these areas.

Once the extents of the visible areas of the screens have been determined, the centers of the viewports corresponding to the center of the screens can be determined. The optical axis of the optical elements of each arm of the stereoscopic display must be aligned with the center of the attached screen. This operation is done using a laser-based alignment method.

In our bench-prototype display, the calibration of the viewport was performed once for all when calibrating each arm. This operation has to be done each time the displays are changed.

8.2 Optical distortions compensation

The focusing lens used to project a virtual image of the screen at nominal viewing distance has the advantage of making the user converge and focus at roughly the same distance. Such a setting reduces eyestrain. However, the focusing lens introduces optical distortion. If the distortion is not corrected, the user observes a distortion of the virtual image around the center of the line of sight that induces depth perception errors.

The optical distortion (3^{rd} order is considered here) is producing a displacement of the position of each pixel relative to its distance from the center of the screen previously aligned on the optical axis of the lens. In our setup, the lens creates a pincushion distortion that produces a concentric displacement of the pixels from the optical center. A grid seen through this lens is shown in the left frame in Figure 23. By pre-distorting the grid with the corresponding invert distortion, which is a barrel distortion shown on the second frame, one can void the effect of

the optical distortion and obtain the correct grid shown on the third frame. Because our display has been modeled using optical modeling software, the distortion coefficients are known and the barrel distortion is completely defined.

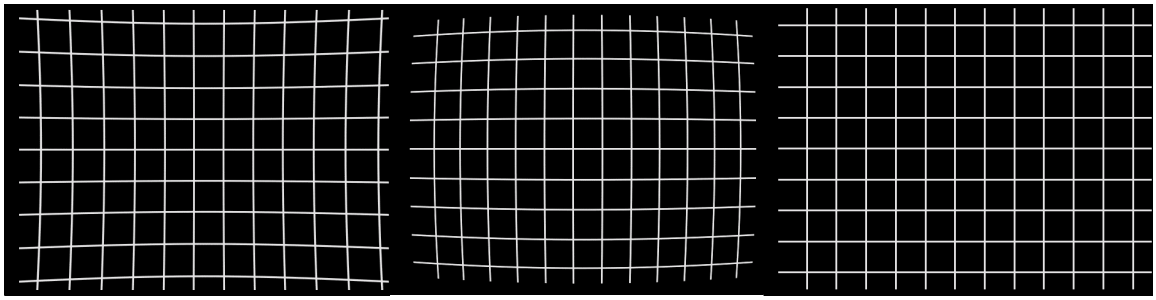

Figure 23: Illustration of the method of pre-distortion to compensate for the optical distortion. On the left frame, the pincushion distortion produced by the optics on a grid. The invert (barrel) distortion is applied on the rendering (grid) before to be displayed. On the right frame, the pre-distorted grid appears correctly through the optics.

This calibration can be done interactively for any optical system whose optical parameters are not modeled. A grid distorted according the distortion model of the optical system is displayed. The grid size and the distortion coefficients must be tunable interactively, so that the user can align the grid graphical grid vertices with the vertices of a reference grid placed perpendicularly to the line of sight. When both grids align, the coefficients that provide the desired distortion for this grid can be recorded.

During the use of the system, the rendering of a frame is terminated by a remapping of the pixels of the rendered image so that the precomputed distortion is performed. This remapping is done in software by rendering into an off-screen buffer the original scene, reading into texture memory, and tilling the texture on a mesh distorted according to the coefficients of distortion. However, on our rendering platform and for most of the current platforms, this processing takes a

non-negligible time and an interactive rate of 30 Hz pleasant to the human eye cannot be reached

Hardware implementation should be considered using the same tilling technique. In such hardware, the mapping from some tessels in a non-distorted grid to others in a predistorted grid could be specified. The configuration could be changed according the display and/or the changes done on it. The manufacturer of a HMD is often the only one to know the characteristics of the optics of the HMD. Therefore, the mapping coefficients of optical system should be hardwired into the display.

8.3 Field of view and eyepoints measurements

The field of view is the first parameter of the calibration procedures that is user-dependent. For our specific system, where the image is not collimated, the optical system is forming a virtual image of the screen at the nominal viewing distance, which is about 0.8 meter in our case. We can imagine to simplify that we have two very large screens placed somewhere at a specific distance for each eye and their display overlap by some amount. The optics can then be put apart from the reasoning under these imaging conditions.

Taking the center of rotation of the eye to render a scene has been shown to be correct when looking at the center of fixation but not in the peripheral vision (Vaissie, 98; Rolland 98). Because eye tracking is still a technological challenge, the center of rotation of the eye is usually taken as the center of projection. Thus,

the center of the perspective frustrum used to render the virtual scene must be attached at the center of rotation of the eyes.

The eyepoints E_L and E_R shown in Figure 20 are changing with respect to the anatomy of every individual. Let's assume that a rectangle of known dimensions is placed at a known distance from the eye-point and perpendicularly to Z, the direction of viewing. If the field-of-view is correctly set, a virtual rectangle of the same dimensions located at the same position is registered with the real rectangle when looking in the display. If this is not the case, the field of view can be adjusted interactively by the current user so that both the real and virtual rectangles appear visually registered. However, the location of the eye-point varies between individuals. Therefore, the calibration technique must be designed to determine both these values at the same time.

