Eigth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

Collision Detection and Tissue Modeling in a VR-Simulator for Eye Surgery

Clemens Wagner, Markus A. Schill, and Reinhard Männer

Institute for Computational Medicine, Universities of Mannheim and Heidelberg, Mannheim, Germany

Abstract

This paper gives a survey of techniques for tissue interaction and discusses their application in the context of the intra-ocular training system EyeSi. As key interaction techniques collision detection and soft tissue modeling are identified. For collision detection in EyeSi, an enhanced image-based approach for collisions between deformable surfaces and rigid objects is presented. By exploiting the computing power of graphics processing units, it achieves higher performance than existing geometry-based approaches. Deformation vectors are computed and used for the biomechanical model. A mass-spring approach is shown to be powerful enough to bridge the gap between low computational demands and a convincing tissue behavior.

Categories and Subject Descriptors (according to ACM CCS): I.6.3 [Simulation and Modeling]: Applications

1. Introduction

Simulators using Virtual Reality must be constructed so that the user forgets her actual surrounding and operates completely naturally in the simulated environment. All necessary senses of the user must be involved in order to achieve this feeling. Virtual Reality aims at merging real and virtually presented sensory perception, thus creating a complete immersion.

The paper presents EyeSi, a training simulator for intraocular surgery[5], which makes heavy use of Virtual Reality techniques (Fig.1). The paper focuses on the issue of updating the objects of the virtual world according to user interactions. In particular the generally required steps of collision detection and object deformation are treated. State of the art techniques are presented together with the way how they are adapted to the requirements of the simulator EyeSi.

2. EyeSi, VR Simulator for Intraocular Surgery

Vitreo-retinal surgery is one of the most demanding microsurgical tasks. The operation is performed in the interior of the eye by using a stereo microscope which makes hand-eye coordination very difficult. The surgeon must operate with submillimeter accuracy; collisions with the highly damageable retina can be fatal and must be avoided in any case. In general, two instruments are inserted into the eye. One is an

Figure 1: *Real vs. virtual eye surgery*

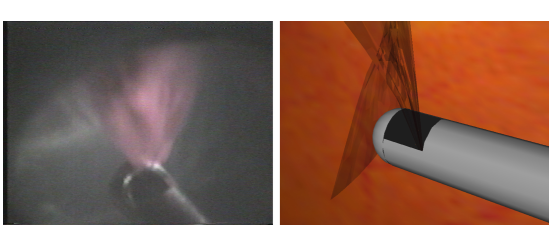

Figure 2: *Video capture (left) and simulation screenshot of vitrector-tissue interaction.*

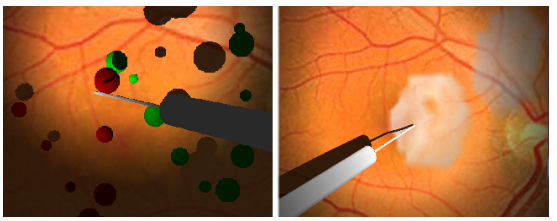

Figure 3: *Screenshots of an abstract training task (left) and of a retina relocation procedure.*

instrument for tissue interaction – e.g. a pick, a forceps or a *vitrector*, which is a hollow needle with an oscillating knife at its tip. The vitrector can suck in, cut and remove lumps of tissue (Fig.2). It is used mainly to interact with the *vitreous humor*, a gelatinuous substance that fills the eye. The other instrument is a lamp that lightens the operation area and casts shadows that are important for estimating distances between objects.

Typical pathologies that make vitreo-retinal surgery necessary include the following:

Vitrectomy. During a vitrectomy, the vitreous humor is removed and the eye is refilled with a clear liquid. This process is performed in almost all vitreo-retinal surgeries to either remove an opaque vitreous humor or to obtain free access to the background of the eye.

Removal of epiretinal membranes. Pathological changes of the vitreous humor can lead to opaque membranes. Besides hindering the sight of the patient, the membranes have a tendency to contract and lift off parts of the retina.

Retina relocation. If the retina is damaged (e.g. from an age-related macular degeneration), a recent therapy relocates the whole retina. During this process, a liquid is injected beneath the retina so that it is lifted off (Fig.3, right). The retina is then rotated so that a healthy part of it replaces the damaged macular region.

2.1. Simulator Overview

In EyeSi a thoroughly modeled mechanical setup provides the sensory perception eye surgeons are used to (Fig.1): original instruments are introduced into a mechanical model of the eye that is mounted under a facial mask. The mechanical eye has the same rotational degrees of freedom as the human eye in its socket. The effect of eye muscles, turning the eye back into its rest position, is modeled by sets of springs on each rotation axis. Instrument and stereo microscope functions are controlled by a footpedal that is attached to the system. A detailed computer graphical model is updated according to user interaction and visual feedback is provided through the mimic of a stereo-microscope; two different views are calculated and displayed on two small LCD displays that are mounted in the shape of a microscope eyepiece. During eye surgery the surgeon has to rely entirely on the visual clues she perceives to estimate e.g. distances between objects. Therefore great effort was spent on a realistic visualization of the operation scenario, including lighting effects and shadows.

