Real-Time Collision Detection Between Cloth And Skinned Avatars Using OBB

Nuria Pelechano

September 9, 2002

Department of Computer Science

University College London

Supervisor: Mel Slater

Acknowledgements

I would like to take this opportunity to thank the following people for their contribution, in one way or another, to the completion of this work.

Firstly, I would like to thank my supervisor Mel Slater for his inspiration, encouragement, and assistance through the process of completing this thesis. Special thanks to Lee Bull for his advice, patience and generous donation of time. Thanks are also due to Jonathan Starck from the university of Surrey for his help and for the donation of the Prometheus Avatar Toolkit code used for part of this project.

I would also like to thank my family for their love and support.

Table of Contents

Table of Figures

ABSTRACT

This project addresses the problem of detecting collision between flexible objects, such as cloth and skin. We are dealing with the collision detection between two sets of points. The first set of points corresponds to the cloth and the second one corresponds to the skin of the deformable body.

The cloth is modelled by a mesh of points. The movement of those points is given by the fundamental law of dynamics and will be affected by different kinds of forces. These forces can be divided in two subsets, internal and external forces. Internal forces are the resultant of the tension of the springs linking a point with its neighbours. External forces can be due to gravity, wind, inertia and the movements of the body.

Skin is modelled by a mesh of triangles. Each individual vertex is related to a particular limb of the skeleton and will change its position depending on the movements of that limb of the skeleton.

Our target is to achieve quick collision detection between those two sets of points in order to obtain a real time animation. For this purpose we have studied different methods of space subdivision such as space partitioning trees combined with the skeleton hierarchy.

The algorithm proposed performs collision detection starting with a coarse approximation of the human body given by oriented bounding boxes, to finally perform a more accurate collision detection test in the areas required.

We have achieved a reduction of about 98% in the number of triangles against which we need to apply collision detection. This reduction allows the algorithm implemented works well for real-time animation of deformable human bodies.

1 INTRODUCTION

Interaction of objects in the virtual world is often an important aspect of computer animation. This interaction between moving objects is usually a dynamic process which main aspects are the collision and the response. Collision detection between two objects involves testing for intersection of these objects and once an intersection has been detected, dynamic equations are applied to simulate collision response.

Collision response involves modifying the velocities of the objects colliding in order to avoid them from passing through each other once the collision occurs. This implies computing new forces for the vertices that collide.

The main problem in collision detection is to control de computational complexity due to the discretization. The surface of the body and the cloth surface are both represented by polygonal meshes that can have several thousand polygons each. Testing each pair of polygons for potential collisions is an unrealistic task.

Collision handling can cause substantial difficulties for any real time system where a maximal simulation speed is required since the computational requirement increase according to a square law with the number of vertices in the modelling scene. In practice, the implementation of collision detection and response methods can account for anything up to 90% of the total computation required for a typical simulation. This complexity problem drives the development of advanced algorithms for collision detection.

In the case of collision detection between cloth and skin we are dealing with deformable objects, so collisions appear not only between different objects but also several parts of the same object, which is known as self-collision detection.

Geometrical collision detection can consist of detecting whether the objects interpenetrate (interference detection), or the objects come closer than a given threshold distance (proximity detection).

When we want to design an efficient collision detection algorithm we need to consider the following:

- If we have more than one test available with different costs, how can we combine them?
- How can we make the test cheaper?
- How can we avoid unnecessary tests?

And so the principles we should follow are:

- Apply first of all a fast simple test in order to eliminate many potential collisions. Eg: test bounding volumes before testing individual polygons of the objects
- Exploit locality to reduce the number of potential collisions. Eg: use cell structures to avoid taking into account distant objects.
- Use as much information as possible about the geometry.
- Exploit frame to frame coherence between successive tests

There are many different collision detection algorithms in the literature, adapted to the geometrical attributes specific to the context in which they are implemented. However they can be classified into groups depending on the general idea used to reduce the complexity of the problem. The main groups are:

- Bounding volumes: Complex objects or groups of objects are enclosed within simpler volumes that can easily be tested for collisions.
- Subdivision methods: they can be based either on the scene space or one the object. The main idea of this method is to decompose the problem into smaller space volumes or objects regions, usually evaluated through bounding volume techniques. Efficiency can be added by using hierarchical subdivision techniques.
- Projection methods: evaluate possible collisions by considering separately the projections of the scene along several axes or surfaces.
- Proximity methods: where we have an arrangement of the scene according to their geometrical neighbourhood and detection between objects is collected depending on their neighbourhood structure.

The deformable surfaces with which we are dealing for this project are given as polygonal meshes. In our case collision detection tries to search efficiently for those surface polygons which are colliding when considering highly refined meshes containing thousands of polygons. The nature of a polygon mesh consists in numerous flat elements, where neighbouring elements are connected by a locally fixed topology and there is also a slightly deforming geometry during animation. It is thus important to consider the nature of the objects we are dealing with in order to chose the best algorithm.

The mesh description accuracy of a curved surface depends directly on the refinement of the elements of that mesh, that is on the size and number of mesh polygons. Collision detection between polygonal mesh surface representations should be able to handle very large number of elements while detecting only a small number of collisions, because complexity of the collision detection increases with the number of elements but the amount of collision involved in a curved surface depends only on the formal geometry of that surface.

The elements consisting a mesh will always keep the same neighbourhood and thus maintain similar relative positions. Collision detection algorithms can therefore take advantage of this consistency during animation.

Finally we have to consider the robustness of the algorithm we want to use, because the fact of taking advantage of the simplified geometrical assumptions to carry out their computation efficiently, may lead to unreliable algorithms that miss some collision or degrade the performance in those cases where the particular geometrical situation may not satisfy these assumptions.

In this project we propose a collision detection method especially efficient for dynamic deformable human bodies. Collision avoidance consists in testing, firstly, cloth vertices against a coarse representation of the body given by an oriented bounding boxes hierarchy. Secondly, vertex-triangle intersections are tested and repulsive forces are applied. The magnitude and direction of such forces are dependent on the normal and distance between the cloth point and the skin surface. Our method handles efficient collision detection between dynamically deforming polygonal meshes since it achieves a reduction of about 98% in the number of triangles to test against for each cloth vertex. The algorithm proposed has cost that is independent of the model complexity.

Chapter 2 provides a brief overview of the existing research on collision detection techniques and human and cloth model. We begin by describing the current methods for collision detection. Secondly we outline the different types of bounding volumes, the evolution of representation and deformation of human bodies and the cloth model techniques currently being used. In the last part of chapter 2 we present the existing environment used as a starting point for the implementation of this project.

Chapter 3 presents our collision detection algorithm and the optimisations that we have applied, we also explain how we tackle the simple collision response problem. Chapter 4 describes the implementation details.

Chapter 5 presents the tests carried out using our algorithm and chapter 6 shows the final results obtained for a deformable human body

Finally we give the conclusions of our algorithm and the future work to be done.

2 BACKGROUND AND LITERATURE REVIEW

2.1 Collision Detection Techniques

Since we want to perform collision detection for real-time simulation, we need to take advantage of the current graphics acceleration hardware. For this purpose we need to work with triangle meshes of vertices.

Therefore we are going to compute collision detection either between a vertex of one object and a triangle of the other object ("point-triangle" collision) or between an edge of an abject and an edge of the other object ("edge-edge" collision) [PRO99]

Knowing the positions and velocities of each node of our model at an instant t_0, it is possible to compute its position at a time $t_0 + \Delta t$, assuming that each node moves at a constant velocity.

In order to detect collision detection between a moving point and a moving triangle or between two moving edges we need first of all to compute the time at which the four points are coplanar [PRO99,BRI02].

2.1.1 "Point-triangle" collision

Considering the moving point P(t) and a moving triangle with vertices A(t), B(t), C(t) with velocities $\vec{V}$, $\vec{V}_A$, $\vec{V}_B$, $\vec{V}_C$ respectively during the time interval $[t_0, t_0 + \Delta t]$. Each of the vertices of the triangle will have a linear movement given by the equations:

$$A(t) = A(t_0) + t\vec{V}_A \ , \ B(t) = B(t_0) + t\vec{V}_B \ , \ C(t) = C(t_0) + t\vec{V}_C$$

In case of collision, the point P(t) will be coplanar with the plane defined by the three vertices of the triangle ABC(t). This can be written by the following relation:

$$\exists t \in [t_0, t_0 + \Delta t] \text{ such that } \exists u, v \in [0,1], u + v \leq 1, \overrightarrow{AP}(t) = u\overrightarrow{AB}(t) + v\overrightarrow{AC}(t) \ (1)$$

In order to solve this non-linear vector equation we can use another condition that will also be satisfied in the moment of collision:

$$\overrightarrow{AP}(t) \cdot \overrightarrow{N}(t) = 0 \qquad (2)$$

Where $\overrightarrow{N}(t)$ is the normal of the plane defined by the triangle ABC(t) obtained by the cross product: $\overrightarrow{N}(t) = \overrightarrow{AB}(t) \times \overrightarrow{AC}(t)$

$\overrightarrow{N}(t)$ is a t^2 term, $\overrightarrow{AP}(t)$ is a t term, thus the dot product is a third degree equation that can be solved easily. By solving this dot product, we can obtain three values of t. We are only interested in those values included in the time interval [t_0, t_0 + Δt] and in case of having several values we will consider the smallest one because is the first collision produced between the point and the triangle. Once we have obtained the values of t we can inject them in equation (1) in order to obtain the values of u and v.

2.1.2 "Edge-edge" collision

There will be a collision between the edges $\overrightarrow{AB}(t)$ and $\overrightarrow{CD}(t)$ if and only if:

$$\exists t \in [t_0, t_0 + \Delta t] \text{ such that } \exists u, v \in [0,1], \quad u\overrightarrow{AB}(t) = v\overrightarrow{CD}(t) \qquad (3)$$

This is also a non linear system, but as we have seen for the point-triangle collision we can use another relation in order to obtain the value of t without solving the equation (3). At the moment of collision, the four points that define the two edges will lie in a same plane, which can be defined with the following equation:

$$(\overrightarrow{AB}(t) \times \overrightarrow{CD}(t)) \cdot \overrightarrow{AC}(t) = 0 \qquad (4)$$

This is a third degree equation. Any roots outside the interval [t_0, t_0 + Δt] are discarded, and then the remaining roots are checked in increasing order. Once we have the value of t we can obtain the values of u and v by injecting t in equation (3).

2.2 Space Subdivision

Collisions are a major bottleneck in cloth simulation. The collision detection between an object of the scene, such as the skin, with N nodes and a cloth model with M nodes has a O(NM) complexity. As the number of vertices of the model increases, this complexity becomes very limitative, and so it is important to find some techniques that can minimise the computation time. These techniques will imply a trade off between speed and accuracy

In order to reduce the complexity of the collision detection we can use a space subdivision that allow as to compute collision test only between those vertices that are potential colliders.

This first level of optimisation is called minimising collision detection test. The most common method is the use of bounding volumes for early elimination of collision test. In this method every object of the scene is enclosed in a bounding volume, then a preliminary collision detection is carried out between bounding volumes, and in case of intersection a more detailed collision detection is computed between the objects within those bounding volumes.

Bounding volumes technique is very effective when objects are simple, but for complex objects it has the disadvantage that it may encompass too much empty space and therefore it losses its effectiveness. Bandi and Thalmann proposed a technique for oc-tree based compact bounding volumes which is not computationally expensive [BAN95].

For the case of cloth-skin collision detection we need to generate a space subdivision structure for the cloth and another one for the skin.

There are different space subdivision structures such as bounding volumes hierarchies and BSP trees. There are efficient algorithms [COH95, PON95] that rely on hierarchical representations of the scene to quickly identify pairs of intersecting objects and capitalise on temporal and spatial coherence between successive frames of simulation.

2.2.1 Bounding Volumes

A bounding volume is a geometric primitive that contains a group of objects, a complex object, or a group of primitives. The application of bounding volumes in computer graphics applications is given in [MOO 88].

There are different kinds of bounding volumes. In choosing the appropriate bounding volume we should consider several criteria:

- Their *geometrical or filling efficiency*, which means how well they contain the objects within minimum empty space .
- *Compactness*, that is how many values are required for describing them.
- *Dependency on the environment,* this refers to if there are any kind of axis-dependency, scaling, etc.
- *Collision detection simplicity*, it means how efficiently the collisions can be detected between two volumes or between a volume and another primitive.
- *Building simplicity,* that refers to how efficiently they can be built from a set of objects or by merging several other bounding volumes.