$$d = \frac{X_1 X_2 Z}{X_1 L_2 - X_2 L_1} \ (pixels)$$

$$\theta = tan^{-1}\left(\frac{X_{max}}{d}\right) \ (degrees)$$

$$EZ_1 = \frac{L_1 d}{X_1} \ (meters)$$

$$EZ_2 = \frac{L_2 d}{X_1} \ (meters)$$

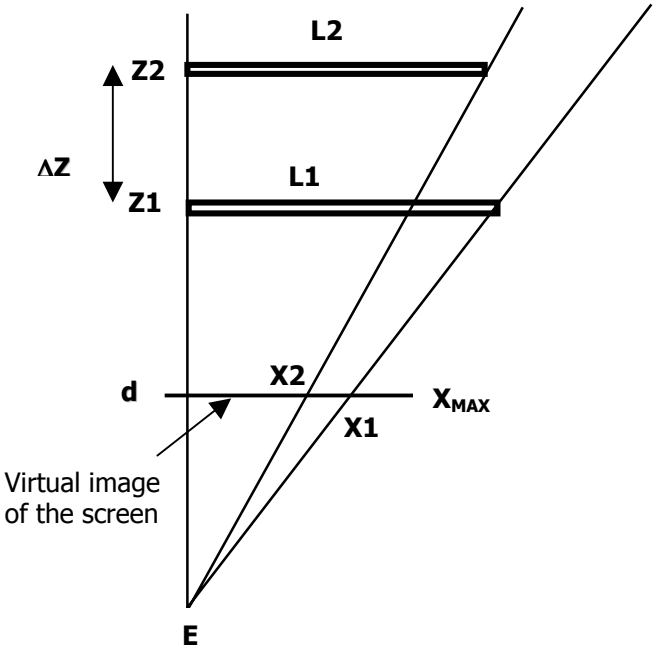

Figure 24: A method to determine both the eye-point location and the field-of-view by interactively registering virtual and real entities.

To solve this problem, we propose a variant of the technique previously described. The technique consists in using two real objects that are placed at two different depths as shown in Figure 24. The depth difference between the two objects is easily accessible in the real world. In our case, it is measured by using our optical tracker.

We use the method described in the viewport calibration to align visually the vertical sides of a real rectangle of known dimension with two vertical lines scanning the rendered area. For this calibration, the rendering of the lines is done by rendering two vertical cylinder with a small diameter in OpenGL using an orthogonal projection and turning the anti-aliasing on. The anti-aliasing allows reducing the resolution of the measurement to sub-pixel. The orthogonal transformation is set so that the pixel values are related to the distance between two three-dimensional lines to register with the side of the real rectangle.

A user ready to use the system can quickly register interactively two virtual lines with the sides of two real rectangles placed at two distances Z_1 and Z_2. The procedure must be done for each eye since both fields of view can be different. The system records the dimension in pixels of the objects in the framebuffer, X_1 and X_2. Using these values, the horizontal field of view for each eye and the location of the center of rotation of the eyes, E_L and E_R, are determined. The determination of these parameters can be done using the equations given in Figure 24. X are in pixels, L and Z are in meters. L is the dimension of the real object along the horizontal direction. Note that the calibration of the field of view has to be done only for each eye in only one dimension since we fixed the ratio

between the horizontal and the vertical field of view when the viewport is calibrated.

Once the fields of view have been determined, they are specified in OpenGL to ensure correct rendering. Using this calibration method, the position of the point E used in the next calibrations can be determined by taking the middle point between E_L and E_R. Because this point is located where the user head is and cannot thus be marked physically, a landmark placed at a known distance from this point is used instead.

8.4 Interpupilliary distance adjustment

The second user-specific parameter is the inter-pupilliary distance (IPD), the distance between the eye-points of the subject. This distance must be applied to the displays so that the optical axis of each arm of the setup is aligned with the optical axis of each eye of the user. This allows consequently aligning the centers of the screens with the optical axes of the eyes. The IPD must also be used in the rendering to place the viewpoints in the virtual world used to render the view for each eye. We measure the IPD using an optical device called pupilometer.

Most commercial HMDs do not offer an adjustment of the IPD distance and thus the perceived depth of virtual objects is incorrect. A solution to this problem is to render the virtual scene using a perspective projection centered at the location of the viewport that is aligned with the entrance pupil of the user. This technique known as off-axis projection corrects the depth perception problem. However, the

pre-distortion of the image exposed in the section 2 of this chapter must be recomputed for each user since the center of projection move with respect to the center of the lens around which the distortion occurs. Therefore, off-axis projection is usually not used.

Our current setup allows adjusting the IPD distance so that the subject can perceive at the correct depth. The position of each arm's optical axis has been marked precisely during a laser-based optical calibration and is used toward this goal. After the field of view has been tuned for each eye individually, a test of the correctness of the calibration can be done by placing a real object at a distance known from E determined in the previous calibration step. Then, its virtual counterpart can be rendered from two viewpoints separated of the IPD distance and registration should be verified.

8.5 Tracking calibration

The tracking calibration is an important component for the registration of real and virtual entities because it conditions the correct measurement of the attitude of the real entities and thus the correct attitude of their virtual counterparts. We used two tracking devices in our experiment, the Optotrak and the parallax ramp, and we will assess their calibration in the following. Several referentials will be used in this section and their location can be seen on Figure 25. For the details of the transformation matrices used in this section, refer to Appendix D.

We track the knee joint using the Optotrak. In our current VRDA prototype, the area viewed by the display is limited and the tracking technique of the subject's

knee is not developed yet. Therefore, we did not use the knee of a subject as real counterpart of our synthetic knee model, but rather we used a mannequin leg. The mannequin leg has some fixed shapes and dimensions and it cannot be flexed. Consequently, the calibration of the tracking of the leg is simplified since it is limited to the tracking of a rigid body. When, the system will be improved and the knee joint of a subject will be tracked dynamically, some new calibration techniques will have to be designed. In effect, the tracking of the position, orientation, and flexion angle of the knee joint will have to account for the variability of knee size and shape among human subjects.

