EG UK Theory and Practice of Computer Graphics (2013)
Silvester Czanner and Wen Tang (Editors)

GPU Simulation of Finite Element Facial Soft-Tissue Models

Mark Warburton and Steve Maddock

Department of Computer Science, The University of Sheffield, UK

Abstract

Physically-based animation techniques enable more realistic and accurate animation to be created. We present a GPU-based finite element (FE) simulation and interactive visualisation system for efficiently producing realistic-looking animations of facial movement, including expressive wrinkles. It is optimised for simulating multi-layered voxel-based models using the total Lagrangian explicit dynamic (TLED) FE method. The flexibility of our system enables detailed animations of gross and fine-scale soft-tissue movement to be easily produced with different muscle structures and material parameters. While we focus on the forehead, the system can be used to animate any multi-material soft body.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Facial modelling and animation is one of the most challenging areas of computer graphics. Currently, most facial animation requires recorded performance-capture data or models to be manipulated by artists. However, using a physically-based approach, the effects of muscle contractions can be propagated through the facial soft tissue to automatically deform the model in a more realistic and anatomical manner.

Physics-based soft-tissue simulation approaches often focus on either efficiently producing realistic-looking animations for computer graphics applications using the mass-spring (MS) method [TW90, KHS01], or simulating models with high physical accuracy for studying soft-tissue behaviour [BJTM08, KSY08] or surgical simulation [KRG*02, ZHD06] using the accurate but complex finite element (FE) method. Various FE software packages are currently available, some of which are specialised for complex biological applications, such as CMISS. Such complex systems are geared towards accurately simulating small areas of soft tissue, whereas FE solvers for computer graphics applications are normally used to simulate less complex models of larger areas, for example, without simulating wrinkles [SNF05]. Using GPU computing architectures, complex FE simulations are now possible in real time [CTA*08].

By modelling more physics-based behaviour than current computer graphics approaches, the aim of this work is to develop an optimised GPU-based FE simulation and visualisation system for efficiently producing realistic-looking animations of facial soft tissue, including animation of expressive wrinkles, focussing on the forehead (see Figure 1). This involves simulating multi-layered forehead soft-tissue models with complex boundary conditions under the influence of muscle contraction using the accurate non-linear total Lagrangian explicit dynamic (TLED) FE method. Our system is optimised for simulating voxel-based models, for which there are various model creation and simulation advantages [WM12], and can be used to animate any inhomogeneous soft body (not just soft tissue).

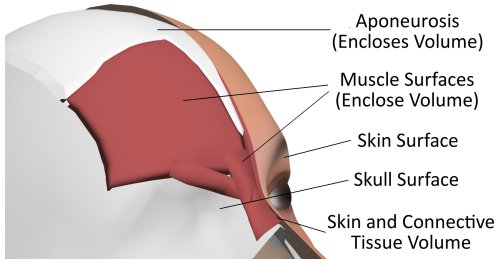

Figure 1: *Surfaces and volumes of a forehead soft-tissue model. The volume between the skull and skin surfaces is discretised and simulated.*

This paper focusses on the CUDA-based implementation and optimisation of our simulation process, the theory of

DOI: 10.2312/LocalChapterEvents.TPCG.TPCG13.001-008

which has been previously presented [WM13b]. The following sections discuss related work, followed by an overview of our physically-based animation approach, and an introduction to CUDA. Implementation details of the simulation system are then presented, finishing with example animations.

2. Related Work

2.1. Physically-Based Facial and Soft-Tissue Animation

Physically-based facial animation systems for computer graphics applications normally consist of muscle and skin models, sometimes along with a skull model and wrinkle models. Contraction of vector-based [LTW95] or volumetric muscles [KHS01] requires a contraction magnitude (e.g. based on a Hill-type model [RP07]) and direction (e.g. based on a fibre field [SNF05]). For simulating the physics of facial soft tissue, efficient and stable physics engines [Fra12], and the MS method [TW90] are popular for computer graphics applications, although heuristic functions are required for incompressibility and volume preservation of MS elements [KHS01]. The more accurate but computationally complex FE method can also be used [SNF05], although this is mainly used with scientific [BJTM08] or surgical applications [KRG*02, ZHD06] that require high accuracy.

For computer graphics applications, skin wrinkles are usually layered onto a mesh using texture-mapping techniques [RBM08], which are computationally efficient and suitable for fine wrinkles, or by using a heuristic function to modify the geometry of the mesh [BBA*07, MC10] for more realistic looking wrinkles on a high-resolution mesh. These can also be layered onto a physics-based model [ZST05]. Multi-layered FE models of small areas of soft tissue have been developed for accurate simulation of expressive or aging wrinkles [KSY08], some of which simulate factors such as anisotropy and viscoelasticity [FM08]. Due to its efficiency, the CPU- and GPU-based TLED algorithms have been used for various non-linear FE soft-tissue simulations of biological organs [MJLW07, CTA*08].