The most widely used bounding volumes are oriented bounding boxes and bounding spheres, but there are also some other techniques.

Axis Aligned Bounding Boxes (AABB)

In a 3D space an AABB is defined by six values giving the minimal and maximal axis coordinates of the box including the object. The axes of the box are aligned with the workspace axes.

The main advantage of this method is that they are very easy to build, to merge and to detect collision. Building an AABB involves only finding the minimal and maximal coordinates of the object that we want to bound. Intersection test are reduced to detect overlap along the axes.

AABB have fast processing but their main disadvantage is that their filling is not very good particularly when the object are elongated in diagonal directions.

Bounding Spheres

A bounding sphere need four values to be defined, which are: its radius and the coordinates of its centre.

Bounding spheres have the advantage of being non axis-dependent and therefore can be transformed by a rigid-body motion with no need of being recomputed. Collision detection test can be easily performed by comparing distances between centres of two spheres with the sum of their radius.

The disadvantages of this technique are that they are difficult to build and merge, because the computation of the optimal centre needs non-linear geometric calculations.

Discrete Orientation polytopes (k-dops)

Another efficient volume is the Discrete Orientation Polytopes (k-dops) which are convex volumes defined by truncate planes that can have normals which comes from a fixed small set of k orientations (k-dops)

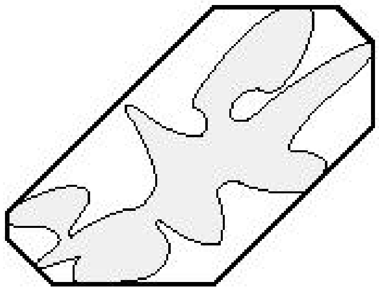


Figure 1. 8-DOPs example

Construction of a k-dop begins by selecting k directions that cover the entire direction-space as evenly as possible. A 6-dop represents a box. After the k directions have been chosen and therefore the planes that define the volume, vertices are projected onto the direction vectors and the maximal projection on each direction is chosen. This k maximal distances represent the k-dop [KLO97].

Merging and detecting collision between k-dops of the same nature are simple, but become more costly as the orientation number k increases.

Other bounding volumes

Using more parameters for the description of the bounding volumes, it is possible to allow a wider range of shapes in optimising their filling efficiency so that their collision becomes more representative of the collision of the contained objects.

The choice depends directly on the shape of the object to be bounded. In the case of elongated objects, a possible solution are ellipsoids and cylinders.

Transformed Volumes

In this case we have a hierarchical structure where each group can be moved and oriented by changing a geometrical transformation matrix. Each bounding volume can be defined in local coordinates so that it is not necessary to recompute its shape. This can lead to great economies in computational time while moving the elements in the scene and also allows to optimise the volume of an object. An example of this are the Oriented Bounding Boxes [GOT96] that we will see in more detail in the next section

2.2.2 Bounding volumes Hierarchies

The bounding volumes can be axis aligned or have an arbitrary orientation. Depending on the object that we are interested in creating a space subdivision we can use a different kind of bounding volumes.

For the cloth model a simple optimisation can be obtained by recursive partitioning of the triangles of the cloth mesh depending on their position in the 2D texture space [PRO99].

For the skin it is possible to use a hierarchical representation of oriented bounding boxes (OBBs). An OBB is a rectangular bounding box oriented in an arbitrary axis in the 3D space [LIN96]. This space subdivision representation is called OBBTree. The fact of using an arbitrary orientation instead of an axis aligned orientation allows us to obtain a tight-fitting OBBs.

We want to have OBBs of each part of the body, and once we have the bounding box of each part we can obtain the OBBTree.

In order to obtain the bounding box of a set of vertices that correspond to a particular part of the skin of the body we can compute its covariance matrix and then obtain its eigenvectors. Since the covariance matrix is symmetric, those eigenvectors are going to be orthogonal. Once we normalise them we will obtain the basis of the bounding box, and afterwards we need to find the extremal vertices along each axis of this basis to obtain the size of the bounding box.

Another method consists in using the convex hull of the vertices of the triangles. The convex hull is the smallest convex set that contains all the points and there are some algorithms of O(n lg n) complexity that are available as public domain packages [BAR93].

Given an algorithm to compute OBB around a group of vertices we need a hierarchical representation of the whole object. There are two categories of methods for constructing a hierarchy: top-down and bottom-up. Top-down methods begin with an OBB that includes the whole object and recursively subdivide until all the leafs nodes are indivisible. Bottom-up methods begin by obtaining the OBB of each polygon and from there recursively merge them until it obtains the OBB that includes the whole object.

2.2.4 Binary Space Partitioning Trees (BSP trees)

A BPS tree is a hierarchical subdivision of n-dimensional space into homogeneous regions, using n-1 dimensional hyperplanes [CHR95]. Those hyperplanes can be defined by the expression:

$$f_n = a_1 x_1 + a_2 x_2 + ... + a_n x_n + a_{n+1}$$

All those points where f_n is greater than 0 are situated in front of the hyperplane, the set of points where f_n is smaller than 0 are at the back of the hyperplane and the point where f_n equals 0 lie on that hyperplane.

A BSP tree has a binary structure where each node corresponds to a region of the space and contains a hyperplane that partitions that region into front and back subregions.

The nodes of the tree that do not contain any hyperplane are known as the leaf nodes and correspond to a non partitioned region of space which is called a cell.

There is different ways of creating a BSPTree for a dynamic scene. We can either add one by one new polygons into the tree and so restructure the hierarchy or we can also merge a BSPTree corresponding to a particular new object with the BSPTree of the scene that we have already created.

2.3 Fast overlap test

Given a space subdivision defined by a hierarchical bounding boxes representation, we can test pairs of bounding boxes to find out which parts of the skin and the cloth may collide and so perform collision detection only between the polygons included in those bounding boxes. There are several simple algorithms for testing overlap between bounding boxes, but they are usually computationally expensive.

It is possible to use a fast overlap test [LIN96] which uses a trivial test for overlapping. The main idea of this algorithm is to project the bounding boxes onto some axis is space. This axis is called “axial projection”. Under this projection each box will be projected onto a segment in that axis. If the segments corresponding to each bounding box do not overlap then the axis is called a "separating axis" for the boxes which are therefore disjoint. The fact that there is an overlap between the segments does not necessary mean that there exist an overlap between the boxes, further test are required.

For the performance of the fast overlap test we can use the "separating axis theorem" described in [GOT96] for testing overlap. This method establishes that two boxes in 3D are disjoint if there exist a separating axis orthogonal from an edge to each box or to a face of either box. For two boxes there can be only 15 possible axes that may be a separating axis.

2.4 Models

The human body can be briefly defined as a composition of skeleton, muscles, fat and skin. The skeleton is formed by bones attached with each other by joints. The skeleton hierarchy is therefore composed of articulated line segments whose movements are described at the joint level. Each joint has associated up to three Degrees of Freedom DOF which correspond to the three possible axis of rotation. Each limb position is defined locally in relation of its node parent in the hierarchy. The muscles are usually modelled by metaballs and the skin by a triangle mesh.

In order to perform the motion of the skeleton it is necessary to assign values to each of the degrees of freedom of the joint. Currently there are three ways to perform human motion. The first way consists in assigning interactively values to each join. The second way is to read and execute the motion form an animation file (key framing tables). The third way is by using tracking devices to perform motion capture.

The other object that we need for the collision detection is the cloth. Cloth pieces are designed in two dimensions with polygonal panels and then are seamed and attached to the avatar's body in three dimensi ons [CAR92]. Then according to the seams, forces are applied to the edges of the panels to put them together. Once the clothes have been created, physical properties are simulated and clothes are animated depending on the avatar's movements in a physical environment.

2.4.1 Cloth

The most generalised model for the representation of a cloth model consists of a deformable surface composed of a network of masses and springs [7,8], which movement is evaluated using the numerical integration of the fundamental law of dynamics:

$$F_{i,j} = \mu a_{i,j}$$

Where μ is the mass of each point $P_{i,j}$ and $a_{i,j}$ is the acceleration created by the force $F_{i,j}$.

In the elastic model described by Xavier Provot [PRO95, LOU95] a mesh of *m*x*n* virtual massed is used. Each mass is linked to its neighbours by massless springs of length at rest non equal to zero. There are three kinds of links between masses in order to give different properties to the cloth object:

- "Structural springs": defined by springs between masses [i,j] and [i+1,j], or between masses [i,j] and [i,j+1]
- "Shear springs": defined by springs linking masses [i,j] and [i+1,j+1], or masses [i,j+1] and [i+1,j]
- "Flexion springs": defined by springs between masses [i,j] and [i+2,j], or masses [i,j] and [i,j+2]

Shear springs provide a shear rigidity to the cloth and prevent it from excessive and unrealistic distortion in its own plane. Flexion springs are under stress when flexion occurs in the cloth object and thus have a three-dimensional role.

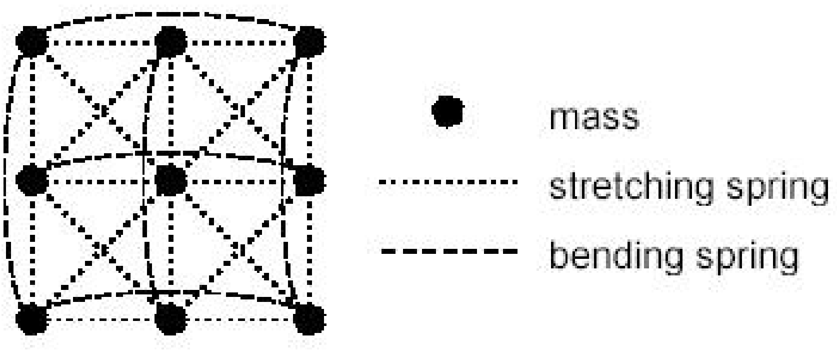


Figure 2. mass-spring cloth model

The forces that affect the movement of the cloth can be divided between the internal and the external forces.

The internal forces result from the tension of the springs linking $P_{i,j}$ to its neighbours.

The external forces are of various natures depending on the types of loads we wish the model to be exposed to. The most common loads are gravity, a viscous damping and a viscous interaction with an air stream or wind. The role of this damping is to model an approximation of the dissipation of the mechanical energy of the cloth model.

All this forces are integrated using a simple Euler method in order to obtain the total force applied to each mass of the cloth. Once computed the total force it is possible to obtain the movement of a mass using the fundamental law of dynamics.

The model described previously uses a regular quadrilateral mesh, but it is also possible to use a model described by an irregular triangle mesh, the dynamics of which are computed by considering the effects of Newtonian mechanics (internal and external forces) acting on their discrete elements [AND98]. The model described by Anderson is considered to be a mass-particle system positioned at the corners of triangular discretizations which are attached to adjoined edges. There are six kinds of forces acting over the mass points which are gravitational force, tensional forces, flexional forces, frictional forces, air resistance and wind effects.

This model is based in the method employed by the MIRALab team at the University of Geneva, Vollino, Courchesne and M. Thalmann [VOL95] which consists in a physical model based on Newtonian mechanics acting on irregular triangle meshes. The forces acting on the edges of each triangle are obtained by calculating during each time-step both the elastic and shearing strains on individual triangles and the bending strains between adjacent triangles. Then the overall change in dynamics is estimated over each time-step using the second-order Euler-Cromer numerical method (Kreyszig, 1998)

2.4.2 Skin

The skin is usually modelled as an elastic triangle-mesh surface generated typically with a modelling program. This skin is then attached to the underlying bone hierarchy of the avatar and in some more complicated avatars is also attached to the underlying muscle model [SCH98].

Changes in the joints of the skeleton result in changes to the position of the bones and therefore in the skin attached to those bones. The vertices of the attached skin move with their underlying components (bones and muscles) resulting in natural-looking deformation to the avatar's model.

The surface model that represents the skin can be either a triangular mesh or a set of surface patches. Deformation of the skin is implemented by using specialised procedures that deform the surface as a function of each joint angle. Similar techniques are used in some of the commercial animation systems available.

Thalmann [AUB00] introduce joint-dependent local deformation operators named JLDs to be able to control deformations in the vicinity of joints. This technique uses a surface mesh and assigns each vertex to a specific joint in the skeleton hierarchy.