To be able to track the mannequin leg with the Optotrak, a rigid body must be built. We attached six markers on the leg so that three are on the top of the femur, and three are on the lower end of the tibia. By distributing the markers on these two extremities, one get the most precision on the orientation of the model around the two axes perpendicular to the long axis of the tibia. The reason to have more precision on these axes is to provide the best alignment along the length of the leg where error will be the most noticeable.

The Optotrak' software tools allow creating a rigid body using different ways. The user can specify directly the coordinates of the markers with respect to the frame of reference of the rigid body. Another more attractive way to construct a rigid body is to show the rigid body once built to the cameras of the tracker so a rigid body can be constructed out of multiple frames. The rigid body is manipulated slowly during this procedure so that all markers are in view during one frame at

least. For each frame, the spatial position of each marker is recorded and after analysis of these data, the tool produces a rigid body.

The cameras of the Optotrak are considered to be precisely calibrated in the range considered for this experiment so that it makes correct position measurements. However, the calibration of the cameras is an issue that should be considered seriously in a custom built apparatus because it conditions the precision of the measurements. The Optotrak recover the position and orientation of a rigid body composed of multiple markers by aligning the model it has in memory with the rigid body formed by the markers it sees.

Because in practice, the markers are fired sequentially for the Optotrak to be able to recognize them individually, there is an error in the detected position of each consecutive marker after the first marker. Consequently, the rigid body is aligned optimally to the markers' positions that are incorrect, and thus the rigid body attitude itself is incorrect. While there is a solution to this problem using prediction of the position of the markers based on the velocity and acceleration of the probe, this error is assumed negligible at this time until further assessment in progress are completed. We make this assumption because of the high measurement rate of the tracker and the slow motion that we will use in our registration experiment.

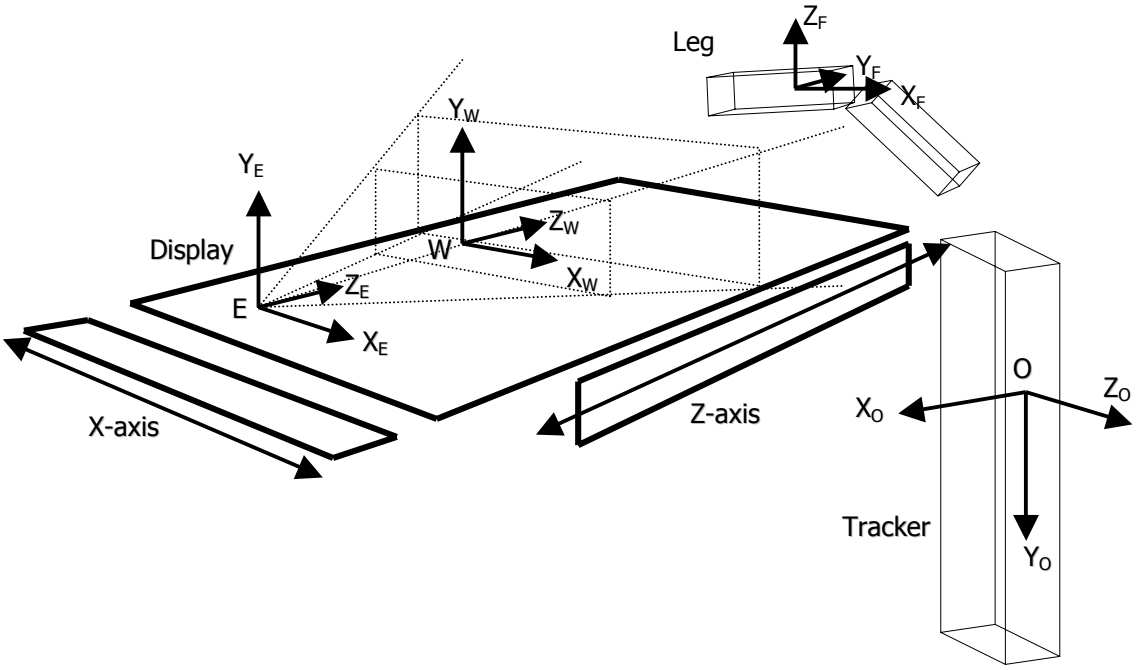

Figure 25: The referential used during the calibration procedure. The axes X and Z are determined by both ramp attached to the optical table. The referential E is attached to the cart of the parallax stage. The referential O is attached to the tracking system. The referential F is attached to the leg to track. Finally, the referential W is temporarily defined using a marker align with a mire rendered on the display. The plans at relative known depth used during the field of view calibration can be seen in dotted line.

Let O be the referential attached to the optical tracking system. Let E be at the middle of the eye-points of the user. Let F be the referential attached to the mannequin leg. To display the virtual knee in 3D at same position and orientation than the real knee, the rendering engine must know the transformation M_{EF}. The knee model can then be translated and rotated from the origin E to achieve this transformation and place the object (knee) in the virtual world. Finally, by shifting the viewpoints of half the IPD distance along the horizontal direction, the views for the user's eyes can be generated.

The referential F attached to the mannequin is tracked by the Optotrak, so the transformation M_{OF} is known at any time. The referential E is tracked by the

parallax ramp as the user moves left and right. Head tracking will later be achieved by using a rigid body attached to the display. The display will be a HMD mounted on the user's head, so that the user will be provided a larger working volume. At this time, the transformation M_{OE} will be known at any time since the tracking of the head will be done by the Optotrak. Therefore, the transformation M_{EF}, needed for the rendering, will be simply computed by combining the invert of the transformation M_{OE} with the transformation M_{OF}.