A self designed optical tracking system links real and virtual worlds in EyeSi. It measures the instrument movements and the eye rotation. Therefore three small cameras are fixed below the mechanical eye and detect small color markers that are mounted on the instruments and on the mechanical eye. For the image processing task an FPGA-based hardware module was developed, that is fast enough to perform the image processing operations without buffering the video data stream. The cameras capture one field with the PAL frequency of 50Hz. About 2 ms later the tracking information is available in the computer, so that nearly no latency is added to the readout time of 20ms. The tracking system works with subpixel accuracy. The reconstruction of the 3D coordinates has a relative accuracy of at least $50\mu m$.

The EyeSi software provides several training modules that range from abstract tasks to whole operations. The abstract tasks emphasize particular training aspects, e.g. bimanual instrument handling, estimating distances by shadows (Fig.3, left) or working with a special instrument type. In the abstract tasks, object interaction only occurs between simple geometric objects, whereas operation modules incorporate the realistic interaction between instruments and pathological tissue.

EyeSi is currently in a clinical validation process. Preliminary results[1, 2] show that the simulator is well suited as a training device.

2.2. Simulator Requirements

EyeSi runs on an off-the-shelf PC, currently Athlon 1,4 GHz AGP4x with an nVidia GeForce 2 GTS graphics processing unit. The given performance measurements refer to this system.

One of the most important requirements on an immersive Virtual Reality is a low system latency[3]. By our experience, a latency over 50ms cancels the effect of immersion in EyeSi. The stereo display has an update rate of 85Hz. As it works field-interlaced, this results in a 42.5Hz refresh of both channels (23.5ms). By synchronizing to the display refresh, EyeSi avoids latencies through temporal aliasing.

The current graphics rendering consists of four passes (two passes for stereo rendering. For each stereo channel, one additional pass for shadow calculation[4]). In total, this takes 15ms, therefore the system cannot synchronize to the full display update rate. Our development goal was to find tissue interaction techniques that allow for synchronizing to the stereo display refresh time of 23.5ms. Subtracting the rendering time, this leaves 8.5ms for collision detection and tissue simulation.

Note that by reaching this goal, the total system latency (simulation, rendering and asynchronuous tracking) is below 50ms, given an average tracking latency of 25ms.

3. Collision detection

The general problem of collision detection is of complexity $O(n^2)$: in a set of n primitives, each element can interfere with any other (*all-pairs problem*). Therefore, collision detection can become a major bottleneck in simulations where complex object interactions have to be computed – in computer graphics, robot motion planning, physical simulations or Virtual Reality systems.

3.1. Geometry-based approaches

A way out of the all-pairs problem is to restrict the spatial and temporal properties of the objects: the Lin-Canny algorithm[8] tracks closest features (vertices or edges) between two convex polytopes. If the motion of the polytopes between two iterations is small (*temporal coherence*), only a local search has to be performed to keep track of the closest feature pair. Then, the algorithm runs in *almost constant time* (Lin and Canny use the term *expected constant time*. The algorithm runs in constant time, if it only has to verify that the pair of closest features did not change. But since a local search on an object with complexity N may take $O(\sqrt{N})$ steps, *almost constant time* is a better description[7]). The Lin-Canny algorithm is part of the collision detection package I-COLLIDE[9], where it is combined with a bounding-box approach.

The GJK-Algorithm[12] finds the closest points between convex polytopes by iteratively approximating the Minkowsky difference. If one only has to decide whether two objects intersect, distance information is not necessary. In this case, GJK can search with lesser computational effort for a *separating axis* between objects, indicating non-collision. For a recent implementation[17] that exploits temporal coherence, the author states that collision detection between convex polyhedra is five times faster than in the implementation of the Lin-Canny algorithm in I-COLLIDE. The separating axis algorithm is integrated into the collision detection libraries SOLID and Q-COLLIDE[11].

Many approaches decompose non-convex objects into a hierarchy of bounding-volumes with convex shape. Each leaf of the bounding-box corresponds to a primitive, e.g. a triangle or a convex polygon. Collision detection between objects is split into a *broad phase* where the bounding volumes are tested against each other and a *narrow phase*, where a collision detection between the primitives is performed. It depends on the configuration whether axis-aligned bounding boxes (AABB) as in SOLID[16], more tightly fitting oriented bounding boxes (OBB) as in RAPID[15] or even bounding spheres[14] are the method of choice. In scenarios where objects are in very close contact, tight-fitting OBBs will be faster. AABBs need less storage and are faster to build or update. Spheres are very easily tested against collisions, but finding a tight-fitting sphere to a given object is a non-trivial problem.

A very simple approach to test object overlaps is to partition the simulation space itself into cells of constant or varying size and maintaining an occupancy map for each cell. If two objects A and B do not share at least one cell, they do not collide[10].

Refer to [13] for a more detailed survey of geometric collision detection algorithms.