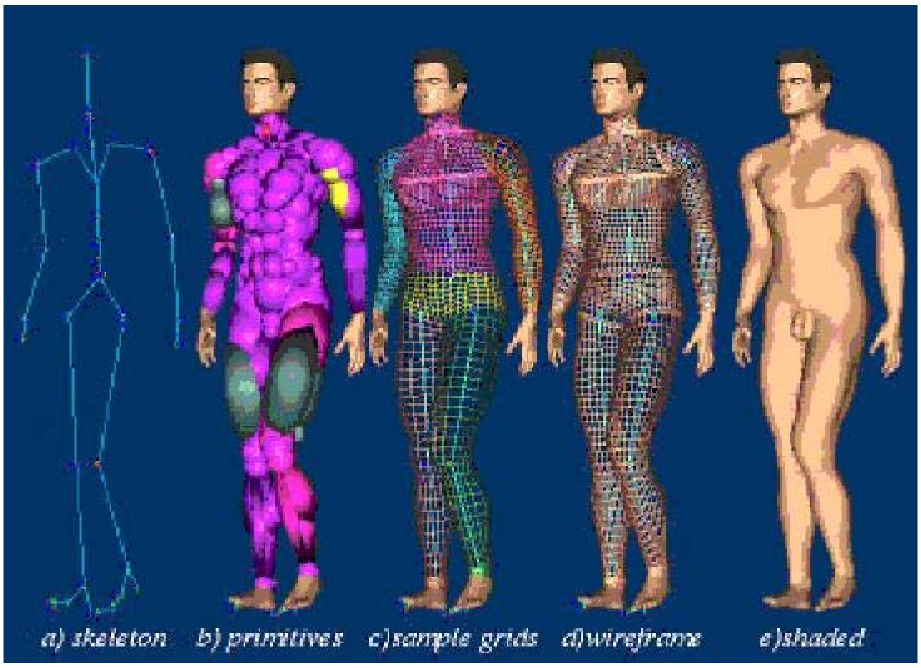


Figure 3. Layered Avatar Construction. Virtual Reality Lab.

It is also possible to group the vertices into *contours* [KAL98] and deform all of them together in order to speed up the deformation process. For the deformation of those contours it is only necessary to compute for each frame the orientation and position of each contour.

Some other techniques with higher computational cost are based in the use of Free Form Deformations (FFDs). In each time step the FFD control points are moved using kinematics or dynamics. Usually those control points deform at joints under several constraints such as elasticity, area preservation and implicit repulsive force fields that mimic bones and muscles [HEN90]. It is also possible to use Rational FFDs, Dirichlet FFD which allow a more accurate control of the deformations.

Another approach is to define the skin by implicit formulations like meatballs. This method provides an efficient way of creating real-looking virtual humans at a reduced storage cost [YOS92].

2.5 Prometheus Avatar Toolkit

The Prometheus project is a consortium which is involved with developing a studio broadcast platform for virtual 3D TV. The aim of the Prometheus project is to allow television production using virtual sets and characters, which are driven by real actors using a motion capture system developed by the University of Surrey. Then there is cloth simulation carried out by UCL and the data is sent out to a central server which integrates all the data from various parts of the system streams it to set top boxes using MPEG4. The system contains also a facial motion capture system [PRO_BBC].

The parts of the project with which we are working in the collision detection project are the software developed by the UCL [PRO_UCL] and the software developed by the University of Surrey.

The software from the UCL contains:

- Rendering engine based on VRML97/H-Anim
- Cloth simulation system using a method based on finite elements
- Voxel collision detection (cloth/avatar) for segmented avatars without skin deformations.

- Animation system which can handle lots of simultaneous motion capture data from files or streamed
- Server architecture and test harness of several temporary servers for motion capture streaming, cloth simulation and playback.

Human animation of 3D models on the internet is currently done in VRML format. In the Prometheus Avatar Toolkint the avatars are defined with the H-Anim 1.1 standard for VRML. The H-Anim specification [H-Anim1.1] defines a method for representing articulated humanoids characters in VRML.

2.5.1 VRML humanoid models

The H-Anim 1.1 specification defines a standard representation for avatars in VRML. This standard defines a set of VRML nodes implemented using the VRML97 PROTO mechanism.

A *Humanoid* node defines the tope level of the humanoid model, which contains a hierarchy of *Joint* nodes describing the skeleton of the avatar. Except for the HumanoidRoot joint, all the joints are children of a higher level joint, and each joint has associated a centre of rotation and a motion limit information to avoid impossible human posses. Joints are only allowed to rotate, not translate, so the skeleton keep the same basic topology and shape as the skeleton moves. Each joint rotation will also rotate in the same way all the children in the same way.

In order to define the shape of the body parts we need another kind of nodes, the *Segment* nodes. Each segment node will have as a parent node a join node and therefore the geometry defined in the segment node will rotate in the same say as its parent joint node.

A standard H-Anim model is thus made up from a number of separate models each attached to a different part of the articulation hierarchy.

2.5.2 Seamless Models

The avatars we are dealing with in this project are seamless models. In order to define these models, a number of few elements have been added to the H-Anim structure defined in the previous section. [SMI00]

The model is defined as a single surface instead of a number of separate segments, so that it gives the illusion of a single surface. This is achieved by using the same articulation structure but only attaching parts of a complete human model instead of attaching the whole rigid objects to each body part. Then, while the skeleton is being animated, each segment affects particular parts the model in order to change the complete shape of the body. The segments store the association between the model and the skeleton, allowing deformation of the surface.

2.5.3 Skin Deformations

The vertices representing the 3D geometry of the model will be deformed based on the rotations and translations that are defined by the skeleton structure. In the Prometheus Avatar Toolkit there three methods have been used to achieve deformations of the avatar's surface. These methods are: rigid animation, simple segment deformations and vertex weighing.

Rigid Animation

In this method, the shape of individual segments is not deformed because the vertices are subjected to rigid transformations [SMI00]. This method for seamless model animation is very fast and can be therefore use for slow computers or large models.

In each frame, the rotation information of each bone is read, and the skeleton is recursively traversed to obtain the complete transformation at each joint and then apply it to their joints children and also to their child segment, which contain the geometry information. In order to calculate the position of the vertices for each frame, it is necessary to store the position of all the vertices in the original model. Since the vertices positions are modified, the body shape will change for each frame of the animation.

Non-rigid Animation

The previous model is fast but unrealistic. It may give problems such as self-intersect on the inside of a joint or triangles being stretched in the outside of a joint. Therefore a more realistic method is used to deform the surface of the body [SMI00].

The method used in the Prometheus Avatar is based in an implementation of the control layered animation method described by Sun et al. [SUN99] for layered model animation.

The method used to deform the body shape based on the skeleton hierarchy parameterises each point in terms of the segment to which it is assigned. Each joint has associated a joint plane that is defined as the plane that bisects the angle of the joint and is orthogonal to the plane defined by adjacent joints. This joint planes will move with the skeleton, always bisecting the joint angle. Each segment will be thus bounded by at least two joint planes. Vertices can be parameterised in terms of a local coordinate system based upon the joint planes. Once a vertex is assigned to a particular segment, it is parameterised in local coordinates of that segment.

As the skeleton layer is animated, the joints will change its local and global position, and so the joint planes will also change. Since the parameterisation of the space between planes is based on those joint planes, and the space is being deformed, therefore the position of the vertices which occupy that space will also change.

The calculations needed in this method are all linear interpolation, hence the reconstruction is highly efficient and can be performed in real time.

Vertex Weighing

In this method , a single vertex can be attached to more than one bone and weights are applied to define the deformations.

This method expresses the deformation of a surface as a function of either the pose of an underlying skeleton, or as a function of some other set or parameters control desirable in face animation such as {raise-eyebrown, smile,...}

The main idea of this deformation approach comes from the observation that several types of deformation can be uniformly represented as mappings from a pose space. These previously disparate deformation can be accomplished within a single unified approach using interpolation [LEW00].

3 ANALYSIS AND DESIGN

The software described in this thesis was not created from scratch. A state-of-the-art anatomically based modelling created by the University of Surrey and a cloth model created by Lee Bull, Research Fellow at UCL, already existed.

Our goal for this project was to detect collision detection between a pre-existing polygonal mesh skin and a cloth model described by an irregular triangle mesh. For this purpose we needed to integrate both systems.

The remainder of this thesis is organised as follows: first of all we review the existing cloth and skin modelling environment, then we discuss the tools we have added to it in order to perform collision detection, next we demonstrate the results that have been obtained and finally outline the future work that still needs to be done.

3.1 Overview of existing environment

As we have previously discussed, for the implementation of this project we have deal with two different systems. On one hand the Prometheus Avatar Toolkit developed by the university of Surrey and on the other hand the cloth model developed by the UCL, in this section we will introduce those systems in more detail.

The primary development platform is Windows but there are also Unix and CAVE versions. The software is written in C++ and uses OpenGL. The project its been implemented using Visual C++

Prometheus Avatar Toolkit

Skin deformations model

The articulated body is represented by a tree structure that represents the parent-child relationship of body segments. Each segment of the body corresponds to a bone and it is thus a section of the body between two successive joints or at the end of a tree branch. Every segment of the body has a default rotation and translation that gives its position in relation with its parent position. The complete model is a layered model given by an skeleton and the skin. Each avatar contains a polygonal mesh that describes the skin, but also each bone of the skeleton has references to those vertices that are attached to it.

For each frame step the position of those vertices attached to a bone that has done a movement is updated in order to obtain natural surface deformations during the animation of the avatar.

In the following figure we can see the main classes of this skinned avatar system:

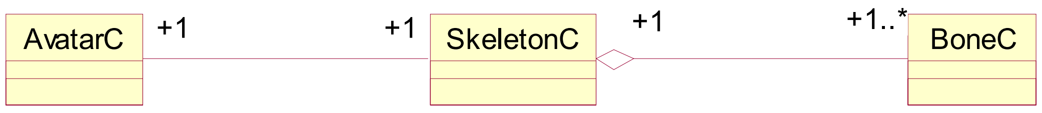


Figure 4. Classes Diagram of avatar

Each avatar will contain a skeleton and that skeleton will be a hierarchy of bones that will change their positions during animation.

Collision Model for the Cloth

The collision and cloth model with which we are dealing has basically the following structure:

