Cloth Simulation within the PROMETHEUS Real-Time Virtual Studio System

Lee Bull and Mel Slater
Virtual Environments & Computer Graphics Group
Department of Computer Science, University College London, United Kingdom
Email P.Bull@cs.ucl.ac.uk, M.Slater@cs.ucl.ac.uk

Abstract

The PROMETHEUS consortium consisted of UK companies and universities who aimed to develop an end-to-end content creation, broadcast and display system for digital 3D programming, based on the MPEG4 standard. This paper gives an overview of the project and details the real-time cloth simulation server developed within this architecture. Collision detection is performed between cloth and rigid bodies using distance fields. A fast algorithm is given for the generation of distance fields of quantized voxel surfaces based on Voronoi geometry. Real-time methods are then given for collision detection between the distance fields and virtual clothing.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]:Physically based modeling

1 Introduction

The Prometheus consortium, lead by BBC research and development, aimed to investigate the feasibility of end-to-end content creation and delivery of 3D media through the design and development of a prototype system.

Our contribution, presented here, has been to research, develop and integrate a Cloth Simulation Server system to provide dynamic 3D clothing for virtual characters.

This document will provide an overview of the project architecture as a whole and where the cloth server system sits within it in section 3. Research performed on the cloth simulation system will then be reviewed in section 4. Results of a test are given in section 5, with future work and conclusions given in sections 6 and 7.

2 Background

Virtual studio systems have been in use for a number of years, based primarily on chromakey, camera tracking systems such as that of Thomas and colleagues [12] and 3D rendering technologies. The Prometheus project aimed to extend these concepts to incorporate virtual characters, with motion from live actors, tracked in real-time. We are not aware of any systems specifically of this nature that have incorporated cloth simulation system for live use, though commercial systems for real-time performance have been developed[1]. A discussion of future virtual production methods can be found in Thomas and Storey [13].

Cloth simulation systems have been researched substantially, with notable systems being those of Volino an Thalmann [16] using a Runge Kutta solver, Provot [11] using an Euler solver and more recently, the most stable, implicitly solved system of Baraff and Whitkin [2]. Although finite element methods and other techniques have been researched, by far the most common method is mass spring models.

Typical collision detection problems have intrinsic complexity O(*nm*) where *n* and *m* are the number of discrete surface elements in an object pair. This complexity does not lend itself well to interactive systems.

Various solutions have been discussed in the literature to alleviate this problem, generally based on hierarchical, polygonal solutions. Examples can be found in [2] and [4]. In particular, cloth self collision is has also been addressed [10], [15]. Image based methods have also been developed [14]. The use of voxels for collision detection has also been investigated by Zhang and Yuen [18] to reduce local searches for geometrical collision tests.

3 Overview of the Prometheus Architecture

A summary of the project can be found in Thomas and colleagues [9].

Prometheus research has focused primarily on studio side content creation, and dynamics for real-time distribution. The body and facial motions of actors in a studio are captured and mapped onto virtual characters (avatars) in the studio system. Virtual sets can be added to provide a desired environment. Avatar models are themselves obtained either through hand modeling or by scanning

[1] Havok's SDK is of particular note http://www.havok.com

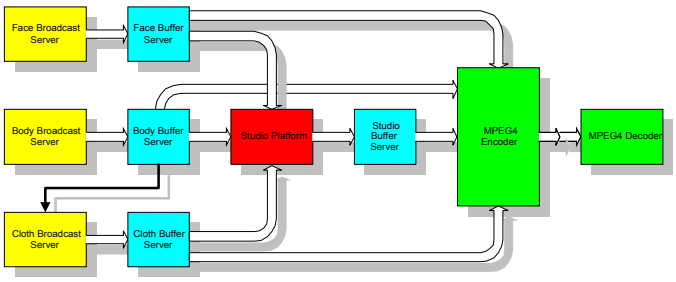


Figure 1 Prometheus Architecture (Basic)

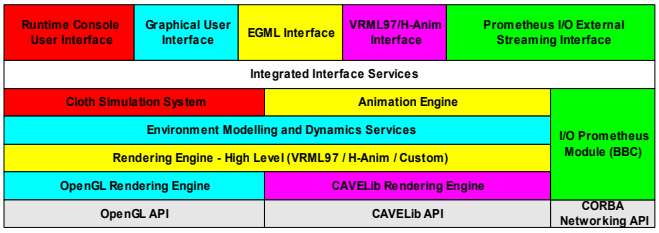


Figure 2 Cloth server software layers

people using image based methods developed by the University of Surrey and industrial partner Avatar-Me. Real-time cloth simulation is performed to make character clothing more dynamic and realistic. The results of the server/studio side processing are passed on to an MPEG4 encoder for broadcast to set top box style systems. The implementation of this encoder has been provided by the University of Ottawa in Canada. Finally, DeMontfort university have researched feasibility issues for 3D content display hardware, targeted at the receiving, end user.

