Virtual Try On: An application in need of GPU optimization

Bart Kevelham
MIRALab – University of Geneva
7 Route de Drize
1227 Carouge, Geneva, Switzerland
+41 22 3790118
kevelham@miralab.ch

Nadia Magnenat-Thalmann
MIRALab – University of Geneva
7 Route de Drize
1227 Carouge, Geneva, Switzerland
+41 22 3790969
thalmann@miralab.ch

ABSTRACT
In this paper we discuss our Virtual Try On (VTO) from the perspective of performance. The VTO is an application which assists users in the evaluation of physically simulated garments on a size-correct virtual representation of their own body. Where in previous work we discussed its features and implementation, here we analyze its performance and identify those components which would benefit most from additional optimization efforts. We then detail our ongoing efforts and achieved results with regards to the optimization of the application, by making use of GPU computation, focused on the accurate physical simulation of garments.

Categories and Subject Descriptors
I.3.8 [**Computer Graphics**]: Applications

General Terms
Performance.

Keywords
virtual try on, cloth simulation, gpu optimization

1. INTRODUCTION
The Virtual Try On is an application that allows customers to evaluate garments through physical simulation of these garments on an animated virtual avatar sized to their measurements. Besides allowing for the evaluation of style, an ideal VTO should also address the issue of "fit". That is, how a certain garment fits on the customer's body, either in a static pose or in a dynamic scenario such as a walk. To this effect, our VTO integrates three main modules:

- A body sizing module which generates size-accurate avatars based on a generic template [1].
- A motion retargeting module which adapts the prerecorded animation to the model [17].
- A garment simulation module which simulates the dynamic behavior of the 3D garments [33].

Conference'10, Month 1–2, 2010, City, State, Country.

Through this integration – which also includes various other functionalities - and by means of a clean API, the VTO has been used in a wide variety of application scenarios. Examples range from web-based applications [18] and public demonstrators [19], to remote AR scenarios [16] and web services.

Though in terms of functionality this VTO has all the features we consider essential, through its various applications we have found that its performance is not optimal. In this paper we will therefore address the issue of VTO performance and our ongoing progress in its optimization using GPU computation.

The paper is organized as follows: We will first take a look at the Virtual Try On and its functionalities in section 2. We will present an analysis of its performance and identify those bottlenecks whose performance improvement would benefit us most. In section 3 we will present previous work related to the GPU optimization of physical simulation, focusing on numerical methods and the related linear algebra operators, as well as mechanics. Subsequently in section 4 we will present our ongoing efforts towards the GPU optimization of our physical garment simulations using the CUDA API [22] as well as the achieved performance results. Conclusions and future work will be detailed in section 5.

2. THE VIRTUAL TRY ON
Where a Virtual Try On system certainly is of value when evaluating garments from the perspective of style, it would be of benefit to also address the issue of "fit". That is, how a certain garment fits on the customer's body, either in a static pose or in a dynamic scenario such as a walk.

The issue of style is relatively easily addressed by allowing the user to "dress" a virtual model with various garment meshes, which can be customized along the options provided by the designers in their collections. One can consider such possibilities as various fabrics, colors and perhaps accessories.

The issue with introducing fit into a Virtual Try On is the additional requirements this imposes. Evaluating fit will only ever work when we base our application on a 3 dimensional avatar which has virtual body measurements exactly matching those of the customer. We have chosen to provide this by means of a resizable template body model [11]. When including animation however, care needs to be taken of the discrepancies that arise between the usually prerecorded body animation and the new morphology of the virtual model. With this issue in mind, we have included a motion retargeting module which maintains the coherence between morphology and animation [17].

One of the bigger issues to address however is that of the garments itself. To evaluate fit, it is no longer enough to make use

of static meshes. The fit of a garment addresses the interplay between a garment and the body. To virtually evaluate this, we use a system for the physical simulation of garments [33] whose input consists of virtual garments constructed from accurate 2 dimensional pattern data, as their real counterparts would be.

With regards to the physical simulation itself there are two competing issues. We ideally would like the system to perform in real-time. But on the other hand, to properly evaluate garment fit, we want our simulation to be accurate. Visually compelling garment simulation in real-time is achievable. Physically accurate simulation is a far more difficult proposition.

2.1 Application Performance

In our extensive use of the VTO under various circumstances we noticed that, despite our best efforts, the performance of the VTO is not what we would like it to be. The question then becomes: "In what way does the VTO underperform and what causes this?"

When regarding VTO performance, there are two areas which take a significant amount of computational effort which might be at the root of our perceived problems. One is the body, including its animation, resizing and animation retargeting. The other is the real-time garment simulation.

With regards to the body and its interactions, we have performed a series of timing runs of the body, in full animation while simultaneously performing sizing operations. The body at this point is not wearing any garments, so there is no influence of the garment simulation. Figure 1 shows a fairly typical graph of application performance of one such instance of measurement over a period of 20 seconds.