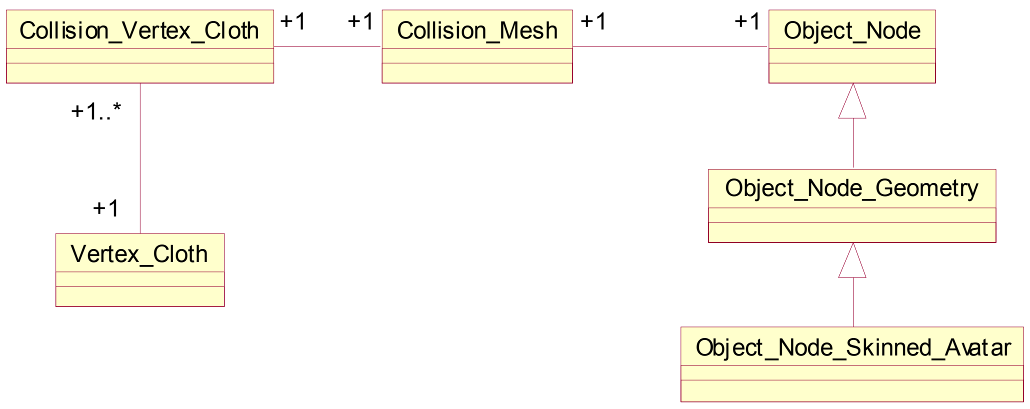


Figure 5 Classes Diagram of Cloth Collision

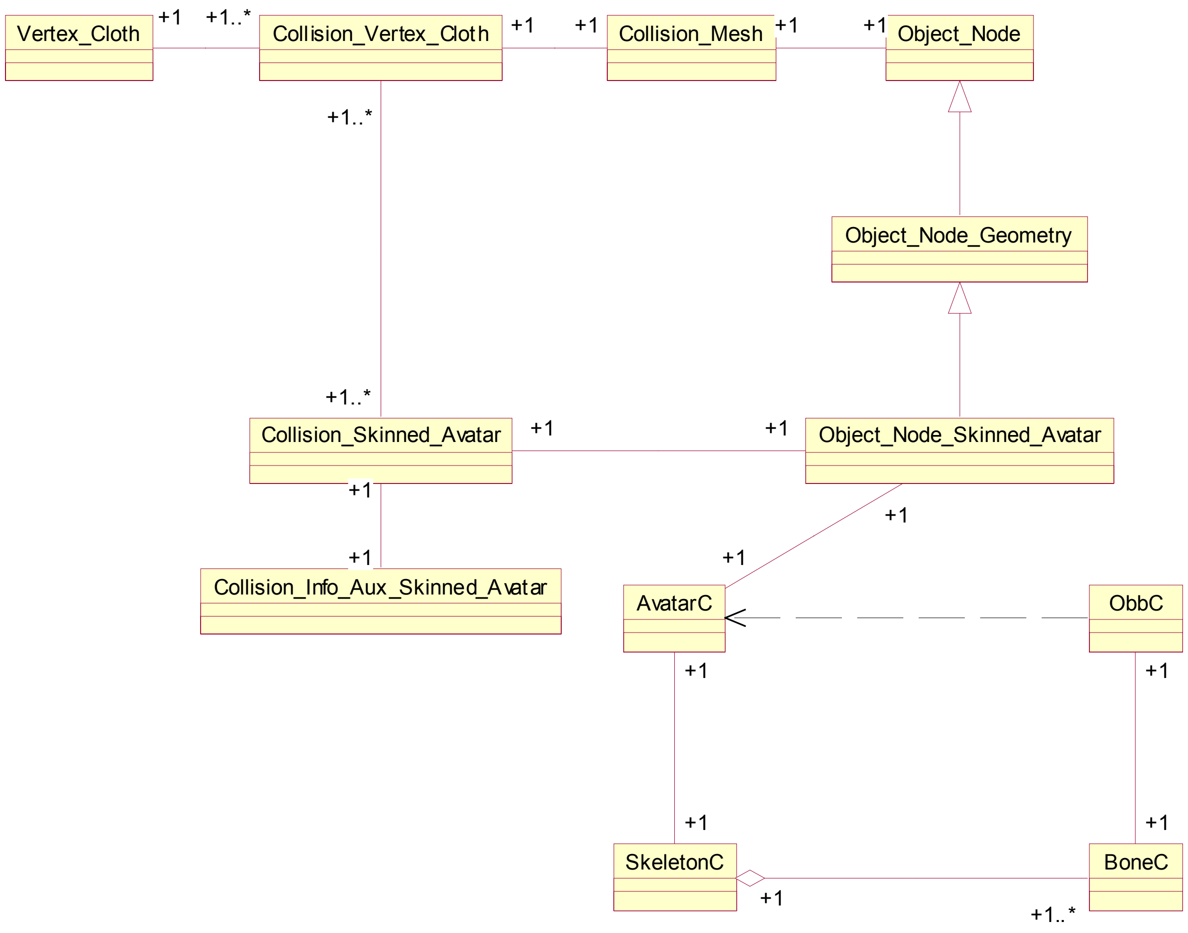
This are the main classes with which are used for the performance of the collision detection test.

Each vertex of the cloth is considered a potential collider that has to be tested against a collision mesh. The class Collision_Vertex_Cloth contains information relevant for each time step collision.

In the original system each vertex cloth can collide with several collision meshes and for each one we need to store some information related with the collision such as position of the vertex before collision, position of collision, acceleration, velocity, feedback force to be applied to the vertex, etc.

OBB Model

In the following diagram we can observe all the classes involved in the collision detection algorithm:



In order to perform the collision detection test, three classes have been added:

Object_Node_Skinned_Avatar: This class contains all the information regarding an object of the type avatar, such as its geometry, global transformations, etc and it is the responsible of the initialisation of the avatar object.

Collision_Skinned_Avatar: This class handles all the collision details concerning the avatar object.

Collision_Info_Aux_Skinned_Avatar: This is an auxiliary class where we can store temporal information for each collision test between a triangle of the skin and a cloth vertex.

3.2 Space subdivision of the body

In order to generate naturalistic cloth animation we need to perform collision detection between cloth and skin of the avatar. Ordinary collision detection methods find intersection between every vertex of the cloth and every vertex of the skin. However this method is impractical when we are dealing with real-time simulations which is our case. It is desirable to apply an efficient detection method whose complexity is independent of the complexity of objects. In this section we propose an efficient method for collision detection using a space subdivision representation for the mesh of points that correspond to the skin based on Oriented Bounding Boxes (OBB).

3.2.1 OBB's vs. Other volumes

There are several methods for space subdivision based on spheres, ellipsoids and axis-aligned bounding boxes (AABB's). The choice of bounding volume is governed by two conflicting constraints [GOT96]:

1.- It should fit the original models as tightly as possible

2.- Testing with such volumes for intersection should be as fast as possible

Simple primitives like AABBs and spheres do very well with respect to the second constraint. But they cannot fit tightly some primitives like long-thin oriented models, which is mainly the kind of object that we are going to find in a human body model.

OBBs and minimal ellipsoids provide tighter fits, but checking for intersection test with them is relatively expensive.

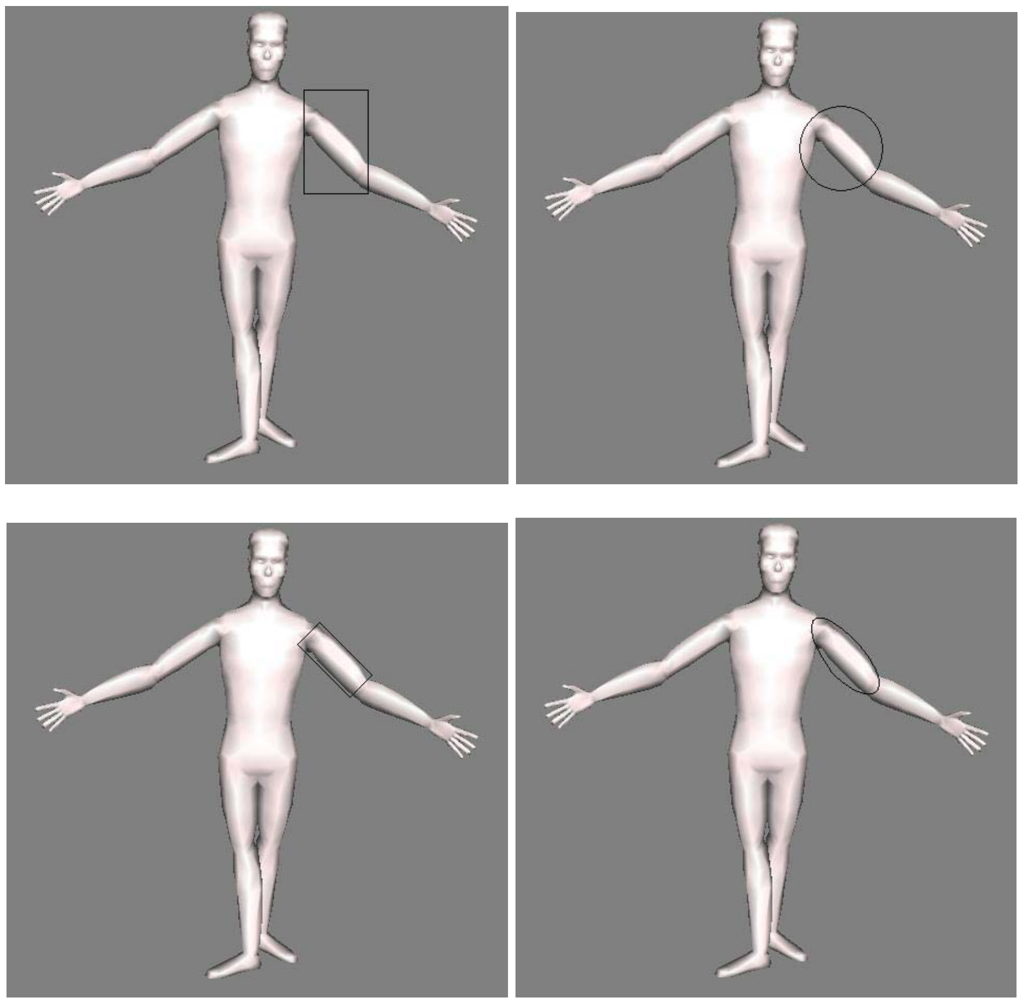


Figure 6. Different kinds of bounding volumes

The primary motivation for choosing OBBs is that not only do they have the advantage of their variable orientation but also they can bound geometry more tightly that AABB or Spheres so we can avoid more potential colliders for the posterior collision detection test between vertex and triangles.

It is difficult to do a general analysis of the performance of collision detection algorithm because performance is situation specific. For our situation it is evident the advantage of OBB vs. AABB and spheres. For the analysis of OBB vs. ellipsoids the main reason why we have chosen OBB is because the intersection test between vertices and OBB has better computational time than intersection between vertices and ellipsoids.

3.2.1 Oriented bounding Boxes

In the first step of our algorithm we create an oriented bounding boxes representation of the human body. Since the human body is very irregular and cannot be described as an unique OBB we need thus to have a subdivision of the body into several parts, and then describe each part with an OBB.

The body we are dealing with is already subdivided in different segments corresponding to the bones of the human skeleton. Each bone has attached a list of vertices that will change their position with each movement of the bone.

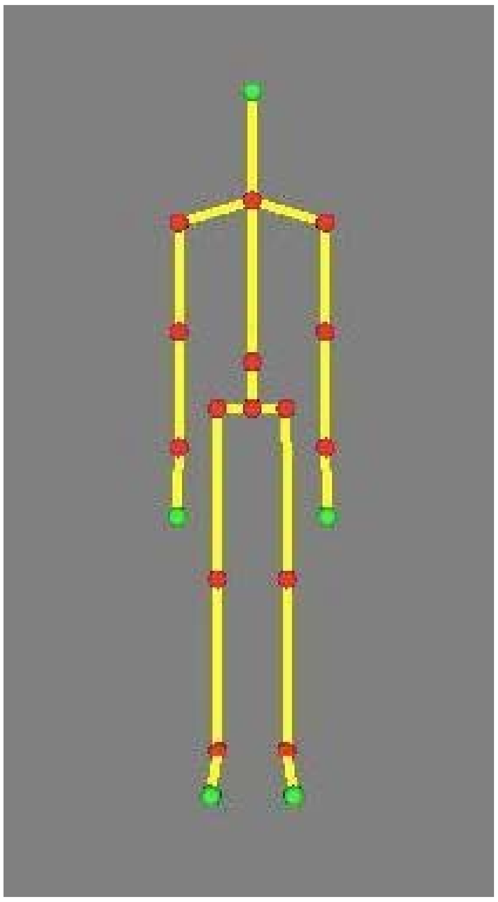


Figure 7. Example of skeleton

An OBB is defined by a centre $\vec{C}$, a set of right-handed orthogonal axes $\vec{A}_0$, $\vec{A}_1$, and $\vec{A}_2$, and a set of extends $a_0 > 0$, $a_1 > 0$, and $a_2 > 0$. As a solid box, the OBB is represented by:

$$\left\{ \vec{C} + \sum_{i=0}^{2} x_i \vec{A}_i : |x_i| \le |a_i| \ \textit{for all } i \right\} \text{a}$$

and the eight vertices of the box are

$$\vec{C} + \sum_{i=0}^{2} a_i \vec{A}_i$$

In order to obtain tight fitting OBB we need to compute the three orthogonal axes that define its orientation and also compute the position and dimension of the OBB. This method is called "Principal Component Analysis", PCA.

The covariance matrix S is calculated by:

$$S = \frac{1}{n} \sum_{i=0}^{n} \left(X_i - \overline{X} \right)^t \left(X_i - \overline{X} \right)$$

Each X_i corresponds to a vertex of the skin attached to the bone we are working with, and it is defined by three variables that are its global coordinates (x_i, y_i, z_i).

Once we have computed the covariance matrix we will calculate its three eigenvectors which are mutually orthogonal. After normalisation those three eigenvectors become axes of the OBB.

Finally we need to find the maximum and minimum extends of the original set of vertices along each axis, and thus the size the OBB.

3.2.2 Subdivision of boxes

With the previous OBB representation we obtain boxes with a certain amount of vertices inside. This allow as to perform an intersection test between the OBB and the cloth to find out if the vertices of the skin within an OBB are potential colliders with the cloth. This method avoids performing collision detection tests with every vertex of the skin, but we can make it also more efficient by subdividing each OBB in smaller boxes.

For this purpose we decided to divide the OBB in a certain amount of boxes along the main axis of the OBB. The main axis is thus divided in regular segments and then the vertices attached to the bone are separated in sub-lists given by the space subdivision obtained by planes orthogonal to the main axis and located in each one of the subdivisions along the axis.