Figure 1 shows the flow of information between server side modules in the simplest configuration of the Prometheus architecture. The decision to develop a distributed architecture was driven by the requirements for substantial processing in all of the modules of the system. Rather than rely on systems having enough CPUs to cope with the architecture internally, the distributed nature of the system allows multiple lower cost systems to work in co-operation on a high speed 1GBit LAN. The system is also scalable in that multiple motion capture or cloth simulation servers can be set up to handle increasing numbers of actors/virtual characters in the scene, network bandwidth permitting. The OMG CORBA API was chosen, as a suitable low level system for the rapid development of middleware for the architecture. This middleware, known as the ioPrometheus system, allows many small buffer servers to be set up across the system. Each server module that produces data, such as facial, body motion capture systems and cloth simulators, all produce data that must be consumed by one or more clients. The system provides a buffer server for each data broadcaster, to which its output is streamed and buffered. Clients that need to receive particular data streams can then connect to these intermediate buffering servers and consume data at their own rate. This buffering process allows different parts of the Prometheus system to execute with a degree of asynchronicity, without global synchronization to the speed of the slowest module. Final frame information must however, be collated when ready, for MPEG4 encoding. Time codes are used throughout to match frame update information from multiple sources.

4 The Cloth Simulation Server

The cloth simulation module must receive real-time streams of motion capture data. Each stream pertains to each virtual character in the scene, for which clothing is to be simulated. Using this data, the system internally animates its own segmented representation of each clothed virtual character and performs simulation tasks for each item of clothing worn by each character. Finally it must stream deformed clothing animation data to clients in the architecture. The system can execute in one of three test modes - animation server, cloth simulator and player, all of which run as separate clients and servers on a network. This permits testing of the cloth simulation server independent of the Prometheus architecture, using the same communication protocols.

The cloth server has its own custom VRML97 compatible rendering engine, along with an animation engine. The system works under Windows, Unix and UCL's ReaCTor (Cave-like environment.) Development of an internal character animation engine allowed the cloth simulation system to be researched in the absence of streamed body motion capture data, two years before data became available in the project. The animation system uses a notion of Event Graphs, separate to the scene graphs, created either through the API or a simple script language we have developed called EGML (Event Graph Markup Language.) The syntax of this language is similar to VRML97. Event graphs are hierarchical, where events can consist of child events, e.g. timelines can consist of child timelines. Executable (effector) events are present at the leaves, where event threads are created during traversal, unique to the traversal path. When event graphs are traversed, parameters for the animation system can be modified and accumulated at nodes during descent and are provided to each leaf effector event thread. Event threads are maintained by an animation thread scheduling system. A range of nodes provide other conditional execution, thread and arbitrary event control. It is possible to create state transition systems with certain sets of nodes. Not all events must be scripted. In particular, the system is designed to allow the injection of event graphs at runtime

which begin immediate execution in addition to any other existing activities.

Rather than research alternative methods for animating 3D clothing, a fairly conventional approach was taken to physically model the clothing dynamics. Forces within the system are evaluated and advanced using an ordinary differential equation (ODE) solver, with additional responses imposed by collisions between the clothing and the animating character.

The most substantial contribution made by this research lies in the area of quickly evaluating cloth-character collisions at runtime. Cloth-cloth collisions are currently ignored to aid realization of real-time performance and are left for future research.

4.1 Overview of the Physical Simulation

This section will give a brief overview of the physical model chosen for use by the simulation. Our physical model is based strongly on the work of Anderson [1], who has reformulated an approximation of a physical model by Volino and Thalmann [16].

Many cloth simulation methods such as [11] require a uniform polygonal grid representation to provide an expected geometry for the formulation of force evaluation equations. These are typically mass-spring representations, where vertices of the mesh have associated masses and polygon edges are notional springs which exert forces on connected mass points. Additional conceptual springs are sometimes included, e.g. diagonally to penalize bending and sheering.

It was considered that regular grid polygonal meshes were too limiting for 3D clothing model designers. This is because such methods require 2D patterns to be designed and virtually stitched together. While this may be appropriate for simulating clothing design processes, it is not easily suited to rapid modeling, where a garment may be designed as a single item. A method was sought which would allow a reasonably accurate simulation of an arbitrary mesh to be performed. This is made possible by the work of Thalmann [16], but force evaluation equations required were deemed too substantial for real-time performance. An approximation of [16] given by Anderson [1] was chosen for use.

Non uniform meshes can be used with this method, using attributes including density, thickness, Young's modulus and the Poisson coefficient, known to material engineering. Vertex masses are calculated by distributing local mesh masses to local vertices. Spring constants also vary across the cloth, calculated individually based on the local geometry of the material and its properties.

Cloth meshes are initially specified as conventional VRML97 IndexedFaceSet nodes, but are identified on startup as they are children of a ClothSet custom scene graph node. These simple indexed face set meshes undergo substantial reprocessing to a more connected form of mesh similar to a winged edge data structure [3] that supports the types of queries needed at runtime.

A custom ClothMaterial node in our scene graphs allow physical attributes and other parameters to be specified for each clothing item. A second Simulation node specifies environmental conditions and simulation parameters.