Currently the parallax stage is used for the head tracking so the problem is more complex. We define an additional frame of reference W to solve it. The referential W is attached to the optical table in our setup. The X-axis of the referential is oriented along the width of the table, the Z-axis is horizontal along the length of the table that is also the direction of sight, and the Y-axis is vertical and going up. Practically, the X-axis is defined by moving the parallax ramp left and right. The Z-axis is defined by moving a translation ramp attached to the long side of the table, and the Y-axis is defined by cross product of the two previous axes. The direction of each of these axes can be defined in the frame of reference of the Optotrak by moving each ramp to its two extreme positions. By recording, for each extreme position, the location of a marker attached to the ramp and subtracting the coordinates of the marker's locations, one can compute a vector director of the corresponding axis. This procedure had to be done only once, each time the relative position and orientation of the optical table with the tracker changes.

During the field of view calibration, E_L and E_R have been recorded to be at a height Y_0 and depth Z_0. A marker or the tip of a probe is placed in W_0, roughly at the nominal distance (0.8 m in our case) and at the middle of the width of the table. The position of W_0 with respect to the eye-points is known after the field-of-view calibration except for the translation along the X-axis. A three-dimensional bull's eye is displayed at the same depth and height than the point W_0 with respect to the eye-points, and the user task is to visually align it with the physical marker by translating the parallax stage. After this alignment, the middle of the eye-points E and W_0 have the same X coordinate. Therefore M_{WE} can be determined.

When the parallax stage is placed at such a location, the linear encoder measures its position, so that the following positions of the parallax stage are given with respect to this origin. During the use of the system, the transformation M_{WE} is biased with an additional translation along the X-axis equal to the position of the stage. The transformation M_{OW} is known by simply measuring the position of the marker with the Optotrak. Therefore, the transformation M_{OE} can be computed by combining the transformations M_{OW} and M_{WE}. Finally, the transformation M_{EF} can be computed as described before so that the virtual entity can be represented with reference to the user head-position.

At this point, the referential F is attached to the mannequin leg and this referential is driving the referential K of the geometrical knee model accordingly in the virtual world. However, generally these two referential are not located and oriented the same way when the virtual model of the bones is actually visually

correctly registered into the mannequin leg. The reason for this is that the tool to construct a rigid body shows only the markers and not the real object supporting them as well as the virtual model that we want to register.

The last calibration step to solve this problem consists in interactively registering the virtual (knee model) and real (mannequin) entities so that the result is visually satisfying. This is done once for any new virtual or real model used by a user for who the system has been calibrated. The task consists in varying the translation and rotation parameters of the transformation M_{FK} so that the real-virtual superimposition appears to be convincing for some positions and orientations of the real-virtual entities. Once the correct transformation has been computed, the referential of the rigid body is biased of M_{FK}. Consequently, the registration is correct next time a user utilizes the system.

The result of the calibration procedure for our current setup can be seen in Figure 26. It can be seen that the subjective illusion of the graphical model of the bones being inside the knee is achieved for several attitudes of the mannequin leg. The quality of the result is however altered because the graphical model of the bones used was from a male, while the mannequin leg is from a female, so that the bones are out of the physically knee for some attitudes of this one. This shows that a scaling of the bones should be done according to the subject knee.