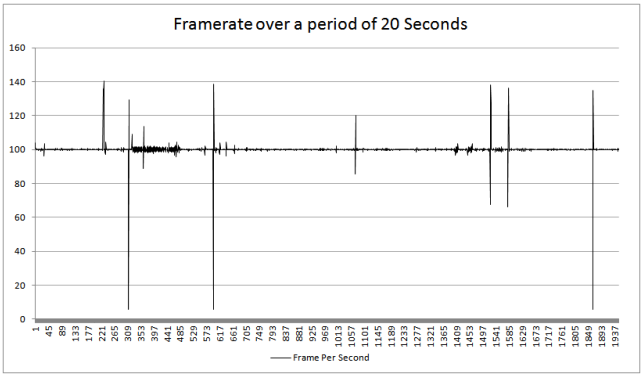

Figure 1. VTO performance of a model undergoing sizing interactions

There are two things to note. First of all, overall performance of just the animated template body lies at a reasonable average of 102 frames per second. As illustrated in Figure 2, sizing the body with respect to a girth measurement triggers the body sizing module to deform the mesh. Sizing any lengths affecting the underlying skeleton triggers both the body sizing module and motion retargeting module. The various dips in the graph are such sizing occurrences with the lower dips caused by a sizing affecting the skeleton. On average such a worst-case manipulation takes around 0.2 seconds to complete for a template body with an animation of 350 key-frames (for a total animation of 14 seconds).

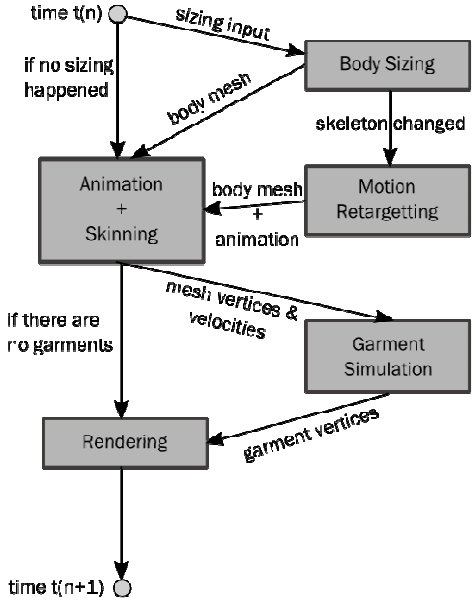

Figure 2. A typical sequence of steps performed by the VTO from one frame to the next

What is more interesting to note however, is not the exact time it takes to perform the operation. While significant in an isolated instance, in the overall scheme of things such interactions are exactly that: isolated instances. At a maximum they occur just once every several seconds. And when performed in a typical static sizing scenario (where the model is not yet animated) such a temporary drop in performance is not disturbing if it is noticeable at all.

Figure 3. A cocktail dress, frilled dress, a pair of pants, a pair of shorts, a flared top and a top with sleeves as used in the VTO performance evaluations

To evaluate VTO garment simulation performance we've taken a particular set of garments as practically used in our VTO demonstrations (see Figure 3). This set of garments has been used in our VTO demonstrator as described in [19].

To aid in the performance of the simulation, the garments use two mesh resolutions. An invisible lower resolution garment mesh is used for the actual simulations, which in turn controls a higher resolution visual mesh which is rendered. Furthermore, the VTO allows for two simulation modes. In real-time a quasi-static simulation mode is used with the aim to improve performance while maintaining simulation accuracy. A fully dynamic simulation mode is used in conjunction with an integrated video recording module. This allows the user to preview a real-time result, while also being able to save a movie with high-quality simulation in a matter of mere minutes. Table 1 lists the number of elements per (combination of) garment(s) as well as the averagely achieved frame-rate and the time needed to go through a

full high-quality non-real-time simulation cycle for an animation of 18 seconds in length.

The hardware used in these evaluations is a desktop PC with an Intel Core i7 X980 CPU at 3.33GHZ, 12GB of RAM and an NVIDIA GeForce GTX 480 GPU.

Table 1. VTO performance per garment (combination)

Garment	Elements	Frames/s	HQ Duration
Cocktail Dress	2425	13.8	492s
Flare Top	2906	18.1	646s
Shorts	2986	17.23	151s
Pants	3970	16.69	175s
Shorts + Flare Top	5892	12.77	1045s
Sleeves Top	6274	13.8	906s
Pants + Flare Top	6876	12.3	1181s
Dress Frills	7092	10.47	401s
Shorts + Sleeves Top	9260	10.43	1273s
Pants + Sleeves Top	10244	9.9	1396s

The simulation timings themselves are influenced by a variety of factors. The density of the simulated garment meshes (that is, the number of elements it has) is one factor. While all garments evaluate both tensile and bending stiffness, their (material) behavior expressed in possibly non-linear strain-stress curves might differ, affecting simulation performance. Materials with a significantly higher stiffness than others will typically take longer to simulation for example.

All these various influences notwithstanding, what can easily be gathered from the listed performance figures is that each and every one of them can at most be called “interactive”. While we have put them to good use, none of them are as real-time as we would like to see. Ideally we would like to achieve the results we currently obtain in our non-real-time, fully dynamic, high-quality simulation mode, but in real-time. The performance figures in Table 1 for the high-quality simulation do however clearly indicate that the current VTO is far removed from achieving that feat.

In summary we can state that the component within the VTO which would most benefit most from improved performance is the garment simulation module. As such, the remainder of this article will discuss how to achieve such improved performance and will particularly focus on the GPU as a computation platform to realize this goal.