3.2. Image-based approaches

A special way of partitioning space is performed in the rasterizing stage of a 3D graphics system. In a parallel projection, the cells are formed by cuboids, in a central projection by frustums. The cells are bounded by the near and far clipping planes; their size is given by the size of the corresponding rasterized pixel. This leads to a trivial rejection rule: *Objects that are not rasterized at the same pixel do not collide*.

The reverse conclusion does not necessarily hold, therefore it was proposed[18] to save all depth values and the corresponding object identifiers for a rasterized pixel into a sorted list. The list entries are then grouped into pairs and the collision criterion reads as follows: *If a pair with two different object identifiers exist, then the corresponding objects do collide.*

The special charm of this approach is that the collision detection problem is formulated in terms of computer graphics. In principle, this enables the algorithm to use graphics processing units (GPU) for computing collisions. Unfortunately the architecture of current graphics accelerators does not allow for all needed operations:

1. It is not possible to render a sorted list of depth values into the depth buffer.
2. Rasterized depth values do not carry information about the originating objects.
3. The content of the buffers cannot be analyzed pixelwise by the GPU in the depth buffer. This has to be done by CPU operations after reading the buffer back to main memory (slow).

Several workarounds have been proposed. [22] uses clipping planes to describe objects with very simple geometries. In [19] a weaker version of convexity is given with the following definition:

An object is called π-convex at point P *if the intersection of a line through* P *with given direction* π *and the object is connected.*

It is shown that stencil buffer comparisons are sufficient to sort the depth values for π-convex objects. This approach

is further enhanced in [20] and [21] by using bounding-box information to compute minimal overlapping regions between the rasterizations, thus minimizing the cost of buffer readback and analysis.

3.3. Collision detection in EyeSi

In EyeSi different collision types have to be considered: (1) collisions between two instruments, (2) collisions between an instrument and the eyeball, (3) collisions between an instrument and geometric bodies in abstract training tasks and (4) interactions between an instrument and pathological tissue in a simulated surgery.

Case 1. No collision detection is necessary: the dimensions of the "real" instruments which the user manipulates inside the eye and the dimensions of the "virtual" instruments are identical. Provided that latency and accuracy of the tracking system are high enough, objects in the "real" and in the "virtual" world collide at the same time.

Case 2. The "real" instruments are inserted into the mechanical eye through two predrilled holes. Therefore, any point on the eyeball they can touch is first reached by the tip of the instruments. We approximate the tip by a point P and calculate the distance d between P and the spherical eyeball. A collision is reported if d falls under a threshold.

Case 3. This case is currently restricted to interactions between a given instrument and simple rigid objects that are composed of only a few convex primitives. Since this situation does not imply high polygon counts or a large number of objects, any of the above mentioned geometry-based libraries is fast enough.

Case 4. Interaction with pathological tissue is more demanding. Geometry-based approaches have proven to be unsuitable for this type of interaction: The tissue's shape is a non-convex surface with a comparably high triangle count and continuously changing geometry and topology. Therefore, it is difficult to decompose it into convex primitives; a pre-built bounding-box tree soon looses its validity.

The library SOLID can dynamically update an AABB-tree. As section 3.4 will discuss, it is fast enough for reasonable triangle counts. However, the provided collision information is restricted to contact points and collision planes, which is not enough for an appropriate collision response of the deformable membranes in EyeSi.

Instead we use an adaption of the image-based approaches discussed above. The extension of π-convexity (Section 3.2) to surfaces leads to the following definition (Fig. 4):

If the intersection of a connected volume V with a surface S subdivides V into two volumes V_1 and V_2 that are π-convex for each point in V, then S is called π-convex in V.

If no volumetric tissue and no self-collisions are considered, the pathological tissue in EyeSi forms π-convex surfaces for volumes V with reasonable sizes. Such a volume

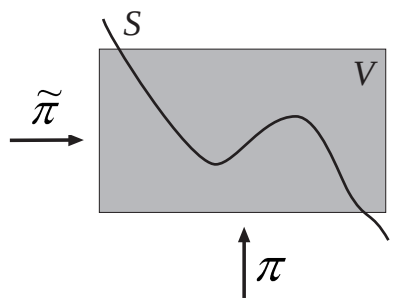

Figure 4: *S is π-convex in V, but not $\tilde{\pi}$-convex in V.*

is e.g. a bounding box around the grabs of a forceps. Due to this observation, collisions between instruments and the membrane can be detected with an image-based collision detection approach.