2.2. Finite Element Solvers

Various FE packages are currently available, and are used widely in science and engineering fields. Most solvers perform FE analysis in three stages: preprocessing (setting up the simulation), computation of the entire simulation, and postprocessing (visualising and analysing the simulation), and are therefore unsuitable for interactive scenarios. Commercial software, such as Abaqus, LS-DYNA and ANSYS, contain a number of solvers and can often be used to solve a wide range of problems; however, they are not geared towards complex biological applications, for example, with complex material models and muscle contraction.

Various packages have been implemented specifically for simulation of skin and soft tissue, such as CMISS and FEBio, and muscle contraction models can be integrated into such software [HMSH09]. For greatly increased computational performance, such GPU-based solvers have also been developed, and one such solver has been implemented into the SOFA framework for interactive soft-tissue simulation [CTA*08], although muscle contraction was not included. FE solvers have been developed for computer graphics applications, although these are normally used to simulate less complex models without wrinkles [SNF05].

Our GPU-based system includes a muscle contraction model, and can handle complex boundary conditions to simulate soft-tissue sliding. While most existing commercial and specialised FE systems consider only a single material per element, our system is optimised for simulation of complex, multi-layer voxel-based soft-tissue models, for which there are various model creation, performance and stability advantages [WM12], and elements of such models may overlap multiple materials and muscles. Our system can be used both interactively, and entire simulations can also be saved to file for later visualisation. Also, rather than simulating models of small areas, we simulate detailed FE facial models, but with enough detail, such as skin layers, to simulate fine details like wrinkles (unlike current such models [SNF05, BJTM08]). As well as computer graphics applications, due to the detail and accuracy of the simulations, our animation approach and simulation system could also be useful in other fields, such as biomechanics and surgery.

3. Overview of our Physically-Based Animation Approach

Figure 2 shows an overview of our entire animation approach, which involves three major stages:

1. Creating the surface mesh for an object
2. Creating a suitable simulation model
3. Simulating and visualising the model over time

The surface mesh can be created using any 3D modelling software. The next stage involves using our model creation system to automatically discretise the volumes enclosed by this mesh into a collection of nodes that are connected to form volumetric elements, and compute FE model parameters to produce a simulation model [WM13a]. These parameters include skin layers and element material composition, muscle fibre directions, and boundary conditions. We use non-conforming (voxel-based) hexahedral models, with bound surface meshes for visual purposes. These models can then be simulated and visualised using a GPU-based FE simulation and visualisation system [WM13b], the implementation of which is the focus of this paper.

4. Model Simulation

The full details of our simulation process have been previously presented [WM13b]. A brief summary is presented

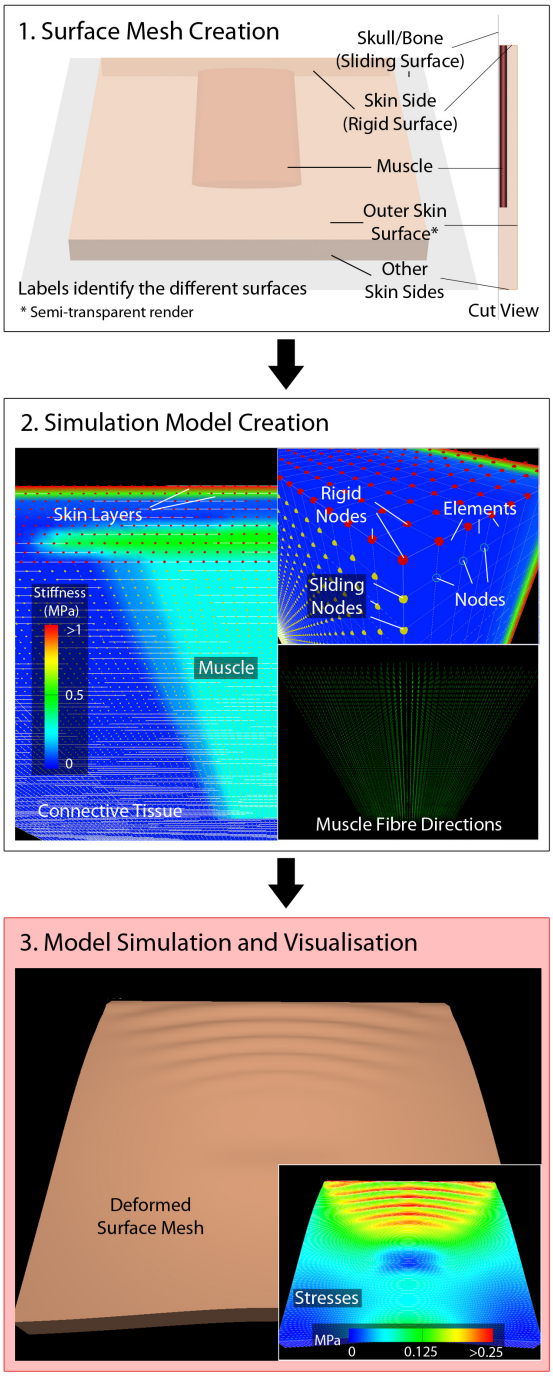

Figure 2: *An overview of our physically-based animation approach. The GPU implementation of the simulation and visualisation process is the focus of this paper.*