3. RELATED WORK

Optimization of garment simulation performance can be approached from two directions. Either you simplify the simulation problem itself. That is, you select a simplified and computationally less intensive mechanical model, such as straight-forward mass-spring model rather than the current (finite) element-based solution. Or you reduce computational load by simplifying the simulated meshes to ones with a lower element density. The problem here is that both solutions will come at the cost of reduced accuracy, which is something we would like to avoid. The second direction would be to simply apply more computational power to the problem.

The VTO in its evaluated implementation is a single-threaded application. And where increased processing power is no longer found in single cores, parallel execution utilizing multi-core hardware or even GPUs seems to be a more viable option.

The parallelization of cloth and garment simulations has seen various platform specific contributions on (what was then) state-of-the-art hardware. Looking at CPU based solutions we see successful parallelization attempts using specialized server hardware [14][27], networked PCs [12][34] and multi-core processors[10][28][32]. While we are predominantly interested in GPU based solutions, what we can take away from these contributions is the following: Both the mechanical and numerical components of physical simulations benefit from a data parallel approach and are of a streaming nature.

Especially the latter point is of importance since, as stated by [23], *“the graphics pipeline is a good match for the stream model…”*. And with the introduction of programmability in the graphics pipeline, graphics processors started to be used – besides their usual rendering tasks – as general purpose stream processors, a concept often referred to as GPGPU (General-Purpose computing on Graphics Processing Units).

3.1 Numerical Algorithms and Linear Algebra operators

When considering the physical simulation of garments, an implicit integrator such as described by Baraff and Witkin [1] is a requirement to ensure simulation stability without having to resort to unfeasibly small time steps. The introduced implicit Euler integration computationally requires the multiplication of a very sparse matrix with a dense vector as part of the employed Conjugate Gradient method. While dense matrix-vector operations are fairly trivially parallelized in a GPU context, sparse matrix operations require more care.

In a shader based setting, both Krüger and Westermann [13] and Bolz et al. [4] map linear algebra operations to a GPU context. Sparse random matrices - as often found in simulations involving unstructured meshes - require that only non-zero elements are stored and used in computation. Both aforementioned papers take a slightly different approach. Non-zero elements in each column are stored as vertex data in [13], with the position ensuring the element gets rendered in the same location as if it were stored as a 2D texture, while the color holds the value of the element. In [4] however, the sparse matrix is split into two textures to hold non-zero elements. One texture holds all entries on the matrix diagonal. A second texture holds all off-diagonal elements of each row, compacted into a single texture. Two additional textures are then needed to hold offsets for the start of each row's data, as well as its column location. In both cases performance improvements of up to an order in magnitude are reported.

For the CUDA platform, Bell and Garland [3] provide various sparse-matrix vector operation implementations, based on standard storage formats among which the CSR format relevant to our random sparse data. Each row of sparse matrix data is handled

by a single warp of 32 threads. The result of these efforts can be found in the CUDA based CUSP [6] library for sparse linear algebra and graph computations. Baskaran and Bordawekar [2] implement a fairly similar CSR based approach, though they use a half-warp per row with added alignment adjustment to ensure coalescing. They report similar or improved performance over CUSP, at the cost of increased memory requirements

3.2 GPU optimized mechanics

Early work in GPU optimized mechanics predominantly focuses on the simulation of 3D (tetrahedralized) objects. The simulation of 3-dimensional deformable bodies at an interactive rate is presented by Georgii et al. in [8]. They employ a linear mass spring model with a Verlet integrator, implemented in fragment shaders. As an extension the authors explore the benefits of an edge centric or point centric approach to the calculation of forces [9]. The point-centric approach (PCA) immediately calculates the forces for each node by looking at connected neighbor nodes. The PCA approach has the downside that spring forces are computed twice. This double computation is avoided with an edge-centric approach (ECA). Using an ECA forces are calculated for each edge/spring and the contributions are added to the connected nodes. This requires a two step approach in which forces are computed per edge and stored in a first pass, while the final nodal forces are then computed by summing the associated edge forces in a second pass. The authors note that the ECA is generally the faster performer and increasingly so for irregular meshes.

In close relation to this work stands the work of Mosegaard et al. [20][21] where a volumetric spring-mass simulator, using Verlet integration, is used in a surgical simulator. The deformable object is a volumetric model based on voxel data from a CT dataset. Two different addressing approaches are taken with regards to spring connections: An explicit approach where each particle maintains a list of particle indices to which it is connected and an implicit approach which makes smart use of the 3D voxel grid by implicitly connecting a node for each voxel with its 18 direct neighbors. Explicit addressing has the downside that an additional lookup is needed to find the neighboring particle positions, which can be a limiting factor in the simulation's performance. Implicit addressing allows for the use of a simple offset to find the connected particle's information. While in their shader based implementation the implicit solution has twice the performance of the explicit solution, this result changes in their later CUDA based implementation [25]. This is mostly due to the inactive particles with in a grid. While these particles could be easily masked out with the shader based approach using a hardware accelerated mask, this is no longer the case when using CUDA. This change in performance behavior does however come with the benefit of a simpler implementation and the support of arbitrary geometry when using explicit addressing.