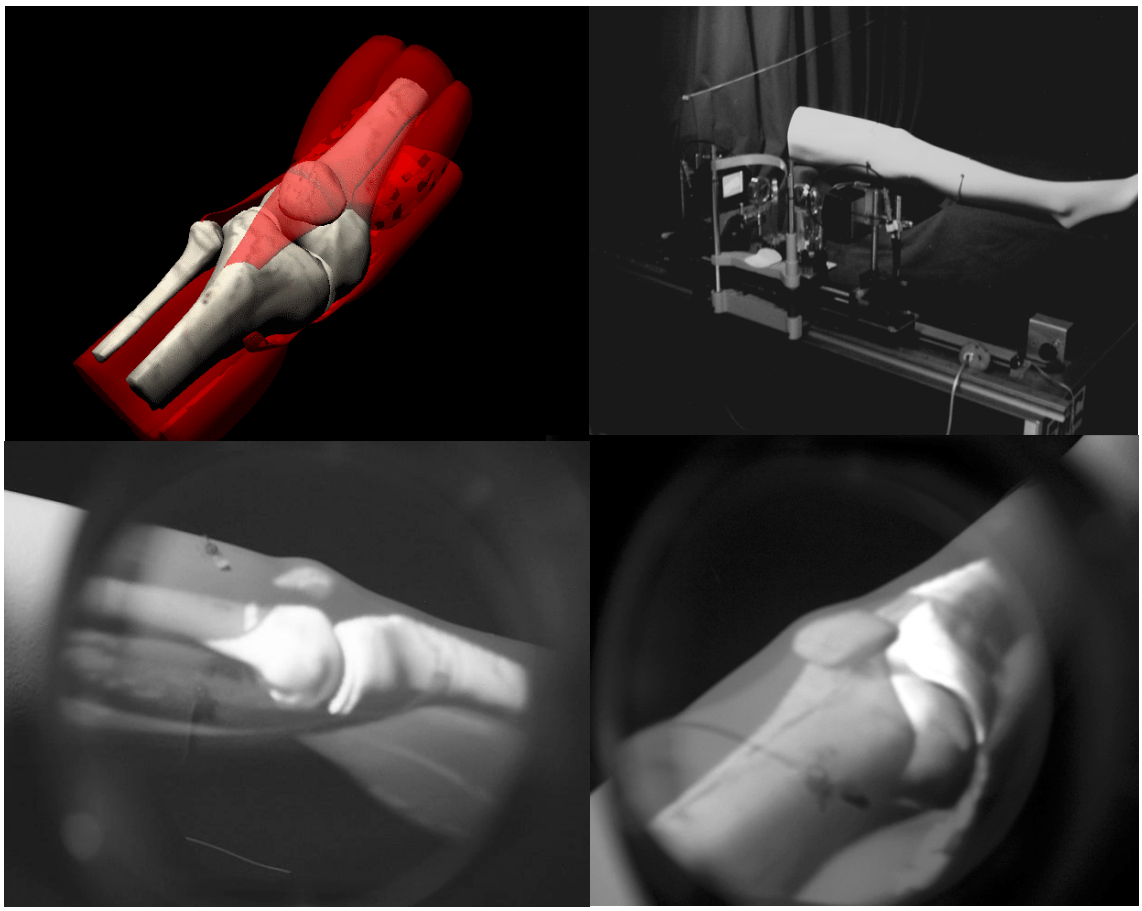

Figure 26: Registration results. The upper frames show the synthetic knee model used, including the bones and the muscles, as well as the complete setup showing the mannequin leg. The lower frames show two photographs taken through the optics with respectively a side view and a diagonal view of the knee joint. Despite these views are monoscopic, the impression that the knee model is inside the mannequin leg can be clearly seen.

9 FUTURE WORK

In this section, we describe future work. The first implementation of the VRDA resulting from this research allows most of this multidisciplinary work to be conducted on a first structure of the system.

A study is currently conducted by Dr. Wright at the Department of Radiology of UNC to determine a relationship between the dimension of the bones and dimension of external landmarks on patients having an X-ray done. This study is going to give us a way to scale the synthetic bones with respect to the subject on which the virtual is superimposed.

Dr. Biocca at the Department of Radiology of the University of Michigan will be assessing the knee joint model we created. Iterative correction of the current and future biomedical joints will be done.

Studies are conducted to develop appropriate tracking algorithms for our tracking system (Davis, 1998). We are currently trying to improve the dynamic accuracy and design some correction algorithms if needed.

Some probes to track the user's head (Baillot, 1999) and the tibia and femur segments of the subject will have to be designed. A calibration of the knee probe and/or some optimization algorithms will be needed to locate correctly the bones of any subject from external landmarks.

Deformable models will be reviewed and some studied will be conducted to attempt to improve them if needed. The ligaments will then be modeled using these deformable models.

A novel lightweight HMD with IPD adjustment and using reflective technology is currently designed and will be certainly used for the VRDA tool. Spherical tracking probes are currently studied to track the user head from any viewpoint.

The software application will be transferred to PC as soon as graphics hardware is efficient enough for normal use of the VRDA tool. The hardware will be interfaced to the PC so that the system can largely available to medical institutions.

Finally, we will have the tool assessed by collecting feedback from some medical institutions trying the tool and the system will be iteratively improved if needed.

CONCLUSION

The study has been divided in two main parts: the modeling and the visualization. A review of the modeling software packages was done to assess their adaptability to the modeling of the knee joint. Moreover, current modeling methods to perform the automatic modeling of the knee joint were reviewed. None of the reviewed modeling packages and methods were found useful to model the knee joint for the VRDA tool.

An algorithm using the actual geometry of the contacting surfaces of the knee model considered as well as some biomedical kinematic constraints was developed. The algorithm automatically searches the stable position and orientation of two objects pushed against each other along a specific direction. The algorithm is based on collision detection to detect the contact points and a robust incremental procedure to move one of the solids toward the other in a stable natural position.

Automatic modeling of the flexion motion of a 3D generic model of the knee bones has been done by applying the algorithm at incremental motion step along the flexion/extension and the varus/valgus angles. Smoothing has been done to obtain some convincing results that have been presented. The motion parameters have been recorded in a lookup table (position and orientation of the tibia). To conform to pure flexion/extension, a procedure has been designed to create a new lookup table used for the simulation where the varus/valgus angle is zero.

A Performer application has been designed as a basis to the rendering engine that will be used for the VRDA tool. Mouse and keyboard interactivities have been added to be able to see the resulting model from any viewpoint on monitor. Realistic rendering of the textured bones and menisci has been achieved.

The components of the virtual environment have been described and their tasks have been identified. The interactions devices as well as the implementation of the interfaces between software and the hardware have been detailed. A calibration procedure of the complete system has been given based on the considerations specific to our system. Both optical and tracking calibrations are taken into account in our approach. Early results of registration using a 3D-bench prototype and a generic rigid leg in extension have been shown.

The presented research gives an initial framework toward the complete implementation of the VRDA tool. This work provides an application that allows calibration of the virtual environment, interaction in the virtual world, and real-time visualization.