here before we concentrate on the GPU implementation using CUDA. We use the non-linear TLED formulation of the FE method with reduced-integration 8-node hexahedral elements (with a single Gauss integration point) [MJLW07], which have efficiency and accuracy advantages for simulating soft tissue. Starting with the principle of virtual work, and using a lumped mass approximation with mass-proportional Rayleigh damping, the uncoupled equation of motion can be derived:

$$\mathbf{M}\,{}^t\ddot{\mathbf{u}} + \mathbf{C}\,{}^t\dot{\mathbf{u}} + k({}^t\mathbf{u})\,{}^t\mathbf{u} = {}^t\mathbf{r} \tag{1}$$

where ${}^t\mathbf{u}$ is the displacements vector, $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $k({}^t\mathbf{u})$ is the stiffness matrix, and ${}^t\mathbf{r}$ is the vector of external forces. Element nodal force contributions, ${}^t\mathbf{f}$, are calculated as:

$${}^t\mathbf{f} = k({}^t\mathbf{u})\,{}^t\mathbf{u} = \int_{{}^0V} {}^t_0\mathbf{B}_L^T\, {}^t_0\hat{\mathbf{s}}\, \mathrm{d}\,{}^0V \tag{2}$$

where ${}^t_0\mathbf{B}_L^T$ is the strain-displacement matrix, ${}^t_0\hat{\mathbf{s}}$ is the second Piola-Kirchhoff stress vector, and V is the element volume. As we use non-conforming models, the stress vector is a weighted sum of those calculated for each material the element overlaps. A stiffness-based technique adds hourglass forces to element nodal forces based on element deformation to reduce hourglass effects that occur with reduced-integration elements [JWM08]. To advance simulations, the central difference time-integration method is used.

To enable active muscle stresses to be generated during simulations, a muscle contraction model has been implemented, which considers both active and transversely isotropic passive stresses in the fibre directions. Weighted stresses are calculated for each muscle overlapping a non-conforming element. To constrain models, rigid (fixed) and sliding nodal constraints can be set. Rigid nodes are simply fixed with zero displacement throughout simulations. Sliding nodes maintain a fixed distance away from the non-conforming surface they are bound by, and can be used to model the sliding of superficial facial soft-tissue layers over the stiff deep layers and skull [WMSH10], which is often neglected [SNF05, BJTM08].

5. CUDA

GPUs are specialised for compute-intensive, highly-parallel computation, consisting of a number of streaming multiprocessors (SMs) that execute in parallel. Each core of an SM executes the same instruction on a different thread (SIMT paradigm). Using CUDA, threads for GPU kernel execution are organised into equally-sized blocks, which are organised into a grid [NVI12]. Each block is executed on an SM in groups of 32 threads, called warps. Depending on memory requirements, various blocks can reside on an SM, and an SM can quickly switch between warps to hide the latency of memory accesses.

Normally, most data on the GPU resides in high-latency

global memory. Each SM has faster shared memory for sharing data between block threads. For local data, threads are allocated extremely fast registers, although, if exceeded, this spills into a section of global memory. With the Fermi architecture, the L1 and L2 cache can increase the efficiency of global memory accesses.

For optimal performance, it is important to balance memory requirements of a block with the ability to hide memory latency. Efficient access patterns should be used to achieve global memory coalescing and avoid shared memory bank conflicts. Also, branch divergence within a warp should be minimised, as this causes each branch to be executed in a serial fashion, with some threads idle during each branch execution. Similarly, uneven loops within a warp will result in idle threads waiting for the longest loop to complete.

6. GPU Implementation of our Simulation Process

The TLED FE formulation we use is inherently parallel, making it suitable for GPU implementation, and enables some variables to be precomputed. Algorithm 1 shows the process to compute a timestep using our simulation system, which has been implemented using C++ with CUDA C API version 4.2 for use with the Fermi architecture.