In each frame, the system iteratively simulates many small time steps of animation. In each iteration, the system evaluates a number of types of force acting on each mass point. This forms an initial value ODE problem. Rather than concentrate on complex solvers, we have opted to keep a simple Euler solver, allowing research to focus on collision detection. Though Euler is known to require more iterations and offers less stability than others such as n'th order Runge Kutta [16] or Implicit Euler [2], our tests show it to be acceptably stable for our current work. Future work will incorporate other solvers.

Forces that are evaluated at the start of each time step include:

- Tensional (Tension, compression, shearing)
- Flexional (Bending)
- Frictional (Between cloth and surfaces)
- Gravitational (Normal earth gravity)
- Viscous drag (Air friction, acts as damper)
- Constant wind (Simple wind model)

An iteration frequency of typically 30 time steps per frame is currently sufficient to keep a reasonable level of stability in the solver, whilst maintaining real-time frame rates for cloth items consisting of a few hundred polygons on a 1GHz PC.

Fixture constraints can be applied to any subset of a clothing item such that it is able to bind to the character. This is needed for waist and cuff areas. To allow constraint areas to be specified at modeling time, a *constraint mask* is designed, which is a subset of the original cloth model. The constraint mask is referred to in the ClothSet node. The system first matches vertices in the mask to vertices in the original model, within a tolerance distance. These vertices are then considered to be constraint candidates. Candidates are constrained to the body part that is closest, within a specified threshold distance. Distance fields are used to perform this proximity measurement and therefore, mass points can only be constrained to objects that have been declared as collidable.

4.2 Character Model Preparation

The Prometheus system has modules that scan a person to obtain an avatar, developed by the University of Surrey and Avatar-Me. These models are seamless, skinned avatars for use in rendering. If these avatars are used, they must currently undergo redesign by hand to create a segmented form of the avatar. The majority of this process can be performed in software, i.e. the identification of segments and their hierarchical export. This leaves a limb capping process to be performed by hand such that gaps will not be revealed under typical articulation.

Secondly, the avatar must have all hierarchical transforms reset, such that all modeling matrices have unit scale and are aligned with the character's root coordinate system. Changes are committed to both the transforms and the vertices in the model, though appearance is preserved. This is necessary to allow the correct application of motion capture data. Secondly, unit scale allows values to be specified, e.g. voxel density in a way which is uniform across the avatar. Thirdly, this solves requirements for re-normalization of surface normals when transforming from local coordinate systems to a global simulation system.

4.3 Overview of the Collision Detection Method

The primary focus of this research was to discover a fast and efficient cloth-avatar collision detection system, suitable for real-time performance.

Cloth self collision detection was not included to reduce overheads at runtime. Typically, the absence of cloth self collisions is not particularly noticeable in dynamic scenes with single layered clothed avatars. Notable work on this subject has been performed by Thalmann [15] and Provot [10].

The expense of full geometric cloth and avatar polygon collision detection was ruled out as this can be expensive unless very few polygons can be determined to be candidates. Cloth mass point (vertex) collisions with the avatar were favored as intersection tests between vertex motion paths and surfaces was considered more likely to achieve real-time performance.

This problem is one of detecting whether the desired motion of a mass point after a time step, as requested by the physical simulation, will collide with any moving avatar body parts or other dynamic objects in the scene. This motion is only desired because the end state may be illegal if cloth mass points penetrate the body during the time step.

If collision tests are to be performed at each time step, the collision detection system must be very efficient. Assuming a simple high frequency solver applied to an example with 30 time steps per frame at 25 frames per second, the system must run 750 times per second. Moreover, the processing time used for collision detection must itself be short to allow additional time for the physical simulation and other overheads such as animation and rendering if required.

The use of vertex-surface collisions, rather than surface-surface collisions gives rise to potential intersections between cloth surface polygons and the character's surfaces, even though all vertices of the cloth may be outside the character's volume. This is most likely to occur in regions of high surface curvature or where cloth surfaces have a low sampling density, i.e. few polygons which is likely for interactive systems. To alleviate this problem, collision detection is performed at a threshold distance from the character's surface, i.e. it must collide with an offset surface of the body. This distance must be calibrated to the avatar and cloth polygon sizes.

The method presented here is based on a distance field representation for local body limb spatial partitioning. Non-hierarchical bounding boxes are used to bound limb segments.

Initially, spatial partitioning of the dynamic characters and scene was considered. There are many ways in which this could be done, including Octrees, BSP trees, K-D Trees or bounding volume hierarchies. Partitioning could be performed in local object or world coordinates. Hierarchical partitioning methods were rejected due to their need for reconstruction resulting from changes and their O(nLoga) performance for tests, where n is the number of vertices, and a the number of partitions. Voxels were favored because they would potentially offer a system which tends towards O(n) complexity with decreasing time step size for collision path tests as motion paths through the voxel space become smaller. One voxel space is created locally for each body part that is potentially collidable with clothing. The voxels would require complete recalculation if skin were allowed to deform at joint areas. Therefore, segmented avatars were chosen for use in the collision system, placing an extra requirement for there to be a segmented representation of characters, in addition to the skinned, seamless representation used for rendering. Offset surface collisions are likely to be capable of covering visual artifacts from geometrical anomalies between the skinned and segmeted avatars for collision purposes.