For collisions between a surface and a solid object, we modified the image-based collision detection algorithm[18, 19, 20, 21] and added a color-based feature identification. For an instrument I and a deformable membrane M it reads as follows:

```
01:proc ZCOLL (I, M)
02:  if (not M.hasColorIDs) M.assignColorIDs;
03:  enableZBuffer (<=);
04:  enableColorBuffer;
05:  I.setCamera;
06:  M.render;
07:  ZM = readZBuffer;
08:  CM = readColorBuffer;
09:  I.render;
10:  ZI = readZBuffer;
11:  for each pixel (x,y) do
12:    if (ZI(x,y)!=ZM(x,y))
13:      M.addCollidingFeature
         (x,y,CM(x,y),ZI(x,y)-ZM(x,y));
14: endproc.
```

First unique colors are assigned to the vertices and triangles of M (line 2). This has to be done only once. After enabling the depth and color buffers (3-4), the parameters of the rendering camera are set so that it points at the tip of the instrument (5, Fig. 5). For a uniform resolution over the rendering area, a parallel projection is applied. M is then rendered (6) and buffers are read back (7,8). After rendering I (9) and again reading back the depth buffer (10), the buffers are analyzed (11-13): a collision has happened if the instrument penetrated the volume between the membrane and the camera. This is reflected by a change in a depth buffer entry after the instrument was rendered. The corresponding color buffer pixel holds the color of the feature that collided with the instrument. As a guess for a deformation, the vector $(x,y,ZI(x,y)-ZM(x,y)$ is added to the corresponding feature $CM(x,y)$ of M. If a feature is rasterized to several pixels, it holds more than one deformation vector. The accu-

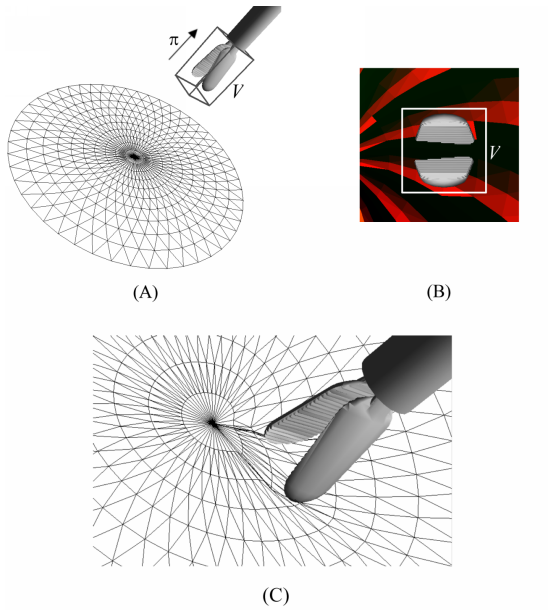

Figure 5: *(A) For the image-based collision detection, the volume V is rendered with a parallel projection in direction π. (B) Each triangle of the membrane has a unique color. Triangles with hidden parts are colliding. (C) Application of deformation vectors (note that, due to its varying grid size, the shown triangulation is not suitable for tissue simulation).*

racy of ZCOLL depends on the resolution of the viewport. If the projection of a feature is smaller than one pixel, it is not guaranteed that the feature is rasterized: Given the lateral rendering resolution R, not more than R^2 colliding features can be detected.

Currently this approach is used in EyeSi for forceps-membrane and, in a slightly modified version, for vitrector-membrane interaction. In both cases, only the triangles of the membrane are used as features.

Note that if no information for tissue deformation is needed, buffer readback can be omitted and buffer analysis left to the GPU. M is rendered in black, I in white. All pixels of the color buffer are XOR-copied by the GPU to one pixel. Iff it is white, a collision has happened.

3.3.1. Forceps-membrane interaction.

If the grabs of the forceps are open, the membrane is pushed away by adding a vector $\vec{v}_i$ to the position of each colliding node i. If grabs are closing and their angle falls below a threshold, the nodes' positions are bound to the position of the instrument tip.

For calculating $\vec{v}_i$, the set D_i of all deformation vectors of the adjacent triangles of node i is considered. The length of $\vec{v}_i$ is calculated by taking the maximal length of the elements

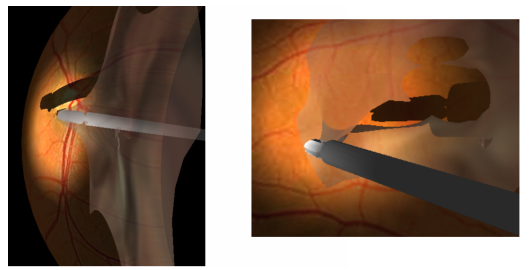

Figure 6: *Forceps-membrane interaction*

of D_i, the direction by averaging the elements of D_i (Eq.1 and Fig.7).

$$\vec{v}_i = \max_{\vec{d} \in D_i} |\vec{d}| \cdot \frac{\sum_{\vec{d} \in D_i} \vec{d}}{|\sum_{\vec{d} \in D_i} \vec{d}|} \tag{1}$$

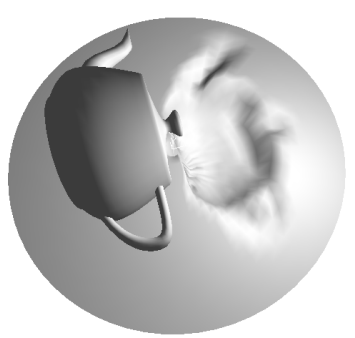

Figure 7: *Application of deformation vectors on a plastic membrane after a collision with a teapot.*

3.3.2. Vitrector-membrane interaction.

We assume that the suction force of the vitrector is applied to those parts of the tissue that fall into a suction cone C. Position and orientation of C are determined by the opening of the vitrector (Fig.8).