6.1. Memory and Data Structures

During a CPU precomputation stage, all simulation values that relate only to the initial configuration, such as shape function derivatives and unscaled hourglass matrices, are precomputed. All simulation values are then copied to the GPU, where they remain throughout simulations to reduce the amount of slow CPU-GPU data transfer. All simulation variables are stored in global memory, which, with the L1 cache and lenient coalescing requirements, is now normally preferred over texture memory. Data is stored using the structure of arrays pattern (e.g. $[[u_1,...,u_n],[v_1,...,v_n],[w_1,...,w_n]]$) where possible, rather than array of structures (e.g. $[[u_1,v_1,w_1],...,[u_n,v_n,w_n]]$), with each row of the arrays aligned to a memory block. This increases spatial locality of the cached accesses between consecutive warp threads, and enables memory coalescing.

Figure 3 shows the main classes used to organise simulation data based on functionality (e.g. to reduce branch divergence and uneven loops). While all nodes of our FE models share the same functionality, element-related computation varies based on element type, material behaviour, and whether active muscle stresses are generated. Figure 4 shows an example of how elements are grouped to reflect this. Similarly, with the surface mesh, the processing of primitives varies depending on the type of primitive, and how these are rendered.

Contractile components are created to generate muscle stresses, and one component is created for each muscle that overlaps an element. As shown by Figure 4, while contractile components exhibit identical functionality, these are grouped by the element collection they reference, the values of which they modify. Within a collection, contractile components are ordered firstly by the muscle, and then by the element they reference, enabling several consecutive warp threads to access the same muscle data and spatially local element data.

Algorithm 1: The GPU-based process to compute a timestep. The major kernels have been identified.

```
   // Element nodal force contributions:
 1 foreach element collection, M do
 2     if active stresses generated for M then
 3         kernel: foreach element in M do
 4             foreach integration point do
 5                 Calculate deformation values;
 6 foreach contractile component collection, A do
 7     kernel: foreach contractile component in A do
 8         foreach integration point do
 9             Calculate muscle stresses;
10 foreach element collection, M do
11     kernel: foreach element in M, m do
12         foreach integration point, i do
13             if active stresses generated with M then
14                 Read deformation and stress values;
15             else
16                 Calculate deformation values and
                   initialise stress;
17             foreach material overlapping m do
18                 Calculate material stresses;
19             Calculate internal nodal forces at i;
20         Add nodal force contributions for m;
21     if hourglass control enabled with M then
22         kernel: foreach element in M do
23             Calculate hourglass forces;
   // Nodal displacements:
24 kernel: foreach node, n do
25     if n not rigid then
26         Calculate nodal displacements;
27 foreach sliding constraint, C do
28     foreach surface in C do
29         kernel: foreach sliding node in C do
30             Perform broad-phase collision detection;
31         kernel: foreach sliding node in C do
32             Compute closest surface point;
33     kernel: foreach sliding node in C do
34         Update nodal displacements;
```

6.2. Timestep Computation

As shown by Algorithm 1, the main simulation system kernels can be grouped into two major procedures: computa-

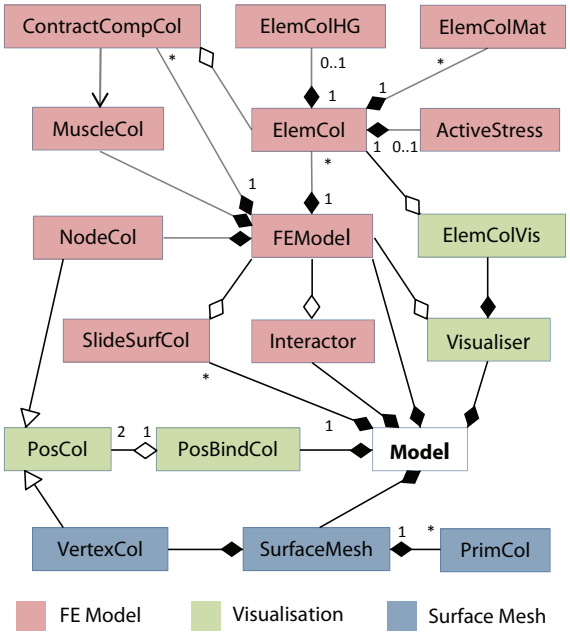

Figure 3: *The relationships between the main classes used with our simulation system. Where not given, the multiplicities equal 1.*