Each joint in the avatar has a single distance field volume. The system must carry out tests to determine which distance fields a mass point's motion may intersect. For this, a method which simply lists collision candidate objects for each clothing item was chosen. Typically, a character's body has about 19 segments. Each garment, e.g.

shirt or skirt will generally only collide with a small subset of these. For example, a skirt generally only collides with hips, thighs, shins and maybe hands. A hierarchy of bounding volumes of these objects would generally have a high degree of overlap and will not have high depth, reducing hierarchical efficiencies. While a non hierarchical method is not a good solution for large environments, it is suitable for this small case where overlapping in hierarchical volumes is prevalent.

The distance field method was also chosen because it may integrate easily with better global spatial partitioning systems, left as future work.

Distance fields are a voxel space, where each voxel contains information about the closest surface items, usually stated as a distance to them. Proximity information is therefore readily available for offset surface and vertex proximity calculations. If this information is in the form of a reference to one or more closest voxels, this forms Voronoi regions around the original surface voxels.

4.3.1 Character Motion Micro Interpolation

Typical motion capture data is sampled at circa 30Hz, which is about 25 times less positional information than is required by a simulation with a high frequency solver. It is not possible to animate the character at 30Hz alone, because frame changes would be received by a single time step, interpreted as very large accelerations. The system would likely become unstable.

Therefore, sub-frame character motion interpolation is necessary, dissipating the energy across all time steps between frames. A linear interpolation system is used, that incorporates spherical linear interpolations (SLERPS) for character joint angles.

Rather than interpolate each local modeling matrix associated with the character's skeleton at each time step, the system interpolates each joint's Local Transformation Matrix LTM and its inverse LTM^{-1}. An advantage is that forward kinematic matrix products are not required to analyze the joint's motion in the world coordinate system. The disadvantage is that the lack of forward kinematic calculation results in the character's joints being interpolated linearly in world space, potentially separating them during the interpolation. The joints then reconvene at the end of the frame. Typical changes between motion capture frames are generally so small that this artifact can be ignored.

4.3.2 Distance Field Generation

Here, we present our method, the *Discrete Voronoi Distance Propagation* (DVDP) for simple, rapid generation of distance fields from a discretized surface.

Initially, the bounding box of an object is selected and used as a basis for the voxel space. The contained polygonal boundary representation is then voxelized into the discretized voxel space using scan line based algorithms [8]. For purposes of rendering, voxel occupancy data could be included to anti-alias the discretized surfaces. For the purposes of collision detection, we relax this requirement and use a binary voxel space, where voxels are considered to be either inside or outside the volume described by the object.

Polygon normals are written into each voxel during the voxelization process, rather than inferring them later from the discretized surface structure. Voxels that contain more than one polygon, e.g. on edges, vertices or multiple sub-sampled polygons, have their normals listed in the voxel to permit multiple surface collision responses.

Naïve generation of the distance field, where the distance between each voxel is compared with every other voxel has a complexity of $O(n^6)$ where n is the size of a cubic voxel space, which clearly does not scale well.

To combat this problem, various Distance Transform (DT) methods have appeared in the literature for calculating distance fields, the most basic being the Chamfer Distance Transforms (CDT) and notably the recent Vector City Vector Distance Transform (VCVDT) by Satherley and colleagues [6], [7] that use a vector form of the CDT. Distance shells are initialized with accurate vectors to closest polygonal surfaces. These vectors are then propagated and modified based on inter-voxel transition directions using multiple pass applications of vector based kernels. A hierarchical variant called an Adaptive Distance Field (ADF) has also been developed Frisken [5].

Our method is based on an incremental propagation of the initial voxel space surfaces, forming a Voronoi diagram of the discretized surface. In particular, no explicit mathematical operators are required to calculate the Voronoi regions, other than those required for simple voxel space data structure access. Accuracy is restricted to the binary, discretized surface unlike methods such as [6] that make corrections based the original polygonal surface.

A global Voronoi solution is determined by considering the aggregate spherical expansions of all voxels in the voxel space, until all free space is filled. The expansion of each Voronoi cell will compete evenly if expansion is sequential and uniform across the voxel space.

To do this, we first consider the iterative expansion of a single voxel at the center of a conceptual voxel space. At each iteration $i+1$, the voxel is expanded by the least radius possible, such that the new spherical shell oriented about the central voxel intersects the centers of new voxels that are not on the surface in the previous iteration i.

A 2D equivalent, forms a sequence of incrementally expanding circles in the plane, centered at the expanding voxel. The 2D voxel space can be reduced by symmetry to quadrants and then further to octants by diagonal symmetry. Within this triangular voxel space, the expansion ordering for a maximal sized voxel space can be predetermined, as shown in Figure 3. Each iteration has an associated Euclidian distance $\|v\|$ from v. Note that each iteration value may add voxels at more than one location. Iterations 13 and 25 are examples of this.