It is straightforward to modify the ZCOLL procedure for this situation: place the camera in the opening of the vitrector and render the membrane only. By using a central projection, the suction cone of the vitrector is approximated by the viewing frustum of the camera. To those nodes that fall into the frustum a force is applied that is proportional to the reciprocal distance to the tip of the vitrector.

Since no deformation vectors are needed, the time-consuming steps (7), (9) and (10) of ZCOLL can be omitted. (12) and (13) are replaced by an analysis of the color buffer only (120,130).

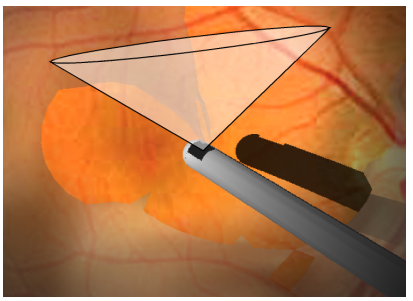

Figure 8: *Suction cone of the vitrector.*

```
120:    if (CM(x,y)!=BLACK)
130:      M.addCollidingFeature
            (x,y,CM(x,y),0);
```

3.4. Performance

The cost of image-based collision detection is the sum of the following complexities:

1. Transforming the geometry of n_T triangles and rasterizing in a quadratic viewport with lateral resolution R takes $O(n_T \cdot R^2)$ steps.
2. The cost of buffer readback is $O(R^2)$ as well.
3. The retrieval of a colliding feature by its color takes $O(1)$, if the features are saved in an optimal hash table. As the number of detected features is bounded by R^2, total cost is $O(R^2)$.

This gives a linear behavior $O(n_T \cdot R^2)$ in the total number of triangles, as long as the resolution R is left constant. If no narrow phase follows and one wants to detect all triangles, the resolution R of the viewport has to be increased with $R^2 \sim n_T$, resulting in the previously discussed quadratic behavior. The method is applicable as long as the surface is π-convex in the bounding volume of the interaction.

Tab.1 shows the computational costs of the different parts of the ZCOLL procedure in EyeSi. The times were measured for forceps-membrane interaction with membranes of two different resolutions (1k triangles and 3k triangles). The forceps is composed of 2k triangles.

Geometry processing and rasterization is done with a triangle throughput of 8 MTriangles/sec. (the full triangle performance of 25 MTriangles/sec. is not achieved because the mesh representation does currently not use triangle strips).

The transfer of the rendered data back to main memory and its analysis by the CPU define an upper bound for the rendering resolution: One color buffer readback is needed for obtaining the colliding triangles, two depth buffer readbacks for calculating the deformation vectors. Although the AGP4x-bus transfer rate is 1066 MBytes/sec., the transfer rate of 32×32 depth values with 4 bytes precision is only 9 MBytes/sec – even for a 400×400 pixel readback where the AGP transfer overhead is less important, we measured 60 MBytes/sec. Apparently, buffer readback is not an issue for the designers of current graphics adapters.

Membrane size Instrument size	1k triangles 2k triangles	3k triangles 2k triangles
Rendering time (A)	0.4ms	0.6ms
ZBuffer readback (B)	0.4ms	0.4ms
Color buffer readback (C)	0.3ms	0.3ms
Analysis & retrieval (D)	0.8ms	0.8ms
Overall (A+2B+C+D)	2.3ms	2.5ms

Table 1: *Computational costs of image-based collision detection with ZCOLL. The rendering throughput is 8 MTriangles/sec., the buffer size 32×32 pixels. 40 triangles are colliding.*

Membrane size Instrument size	1k triangles 2k triangles	3k triangles 2k triangles
ZCOLL run	2.3ms	2.5ms
RAPID preprocessing	50ms	80ms
RAPID run	0.7ms	1.0ms
SOLID tree update	1.5ms	4.5ms
SOLID run	<0.1ms	<0.1ms

Table 2: *Comparison of ZCOLL with the geometry-based libraries RAPID and SOLID. For deformable objects, RAPID preprocessing and SOLID tree update have to be repeated in each step.*

The results are compared in Tab.2 with the geometric collision detection libraries RAPID and SOLID. The construction of an OBB tree as in RAPID needs a costly preprocessing phase that is not suitable for realtime deformations. SOLID can refit its AABB tree after an object deformation in a time that is comparable to the runtime of ZCOLL. For collisions with few triangles (as given in EyeSi), large parts of the bounding box tree can be clipped away so that collision detection itself is fast enough both with RAPID and SOLID. However, none of the packages can be used in EyeSi: SOLIDs collision information does not include colliding triangles whereas RAPID cannot be used for deforming objects. In addition, for situations with a higher triangle count or more colliding triangles, the performance of both packages falls considerably behind the image-based approach.