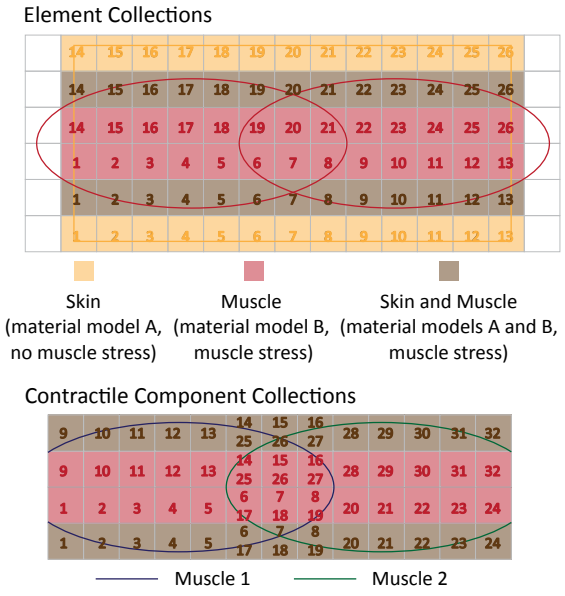

Figure 4: *Element and contractile component groupings for a voxel-based model with 2 muscles and 2 material models, showing the element and contractile component orderings within these groups. Note this is a 2D illustration of a 3D process.*

tion of element nodal force contributions, and computation of nodal displacements. Various other kernels are also used within a timestep, for example, to initialise element stresses and internal nodal forces.

6.2.1. Element Nodal Force Contributions

First, muscle stresses are computed by launching a kernel for each contractile component collection. Additions of the stress values are performed atomically as there may be multiple contractile components per element. The element nodal force contributions kernel can then be launched for each element collection. For collections that overlap muscles, values relating to the deformation of elements that are used in the muscle and material stress computations, such as the deformation gradients, are computed in a separate kernel, and stored before the computation of muscle stresses. For other collections, these values are computed and used only in the element nodal force contributions kernel, which uses a function pointer to the relevant GPU function to compute or read such values.

As well as obvious optimisations with efficient storage and computations using symmetric matrices like the right Cauchy-Green deformation tensors, when computing, for example, a deformation gradient, ${}^t_0\mathbf{X}_{ij}$, efficient global memory accesses can be achieved by computing matrix multiplications in stages such that element and nodal values are only accessed once:

$$ {}^t_0\mathbf{X}^{(\mathbf{a})}_{ij} = \begin{pmatrix} {}^tu_{1_a} \\ {}^tu_{2_a} \\ {}^tu_{3_a} \end{pmatrix} \begin{pmatrix} \frac{\partial N_a}{\partial {}^0x_1} & \frac{\partial N_a}{\partial {}^0x_2} & \frac{\partial N_a}{\partial {}^0x_3} \end{pmatrix} \tag{3} $$

$$ {}^t_0\mathbf{X}_{ij} = \sum_{a=1}^{n} {}^t_0\mathbf{X}^{(\mathbf{a})}_{ij} \tag{4} $$

where a is the node number and n is the number of nodes.

For each material model used by a non-conforming element collection, a weight and average parameters are computed for each element. A pointer to the relevant GPU function that calculates material stresses is also stored, which enables easy integration of material types without modifying the element nodal force contributions kernel. The weighted combination of stresses calculated for each material model is added to any current (e.g. muscle) stresses. As this kernel makes use of many variables, to minimise both register spilling and multiple same-location global memory accesses, shared memory is used for temporary storage of the stress vectors being used by each thread in a block, and these are organised using the same structure of arrays pattern that is used for global memory storage.

When computing forces at an integration point, instead of computing the large strain-displacement matrix for the whole element, which would increase register spilling, rows, r, of the force vector can be constructed independently:

$$ \mathbf{f}_r = \int_{{}^0V} {}^t_0\mathbf{B}^T_{L_r} \, {}^t_0\hat{\mathbf{s}} \, \mathrm{d}\,{}^0V \tag{5} $$

Element nodal force contributions are computed using Gaussian quadrature with the forces computed at each integration point, and additions of these to the internal nodal forces are performed atomically, as nodes are usually attached to multiple elements. Hourglass forces can then be computed for each necessary element collection.

6.2.2. Nodal Displacements

A kernel is launched to update the unfixed nodal displacements for each node independently, using the central difference time-integration scheme, and boundary conditions are then applied to update these according to boundary constraints. For sliding constraints (see Section 4), computation of the distance and direction of a node to the closest bound surface point is required. A GPU-based semi-brute-force broad-phase collision detection algorithm with regular spatial subdivision has been implemented to prune the number of polygons to be tested [AGA12], using which nodes are represented as AABBs with lengths of $2 \times d_q$, where d_q is the fixed distance for node q. For a particular sliding constraint, there may be several sliding surfaces, for which primitive and collision data is precomputed as these surfaces don't move.

After performing broad-phase collision detection, a kernel is executed for each surface to compute the closest surface point to each sliding node based on the broad-phase collisions, followed by a kernel to update the displacements for each sliding node based on these values. The former has a loop with a variable number of iterations per thread depending on the number of broad-phase collisions per surface. This could be avoided if the kernel launched a thread for each collision, rather than the group of collisions between a node and a surface, although race conditions could occur when comparing and overwriting the closest surface positions when there are multiple collisions per surface.