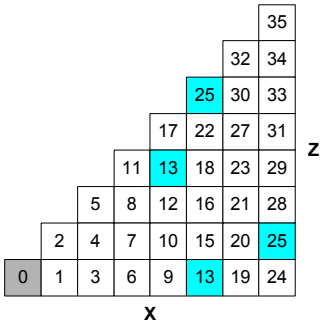


Figure 3 2D Octant incremental expansion of voxel 0

The expansion is mirrored in each octant in the plane. This example would cater for a voxel space of maximum size 8, but can be extended as required.

An adjacency rule base can be formed, to locate the relative positions of new voxels V_{i+1} in iteration $i+1$ from those V_i already introduced in i.

Adjacency rules are based on the type of connectivity between two voxels. In 2D, two types of 4-connections are possible, through edges and corners.

Each iteration i has a set of k associated action rules A_i, such that each action $a \in A_i$ introduces one new voxel $v \in V_i$ in this symmetrical subset of the voxel space. For example, A_7 has only one action, but A_{13} and A_{25} have two. Each action has a source voxel v_s and a destination voxel v_d that is dependant on v_s.

To simplify action rules and cater for lookup table symmetry through orientational invariance, a source voxel v_s is expanded to all neighbouring voxels N_{v_s} with the same connectivity as it has with the destination voxel v_d. The chosen v_s must itself, be contained in the lookup table. Care must be taken when choosing connectivity. For example, if $v_d = v_{10}$ as the iteration count has reached 10, v_8 cannot be chosen as v_s based on a diagonal 4-connectivity due to v_8's potential expansion to v_{13} by the same adjacency type. To ensure correct ordering in the expansion, a source voxel v_s is chosen for each v_d in an iteration, based on a chosen connectivity such that:

$$\|v_s\| < \|v_d\| \qquad \textbf{(1)}$$

$$\|v\| \le \|v_d\| \ , \forall v \in N_v \qquad \textbf{(2)}$$

where N_v is set of voxels neighbouring v_s with the chosen connectivity. This guarantees that the chosen v_s exists because it was introduced in an earlier iteration. Secondly, it will not under any circumstances, expand to iterations higher than that of the destination voxel due to orientational invariance.

It can be shown through inequalities that for any v_d that does not lie on the diagonal, a suitable v_s is the left edge 4-connected voxel as they satisfy (1) and (2), i.e. for $v_d = (x,z)$, $v_s = (x-1,z)$. Similarly, for those $v_d = (x,z)$ that lie on the diagonal, it can also be shown through inequalities that the lower left diagonal can be used, i.e. $v_s = (x-1,z-1)$ such that (1) and (2) are satisfied.

The 2D form of this rule base can be reinterpreted in 3D with a lookup table space in the shape of a tetrahedron bounded by the following inequality:

$$y \le z \le x \qquad \textbf{(3)}$$

as shown in Figure 4. Other tetrahedral configurations would also be possible through symmetry.

There is a 1/48th fold symmetry of the tetrahedron in the voxel space, shown in Figure 5. Each octant of the voxel

space is subdivided into sextants, mappable through symmetry onto the tetrahedron of Figure 4.

Note that each of the sextant tetrahedrons share the same origin at the center of the voxel space, equivalent to the voxel 0 origin of Figure 4.

In the 3D case, there are three types, of voxel connectivity, 6-connection (face), 12-connection (edge) and 8-connection (corner).

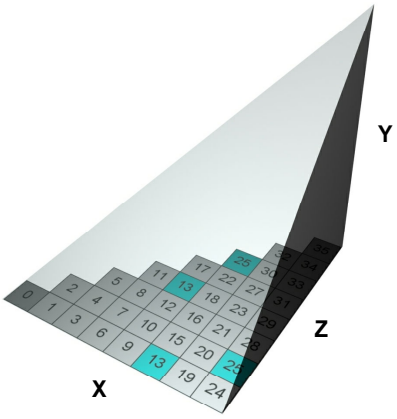


Figure 4 Extension of 2D rule base triangle (base) to 3D Tetrahedral rule base (only 2D shown)

Again, it can be shown through inequalities that the source voxels in Table 1 satisfy equations (1) and (2) subject to v_s being contained within (3) with selected connectivity such that v_s still maps into the tetrahedron. Table 1 shows which central voxels and connectivity are used for actions based on the position of $v_d = (x, y, z)$.

It is possible to use other rules, such as 8 or 12 connected rules in place of 6-connected rules. However, it is advantageous to use a connectivity with the least number of adjacent voxels as the expansion of each source voxel will address all adjacent voxels with the chosen connectivity. A number of these will be redundant as some of the neighbours will belong to previous iterations.

The voxel is initialized such that all transparent voxels are tagged as *unprocessed*. The Voronoi distance field algorithm incrementally expands the area around each discretized voxel in equal amounts at each iteration. Each expansion step is the smallest possible.