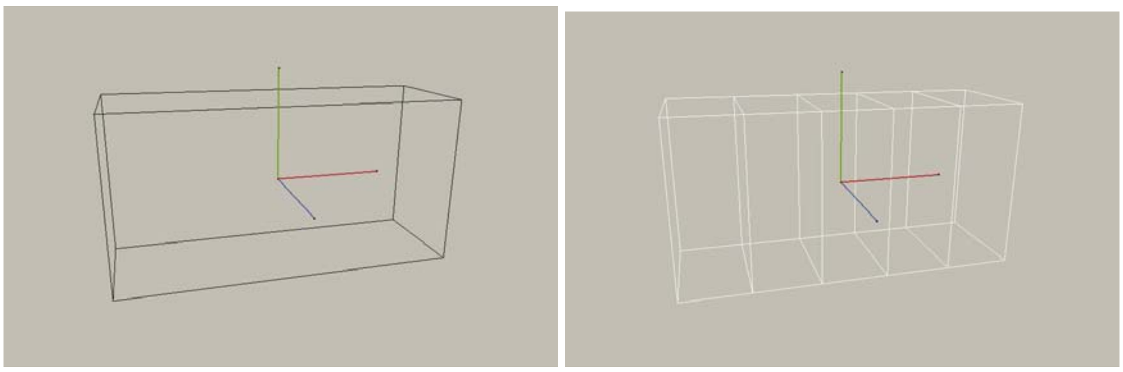


Figure 8. Original OBB and its correspondent subdivision along the main axis

3.2.3 OBB updates during animation

In section 3.2.1 we have introduced how the OBB are obtained from the list of vertices attached to each bone. The method used to obtain the axis of the OBB from the covariance matrix has cost O(N) where N is the number of vertices. Since we are dealing with real time animation we cannot afford to compute the OBB for each step of the animation so we need to find a method to modify the OBB during animation with a computational cost independent of the complexity of the model we are working with.

As we have briefly explained in section 1.3, the avatars are represented by a hierarchy of nodes. Each node in the graph represents an object. Nodes carry two types of information:

- Joint data
    - Rotation
- Segment data
    - Translation
    - Graphical data

The rotation and translation together give the transformation matrix which specifies how the object at that node is related to its parent node. The rotation refers to the joint rotation and will depend on the degrees of freedom of the joint and the translation is the position of the segment relative to its parent, that is the length of the bone.

Each node of the graph has a local coordinate system (LC) which is independent of the rest of the nodes in the hierarchy. The matrix associated with the root can be thought of as transforming the root object into world coordinates (WC)

In each bone we have stored its global transformation matrix and its local rotation and translation. In order to update the skeleton we only worry about the rotation, since the translation wont change during the animation.

Every time the rotation matrices are updated we need to update also the transformation matrices. The global transformation matrix (GTM) of each bone is equal to the GTM of its parent multiplied by its local transformation matrix (LTM), so in order to update all the GTM we need to traverse the tree starting from the root and multiplying all the LTM.

Once we have this transformations updated it is obvious that we can use them to update also the axis of the OBB so that the OBB will be transformed as the skeleton is animated. This method needs only a multiplication of each axis by the GTM of the bone, so it does not depend on the complexity of the model.

Having the OBB reoriented we would now need to compute its size since some of the vertices may have change its position relative to the bone. The method explained in section 3.2.1 needs also to test all the vertices associated with the bone, so again the cost is O(N). In order to avoid this we can allow some error in the size of the OBB by having the OBB slightly bigger than its correct size, so we do not need to compute its size for each step of the animation.

By applying this method we may need to do some additional collision detection tests afterwards, but we avoid having to compute the size of the OBB each time the skeleton is animated which will save computational time.

3.4 Cloth Collision

Collision detection is in our case to determine whether a vertex of the cloth is inside of the skin. A correct test must consider line segments which will be the vertices cloth trajectories between time steps t_{i-1} and t_i against faces of the skin in time step t_i. But in order to reduce the number of collision detection tests we previously perform some easier test where the elements colliding are vertices against OBB.

3.4.1 Detection vertex-OBB

A OBB is defined, as we have seen before, by a centre and three orthonormal axis that can be considered as a coordinate system. The OBB is bounded by 6 planes, so the first idea in order to detect if a vertex is inside a OBB would be to compare the vertex coordinate against these 6 planes.

There is a better way to solve this problem. Assuming that the vertices coordinates are given in global coordinates, if we apply a mapping from the global coordinate system to the OBB coordinate system then we just need to compare the coordinates of the vertex against the right top corner of the OBB.

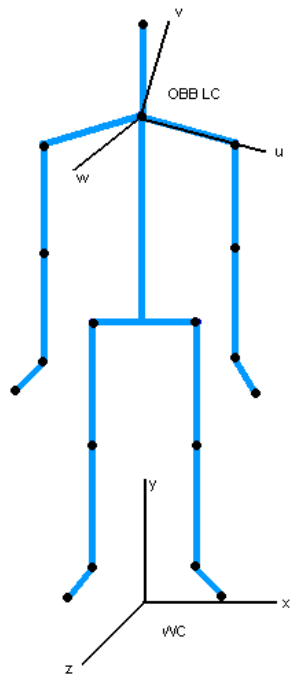


Figure 9. Coordinates transformation mapping

Considering the following skeleton where we have the world coordinates system (WCS) at the bottom of the skeleton and the local coordinate system (LCS) of the OBB associated with the right shoulder.

We need to obtain the transformation matrix which maps WC into LC of the OBB:

$$M = \begin{bmatrix} R & 0 \\ t & 1 \end{bmatrix}$$

The vectors u, v and w of the LCS must rotate under transformation matrix into the unit principal vectors i, j, k of VC [SLA02], therefore:

$$uR = i = (1,0,0)$$

$$vR = j = (0,1,0)$$

$$wR = k = (0,0,1)$$

that is the same as: $\begin{bmatrix} u \\ v \\ w \end{bmatrix} R = I$, where I is the 3x3 identity matrix

and $R = \begin{bmatrix} u^T & v^T & w^T \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{bmatrix}$ because u, v, w are orthonormal vectors

once we have the rotation matrix we need to compute the translation which transforms the OBB into the origin of the VC system. Lets call the OBB q then:

$$(q,1)M = (0,0,0,1)$$

therefore:

$$qR + t = 0$$

so: $t = -qR = -\left(\sum_{i=1}^{3} q_i u_i, \sum_{i=1}^{3} q_i v_i, \sum_{i=1}^{3} q_i w_i \right)$

Finally the transformation matrix is thus:

$$M = \begin{bmatrix} u_1 & v_1 & w_1 & 0 \\ u_2 & v_2 & w_2 & 0 \\ u_3 & v_3 & w_3 & 0 \\ -\sum_{i=1}^{3} q_i u_i & -\sum_{i=1}^{3} q_i v_i & -\sum_{i=1}^{3} q_i w_i & 1 \end{bmatrix}$$

This matrix is computed once at the initialisation part and then stored so that for every animation step it can be multiplied directly by the cloth coordinates to go from world coordinates to the OBB coordinates, so the only thing we have to compare to know if the vertex is inside the OBB is the coordinates of that vertex against the coordinates of the right top corner of the OBB

3.4.2 Collision Detection vertex-triangle

Once we have the two sets of points between which we need to compute collision detection vertex-triangle, we need first of all to compute the distance from the vertex to the triangle and if it is bellow a certain threshold then we consider that the point lies on the plane and therefore there may be an intersection.

The next step is to compute the projection of this point on the plane. This can be obtained by computing the intersection between the plane and the ray that passes through that vertex with direction normal to the plane [SLA02].

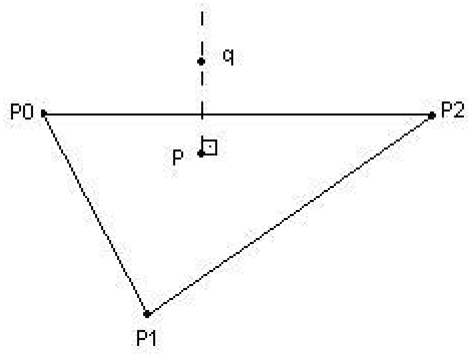


Figure 10. Vertex projection onto triangle plane

The equation of the plane is given by:

$ax + by + cz - d = 0$

Where (x, y, z) are coordinates and a, b, c and d are know. The cross product $(P1-P0)\times(P2-P0)$ gives a normal vector to the plane $\vec{n}=(a,b,c)$. It is obvious that there are two normal vectors to the plane pointing in opposite directions, this direction depends in the order of the points. Therefore it is important the particular labelling of the points which is usually done in counterclockwise order when looking the polygon from the front side.

One useful fact about the equation of a plane is that it can determine the relationship between the plane and any other point in space. A plane divides the space in three subspaces which are the positive half-space, the negative half space and the space that contains all those point lying on the plane.

Supposing we have the point (x,y,z) then:

if $ax+by+cz-d>0$ then (x,y,z) is in the positive half-space (that is on the same side as the one which the normal vector (a,b,c) points)

if $ax+by+cz-d<0$ then (x,y,z) is in the negative half-space, and finally

if $ax+by+cz-d=0$ then the point (x,y,z) lies on the plane.

These facts give allows as to determine whether any particular point is located in front or behind the plane.

Once we know the equation of the plane given by the three vertices that define the triangle we need to compute the intersection of the plane with the ray that goes through the point P in direction perpendicular to the plane.

The equation of that ray is:

$q(t)=P+t\vec{n}$ with $t\geq 0$

and the direction of the ray is given by the vector:

$dq=(du,dv,dw)$

where the plane and polygon meet thus:

$a(u_0+tdu)+b(v_0+tdv)+c(w_0+tdw)-d=0$

Therefore t can be obtained from:

$$t=\frac{d-au_0-bv_0-cw_0}{adu+bdv+cdw}$$

Then substituting t in the equation of the ray we obtain the point P=(x,y,z) which is the projection of the point q into the plane.

Once we have the point p which obviously lies on the plane, we have to find out whereas p is inside the polygon itself or outside. The problem now is thus to determine if the point *p* given in 3D space is inside a polygon given also in 3D space. Since this is not an easy problem to be solved in 3D we are changing the problem to work in 2D space by using projection.

3.4.1 Projections

This method consists in projecting both the point *p* and the vertices of the polygon onto one of the principal planes (XY, XZ or YZ) [SLA02].

The inside/outside relationship between the projection point and the projected polygon would be the same as the original point did to the original polygon. The problem here is choosing the best plane to do the projection. The worst situation that we have to avoid is the one where the projection plane turns out to be orthogonal to the triangle plane, because in this case the projected plane will degenerate into a line. The main idea will be to choose the plane that is somehow most parallel to the triangle plane. This can be obtained by choosing the plane that minimises the angle between the normal to the plane of the triangle and the normal to the principal plane of projection.

Assuming that the normals are normalised, the minimisation of the angle can be obtained by maximising the dot product of the normals.

Since we are working with the principal axis, the dot product turns to be very easy to obtain:

If plane is XY then the plane equation is Z=0 thus its normal vector of the principal plane is $\vec{n}_{pp}$=(0,0,1) and so the dot product $\vec{n}_{pp} \cdot \vec{n} = c$, where $\vec{n}$ is the normal of the triangle plane.

If the plane is XZ then by the same reason the dot product turns out to be *b* and is the plane is YZ then the dot product is *a*.

Therefore the principal plane that we have to choose for the projection corresponds to the maximum absolute value of the coefficient in the plane equation of the triangle.

The algorithm to obtain the projection is very simple:

If XY is chosen then drop the z-coordinate for the triangle vertices and point p

If XZ is chosen then drop the y-coordinate for the triangle vertices and point p

If YZ is chosen then drop the x-coordinate for the triangle vertices and point p

In other words, we can carry out the projection to 2D by dropping the coordinate *x* if |a| is the maximum value, drop *y* if it is |b| the maximum or drop *z* if |c| is the maximum.

Once we have the 2D projection we need to determine if the point is inside the polygon. This is not an easy problem even in 2D if the polygon is allowed to have any shape, but this is not our case, since we are dealing with triangles and thus convex polygons.