Tejada and Ertl [31] address the inherent instability of explicit integration with an implicit Euler integrator along the lines of [1] to be able to simulate stiff materials at larger time steps and higher stability. The solver's matrix operations are implemented along the lines of [13], exploiting new hardware capabilities to reduce the number of passes needed for reduction and multiplication operations.

In the context of garment simulation, the work of Rodriguez-Navarro and Susin [26] is the first to consider a co-rotational Finite Element Method approach to simulation on the GPU. Based on the work by Etzmuss et al. [7] they implement a garment simulator for virtual avatars, using implicit integration and GPU based collision detection. Their reported speedup over an equivalent CPU implementation lies in an order of a magnitude.

In the work of Comas et al. [5], the CUDA API is used to implement soft tissue simulation based on the TLED algorithm for nonlinear dynamic finite elements. The work is implemented as part of the open source SOFA medical simulation framework. TLED had had previously been implemented using a shader based approach [29][30]. The authors describe a two stage solution to avoid the inherent difficulties of creating a fully functional scattered write solution. First element stresses are converted to nodal force contributions. In a second stage for each node the contributions by the various elements it's attached to are gathered to obtain the final nodal forces. Given the use of unstructured meshes, memory access is fairly random. Texture memory is used for global memory access, which - due to a texture's cache - should reduce the performance hit usually associated to uncoalesced reads. The authors use page-locked host memory to transfer data between CPU and GPU.

Li et al. [15] present a hybrid CPU-GPU approach to the simulation of clothing. Using a mass-spring approach to the simulation of clothing, they employ a spatial subdivision approach to collision detection. All states are handled on the GPU via the CUDA API. The authors find however that the handling of deformation constraints as presented by [24] is more time-consuming when performed on the GPU than when this is done on the CPU. They therefore handle this stage on the CPU, once the GPU force computation has finished. To hide some of the costs associated with CPU/GPU data transfer, the ability to asynchronously launch kernels is used. In their evaluation they note a performance improvement in the order of a magnitude of their hybrid approach, over a CPU only solution. A GPU only solution falls somewhere in between.

4. GPU OPTIMIZATION

In this section we discuss our GPU optimized framework for the physical simulation of cloth and garments. The main goal of this framework, as should have become apparent, is to achieve improved performance with respect to our VTO platform. As such we desire ideally real-time performance in a fully dynamic simulator, which is able to accurately simulate the nonlinear behavior of fabrics. We will focus on the simulation itself, but of course there are additional requirements to our framework such as the seamless integration with our existing data and content pipeline, as well as ease of use for application developers through a clean API.

4.1 The framework

Our simulation framework, written in C++, consists of a templated set of objects which allow for instantiations based on floating point precision (single or double) and backend (either CPU or GPU). The GPU implementation which we'll discuss here is based on the CUDA API. With regards to the mechanical and numerical components of our framework, an abstract and simplified overview of their organization is given in Figure 4. This overview ignores all other components such as those related to I/O, rendering and other functionalities.

The GarmentObject is the main object within our framework and is what controls the full simulation cycle. It holds the garment's

mechanical state (positions, velocities, masses, etc.), one or more Elements, a Solver and possibly an external ForceField (for such effects as wind and drag.) It is this GarmentObject which is used by the application developer and which hides all the necessary functionality behind an easy to use API. The main computational components are listed on the right side of the diagram and effectively hold the CUDA kernels launched for the computation of forces, force Jacobians (Elements) and numerical integration (Solvers).

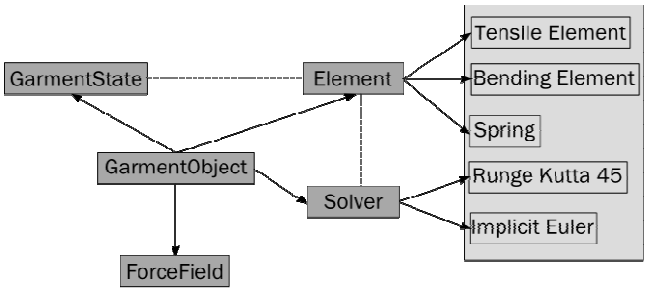

Figure 4. A structural overview of our GPU simulation framework

4.1.1 The Solvers

The responsibility of a solver is to update a system's state from its current frame/time to the next. As such, once invoked, it will request a GarmentObject's Elements to compute the forces and possibly force Jacobians based on a specific state. Our framework supports two solvers; an explicit Runge-Kutta-Fehlberg solver, as well as an implicit Euler solver. In practice, the implicit solver is our solver of choice. The explicit solver is mainly reserved for those cases where we need the associated high accuracy or when we want to establish a ground truth.

The explicit solver has a fairly straight-forward structure. Its computation consists of dense vector operations and allows for a trivial implementation.