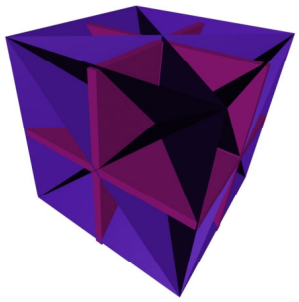


Figure 5 Tetrahedral symmetry in voxel space

Table 1 3D Tetrahedral rule base action connectivities

$v_d(x,y,z)$	$v_s(x,y,z)$	Connectivity
x>z	(x-1,y,z)	6-connected
x=z, y=0	(x-1,y,z-1)	8-connected
x=z, y>0	(x-1,y-1,z-1)	12-connected

A set of voxel sets $S_i \in S$ stores references to voxels generated in each iteration. The initial surface set S_0 is the initial voxelized surface. When a set is completed, all its voxels are tagged as *processed*. Therefore, the initial surface is also considered processed before iteration starts. Sets can be discarded to reduce memory overheads when there are no longer any dependant voxels.

Each action in A_i specifies which set will be used as the source, the nature of the connectivity and the distance that will be assigned.

Each iteration has one or more actions. Each action is executed on the source set that it specifies. During a source voxel's expansion, an attempt is made to expand to all neighbours with the connectivity specified by the action. Due to (1) and (2), these voxels will either be valid destinations for the current iteration, or they will have already been included in a previous iteration and will already be tagged as processed. In this case, they are not included in the iteration.

Several voxels may legally expand to the same destination voxel in a single action, or during multiple actions of an iteration. In these cases, the distance and closest voxel information are passed on and accumulated within the destination voxel allowing multiple Voronoi cell membership.

Voxels of the original surface have an associated Voronoi cell when completed. To reduce memory storage overheads, we limit the propagation of Voronoi references to a shell distance around the initial surface. Voxels further in the Voronoi cell contain only distance information that could apply to multiple surfaces at that distance. This is important as voxels in volumes within objects and concavities can belong to a large number of Voronoi cells if equidistant to multiple voxels in S_0.

Offset surface normals are currently derived from the normals associated with the Voronoi cell. A number of methods have been developed in the literature for offset surface normal approximation [17], but we currently use the Voronoi cell normals as they result in a collision response representing their closest surface voxels in a way that has shown to be acceptable.

4.3.3 Collision Detection in the Distance Field

This section will look at how collision detection can be performed between a single mass point and a single distance field. During a time step from t to $t+\Delta$, both the mass point and the body segment will move, with a mass point's relative starting position $p(t)$ in the previous time step to a relative end position $p(t+\Delta)$. The motion of the mass point relative to the distance field over the short time step is treated as a linear approximation, to simplify collision calculations and attempt to provide a fast algorithm.

A cloth mass point can have one of three assigned states with respect to its position relative to the distance field volume and object offset surfaces:

$$State(p) \in \{OutsideVol, InsideVol, OnSurf\}$$

The first two possibilities are defined by the mass point's containment status in the distance field volume. Whether the mass point is on the object surface or not is defined during the course of the collision detection, though it must also be contained within the volume.

Each mass point has a data structure that records its current state with respect to each potential collidable object, also caching other data such as its last position in the world, local and voxel coordinate systems.

The initial state of all cloth mass points with respect to all distance fields each is potentially collidable with, must be established before the simulation starts.
At the start of a time step, the mass point may be in any of these states. At the end of the time step, it may finish in any of these states, therefore resulting in 9 possible state transitions.

The relative linear motion of the mass point in the distance field is traced using an algorithm that is similar to that of plotting a 3D line. Currently, we use a 3D Bresenham algorithm or alternatively, a potentially faster lookup table method. As the line trace advances, the distance of the voxel is checked, relative to a number of offset surfaces that provide a response model.

State transitions from $OutsideVol \rightarrow OutsideVol$ do not require any tracing to be performed as this form of intersection is currently ignored. Similarly, once a trace has exited a distance field volume, i.e. the transition is $\{InsideVol \mid OnSurf\} \rightarrow OutsideVol$ it cannot re-enter due to the convex nature of the volume and thus requires no further processing during the time step.

A mass point that is entering the volume $OutsideVol \rightarrow \{InsideVol \mid OnSurf\}$ needs to establish the voxel encountered upon entry such that the trace can commence. This would be possible by calculating the face intersected, establishing the point of intersection and transforming the position into the local integer coordinate system of the distance field. Rather than perform this potentially lengthy calculation, the method simply traces the path in reverse until the trace exits the volume. In this case, it is not the first intersection encountered that is of interest, but the last, which is in fact the first intersection as the trace is reversed.

During tracing it is necessary to establish whether the trace has exited the volume. Performed naively, this would require clipping or a dimensions bounds test at each trace step, or knowledge of the number of steps until exit. To speed up this process, extra voxels are placed on the outer border of the distance field that are tagged as being on the border. Each step of the trace can then perform a single, fast exit test.

A four offset surface model is used to manage collisions and sliding with the distance field, shown in Figure 6. Offset surfaces can be easily determined at runtime as the distance field already encodes surface proximity information for all voxels. References to normals of surface cells are passed on through their Voronoi cell to a shell distance that contains the chosen outer threshold distance T_3.