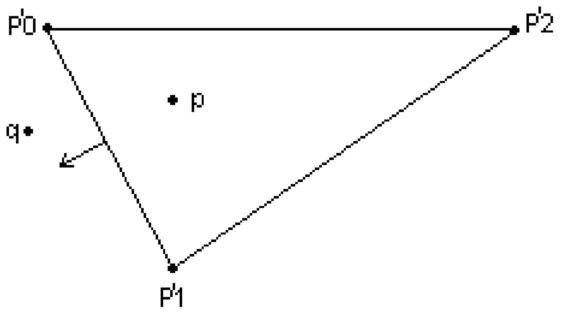


Figure 11. Triangle intersection

Assuming the triangle vertices are given in a counterclockwise order and supposing $p_i' = (x_i, y_i)$ then the equation of the lines forming the triangle edges are:

$$e_i(x,y) = (x - x_i)dy_i - (y - y_i)dx_i = 0$$

$$dx_i = x_{i+1} - x_i$$

$$dy_i = y_{i+1} - y_i$$

For each line equation defining the triangle it is satisfied that those points (x,y) such that $e_0(x, y) > 0$ lie on the positive half-space of the line, that is on the side where the normal to the vector lies. Those points which are outside the triangle will have a positive value for some line equation and a negative value for others, but for points located inside the triangle the values will be negative for all the line equations (they may also have some value 0 if the point lies on any edge).

Hence the algorithm for determining if a point is inside a triangle will be:

If $e_i(p) \leq 0$ *for each edge i = 0,1,...,n-1 then the point is inside*

Otherwise it is outside

4 IMPLEMENTATION

Our main goal in this project is to achieve real time performance of cloth simulation for skinned avatars, so we need an efficient algorithm to detect collision between the cloth and a deformable avatar. One of the optimisations done to avoid a squared cost in the collision detection algorithm is represent the body by OBB in order to have a simplification of the characters for the purpose of computing a test for intersection between the vertices and these OBB, previous to the vertex-triangle intersection.

In our system, the character to be dressed is first approximated by OBB that closely match the character's shape and its hierarchy is arranged in the exact same way as the character's hierarchy. There is therefore one OBB associated with each bone and so in order to go through all the OBB we just need to traverse the skeleton hierarchy.

This OBB representation greatly enhances the speed at which collision detection is performed in our system since it is very quick to compute the intersection by simply applying the transformation matrix to go from world coordinates to local coordinates of the OBB.

The garments are based on a mass-spring particle system. The cloth is given by an irregular triangle mesh where each vertex can have a different mass. This mesh representing the cloth can also have textures applied to give more realism to the final image. The shape and position of the cloth pieces must match the shape and position of the characters to be dressed.

Our main concern during the implementation part of the project has been to obtain the best computational time as possible. In order to obtain our target it has been necessary to take some assumptions and make some generalisations that are explained in detail in this section.

4.1 Space subdivision

First of all we allow some tolerance error in the space subdivision to avoid recomputing it for each time step. Since the computation of the OBB has high computational time, we can not afford all the process of obtaining the tightest OBB for each time step, so when we calculate the size of an OBB we will store a slightly bigger box so that it can be used for every position of the bone in the skeleton.

This assumption can be made since the object we are dealing with (the skin) is not going to change its shape in an unknown way. Skin vertices will have their position updated for each movement of the skeleton, but it is easy to estimate the final shape we can obtain even in an extreme situation. So it can be considered a good decision to work with a slightly bigger size of the OBB.

This decision involves a higher number of collision detection tests between cloth vertices and skin triangles since there will be more cloth vertices satisfying the intersection test with the OBB. But even though we may have a higher number of collision detection tests to be done, we avoid updating the size of the OBB which has a cost of O(N) where N is the number of vertices attached to a bone.

In the analysis section we explained that in order to reduce the amount of triangles to compare for each OBB we also perform a subdivision along one of the axis of the OBB. This subdivision consists mainly in projecting each vertex of the skin onto one of the axis and depending on the segment of the axis where it lies it will belong to a particular subdivision of the OBB. This subdivision can be seen in the following figure:

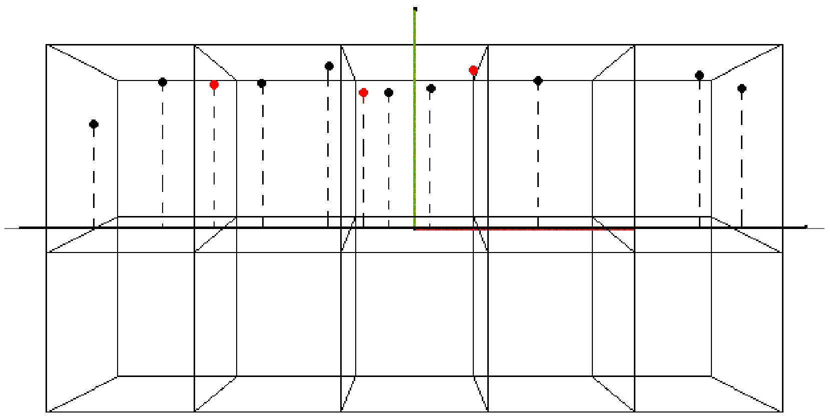


Figure 12. Subdivision of the OBB. Red vertices are the ones that belong to two adjacent subspaces of the OBB.

From the previous figure we can appreciate that the vertices that lie near the border of a segment may be projected in the adjacent segment after the skin has been modified. Since our main goal is to avoid unnecessary updates that require a lot of computational time we can allow certain modifications of the vertices without needing to update the space subdivision. This can be done by allowing a vertex near the border to belong to two adjacent subspaces. The only problem of this method is that we will need to apply the collision detection test twice for the triangles containing that particular vertex, but this will occur in very few situation. Therefore this method will manage to compute collision detection without any need of updating the space subdivision structure.

4.2 Cloth implementation

At the start of the simulation the mesh representing the cloth is given in global coordinates and it has associated the objects of the scene with which it can collide. In this case the object associated with the cloth mesh is an object representing the entire avatar.

For every frame in the simulation, forces such as gravity, wind and internal damping are applied to the particles representing the cloth in order to modify the position of its vertices. After the dynamic forces are accumulated, an explicit integration is performed and the velocity and position of every particle is obtained. Once the new position for a cloth particle is acquired, it is verified that the particle does not penetrate any OBB. If this happens then a more accurate collision detection test between the particle and the skin has to be performed and in case we detect an intersection of the cloth with the skin then the cloth vertex position has to be calculated. The particle is moved to that position and a feedback force is applied to avoid collision in the following frames.

4.3 Collision Detection Algorithm

Our algorithm tests first of all the intersection of each cloth vertex against all the OBB. Once an intersection between a cloth vertex and an OBB is detected then we determine the subdivision inside the OBB where the vertex lies and therefore we will compute vertex-triangle intersection with only the triangles that lie in the same subdivision.

The algorithm is the following:

```
For each OBB detect vertex-OBB intersection
{
If intersection occurs:
    {
    Project vertex onto main axis of the OBB to determine the subdivision
    For all the triangles in the subdivision:
        {
        Compute the distance between the vertex and the triangle plane
        If distance smaller than previous_distance then:
            If vertex projection inside triangle
                Store triangle reference and distance
        }
    }
}
```

This collision detection algorithm will test all the OBB and then will compute intersection vertex-triangle where vertex-OBB intersection occurs.

Two optimisations have been done for this algorithm. First of all we don't need to test intersection against all the vertices within an OBB since we use the projection method described in section (4.1) to narrow the number of triangles to compute intersection.

The second optimisation of the algorithm is that since we are only interested in the closest triangle, after an intersection occurs we store the distance and therefore in the following possible intersections we will only compute the vertex-triangle intersection when the distance between the vertex and the triangle is bellow the previous intersections detected for that particular vertex cloth

Once we have acquired all the information related with the intersection we need to calculate the new position of the cloth vertex and the feedback force needed to be applied.

4.4 Cloth Vertex Update

Once a intersection between a cloth vertex and a OBB has been detected, we need to compute whether the particle intersects any of the triangles within the OBB. To compute the intersection we need to have all the elements in global coordinates, so first of all we need to modify the coordinates of the skin from global to local coordinates. This transformation is given by multiplying the local transformation matrices applied to each bone on the traversal from the root bone to the bone associated with the OBB where the skin vertex is included.

Once we have all the data in global coordinates we have to compute the distance between the cloth vertex and the triangles of the skin with which it may collide. We can have several situations depending on this distance.

The first situation we may have is that the vertex does not approach enough any of the triangles and therefore there is no intersection vertex-triangle as we can see in the following picture:

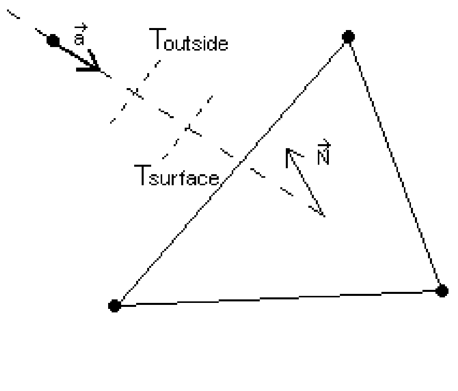


Figure 13. Triangle with no intersection

In this case we do not need to modify the forces applied to the vertex nor its position, and the collision test will return a negative value.

If the vertex approaches any of the triangles we use three different distance thresholds to determine the action we should perform with the cloth vertex colliding.

Those thresholds are:

- OUTSIDE
- ON SURFACE
- INSIDE

Those thresholds will be selected depending on how accurate the final result is required . The bigger the thresholds the easier the intersections are to detect but the higher the computational time required will be.

The OUTSIDE threshold will determine when the cloth is approaching a triangle and therefore it may have a collision within the following frames. This case can be seen in the following figure:

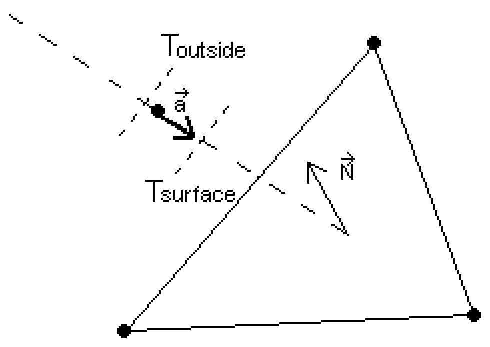


Figure 14. Triangle with intersection but without feedback force

When this occurs we need to compute a feedback force since we want to modify the total force order to avoid the vertex moving towards the skin, but we will not modify the vertex collision position.

The feedback force will be given by Newton's Law:

$$F=m\cdot a$$

Where *m* is the mass of the cloth vertex and *a* is the acceleration that we want to apply to the movement of the vertex.

The module of the acceleration will be inversely proportional to the distance between the vertex and the surface and the direction of the acceleration will be equal to the normal of the surface.

$$|\vec{a}| = SURFACE\_THRESHOLD - distance$$

and $\vec{a} = \vec{n}$

Thus the feedback force is:

$$\vec{F} = m \cdot \vec{a}$$

This feedback force returned by the collision detection algorithm is the response force and will be added to the total force applied to the vertex due to gravity, wind, etc. in order to modify the direction of the movement for the following frames of the animation and therefore avoid the intersection with the skin.

The last two possibilities are shown in the following figures:

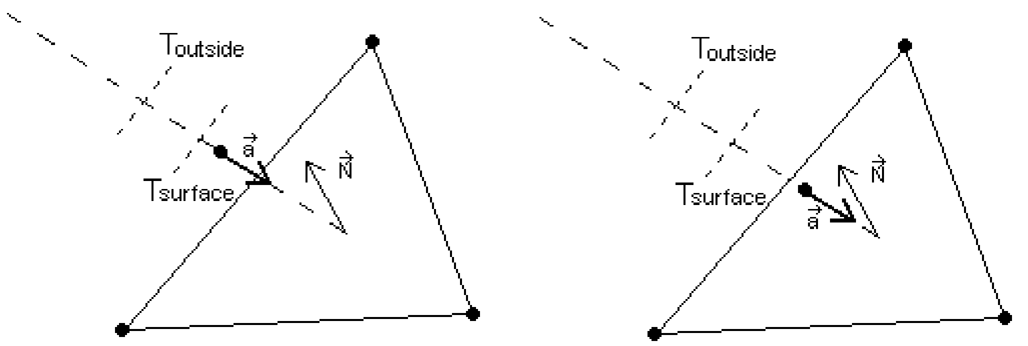


Figure 15. Triangle with intersection and feedback force

The figure on the left shows the case when the distance between the cloth vertex and the plane is bellow the SURFACE threshold, and the figure on the right shows the case when the cloth vertex has already penetrate the triangle. Both cases will be treated in the same way.