For the implicit solver we take a somewhat different approach than the methods described in section 3.1. For one, we don't explicitly compute and store a sparse matrix. Instead we compute the sparse-matrix/vector product on the fly by deferring the responsibility to each individual element, be it a triangular element or a spring. Taking note of the fact that each element's contribution to the overall Jacobian matrix can be formulated as the outer product ($\boldsymbol{a}\,\boldsymbol{b}^T$) of two vectors, its multiplication with an arbitrary input vector ($\boldsymbol{a}\,\boldsymbol{b}^T)\boldsymbol{v}$ can be rewritten to $\mathbf{a}(\boldsymbol{b}^T\,\boldsymbol{v})$. This is a dot-product and a scalar product, rather than a more complex matrix/vector product.

Due to this structure, the computation of the sparse-matrix/vector product uses two CUDA kernels. The first kernel computes the product of an element's Jacobian contributions with the relevant parts of the vector, while a second kernel sums all the individual contributions into the final dense result vector.

As such, we avoid the need for the setup or update of a sparse matrix data structure. What is more, this computational structure in the end closely resembles the computational structure for element forces as we will come to discuss, making computational organization fairly straight-forward.

4.1.2 The Elements

To evaluate the tensile and bending behavior – and the associated forces – of the garments we simulate, we integrated three different element types. Each element maintains its own specific topology information and other (possibly pre-computed) per element values.

In line with [CTO08] we have found that an element centric approach using two kernels for force computation is relatively straight-forward to implement and allows for good performance. As was discussed for our implicit solver, the computation of forces takes two steps. A first kernel computes the forces for the nodes in each triangular element or for the individual spring. A second kernel then combines all the forces for each node by summing the contributions of each attached element. This second kernel relies on a lookup table which for each node stores which elements it attaches to. This same lookup table is also used by our earlier discussed implementation of the implicit solver.

Memory access with regards to the second kernel, but also with regards to the state values accessed in the first kernel is largely unstructured. In an attempt to alleviate most of the performance penalty involved in unstructured memory reads using CUDA, we bind all the arrays accessed in an unstructured manner to a texture. Through texture fetching and the associated caching available, we find that we still obtain good performance.

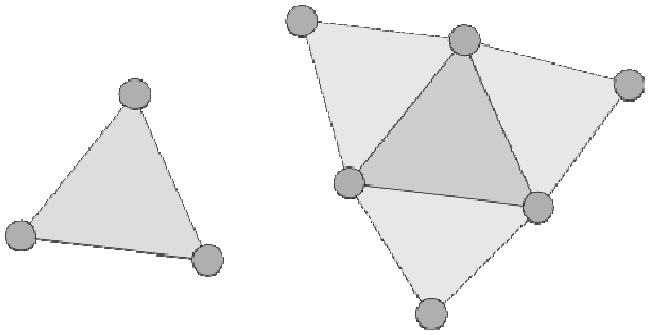

Figure 5. Our framework evaluates tensile stiffness based on a triangular element (left) and bending stiffness based on a "basic shell triangle" (right).

The tensile element is based on the mass-lumped particle system as described in [33]. The triangular element (Figure 5, left), has a fairly straight-forward expression of the element forces and force Jacobian. As such it is straight-forward to implement, while it still allows us to simulate the highly nonlinear behavior of fabrics.

Bending stiffness within fabrics is often low – especially compared to tensile stiffness – but its accurate evaluation is necessary for realistic simulation results related to folds and wrinkles. To evaluate bending stiffness we support two element types; a so-called "basic shell triangle" (Figure 5, right) as well as a straight-forward spring. In the former case bending stiffness is evaluated through the positioning of the 3 outer nodes with respect to the inner triangle, while in the latter case two nodes are connected by a spring across the shared edge of two triangles. In practice we usually resort to the basic shell triangle element for our bending behavior.

To model material behavior we support non-linear strain-stress curves. These curves are implemented as piecewise cubic polynomial splines whose constants are easily stored within float4

elements. While splines of a higher degree would be possible, we find in practice this is not necessary. The evaluation of both the stress and stress-derivative has a simple structure and is implemented as a straight-forward device function. Our platform allows for both a global material behavior as well as different materials per triangular or spring element to allow varying material behavior across a fabric.

4.2 Results and performance

To evaluate the performance of our framework, we have taken a $1m^2$ square piece of fabric which is horizontally clamped at its top edge. The fabric consists of between 200 and 20k triangles. We used an implicit Euler solver fixed at 16 Conjugate Gradient iterations with a fixed time step of 0.01 seconds.

We tested two scenarios as illustrated in Figure 6; A fabric with only tensile stiffness and a fabric with both tensile and a high bending stiffness.

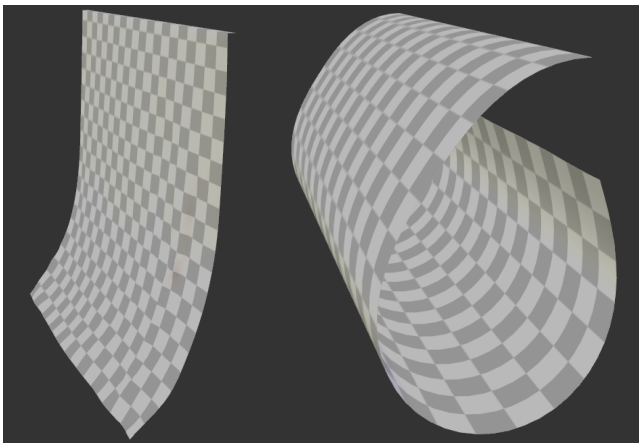

Figure 6. Simulation of a 5000 element $1m^2$ fabric. Tensile only (left) and tensile + bending (right).