APPENDIX A

RIGID BODY DYNAMICS

The implementation of the laws of dynamics is problematic because the equations used need to be integrated with respect to the time to obtain an analytical solution. Here are the equations of the rigid body dynamics method:

$\sum \vec{\mathbf{F}} = m\vec{\mathbf{a}}$ and $\sum \boldsymbol{\tau} = \vec{\mathbf{I}}.\dot{\vec{\omega}}$ where :

$\vec{\mathbf{F}}$ a force acting on the rigid body

$\vec{\mathbf{a}}$ the linear acceleration of the rigid body

$\boldsymbol{\tau}$ a torque produced by a force acting on the rigid body and defined as :

$\boldsymbol{\tau} = (\vec{\mathbf{p}} - \vec{\mathbf{x}}) \times \vec{\mathbf{F}}$ with :

$\vec{\mathbf{p}}$ the application point of a force

$\vec{\mathbf{x}}$ the position of the center of mass of the rigid body

$\vec{\mathbf{I}}$ the inertial tensor associated to the rigid body and defined as :

$$\vec{\mathbf{I}} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \text{ with } \begin{cases} I_{xx} = m \int_V (y^2 + z^2) dV \text{ (diagonal terms)} \\ I_{xy} = -m \int_V xy dV \text{ (off - diagonal terms)} \end{cases}$$

$\dot{\vec{\omega}}$ the angular acceleration of the rigid body defined as :

$$\dot{\vec{\omega}} = \frac{d\vec{\omega}}{dt} = \frac{d}{dt} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \text{ where } \vec{\omega} \text{ is the angular velocity}$$

By using these equations, the linear and rotational acceleration of an object submitted to some given forces at a time t0 can be found. By integrating these equations, the linear and rotational velocity of the object at the time t0+Δt can be determined. This computation assumes that the acceleration of the object is constant during Δt, the integration step time. Integrating another time will lead to the attitude of the object at time t0+Δt assuming that both the accelerations and velocities are the same during the integration step time. If the velocity and acceleration are not constant during a time step, errors are accumulated and

divergent oscillations can be produced. Moreover, the result obtained is largely dependent of the initial conditions of speed and position, as well as the time step used. Rigid body dynamics also assume that the center of mass of the rigid body is known, but it is difficult to determine it in the case of the knee. The equations detailed above are for unconstrained rigid body dynamics, i.e. the rigid body is supposed to be able to move freely in space because of the forces acting on it. However, when a rigid body is constrained to a certain motion because there is contact with another surface, one must use constrained rigid body dynamics. In effect contact forces are not defined in amplitude and cannot be implemented in the general framework as normal forces. This additional operation will make the whole process even more challenging given that the contact forces are changing at each step of the modeling. In front of the difficulty of this implementation, there is at our knowledge no published result demonstrating a knee joint modeled on a full range of motion using rigid body dynamics.

APPENDIX B

MODELING CONFIGURATION FILE

```
MODEL                0          femur
MODEL                1          tibmen

INIT_POS             1          0          -0.02      0
INIT_STEP            1          0          0.005      0
INIT_ROT             1          0          0          0

SHIFT_MODEL          0          -0.5       0          0
ORIENTATION_MODEL    0          0          180        0
SHIFT_MODEL          1          0.4        0          0
ORIENTATION_MODEL    1          0          -90        0

DISPLAY              640        0          640        486
ZOOM                 8          8          8
ANGLE_RANGE          0          0          90         1
ANGLE_RANGE          1          -15        15         1
CONTACT_RESOLUTION   0.0001
MODEL_STEP           0.001
RENDER               WIRE
```

This file is a sample configuration file used for the modeling of the knee joint. The keyword MODEL in the first line specifies to load the model *femur* and to assign it to the reference rigid body, which we set to have the id 0. Upon the arrival of this keyword, the OBJECT file *femur.obj* is loaded from the directory *models*. The second model loaded here is the *tibmen* rigid body, described earlier as the rigid body containing the tibia, the menisci, and the fibula.

The three following keywords specify respectively the initial position, motion, and orientation to apply to this rigid body. Here, the initial position of the rigid body 1, the *tibmen,* is such that the origin of the *tibmen* object is 2 centimeters below the origin of the *femur* since the Y-axis is vertical. Remark that for the *femur* no parameter have been specified. Therefore, they are all initialized to null values. The initial motion parameter specifies that the rigid body *tibmen* will initially approach the *femur* along the vertical axis at a rate of 5 millimeter per modeling

step. The orientation parameter gives the orientation of the rigid body *tibmen* by specifying the angle of rotation around the X, Y, and Z-axis.

The keyword SHIFT_MODEL and ORIENTATION_MODEL allow creating several view of the scene and the values are identifying the view number, the shifting of this view in meters along the three axes, and the orientation of the object. This allows perceiving complex modeling processes more clearly. The DISPLAY and ZOOM keyword are used to setup the display area and setup the scaling of the object.

The CONTACT_RESOLUTION (i.e. 0.1 mm) setup the modeling resolution in translation as well as the contact resolution employed in the algorithm. The modeling resolution, one millimeter here, is used as modeling step for the translation after the solids touch for the first time. The orientation resolution and modeling step are not specified here but are set by default to 0.1 degree and one degree.

The ANGLE_RANGE lines are used to setup the rotation increments to apply on the moving rigid body (i.e. the *tibmen*) after each new stable position and orientation has been found. These lines specify that the flexion angle will be changed from 0 to 90 degrees, and for each of these angles the varus/valgus will be spanned from –15 to 15 degrees. The coding of 0 as the flexion and 1 as the varus/valgus is done internally in the program.

The rendering type can be set to smooth shaded or wireframe by using the keyword RENDER followed by the values *fill* or *wire*. The smoothing of the model

depends on the correct specification of the normals. The axes of the frame of reference as well as the normal can be displayed when the modeling program is running by setting macros in the code. The rendering can be turned off to increase the modeling speed, or only the stable motion steps can be shown as they are computed.