4. Tissue Simulation

Tissue simulation in medical simulators has to meet two requirements: accuracy and computing time. Unfortunately there is an inherent trade-off between the physical correctness of a model and its computational costs. One can distin-

guish *descriptive* and *physical* models[6]. Descriptive models mimic a desired property without exploiting physical knowledge about the observed system. Free parameters in the descriptive model are used to fit modeled and observed behavior. There is no straightforward mapping between physical properties and model parameters[25]. In contrast, physical models are constructed using physical laws. Therefore the parameters of a physical model reflect physical properties that can be measured in the real-world process.

4.1. The Finite Element Method

The standard approach for physical modeling is the finite element method (FEM), where a problem is stated in a continuous way, but solved for each element of a discretized definition space[23]. Simple interpolation functions within these elements make the problem numerically tractable; appropriate boundary conditions guarantee a physically correct solution. FEM is widely used in offline simulations, e.g. for surgery planning, where high precision is prior to realtime response.

4.2. Mass-Spring Models

The most common approach for realtime tissue simulations are mass-spring systems. In contrast to FEM, they discretize not only the definition space, but the problem formulation itself. An object is decomposed into a mesh of vertices and springs. Mass and damping are assigned to the vertices; the behavior of the springs is governed by a deformation law (typically Hooke's law). Despite their physical terminology, mass-spring meshes are descriptive: in their standard formulation they lack physical properties like volume preservation, anisotropy or a mapping of given elasticity properties onto the model.

The behavior of a mass-spring mesh depends heavily on its topological and geometric configuration. In addition, configurations with large forces (e.g. nearly rigid objects) lead to stiff differential equations with a poor numerical stability, requiring small time-steps of the integration. Nevertheless a mass-spring model is an easily understandable concept, which is simple to implement and has low computational demands. Therefore, mass-spring models are used in a wide range of computer graphics and VR applications, e.g. in the animation of facial expressions[28, 29], the modeling of inner organs[24], or the simulation of cloth[26].

4.3. Hybrid approaches

There is a strong tendency to narrow the gap between FEM and mass-spring modeling: FEM simulations are accelerated by precomputing parts of the solution or by simplifying the model. In the context of hepatic surgery simulation, [30] suggest a hybrid model, where linear combinations of precomputed elementary deformations are used if the FEM configuration does not change. Regions with a change in configuration (as it appears when cutting or tearing) are deformed by a *tensor-mass-model*, where masses and dampings are assumed to be concentrated on the vertices (as in a mass-spring model) whereas the elasticity properties of the tissue are modeled in a continuous representation (as in the model of a FEM simulation). By using an explicit integration scheme, the costly inversion of the rigidity matrix is avoided.

Mass-spring-models are enhanced to overcome some of their physically inadequate properties. For stiff materials, [26] improves the behavior of a mass-spring-model by readjusting the vertices after each explicit integration step, if forces are too high to guarantee a stable solution in the next step. In [32], a mass-spring model is adapted to the elements of a volumetric mesh, so that both volume preservation and discretization-independent anisotropy can be modeled. [27] uses a more advanced implicit integration scheme for mass-spring-models that allows for large time-steps even for stiff equations. Calculating an iteration step in an implicit scheme implies solving a linear system of equations or inverting a (sparse) matrix, which makes this approach more stable, but as well more time-consuming in comparison to a one-step explicit integration.

4.4. ChainMail

In a mass-spring model, time as a variable plays an important role: deformations are calculated in a certain timestep by considering first and second temporal derivatives of the point masses' positions. In [31] the *ChainMail* algorithm is proposed where time-dependent dynamic behavior is replaced by the notion of *displacement*. As a mass-spring model, ChainMail works on a connected neighborhood of nodes. Each node must satisfy a given maximum and minimum distance constraint to its adjacent nodes. When an element is moved and one of its constraints is violated, a displacement of the respective neighboring element takes place. In an iterative process the initial deformation is propagated through the structure. Within one $O(n)$ iteration, the algorithm finds a new valid configuration for any given disturbance. Therefore, ChainMail can handle arbitrarily stiff objects with low computational effort. Limitations of the original approach include the incapability of handling inhomogeneities, relaxation and volume preservation and a strong anisotropic behavior. Extensions for ChainMail that handle these problems have been developed[33, 34].

4.5. Tissue simulation in EyeSi

The choice of tissue simulation algorithms for EyeSi is strongly restricted by its real-time requirements. Although we showed that Enhanced ChainMail, which can work on inhomogeneous tissue, is a suitable approach for a volumetric model of the vitreous humor[33], it could not yet be incorporated into the EyeSi simulator because of the high computational demands of volume visualization.

Currently pathological membranes in EyeSi are simulated

with a mass-spring model and a simple explicit euler integration. Stability problems as discussed above are avoided by several changes on the standard mass-spring formulation: maximum values a_{max} and v_{max} for acceleration and velocity of each node can be set. If a_{max} is exceeded by a certain percentage p_{tear}, the spring exerting the highest force on the node is deleted. Note that $v_{max} = 0$ is allowed. This describes a system where the kinetic energy of the nodes is in any case damped to zero within a timestep's length.