Performance was evaluated on a NVIDIA GeForce GTX 480, which is a GPU based on the Fermi architecture. All timings reported are the exact simulation times. Data transfer between CPU and GPU is not taken into account.

All performance tests simulated cloths of varying element densities through a simulation cycle of several simulated seconds. Timing results were then averaged for 100 timesteps, or 1 full simulated second.

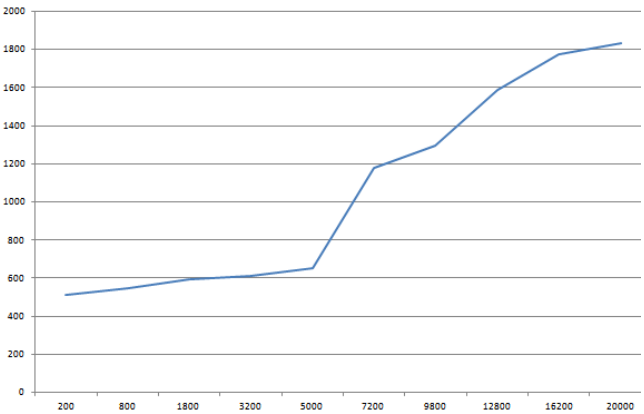

Figure 7. Tensile: average time taken in ms for 100 timesteps

4.2.1 *Tensile Element*

When considering the scenario with only tensile stiffness as various fabric resolutions, we get a performance graph as illustrated in Figure 7. Even at 20k elements, we achieve a performance of about 55 time steps per second.

Comparing this to the performance results mentioned in [33] is somewhat difficult since no exact hardware details are listed. However, where they report the ability to iterate over 17.5k elements per second, a conservative estimate of at least an order of magnitude improvement seems reasonable.

If we evaluate the computation time spent per triangular element, we get a performance graph as illustrated in Figure 8. What this graph illustrates is that the performance starts to achieve its peak from around 5000 elements and higher. Though the exact shape of this graph depends on the hardware used, its global shape is explained by the fact that only at a significant amount of elements a reasonable level of occupancy is achieved and some of the latencies associated with memory access are hidden.

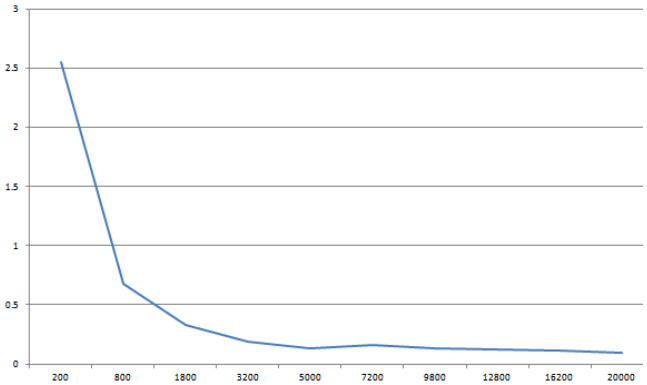

Figure 8. Simulation time per triangular element per 100 iterations in ms

Overall the simulation performance seems to keep up fairly well with increasing mesh densities. Pushing our simulation to the limit, we see that in the scenario mentioned we still achieve 2 time steps per second for a 1million element cloth. Although the 16 CG iterations are then by no means enough for proper simulation.

4.2.2 *Tensile and Bending Element*

When adding bending elements into the mix, the situation becomes slightly different. Of course the computational complexity increases which has an effect on performance. As shown in Figure 9, the performance at 20k elements is still 37 time steps per second. Performance behavior sometimes becomes slightly irregular. With varying amounts of elements we see certain repeatable performance sweet spots such as the sudden dip at 7.2k elements.

Given the particular bending element as shown in Figure 5 (right) there is relatively more irregular memory access to computation than with the tensile elements alone. There are for example twice as many nodal values to retrieve per element. With the computation itself remaining fairly simple, it becomes more difficult to find a good balance between memory accesses and computation.

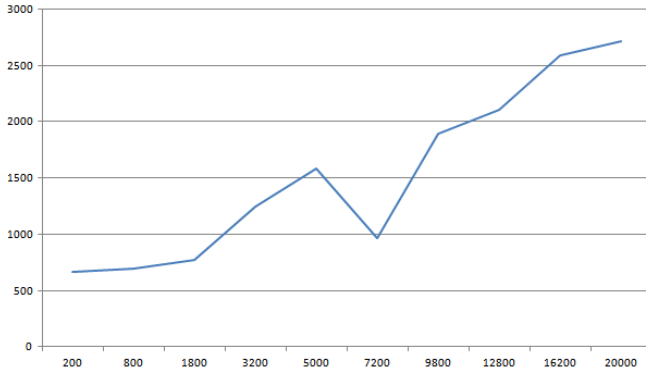

Figure 9. Tensile + Bending: average time taken for 100 timesteps

Overall though the per-element performance (Figure 10) follows a similar pattern as in the tensile-only case. Optimal performance starts to be achieved slightly later, from 7.2k elements and higher.