6.3. Visualisation, Output and Interaction

Before rendering a frame, kernels are executed to update nodal positions using the rest positions and displacements, and calculate the new positions and normals of any bound vertices. These updates, and outputting graphics frame data to screen is done after computation of a number of timesteps. These output intervals depend on the desired frame rate if the simulation runs in real time, or the desired simulation time between frames otherwise. Simulation data at such intervals can also be saved to file for later playback and analysis in real time. During playback, simulation data is updated using values read from the file, requiring no modifications to the visualisation component to visualise this data, and enabling simulations to be continued after all frames have been read.

CUDA-OpenGL interoperability enables necessary simulation data on the GPU to be directly read by OpenGL. A vertex buffer object (VBO) is created for an FE model, which

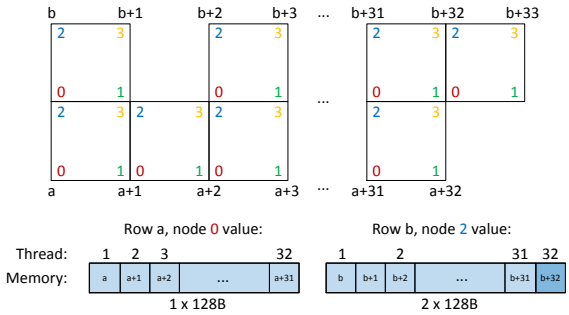

Figure 5: *An example of efficient global memory accesses of nodal values during a kernel that loops over elements, when using a voxel-based model with ordered node numbering. Each warp thread is accessing a 4B nodal value in cached global memory. Note this is a 2D illustration of a 3D process.*

consists of all nodal positions, followed by all nodal colours, both of which are memory aligned for efficient writes using CUDA, and in an appropriate array of structures format for rendering. Two copies of nodal positions are therefore maintained, one of which as structures of arrays for efficient access using CUDA that is used to update data for the VBO. To visualise a vector, such as stresses, the magnitude is clamped and converted to a colour, which is written directly to the VBO. As the visualisation component is uncoupled from the simulation process, it can be easily extended to visualise other variables. A precomputed static index buffer object (IBO) contains a group of node indices to render each element in a collection. A similar approach is used for rendering a surface mesh.

After computation of a timestep, any updates to simulation parameters, such as muscle contraction parameters, are computed on the CPU, and copied from host (main) memory to the GPU. Additional boundary conditions are also set; for example, our system supports user interaction, whereby external forces are applied to a group of nodes. For efficiency, all memory copies during simulations are done via page-locked host memory.

6.4. Optimisation for Voxel-Based Models

Our system has been optimised for use with voxel-based models. For example, only a single set of the various element values, such as a single set of shape function derivatives (12 values) and one unscaled hourglass stiffness matrix (64 values), needs to be stored, rather than a set for each element. While greatly reducing memory usage, this also enables more efficient memory accesses as the same set of values are accessed by each thread, reducing the required number of slow global GPU memory accesses.

Elements and element nodes can also be easily efficiently

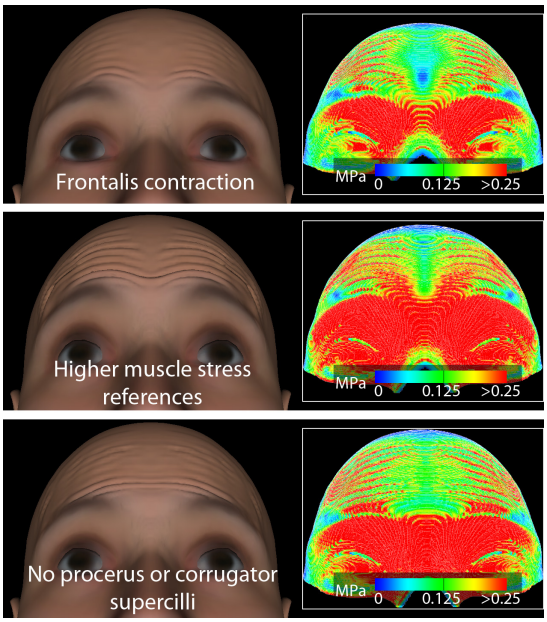

Figure 6: *Variations of a forehead model under contractions of the frontalis. Insets show the stresses, where red indicates high, and blue indicates low stress.*

Layer	ρ* (kg/m^3)	E (MPa)	v	Depth (mm)
SC	11,000	48	0.49	0.02
D	11,000	0.0814	0.49	1.8
H	11,000	0.034	0.49	Remains
M	11,000	0.5	0.49	∼1
T	11,000	24	0.49	∼1

Table 1: *The neo-Hookean material properties used for the animation examples. Key: ρ: Density, E: Young's Modulus, v: Poisson Ratio, SC: Stratum Corneum, D: Dermis, H: Hypodermis, M: Muscle, T: Tendon. *Includes mass scaling.*