First of all we need to compute the feedback force that will be added to the total force applied to the vertex, so that we will modify the direction of the vertex movement for the following frames of the animation. This is done in the same way as we have seen for the previous case.

For the case where the vertex does intersect the skin we need to compute the collision position to be at a certain threshold distance from the surface of the skin.

The new position will be computed as the intersection of the line with direction equal to the normal of the plane and passing through the projection of the point on the triangle plane.

$P' = (x', y', z')$ where P' is the projection of the point P onto the plane

$Pos = P' + \vec{N}$ where $\vec{N}$ is the normal to the plane

4.5 Public Software

Collision Detection Libraries:

There are several algorithms for collision detection that have been already implemented and are available for public domain, for example:

I_COLLIDE Collision detection package available at:

http://www.cs.unc.edu/~geom/I_COLLIDE.html

RAPID Interference detection package available at:

http://www.cs.unc.edu/~geom/OBB/OBBT.html

GSL – GNU Scientific Library:

In order to compute the axis of the OBB the GSL library has been used to compute the eigenvectors of the covariance matrix.

This public software can be found in:

http://www.gnu.org/software/gsl/

5 TESTING

In order to detect the capability of our collision system we have two models which can be seen in the following figures:

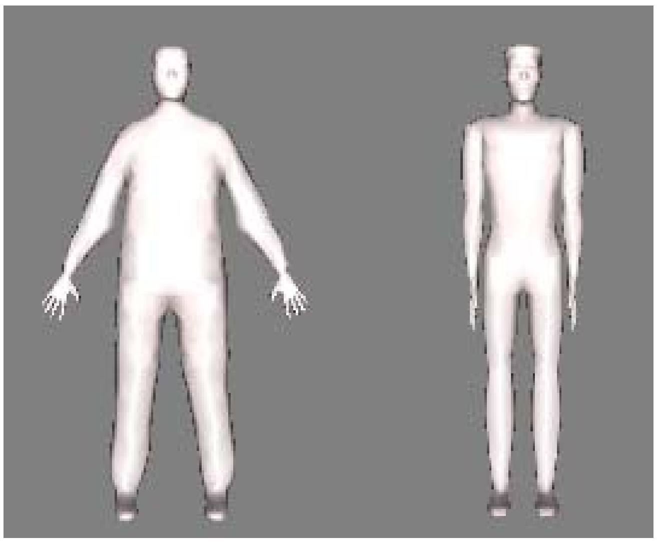


Figure 16. Models

First of all we want to compute the reduction of potential colliders of the skin model for each animation step. Both model consist of a polygon mesh. The number of vertices and polygons (triangles in our case) are:

No. of vertices = 1290

No. of polygons = 2576

The model on the left contains 240 OBB and the one on the right 255.

5.1 OBB

The first step of our algorithm is to compute the OBB associated with each bone. In the following figures we can observe the OBB associated with all the bones of the skeleton.

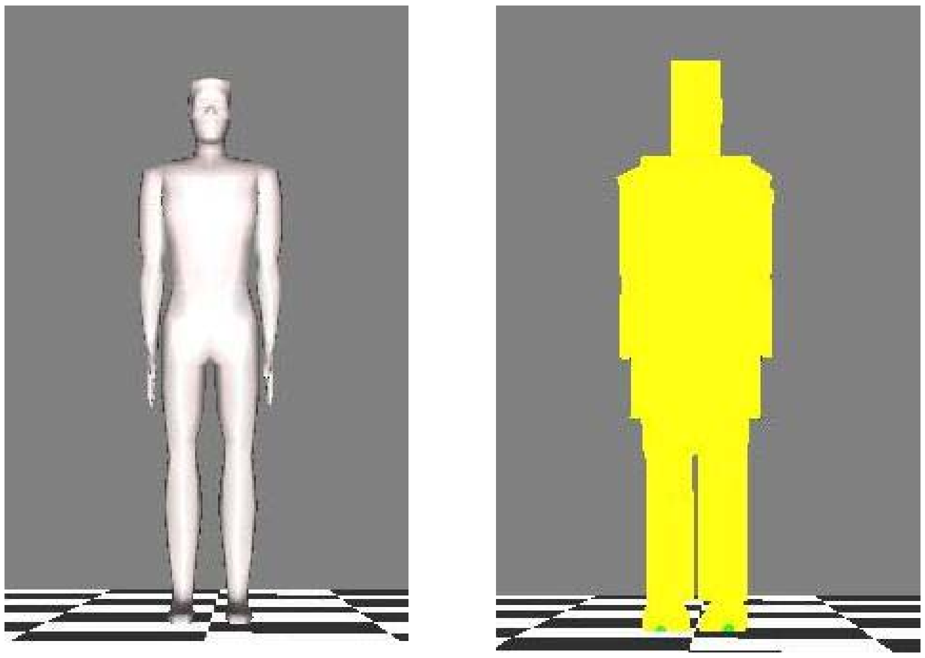


Figure 17. Test of OBBs

In the following figures we can see how the OBB are reoriented depending on the movement of the bones:

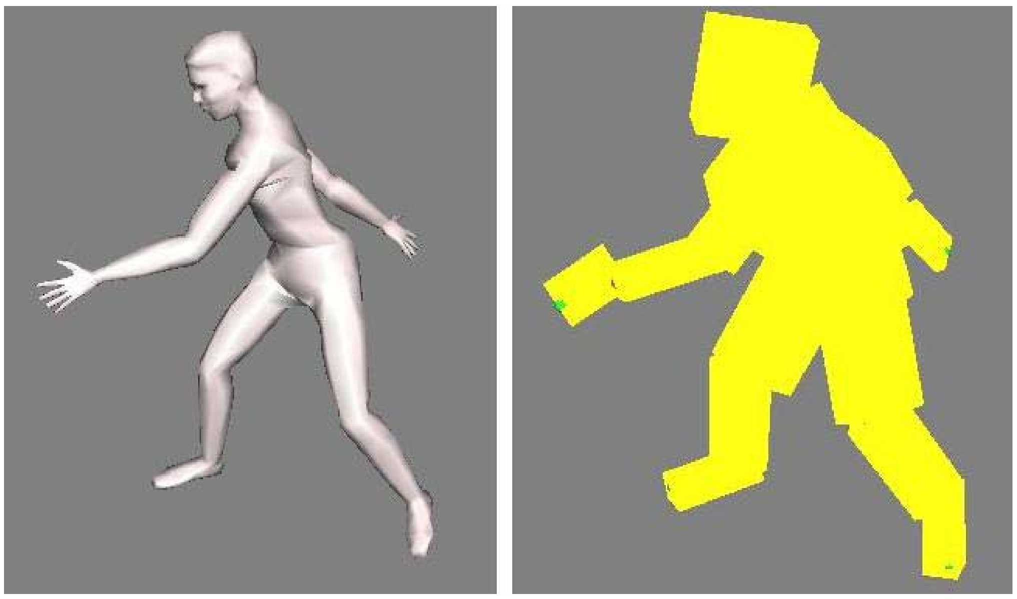


Figure 18. OBB updated during animation

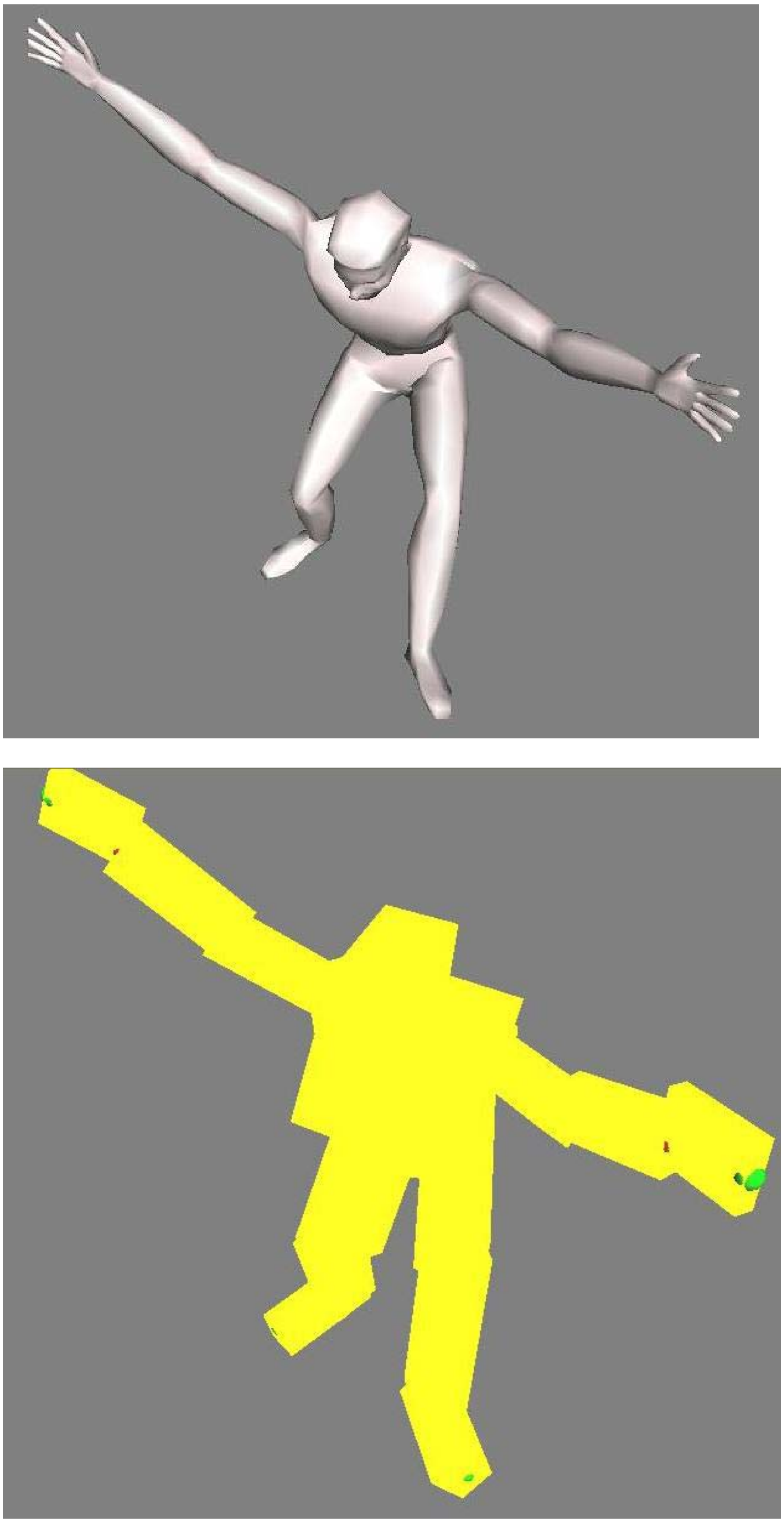


Figure 19. OBB update

5.2 Reduction of the number of triangles

Once competed the OBB associated with each bone we perform a subdivision of the OBB along one of the three axes of its coordinates system. For each OBB we have 15 subdivisions in order to reduce the number of vertices and polygons within each space partitioning so the later collision detection test will perform quicker since there are less triangles to test.

Depending on the axis selected to do the subdivision we obtain slightly different results. In the following tables we can observe the values obtained for each model.

Results for the left figure:

Axis	No. polyg/OBB (Average case)	No. polyg/OBB (Worst case)	No. vertic/OBB (Average case)	No. vertic/OBB (Worst case)
0	29.3	163	6.6	27
1	27.2	128	6.4	28
2	26.3	189	6.6	28

Results for the right figure:

Axis	No. polyg/OBB (Average case)	No. polyg/OBB (Worst case)	No. vertic/OBB (Average case)	No. vertic/OBB (Worst case)
0	27.3	158	6.0	22
1	25.5	220	6.1	23
2	24	203	5.9	27

After testing the cloth mesh against the OBB enclosing the skin the number of collision detection test to be done will depend only on the average number of polygons for each OBB, since in most of the situation one cloth vertex will collide with only one space subdivision. Therefore the number of triangles against which we apply collision detection for each cloth vertex has been reduced from 2576 to approximately 26 which means a reduction of about 99%.

In the worst situation we will have a cloth vertex that can collide with two adjacent space subdivision, since we have allow a tolerance error to avoid the update of the OBB in each time step. In this case the number of triangles to be tested against each cloth vertex will be about 52 (the correspondent to sum the average number of polygons of two space subdivisions). This implies a reduction of about 98%

6 RESULTS

Our system has been used to perform collision detection between a garment with skirt shape and a virtual 3D mannequin obtained from the scanning of a real person. The cloth model contains 264 vertices and the avatar model contains 1290 vertices.