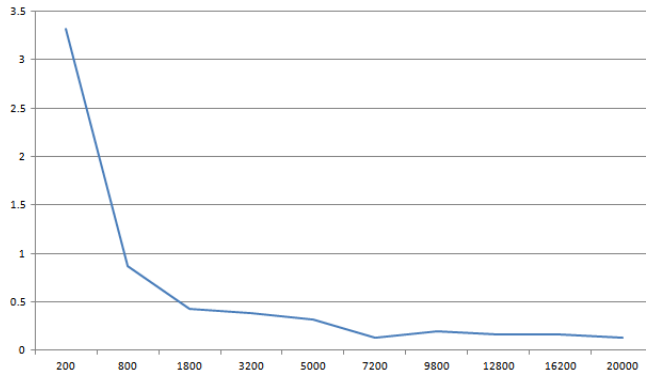

Figure 10. Simulation time per tensile+bending element per 100 iterations in ms

While material nonlinearities are evaluated through our piecewise polynomial splines, we do find that in their current implementation performance is not ideal due to possible branch divergence between threads.

5. CONCLUSION

With our framework in place, we have found that we can achieve a considerable speedup compared to our CPU based simulations. As such, the GPU optimization of our garment simulations seems to be a valuable avenue to pursue. Real-time performance is promising.

Because the performance scales well with increasing mesh density, we also find that the framework might be valuable in non-real-time scenarios with high garment mesh densities.

Future work will focus on the deeper evaluation of achieved performance and comparison to other platforms, as well as the improvement of our framework with respect to some of the noted performance bottlenecks.

Further avenues for investigation are the integration of collision detection and handling, either based on a full GPU solution or by means of a hybrid CPU-GPU alternative.

6. REFERENCES

[1] Baraff, D. and Witkin, A. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98. ACM, New York, NY, USA, 43-54. DOI= http://dx.doi.org/10.1145/280814.280821

[2] Baskaran, M. and Bordawekar, R. 2009. Optimizing Sparse Matrix-vector Multiplication on GPUs. *IBM Research Report RC24704*, Apr. 2009.

[3] Bell, N., Garland, M. 2009. Implementing Sparse Matrix-Vector Multiplication on Throughput-oriented Processors. In *SC'09: Proceedings of the Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, New York, NY, USA. 1-11. DOI= http://dx.doi.org/10.1145/1654059.1654078

[4] Bolz, J., Farmer, I., Grinspun, E. and Schröder, P. 2003. Sparse Matrix Solvers on the GPU: Conjugate Gradient and Multigrid. *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03. ACM, New York, NY, USA. 917-924. DOI= http://dx.doi.org/10.1145/1201775.882364

[5] Comas, O., Taylor, Z.A., Allard, J., Ourselin, S., Cotin, S. and Passenger, J. 2008. Efficient Nonlinear FEM for Soft Tissue Modelling and its GPU Implementation within the Open Source Framework SOFA. In *Proceedings of the 4th International Symposium on Biomedical Simulation, ISBMS'08*, Springer-Verlag, Berlin, Heidelberg. 28-39. DOI= http://dx.doi.org/10.1007/978-3-540-70521-5_4

[6] CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. http://code.google.com/p/cusp-library/

[7] Etzmuss, O., Keckeisen, M. and Strasser, W. 2003. A Fast Finite Element Solution for Cloth Modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. Oct. 2003, 244-251. DOI= http://dx.doi.org/10.1109/PCCGA.2003.1238266

[8] Georgii, J., Echtler, F. and Westermann, R. 2005. Interactive Simulation of Deformable Bodies on GPUs. In *Proceedings of Simulation and Visualization 2005*, 247-258.

[9] Georgii, J. and Westermann, R. 2005. Mass-Spring Systems on the GPU. *Simulation Modelling Practice and Theory*, 13, 8, 693-702. DOI= http://dx.doi.org/10.1016/j.simpat.2005.08.004

[10] Hughes, C., Grzeszczuk, R., Sifakis, E., Kim, D., Kumar, S., Selle, A., Chhugani, J., Holliman, M. and Chen, Y. 2007. Physical Simulation for Animation and Visual Effects: Parallelization and Characterization for Chip Multiprocessors. In *Proceedings of the 34th Annual International Symposium of Computer Architecture, ISCA '07*. ACM, New York, NY, USA, 220-231. DOI= http://dx.doi.org/10.1145/1273440.1250690

[11] Kasap, M. and Magnenat-Thalmann, N. 2007. Parameterized Human body Model for Real-Time Applications. In *Proceedings of the 2007 International Conference on Cyberworlds*. IEEE Computer Society, 160-167.

[12] Keckeisen, M. and Blochinger, W. 2004. Parallel Implicit Integration for Cloth Animations on Distributed Memory

Architectures. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*.

[13] Krüger, J. and Westermann, R. 2003. Linear Algebra Operators for GPU Implementation of Numerical Algorithms. *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03. ACM, New York, NY, USA. 908-916. DOI= http://dx.doi.org/10.1145/1201775.882363

[14] Lario, R., Garcia, C., Prieto, M. and Tirado, F. 2001. Rapid Parallelization of Multilevel Cloth Simulator using OpenMP. In *Proceedingsof European Workshop on OpenMP (EWOMP 2001)*.