Detail	Face	Skin Block	Armadillo
Nodes	629,178	146,410	19,698
Elements	503,530	129,600	15,107
Timestep (ms)	0.005	0.005	0.15
Timestep Computation Time (ms)	6.83	1.48	0.14

Table 2: *Statistics of the examples, using an NVIDIA GTX 680 GPU.*

numbered, which not only means that only 4 node indices per element need to be stored, from which the other 4 indices can be deduced, but the more efficient data storage of nodal values can also improve memory coalescing and global memory cache hits, for example, during the element nodal force computations, as shown by Figure 5. Using such an approach to access element nodal values during a kernel that loops over elements, in the worst case scenario (when none of the neighbouring elements are connected in series), just 2×128B memory accesses per warp are required, as opposed to potentially 32×128B (when none of the nodal values are in the same memory block) with unordered and inefficient node organisation. However, this value increases at sections where consecutive elements don't belong to the same element collection.

7. Results

Figure 6 shows example animations using a forehead model (generated using the surface mesh in Figure 1 with our model creation system [WM13a]), while Figure 2 contains an example animation of a simpler soft-tissue-block model. Tables 1 and 2 show the material properties that were used (based on those reported in literature [KSY08]), and some model statistics. Muscle parameters were estimated based on literature [RP07] and from testing. Each muscle was assigned stress references of 5MPa, and an optimal fibre stretch of 1 (rest length). An estimated mass-proportional damping scale factor of 2Ns/m was used, and external forces that have little visual effect on the animations, such as gravity, were neglected. Mass and time scaling were used to increase stability and improve performance [WM13b]. Figure 7 shows an example of a more generic multi-material object.

Using a simplified version of our GPU-based simulation system, early tests showed performance increases of over 130x with models of up to 250,000 4-node tetrahedral elements compared with our CPU-based solver. This made the implementation of the full system on the GPU an obvious choice. Using our current GPU-based system, optimisations

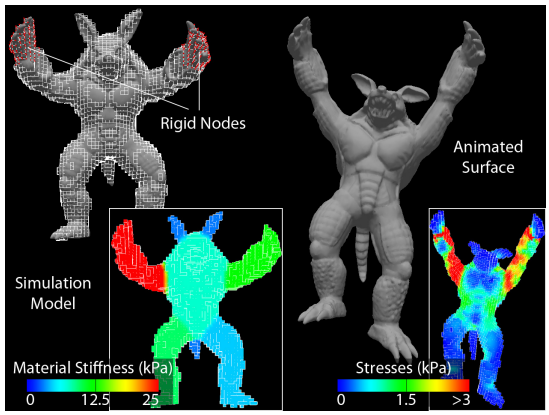

Figure 7: *Animation of a multi-material Stanford Armadillo under gravity. With the material stiffness, red indicates high, and blue indicates low stiffness.*

when using voxel-based models have led to performance increases, even of almost 2x with low resolution models containing around 3,000 elements, compared with using a conforming hexahedral model [WM12].

Despite the performance advantages, the soft-tissue simulations aren't real time due to the required model complexity to capture the necessary detail for wrinkle simulation, such as skin layers. One computational bottleneck is the processing of sliding constraints, which contributed to over 40% of the timestep computation time with the forehead simulations, containing 118,276 sliding nodes. This procedure could be further optimised to even out the workload between threads (see Section 6.2.2). However, real-time frame rates can be achieved using lower-resolution models for simulating gross deformation, like the multi-material Armadillo.

8. Conclusion

This work has presented a GPU-based FE simulation and visualisation system for efficiently producing realistic-looking animations of facial movement, including animation of expressive wrinkles. Focussing on the forehead, this involves simulating multi-layered voxel-based soft-tissue models with complex boundary conditions under the influence of muscle contraction using an optimised GPU-based nonlinear TLED FE solver. Simulations can be visualised during computation, and can be interacted with. Animation examples have demonstrated the ability of our system to produce animation of realistic large and fine-scale soft-tissue behaviour, and the flexibility to deform different soft-bodies. For improved performance, future work will focus on extending our system to support running simulations using multiple GPUs.