APPENDIX C

RENDERING TYPES

We shall describe here the different ways to visualize a 3D model selected by the current rendering mode. There exist two types of model: polygon-based and NURB-based. Polygon-based stands for a type of model described discretely as a list of polygons described themselves as a list of vertices. NURB-based models are described continuously by bicubic B-spline patches deformed by control vertices. Current rendering hardware transform at rendering time each patches of a NURB model into polygons meshes approximating the surface within a given tolerance.

Given that all the models are described as polygons at rendering time, there are several ways to visualize them. Wireframe rendering is a mode allowing fast computation because only the sides of the polygons are traced. It also allows seeing the geometry underlying the model. A variant of this mode is the hidden line mode where the polygon that are supposed to be hidden by other in front of them are not rendered, giving more depth to the model. Another rendering mode is filled, where each polygon is colored by a user given color. This mode does not give any depth impression because the color of the surface is not changing according to the environment. In order to give a depth impression due to lighting conditions, one must use shading.

Shading uses the incoming light, the material property of the object, as well as the orientation of the current face to compute the color of each pixel. The rendering hardware is typically dividing each polygon into triangles to perform this operation. Then, the color of each of the three vertices of the triangle are computed using a shading algorithm using the incoming light, the material

property, and the normal at each vertex, giving the orientation of the surface. Each triangle is painted using a trilinear interpolation of the color of the three vertices of the triangle to find out the color of each pixel. While the geometry is discrete because it is described as polygons, this method gives the impression of a smooth surface because trilinear interpolation is used. There exist several shading algorithms that can be used according the type of surface to simulate; Gouraud, Blint, and Phong are examples. Phong is usually available in OpenGL. Each type of shading is more or less adapted to the lighting conditions and material of the model rendered.

A last improvement that one can add to a shaded model is texture. Texture allows showing default and variability of the attribute of a surface in a realistic way without having to compute new attributes at every pixel. An original texture image is employed and is tilled on the model by the hardware. The filtering of the texture due to its deformation is done in hardware. For more details on the types of rendering, the reader should refer to the book by Foley & Van Dam (Foley, 91).

APPENDIX D

TRANSFORMATION MATRICES

The transformations used in section 8.5 are homogeneous transformations. Homogeneous transformation includes additional parameters allowing to manipulate rotation and translation matrices together to perform the following operation:

$$\mathbf{P}' = \mathbf{RP} + \mathbf{T}$$

Where X is the 1x3 column-vector locating a vertex to transform and X' is the resulting vertex after transformation. R is the 3x3 matrix representing the rotations to perform on the vertex around X, Y and Z, and T is the 1x3 translation vector added to the vertex transformed by the rotations. To encode successive translations and rotations as well as other transformation, the hardware transforms the translation and rotation matrix into a single 4x4 homogeneous transformation matrix M:

$$\mathbf{M} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By using such a general representation for the matrix, successive homogeneous transformations can be combined into a single homogeneous matrix by multiplying them. Further, the transformation of a vertex is done by multiplying the resulting matrix by the vector representing the vertex expressed in homogeneous coordinates:

$$\mathbf{P} = \begin{bmatrix} P_X \\ P_Y \\ P_Z \\ 1 \end{bmatrix}$$

and the first operation can then by simplified as:

$$\mathbf{P}' = \mathbf{MP}$$

LIST OF REFERENCES

Ateshian GA (1993) "B-spline least-square surface fitting for articular surface of diathroidal joints", J Biomech Eng 115(4), 366-373.

Azuma R (1995) "Predictive tracking for Augmented Reality", PhD Dissertation, UNC-CH, TR95-07.

Baillot Y (1999) "Algorithm to find the stable position and orientation of two rigid bodies". US patent filed.

Baillot Y (1999) "Study on spherical tracking probes design", CREOL/UCF, Technical Report 99-01, December 1998

Baillot Y, Rolland JP & Wright DL (1999) "Automatic Modeling of knee-joint motion for the Virtual Reality Dynamic Anatomy Tool (VRDA) ", Proceedings of Medicine Meets Virtual Reality 99, IOS Press

Bajura M and Neumann U (1995) "Dynamic registration correction in video-based augmented reality systems", Proc of the IEEE VRAIS'95, 189-196.

Blankevoort L, Huiskes R and Lange A de, (1990) "Helical axes of passive knee joint motions", J Biomech 23, 1219-1229.

Blankevoort L, Huiskes R, Lange A de (1988) "The envelope of passive knee joint motion", J Biomech 21, 705-720.

Blankevoort L, Kuiper JH, Huiskes R and Grootenboer HJ (1991) "Articular contact in a three-dimensional model of the knee", J Biomech 24(11), 1019-1031.

Bylski-Austrow DI, Ciarelli MJ, Kayner DC, Mattews LS, and Goldstein SA (1994) "Displacemens of the menisci under joint load: an in-vitro study of human knees", J Biomechanics, 27 (4), 421-431

Conlay MD and Long GL (1994) "ligament forces, condyle reactions, line geometry, screw theory and human knee stability", Advances in Bioengineering American Society of Mechanical Engineers, Bioengineering Division (publication) BED 28, AMSE, New York, NY, USA, 169-170.

Davis, L., Rolland, J.P., & Baillot, Y. (1998) "Probe design for Tracking Based on LED imaging", CREOL/UCF, Technical Report 98-03

Delp SL and Loan JP (1995) "A graphics-based software system to develop and analyze models of musculoskeletal structures", Comput Biol Med 25(1), 21-34.

Delp SL, Loan JP, Hoy MG, Zajac FE, Topp EL and Rosen JM (1990) "An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures", IEEE Transactions on biomedical eng, 37(8), 757.

Foley J, van Dam A, Feiner S, and Hugues J (1991) “Computer Graphics, principle and practice”, second edition, Addison Wesley Edition.