Each offset surface performs a different function. Collisions are registered on impact at T_2 which is contained between envelope surfaces T_3 and T_1.

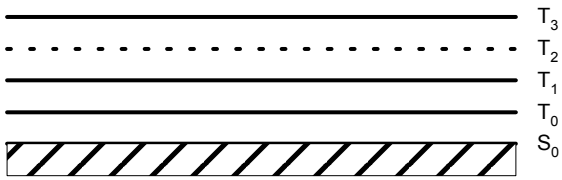


Figure 6 Four offset surface collision model

This envelope allows for imprecision in the simulation's collision response, e.g. due to inaccurate normals or rounding errors. It also attempts to counter the problematic effects of sliding along jagged, voxel offset surfaces. If a cloth mass point ventures below, a penalty force is applied to persuade the vertex to move away from the surface. If the mass point gets closer still, a collision is registered at until later conditions allow forces to be applied that move it away from the surface.

5 Results

Test scenarios have been developed, primarily consisting of clothed avatars walking, to evaluate the system's performance. Here, we will present results of the walking sequence shown in Figure 7 for both distance field generation and runtime performance. The character in sequence walks from (a) to (g) using a motion sequence of 203 frames. Image (h) shows a continuation of the sequence after the test period. Hips, thighs and shins are set up as collision candidates and therefore require distance field representations. The cloth consists of 480 triangles. The hardware used throughout is an Intel P4 2.8GHz system with an nVidia Quadro4 900XGL graphics card.

Times for distance field generation after voxelization for each collision candidate body part are shown in Table 2. The distance field is calculated for the whole voxel space, but Voronoi cell membership references are only propagated up to the outer offset surface T_3=6 to conserve memory. Also included is the time required to generate the distance field without Voronoi references. Though the resulting volume not used in the cloth simulation, it serves as a comparison. Additional empty voxels are placed on the borders of the volume to cater for the offset surfaces, such that they are guaranteed to be contained.

The simulation was run at 33 time steps per frame. Runtime performance details for this animation sequence are given in Table 3. Here, we have rendered just the character and cloth without a scene, to realize a frame rate of 26.62 fps. In a second test, rendering has been disabled to show calculation time for the animation and cloth simulation, resulting in a frame rate of 30.2 fps. The sequence uses 17.1% of its simulation time performing collision tests, disregarding rendering, animation and animation interpolation.

Table 2 Collision candidate distance field generation

Character Body Part	Voxel Space Dimensions & No. Voxels	Distance Field (6) Generation Time (s)	Distance Field (0) Generation Time (s)
Hips	109x99x107 (1,154,637)	2.765	2.422
Thigh	75x190x73 (1,467,750)	3.641	2.797
Shin	57x177x61 (615,429)	3.422	1.656

Reduced frequency collision tests have been shown to work well, e.g. performing 10 collision test cycles per second, with multiple time steps in between, though some accuracy will be lost and the resulting simulation differs.

Table 3 Real-time performance for sequence

Sequence Performance Detail	Value
Rendered Frame Rate (fps)	26.62
Sim. Frame Rate (No Rendering) (fps)	30.2
Average Collision Path Distance (Voxels)	0.166036
Total Collision Detection Time Percentage	17.1 %

6 Future Work

Additional work could focus on:

- General stability issues and compatibility with lower frequency solvers, such as Runge Kutta [16] and Implicit methods [2].
- Compression of volume data and its use in real-time
- Application of distance field collision detection to seamless avatars.

7 Conclusion

We have developed a real-time cloth simulation sever, integrated into a distributed network architecture for 3D multi-media content creation and delivery. Real-time, interactive frame rates have been achieved on low cost equipment, suitable for multiple cloth server implementations. We have introduced the DVDP algorithm as a fast and simple method for the generation of Voronoi geometry of volumetric object representations. A fast collision detection method has been developed, based on the results of the DVDP. The system uses a four offset surface collision response model, which has also been shown to function at real-time rates, with very low percentage overhead in comparison to the physical simulation time.