References

[AGA12] AVRIL Q., GOURANTON V., ARNALDI B.: Fast Collision Culling in Large-Scale Environments Using GPU Mapping Function. In *Proc. EGPGV* (2012), pp. 71–80. 6

[BBA*07] BICKEL B., BOTSCH M., ANGST R., MATUSIK W., OTADUY M., PFISTER H., GROSS M.: Multi-Scale Capture of Facial Geometry and Motion. *ACM Trans. Graph. 26*, 3 (2007). 2

[BJTM08] BARBARINO G., JABAREEN M., TRZEWIK J., MAZZA E.: Physically Based Finite Element Model of the Face. In *Proc. ISBMS* (2008), pp. 1–10. 1, 2, 3

[CTA*08] COMAS O., TAYLOR Z. A., ALLARD J., OURSELIN S., COTIN S., PASSENGER J.: Efficient nonlinear FEM for soft tissue modelling and its GPU implementation within the open source framework SOFA. In *Proc. ISBMS* (2008), pp. 28–39. 1, 2

[FM08] FLYNN C., MCCORMACK B. A. O.: Finite element modelling of forearm skin wrinkling. *Skin Res. Technol. 14*, 3 (2008), 261–269. 2

[Fra12] FRATARCANGELI M.: Position-based facial animation synthesis. *Comput. Animat. Virtual Worlds 23*, 3-4 (2012), 457–466. 2

[HMSH09] HUNG A., MITHRARATNE K., SAGAR M., HUNTER P.: Multilayer Soft Tissue Continuum Model: Towards Realistic Simulation of Facial Expressions. In *Proc. WASET* (2009), pp. 134–138. 2

[JWM08] JOLDES G. R., WITTEK A., MILLER K.: An efficient hourglass control implementation for the uniform strain hexahedron using the Total Lagrangian formulation. *Commun. Numer. Methods Eng. 24*, 11 (2008), 1315–1323. 3

[KHS01] KÄHLER K., HABER J., SEIDEL H.-P.: Geometry-based Muscle Modeling for Facial Animation. In *Proc. GI* (2001), pp. 37–46. 1, 2

[KRG*02] KOCH R. M., ROTH S. H. M., GROSS M. H., ZIMMERMANN A. P., SAILER H. F.: A Framework for Facial Surgery Simulation. In *Proc. SCCG* (2002), pp. 33–42. 1, 2

[KSY08] KUWAZURU O., SAOTHONG J., YOSHIKAWA N.: Mechanical approach to aging and wrinkling of human facial skin based on the multistage buckling theory. *Med. Eng. & Phys. 30*, 4 (2008), 516–522. 1, 2, 7

[LTW95] LEE Y., TERZOPOULOS D., WATERS K.: Realistic Modeling for Facial Animation. In *Proc. SIGGRAPH* (1995), pp. 55–62. 2

[MC10] MÜLLER M., CHENTANEZ N.: Wrinkle Meshes. In *Proc. SCA* (2010), pp. 85–92. 2

[MJLW07] MILLER K., JOLDES G., LANCE D., WITTEK A.: Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Commun. Numer. Methods Eng. 23*, 2 (2007), 121–134. 2, 3

[NVI12] NVIDIA CORPORATION: *NVIDIA CUDA C Programming Guide Version 4.2*, 2012. 3

[RBM08] REIS C. D. G., BAGATELO H., MARTINO J. M.: Real-time Simulation of Wrinkles. In *Proc. WSCG* (2008), pp. 109–116. 2

[RP07] RÖHRLE O., PULLAN A. J.: Three-dimensional finite element modelling of muscle forces during mastication. *J. Biomech. 40*, 15 (2007), 3363–3372. 2, 7

[SNF05] SIFAKIS E., NEVEROV I., FEDKIW R.: Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data. *ACM Trans. Graph. 24*, 3 (2005), 417–425. 1, 2, 3

[TW90] TERZOPOULOS D., WATERS K.: Physically-Based Facial Modeling, Analysis, and Animation. *J. Vis. Comput. Animat. 1*, 2 (1990), 73–80. 1, 2

[WM12] WARBURTON M., MADDOCK S.: Creating Animatable Non-Conforming Hexahedral Finite Element Facial Soft-Tissue Models for GPU Simulation. In *Proc. WSCG* (2012), pp. 317–325. 1, 2, 8

[WM13a] WARBURTON M., MADDOCK S.: Creating Finite Element Models of Facial Soft Tissue. In *Proc. WSCG* (2013). 2, 7

[WM13b] WARBURTON M., MADDOCK S.: Physically-Based Forehead Animation including Wrinkles. In *Proc. CASA* (2013). 2, 7

[WMSH10] WU T., MITHRARATNE K., SAGAR M., HUNTER P. J.: Characterizing Facial Tissue Sliding Using Ultrasonography. In *Proc. WCB* (2010), pp. 1566–1569. 3

[ZHD06] ZACHOW S., HEGE H.-C., DEUFLHARD P.: Computer-Assisted Planning in Cranio-Maxillofacial Surgery. *J. Comp. Inf. Technol. 14*, 1 (2006), 53–64. 1, 2

[ZST05] ZHANG Y., SIM T., TAN C. L.: Simulating Wrinkles in Facial Expressions on an Anatomy-Based Face. In *Proc. ICCS* (2005), pp. 207–215. 2