In EyeSi, this model is currently used for two different problems: (1) A membrane that is situated in front of the retina must be removed by using a forceps, a pick and a vitrector (Fig. 6). As in reality, the rim of the membrane is connected to the retina. In the mass-spring model, the corresponding nodes are not moved. If the force on the rim nodes exceeds a certain limit, the simulator warns the surgeon that he is going to lift off the retina (the treatment of an accidentally detached retina will be incorporated into this training module).

(2) By injecting a balanced salt solution beneath the retina, it can be intentionally lifted off and relocated (Fig.3). Therefore the whole retina was modeled as a mass-spring mesh. At the start of the procedure, every node is fixed on the rear hemisphere of the eye (the *choroidea*). Retina detachment is started when the injection needle penetrates the retina by a small amount at a point x_0 and the surgeon starts to inject salt solution with a certain pressure p. If the pressure exceeds a given limit, nodes below a certain distance d to x_0 are detached (i.e. they are considered in the mass-spring-simulation). To each node N_i with position x_i, a constant external force with direction $x_i - x_0$ is applied. If p increases, d is increased proportionally. By withdrawing the needle and repeating the procedure at another position, the retina is lifted off and forms several hemispherical bubbles. Currently, the smooth handling of the needle is one of the main training goals. If the projection of the needle's velocity vector onto the surface of the retina exceeds a given value, the simulator warns the surgeon that he is going to tear the tissue. In a second step that is currently under development, the re-attachment of the retina will be trained.

For both situations, the mass-spring parameters were determined in an iterative process in co-operation with our clinical partners. Although we currently can not cover the whole range of occurring tissue behaviors, we can already get very close to a significant part of them.

4.6. Performance

Table 3 shows the times that is needed by mass-spring-meshes of different resolutions. As the geometry of the mesh is changed after each simulation step, it has to be transferred to the graphics adapter again. Remarkably, with a performance of 600 ktriangles/sec., this takes more time than the simulation itself.

Membrane size	Simulation	Transfer
400 triangles (600 springs)	0.5ms	0.7ms
1k triangles (1.5k springs)	1.5ms	1.7ms
3k triangles (4.5k springs)	4.0ms	5.0ms

Table 3: *Performance of mass-spring-simulation and time to transfer a deformed mesh to the GPU.*

Collision detection	2.3ms
Tissue simulation	1.5ms
Geometry transfer	1.7ms
Misc	1.0ms
Graphics rendering	15.0ms
Overall	22ms (45Hz)

Table 4: *Computational requirements for the different tasks of the EyeSi software.*

The transferred mesh is used both for the visualization of the current timestep and as a basis for collision detection for the next timestep.

5. Summary

We presented state-of-the-art approaches for collision detection and tissue simulation and discussed their application in EyeSi, a Virtual Reality system for the training of vitreo-retinal surgery. It was shown that geometry-based approaches for collision detection are not suitable for interaction with deformable objects. An image-based approach was enhanced for interactions between deformable membranes and instruments and showed sufficient performance. The mass-spring model that was used for tissue deformation is stable enough to model the properties of pathological tissue in the eye and has very low computational demands. For simulation of volumetric deformable objects, we previously implemented a ChainMail approach[33]. However, due to the high visualization costs of volumetric objects, it is currently not in use.

Table 4 shows how the performance of the current EyeSi system distributes on collision detection, tissue simulation and visualization. The complexity of the deformable meshes is 1500 springs and 1000 triangles.

By the tissue interaction approaches that were described in this paper, the initial goal of achieving an update rate below 23.5ms (Sec.2.1) is reached.

The VR techniques that are used in EyeSi are well suited for a convincing training simulation. EyeSi will be complemented by advanced scoring functions and a GUI and brought to market in Summer 2002 by the startup company VRMagic (www.vrmagic.de).