Frankel VH (1997) Personnal communications.

Garg A and Walker PS (1990) "Prediction of total knee motion using a three-dimensional computer-graphics model", J Biomech 23(1) 45-58.

Gottschalk S (1996) “OBB Tree: A hierachical structure for Rapid Interference detection”, Proc. of ACM Siggraph'96.

Hefzy M and Grood E (1996) "Review of knee models: 1996 update", App. Mech. Rev 41, 1-13.

Holloway R (1995) Registration errors in augmented reality systems, PhD dissertation, University of North Carolina at Chapel Hill. TR95-016.

Huiskes R and Blankevoort L (1990) "The relationship between knee joint motion and articular surface geometry", Biomechanics of Diarthrodial Joints, Springer-Verlag, NY, II, 269-286.

Huiskes R, Kremers J, Lange A de, Woltring HJ, Selvik G and Rens Th JG van (1985) "Analytical stereophotogrammetric determination of the three-dimensional knee-joint geometry", J Biomech 18, 559-570.

Janin AL, Mizell DW and Caudell TP (1993) "Calibration of a head-mounted display for augmented reality applications", Proc of the IEEE VRAIS'93, 246-255.

Kuiper JH (1988) "Modeling of articular contact in a mathematical knee model", M Sc Thesis, University of Twente, Enshede, The Netherlands.

Kurosawa H, Walker PS, Abe S, Garg A and Hunter T (1985) "geometry and motion of the knee for implant and orthotic design", J Biomech 18, 487-499.

Loch DA, Luo Z, Lewis JL and Stewart NJ (1992) "A theoretical model of the knee and ACL: Theory and experimental verification", J Biomech 25(1), 81-90.

Minami M, Yoshikawa K, Matsuoka Y, Itai Y, Takashi Kokubo T, and Iio M (1991) "MR study of normal joint function using a low field strength system", J Computer Assisted Tomography 15(6), 1017-1023

Nisell R (1985) "Mechanics of the knee, a study of joint and muscle load with clinical applications", Acta Orthop Scand, 56 (216).

Nordin M and Frankel VH (1980) "Basic Biomechanics of the Skeletal System", Lea and Febiger, Philadelphia, Chap 4.

Northern Digital (1992) "Accuracy of digitizing probes", Technical report #3.

Ounpuu S, Gage JR, Davis RB (1991) "Three dimensional lower extremity joint kinetics in normal pediatric gait", J. Pediat Orthop. 11, 341-349.

Ponamgi M, Manocha D and Lin M (1995) "Incremental algorithms for collision detection between general solid models", Proc of ACM/Siggraph Symposium on Solid Modeling, 293-304

Rehder U (1983) "Morphometrical studies on the symmetry of the human knee joint: femoral condyles", J Biomech 16, 351-361.

Rolland JP, Wright DL, and Kancherla AR (1997) “Towards a novel augmented-reality tool to visualize dynamic 3-D anatomy”, Proceedings of the Medicine Meets Virtual Reality: 5.

Rolland, J.P., Baillot, Y., Davis, L., Vaissie, L., & Wright, D.L. (1998) "Role of optics in virtual environments", Invited Proceeding of the International Lens Design Conference

Rolland, J.P., Baillot, Y., & Goon, A.A. (1999) "A Survey of tracking technology for virtual environments", book chapter in Augmented Reality Wearable Computer, Edition Bardfield and Caude l, Mahwah NJ

Seedhom BB, Longton EB, Wright V and Dowson D (1972) "Dimensions of the knee; Radiographic and autopsy study of sizes required for a knee prosthesis", Ann Rheum Dis 31, 54.

Shiavi R, Limbird T, Frazer M, Stivers K, Strauss A and Abramovitz J (1987) "Helical motion analysis of the knee-I. Methodology for studying kinematics during locomotion", J Biomech 20(5), 1987, 459-469.

Shiavi R, Limbird T, Frazer M, Stivers K, Strauss A and Abramovitz J (1987) "Helical motion analysis of the knee II. Kinematics of uninjured and injured knees during walking and pivoting", J Biomech 20(7), 653-665.

Soudan K, Van Audekercke R and Martens M (1979) "Methods, difficulties and inaccuracies in the study of human joint kinematics and pathokinematics by the instant axis concept. Example: the knee joint", J Biomech 12, 27-33.

Thompson WO, Thaete FL, Fu FH, and Dye SF (1991) "Tibial meniscal dynamics using three-dimensional recontruction of magnetic resonance images", The American Journal of Sports Medicine, 19 (3), 210-216

Vaissie L and Rolland JP (1998) "Analysis of Eyepoint Locations and Accuracy of Rendered Depth in Binocular Head-mounted Displays", CREOL/UCF, Technical Report 98-01.

Walker PS, Rovick JS, Roberston DD and Schrtager RJ (1988) "The effects of knee brace hinge design and placement on joint mechanics", J Biomech 21, 965.

Welch G and Bishop G (1997) "SCAAT: Incremental Tracking with Incomplete Information," SIGGRAPH 97 Proceedings

Wismans J, Veldpaus F, Janseen J, Huson A and Struben P (1980) "A three - dimensional mathematical model of the knee-joint", J Biomech 13, 677-685.

Wright DL, Rolland JP, and Kancherla AR (1995) "Using virtual reality to teach radiographic positioning", Radiologic Technology, 66(4), 233-238

Yamaguchi GT and Zajac FE (1989) "A planar model of the knee joint to characterize the knee extensor mechanism", J Biomech 22(1), 1-10.