[15] Li, H., Wan, Y. and Ma, G. 2011. A CPU-GPU Hybrid Computing Framework for Real-time Clothing Animation. *2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 391-396.

[16] Lim, M., Kevelham, B., Nijdam, N. and Magnenat-Thalmann, N. 2011. Rapid Development of Distributed Applications using High-Level Communication Support. *Journal of Network and Computer Applications*. 34, 1, 172-182. DOI= http://dx.doi.org/10.1016/j.jnca.2010.08.003

[17] Lyard, E. and Magnenat-Thalmann, N. 2008. Motion Adaptation based on Character Shape. *Computer Animation and Virtual* Worlds. 19, 3-4 (Sep. 2008), 189-198. DOI= http://dx.doi.org/10.1002/cav.v19:3/4

[18] Magnenat-Thalmann, N., Kevelham, B., Volino, P., Kasap, M. and Lyard, E. 2011. 3D Web-Based Virtual Try On of Physically Simulated Clothes. *Computer Aided Design and Applications*, 8, 2, 163-174.

[19] Magnenat-Thalmann, N., Volino, P., Kevelham, B., Kasap, M., Tran, Q., Arevalo, M., Priya, G. and Cadi, N. 2011. An Interactive Virtual Try On. In *Proceedings of Virtual Reality Conference (VR), 2011 IEEE*. IEEE, (Mar. 2011), 263-264.

[20] Mosegaard, J., Herborg, P. and Sorensen, T.S. 2005. A GPU Accelerated Spring Mass System for Surgical Simulation. *Studies in Health Technology and Informatics*, 111, 342-348.

[21] Mosegaard, J. and Sorensen, T.S. 2005. GPU Accelerated Surgical Simulators for Complex Morphology. In *Proceedings of IEEE Virtual Reality (VR), 2005*. 147-153. DOI= http://dx.doi.org/10.1109/VR.2005.1492768

[22] NVIDIA Corporation. 2012. NVIDIA CUDA C Programming Guide. Version 4.2.

[23] Owens, J. 2005. Streaming Architectures and Technology Trends. *ACM SIGGRAPH 2005 Courses*. ACM, New York, NY, USA.

[24] Provot, X. 1995. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. In *Proceedings of Graphics Interface '95*. 147-154.

[25] Rasmusson, A., Mosegaard, J. and Sorensen, T.S. 2008. Exploring Parallel Algorithms for Volumetric Mass-Spring-Damper Models in CUDA. In *Proceedings of the 4th International Symposium on Biomedical Simulation, ISBMS '08*. Springer-Verlag, Berlin, Heidelberg. 49-58. DOI= http://dx.doi.org/10.1007/978-3-540-70521-5_6

[26] Rodriguez-Navarro, J. and Susin, A. 2006. Non Structured Meshes for Cloth GPU Simulation using FEM. *3rd Workshop in Virtual Reality, Interactions and Physical Simulations, VRIPHYS'06*, Eurographics Ed., 1-7.

[27] Romero, S., Romero, L. and Zapato, E. 2000. Fast Cloth Simulation with Parallel Computers. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing, Euro-Par '00.* Springer-Verlag, 491-499.

[28] Selle, A., Su, J., Irving, G. and Fedkiw, R. 2009. Robust High-Resolution Cloth Using Parallelism, History-Based Collisions and Accurate Friction. *IEEE Transactions on Visualization and Computer Graphics*. 15, 2 (Mar. 2009), 339-350.

[29] Taylor, Z., Cheng, M. and Ourselin, S. 2007. Real-Time Nonlinear Finite Element Analysis for Surgical Simulation using Graphics Processing Units. *Medical Image Computing and Computer-Assisted Intervention: MICCAI 2007*, Lecture Notes in Computer Science, 4791, 701-708.

[30] Taylor, Z., Cheng, M. and Ourselin, S. 2008. High-Speed Nonlinear Finite Element Analysis for Surgical Simulation using Graphics Processing Units. *IEEE Transactions on Medical Imaging*, 27, 5 (May 2008), 650-663.

[31] Tejada, E. and Ertl, T. 2005. Large Steps in GPU-based Deformable Bodies Simulation. *Simulation Modelling Practice and Theory*, 13, 8, 703-715.

[32] Thomaszewski, B., Pabst, S. and Blochinger, W. 2008. Parallel Techniques for Physically Based Simulations on Multi-core Processors Architectures. *Computers and Graphics*. 32, 1, 25-40. DOI= http://dx.doi.org/10.1016/j.cag.2007.11.003

[33] Volino, P., Magnenat-Thalmann, N. and Faure, F. A Simple Approach to Nonlinear Tensile Stiffness for Accurate Cloth Simulation. 2009. *ACM Transactions on Graphics*. 28, 4 (Sep. 2009), 105:1-105:16. DOI= http://dx.doi.org/10.1145/1559755.1559762

[34] Zara, F., Faure, F. and Vincent, J-M. 2004. Parallel Simulation of Large Dynamic Systems on a PCs Cluster: Application to Cloth Simulation. *Int. J. Comput. Appl.* 56, 3 (Mar. 2004), 173-180.