References

1. S. Rabethge, H.-J. Bender, J.B. Jonas. "In-vitro-training of Membrane-Peeling using a Pars-Plana-Vitrectomy-Simulator", 99. Jahrestagung der Deutschen Ophthalmologischen Gesellschaft (2001).
2. B. Harder, H.-J. Bender, J.B. Jonas. "Pars-Plana-Vitrectomy-Simulator for learning vitreoretinal surgical steps: first experiences", 99. Jahrestagung der Deutschen Ophthalmologischen Gesellschaft (2001).
3. M.C. Jacobs, M.A. Livingston, A. State. "Managing latency in complex augmented reality systems", *1997 Symposium on Interactive 3D Graphics*, pp. 49–54 (1997).
4. M.J. Kilgard. "Improving shadows and reflections via the stencil buffer", *Technical report, nVidia Corporation, www.nvidia.com* (2000).
5. M.A. Schill, C. Wagner, M. Hennen, H.-J. Bender, R. Männer. "EyeSi – a simulator for intra-ocular surgery", *Medical Image Computing and Computer Assisted Intervention – MICCAI '99 proceedings*, pp. 1166–1174 (1999).
6. M.A. Schill, C. Wagner, R. Männer, H.-J. Bender. "Biomechanical modeling techniques and their application to the simulation of brain tissue", *Navigated Brain Surgery*, Aachen (1999).
7. B. Mirtich. "Efficient algorithms for two-phase collision detection", *Technical Report TR-97-23, Mitsubishi Electric Research Laboratory* (1997).
8. M.C. Lin, J.F. Canny. "A fast algorithm for incremental distance calculation", *Proceedings of the IEEE International Conference on Robotics and Automation* **2**, pp. 1008–1014 (1991).
9. J.D. Cohen, M.C. Lin, D. Manocha, M.K. Ponamgi. "I-COLLIDE: An interactive and exact collision detection system for large-scale environments", *Proceedings of the ACM Interactive 3D Graphics Conference*, pp. 189–196 (1995).
10. S.F. Gibson. "Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects", *Proceedings of the Sixth Eurographics Workshop on Visualization in Scientific Computing*, pp. 10–24 (1995).
11. K. Chung, W. Wang. "Quick collision detection of polytopes in virtual environments", *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 125–131 (1996).
12. E.G.Gilbert, D.W. Johnson, S. Keerthi. "A fast procedure for computing the distance between complex objects in three dimensional space", *IEEE Journal of Robotics and Automation* **4**, 2, pp. 193–203 (Apr. 1988).
13. M.C. Lin, S. Gottschalk. "Collision detection between geometric models: a survey", *Proceedings of the IMA Conference on Mathematics of Surfaces* (1998).
14. P.M. Hubbard. "Collision detection for interactive graphics applications", PhD thesis, Department of Computer Science, Brown University (1994).
15. S. Gottschalk, M.C. Lin, and D. Manocha. "OBBTree: A hierarchical structure for rapid interference detection", *Proceedings of SIGGRAPH '96*, pp. 171–180 (1996).
16. G. van den Bergen. "Efficient collision detection of complex deformable models using AABB trees", *Journal of Graphics Tools* **2**(4) (1997).
17. G. van den Bergen. "A Fast and robust GJK implementation for collision detection of convex objects", *Journal of Graphics Tools* **2**(2) (1999).
18. M. Shinya, M. Forge. "Interference detection through rasterization", *Journal of Visualization and Computer Animation* **(**2), pp. 132–134 (1991).
19. K. Myszkowski, O.G. Okunev, T.L. Kunii. "Fast collision detection between complex solids using rasterizing graphics hardware", *The Visual Computer* **11**, pp. 497-511 (1995).
20. G. Baciu, W.S-K. Wong. "Rendering in Object Interference Detection on Conventional Graphics Workstations", *Proceedings of Pacific Graphics '97* (1997).
21. G. Baciu, W.S-K. Wong, H. Sun. "RECODE: An Image-Based Collision Detection Algorithm", *Proceedings of Pacific Graphics '98* (1998).
22. J.-C. Lombardo, M.-P. Cani, and F. Neyret. "Real-time collision detection for virtual surgery", *Proceedings of Computer Animation '99*, pp. 33–39 (May 1999).
23. K.L. Bathe. "Finite element procedures in engineering analysis", Prentice Hall (1982).
24. U. Kühnapfel, H.K. Cakmak, H. Maaß. "Endoscopic surgery training using virtual reality and deformable tissue simulation", *Computers & Graphics* **24** pp. 671–682 (2000).
25. A. Radetzky, A. Nürnberger, D.P. Pretschner. "Elastodynamic shape modeler: A tool for defining the deformation behavior of virtual tissues", *RadioGraphics* **20** pp. 865–881 (2000).
26. X. Provot. "Deformation constraints in a mass-spring model to describe rigid cloth behavior", *Graphics Interface*, pp. 147–155 (1995).
27. D. Baraff, A. Witkin. "Large steps in cloth simulation", *Proceedings of SIGGRAPH 98*, pp. 43–54 (1998).
28. S. Platt, N.I. Badler. "Animating facial expressions", *Proceedings of SIGGRAPH 81*, pp. 245–252 (1981).

29. Y. Lee, D. Terzopoulos, K. Waters. “Realistic modeling of facial animation”, *Proceedings of SIGGRAPH 95*, pp. 55–62 (1995).

30. S. Cotin, H. Delingette, N. Ayache. “A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation”, *Visual Computer* **16**(8), pp. 437–452 (2000).

31. S.F.F. Gibson. “3D ChainMail: A Fast Algorithm for Deforming Volumetric Objects”, *1997 Symposium on Interactive 3D Graphics*, pp. 149–154 (2000).

32. D. Bourguignon, M.-P. Cani. “Controlling anisotropy in mass-spring systems”, *Proceedings of the 11th Eurographics Workshop*, pp. 113–123 (2000).

33. M.A. Schill, S.F.F. Gibson, H.-J. Bender, R. Männer. “Biomechanical simulation of the vitreous humor in the eye using an enhanced ChainMail algorithm”, *Lecture Notes in Computer Science*, Vol. 1496, pp. 679–687, Springer (1998).

34. M.A. Schill. “Biomechanical soft tissue modeling – techniques, implementation and applications”, PhD Thesis, Institute of Computer Science V, University of Mannheim (2001).