Our algorithm has been implemented using an OBB hierarchical representation. These OBBs allow a preliminary test between vertices of the cloth and skin triangles to eliminate them for the intersection tests. This method dramatically reduces the number of triangles against which the collision detection test has to be done. This optimisation, as we have previously mentioned, achieves a reduction of about 98%.

The method we have implemented not only optimises the number of collision detection tests, but also it is scalable since it is independent of the complexity of the model.

In practice, the implementation of collision detection and response methods can account for anything up to 90% of the total computation required for a typical simulation.. Our algorithm reduces the frame rate by 65% when performing both, collision detection and response in an 800MHz machine running Windows 2000 with a GeForce2 card.

The model previously presented has been also tested when collision detection was being performed without our optimised algorithm. The results showed a decrease in the frame rate of about 99.8%

If we consider only collision detection without applying feedback forces for the response, our reduces the frame rate by about 37%.

From the values obtained we can state that our optimisation significantly improves the frame rate when collision detection is being applied. In the following figure we illustrate the result of our algorithm.

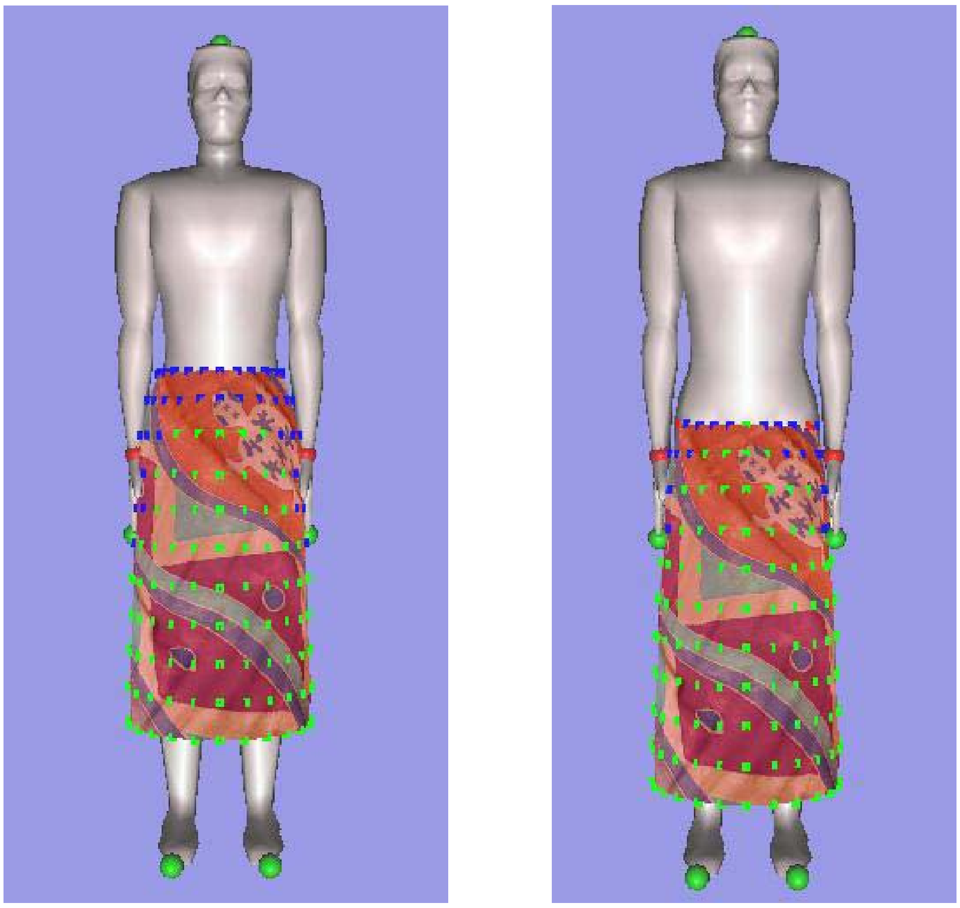


Figure 20. Visual results of collision detection

The regular distribution of points that can be seen in the cloth, shows the cloth vertices against which the collision detection algorithm is being applied.

The green cloth vertex represent those vertices that do not intersect with any OBB and therefore they are not being tested for intersection with the skin. The blue vertices represented those vertices of the cloth that are inside one or maybe more OBB but the collision detection test has given a false result, which means that they are no close enough to the skin so there is no intersection.

The red vertices represent those vertices that are inside an OBB, and intersect a triangle of the skin polygon mesh or are bellow a threshold so that they may intersect in the following frames. In this case a feedback force is applied so that the vertices will be separated from the skin. This can be observed in the waist of the skirt from the image in the left to the image on the right. It can be seen that the upper part of the cloth gets wider as the skirt falls around the body due to the gravity force.

7 CONCLUSIONS, EVALUATION AND FURTHER WORK

Simulation of cloth deformation has many applications in engineering, the textile industry and the entertainment industry. It involves both computer graphics and the understanding of the physical properties of such fabric materials. We need therefore to have efficient collision detection methods to determine the deformations of the cloth depending on the underlying human body.

Real-time graphics simulations, where the shapes of the bodies deform continuously over time, constitute a particular challenge since the possibilities of using pre-calculated data and data structures are dramatically decreased. The result of this work is an efficient collision detection algorithm that works well for real-time simulations of deformable human bodies represented by polygonal meshes.

The method proposed in this project its based on a bounding volume hierarchy whose bounding volumes are OBB. This space subdivision is suitable to be prebuilt before simulation time for many types of deformable bodies and very fast to update during simulation time.

In computer graphics, the emphasis is placed on the possibility of detecting collision in real-time, specially in computer animation applications, even if speed is gained at the cost of losing precision. This is the main reason why hierarchical representations are often used in this domain, since they provide higher precision as one moves down the hierarchy, therefore allowing to adapt output precision to the computing time available. A further work could be then to compute approximate collision detection given the intersection with the OBB and the distance to its main axis in order to avoid computing accurate detection with the polygons of the skin. This approximation could be done depending on the computational time available for each frame and considering for example the viewpoint as a factor when choosing the level of detail for each space subdivision.

In our project we compute the OBB directly from the polygonal mesh associated with each bone. We have seen in the results that this method leads to an orientation of the axes dependent on the distribution of the vertices along the bone. It could be possible to obtain a more accurate OBB if before starting to compute the eigenvectors we carried out some process to obtain an homogeneous level of detail in the mesh of vertices representing the skin of the avatar.

Even though the OBB are updated during the animation, so far we have conducted collision detection experiments only on a static human body. A further step would be therefore to implement dynamic simulation of cloth on moving virtual humans using the presented approach, evaluate its efficiency and see how it compares to other existing methods.

There is much more interesting work to do regarding collision detection between cloth and deforming bodies, as for example:

- Use some kind of space subdivision also for the cloth model or use adjacency graphs to narrow the number of collision detection test.
- Use frame to frame coherence to guess the most likely vertices to collide in the following frames of the animation and discard those that are far away from a potential collider.
- Use information about the viewpoint to perform collision detection with different levels of accuracy depending on how visible a vertex is from the current viewpoint.
- Improve the quality of the physics simulation and avoid also self collision detection among vertices of the cloth.

REFERENCES

[AND98] Anderson, J. N. *Fast Physical Simulation of Virtual Clothing based on Multilevel Approximation Strategies*. Doctor of Philosophy thesis, University of Edinburgh (1998)

[AUB00] Aubel, A., Thalmann, D., *Realistic Deformation of Human Body Shapes*. Computer Graphics Lab, Swiss Federal Institute of Technology (EPFL) Lausanne, Switzerland. (2000)

[BAN95] Bandi, S., Thalmann, D., *An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animating Rigid Bodies*. Computer Graphics Laboratoty, Swiss Federal Institute of Technology, Laussane, Switzerland. (1995)

[BAR93] Barber, B., Dobkin, D., and Huhdanpaa, H., *The quickhull algorithm for convex hull*. Technical Report GCG53, The geometry Center, MN, 1993.

[BRI02] Bridson, R., Fedkiw, R. And Anderson, J., *Robust Treatment of Collision, Contact and Friction for cloth Animation*. Standford University and Industrial Light & Magic, (SIGGRAPH 2002)

[CAR92] Carignam, M., Yang, Y., Thalmann, N. M., Thalmann, D., *Dressing Animated Synthetic Actors with Complex Deformable Clothes*. MIRALab, University of Montreal, Canada. MIRALab, University of Geneva. Computer Graphics Lab, Swiss Federal Institute of Technology. (1992)

[CHR95] Chrysantou, Y., *Shadow Computation for 3D Interaction and Animation*. Department of Computer Science, Queen Mary and Westfield College, University of London. December 1995.

[COH95] Cohen, J. D., Lin., M. C., Manocha, D., Ponamgi, M. K., *I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments*. Department of Computer Science, University of Carolina. (1995)

[GOT96] Gottschalk, S., Lin, M. C., Manocha, D., OBBTree: *A Hierarchical Structure for Rapid Interference Detection*. Department of Computer Science, University of North Carolina. 1996

[H-Anim1.1] VRML Humanoid Animation Working Group, H-Anim 1.1 Specification, 1999. http://www.h-anim.org/

[HEN90] Henne, M., *A Constraint-Based Skin Model For Human Figure Animation*. Master's thesis, University of California, Santa Cruz, June 1990.

[KAL98] Kalra, P., Thalmann, N. M., Moccozet, L., Sannier, G., Aubel, A., Thalmann, D., *Real-time Animation of Realistic Virtual Humans. Computer Graphics and Applications*, Vol 18, No. 5 , (1998)

[KLO97] Klosowski, J.T., Held, M., Mitchell, J.S.B., *Efficient Collision Detection Using Bounding Volume Hierarchies of k-dops*, IEEE transactions on Visualization and Computer Graphics, 1997.

[LEW00] Lewis, J. P., Cordner, M., Fong., N. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation, 2000

[LIN96] Lin, M.C., Manocha, D., Cohen, J., Gottschalk, S., *Collision Detection: Algorithms and Applications*. University of North Carolina (1996)

[LOU95] Louchet, J., Provot, X., Crochemore, D., *Evolutionary identification of cloth animation models*. ENSTA Laboratorie d'Elecronique et d'informatique, Paris, France. INRIA project SYNTIM, Le Chesnay, France. (1995)

[MOO88] Moore, M., Wilhelms, J., *Collision Detection and Response for Computer Animation*, Computer Graphics (SIGGRAPH'88 proceedings), Addison-Wesley, pp 289-298, 1988.

[PON95] Ponamgi, M. K., Manocha, D., Lin, M. C. *Incremental algorithms for collision detection between solid models*. Department of Computer Science, University of Carolina. (1995)

[PRO95] Provot, X., *Deformation Constraints in a Mass-Spring Model to describe Rigid Cloth Behavior*. Institut National de Recherche en Informatique et Automatique (INRIA) Le Chesnay Cedex, France. (1995)

[PRO99] Xavier Provot. *Collision and self-collision handling in cloth model dedicated to design garments*. Institut National de Rechearche en Informatique et Automatique (INRIA) Le chesnay Cedex, France (1999)

[SCH98] Schneider, P. J., Wilhelms, J., *Hybrid Anatomically Based Modeling Of Animals*. University of California, Santa Cruz, California, USA (1998)

[SLA02] Slater, M., Steed, A., Chrysanthou, Y., Computer Graphics and Virtual Environments, From Realism to Real-Time. Addison Wesley

[SMI00] Smith, R., Hilton, A. And Sun, W. *Seamless VRML Humans* CVSSP, University of Surrey, UK

[SUN99] Sun, W., Hilton, A., Smith, R. and Illingworth, J. *Building layered animation models from captured data*. In *Eurographics Workshop on Animation and Simulation '99*. Springer, 1999.

[VOL95] Vollino, P., Courchesne, M., Thalmann. M., *Versatile and Efficient Techniques for Simulating Cloth and Ot her Deformable Objects*, MIRALab, University of Geneva (1995)

[YOS92] Yoshimito, S. *Ballerinas Generated by a Personal Computer*. The journal of visualization and Computer Animation, Vol.3 (1992)

[PRO_BBC] http://www.bbc.co.uk/rd/projects/prometheus/

[PRO_UCL] http://www.cs.ucl.ac.uk/research/vr/Projects/Prometheus/index.html