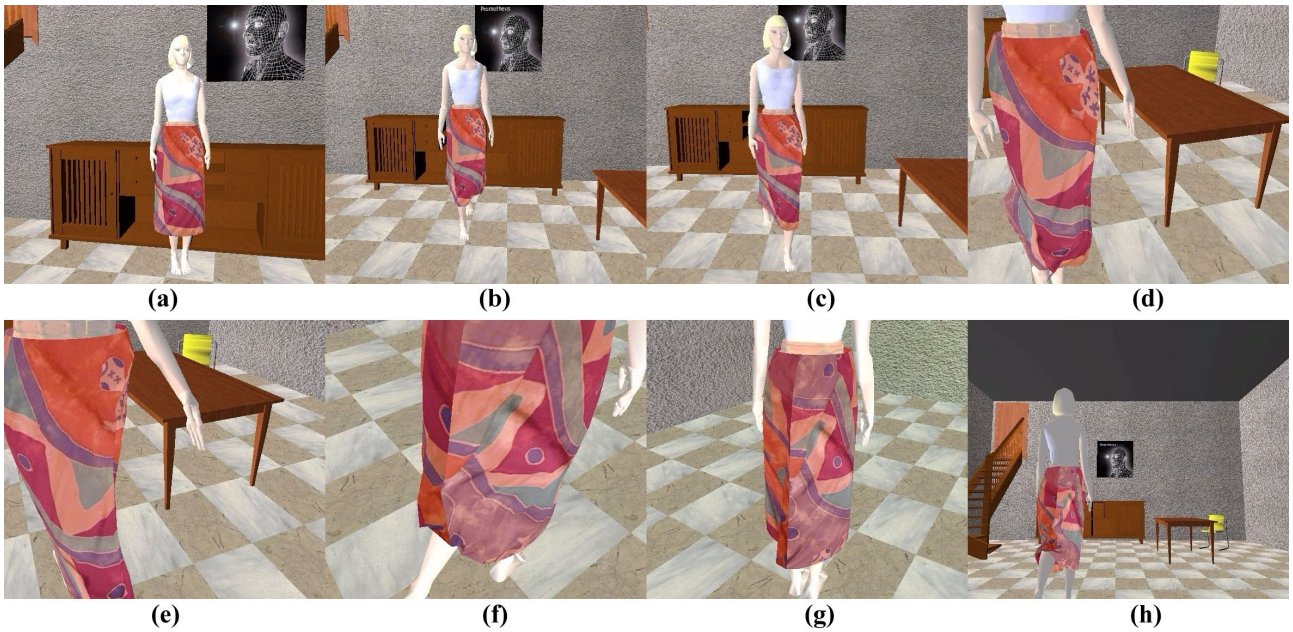


Figure 7 Basic character walking animation sequence

Acknowledgements

The Prometheus project partners include:
BBC, British Telecom, UCL, University of Surrey, QMUL, DeMontfort University, Snell & Wilcox, Avatar-Me.

This research was partially funded by the BBC and partially by the EPSRC under research grant GR/M88068/01.

References

1. J. Anderson.
Fast physical simulation of virtual cloth based on multilevel approximation strategies
Ph.D. 1998. Thesis, University of Edinburgh

2. D. Baraff, A. Whitkin
Large steps in cloth simulation
Proceedings of SIGGRAPH 1998, pp 43-54

3. B. Baumgart
Winged edge polyhedron representation
Technical report AIM-179 (CS-TR-74-320) 1972, Computer science department, Standford University

4. R. Bridson, R. Fedkiw, J. Anderson
Robust treatment of collisions, contact and friction for cloth animation
Proceedings of SIGGRAPH 2002, pp 594-603

5. S. F. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones
Adaptively sampled distance fields: A general representation of shape for computer graphics
ACM SIGGRAPH Proceedings July 2000, pp 249-254

6. M. Jones, R. Satherley
Using distance fields for object representation and rendering
Eurographics UK 19'th Annual Conference Proceedings, pp 37-44

7. M. Jones, R. Satherley
Voxelization: Modelling for volume graphics
In Vision, Modeling, and Visualization 2000, pp 319-326

8. A. Kaufman
An algorithm for 3D scan-conversion of polygons
Proceedings of Eurographics, August 1987, pp 197-208

9. M. Price, J. Chandaria, O. Grau, G. Thomas, D. Chatting, J. Thorne, G. Milnthorpe, P. Woodward, L. Bull, E-J. Ong, A. Hilton, J. Mitchelson, J. Starck
Real-time production and delivery of 3D media
Proceedings of the International Broadcasting Convention (IBC) 2002, pp 348-356

10. X. Provot
Collision and self-collision handling in cloth model dedicated to design garments
Graphics Interface 1997, pp 177-189

11. X. Provot
Deformation constraints in a mass-spring model to describe rigid cloth behavior
Graphics Interface 1995, pp 147-155

12. G. Thomas, J. Jin, T. Nibblet, Urquhatr
A versatile camera position measurement system for virtual reality TV production
Proceedings of the International Broadcasting Convention (IBC), pp 284-289

13. G. Thomas, R. Storey
TV Production in the year 2005
Montreux Symposium 1999, pp 234-238

14. T. I. Vassilev, B. Spanlang, Y. Chrysanthou
Efficient cloth model and collision detection for dressing virtual people
Proceedings of GeTech Hong Kong 2001

15. P. Volino, M. Courchesne, N. Thalmann
Versatile and efficient techniques for simulating cloth and other deformable objects
Proceedings of SIGGRAPH 1995., pp 137-144

16. P. Volino, N. Thalmann
Developing simulation techniques for an interactive clothing system
IEEE International Conference on Virtual Systems and Multimedia, September 1997, pp 109-118

17. R. Yagel, D. Cohen, A. Kaufman
Normal estimation in 3D discrete space
The Visual Computer, Vol. 8, No. 5-6, June 1992, pp 278-291

18. D. Zhang, M. Yuen
Collision detection for clothed human animation
Proceedings of Pacific Graphics 2000, pp. 328-33