Adaptation and Integration of GPU-Driven Physics for a Biology Research RIS

Andreas Knote * Sebastian von Mammen †

Games Engineering
University of Würzburg
Germany

ABSTRACT

Developmental biology studies biophysical processes that lead to the development of tissues, organs, and organisms. Like other complex scientific domains, developmental biology can greatly benefit from real-time interactive systems (RIS). In addition to utilizing various innovative RIS technologies, highly efficient domain models have to be provided as well. In this paper, we present a prototypical RIS for developmental biology research that achieves this goal by adapting an existing, GPU-driven, position-based physics engine called FleX to support the required biological interaction mechanisms. The adapted cell model is further augmented by a GPU-based substance diffusion system that simulates biochemical signals that allow cells to communicate and react to their environment. We present model specifics with an emphasis on their efficient integration in an existing game engine, and we elaborate on future improvements.

Index Terms: Computing methodologies—Interactive simulation; Computing methodologies—Agent / discrete models; Computing methodologies—Real-time simulation;

1 INTRODUCTION

Developmental biology seeks a comprehensive understanding of the processes that lead from single cells to complex organisms. Accordingly, the focus of attention rests on cells and their interactions [19]. The cell also provides a natural level of abstraction for according mathematical and computational models. These models have to consider the cells' physical representation but also their behaviors: "Cells can move, divide, die, differentiate, change shape, exert forces, secrete and absorb chemicals and electrical charges, and change their distribution of surface properties" [19].

Composing simulation models of multiple cells, complex interaction patterns ensue potentially yielding well-studied morphologies, pathologies or unforeseen phenomena. Having a RIS to quickly prototype model instances and to see their results can greatly benefit empirical work. Real-time feedback is all the more important as it allows the biologists to follow different intuitions, to not only statistically analyze cohorts of data but to understand and embrace the systems' dynamics interactively and directly. Hence, before the experimenters invest in costly apparatuses and long, tedious lab routines, various hypotheses can be explored "in-silico" and thereby minimize the effort and maximize the research outcomes. We have created a RIS prototype that combines a virtual biological cell model with an immersive visualization. The agent-based cell model provides mechanical interactions powered by a GPU-driven physics simulation, chemical signaling through a diffusion simulation, and high-level behavior programmable through visual scripting. The physics simulation repurposes the particle-based NVidia FleX physics solver [22] originally developed for visual effects. Interactivity and immersion are realized through UnrealEngine 4 (UE4) [9], a state-of-the-art game engine and a VR interface. This work focuses on the integration of the mechanical cell model and the diffusion simulation with the game engine and evaluates performance and difficulties encountered.

The remainder of this paper is structured as follows. We introduce related research on simulation models and RIS in Section 2. Our current RIS prototype is described in Section 3. The physical cell model and the diffusion model are presented in appropriate depth in Section 4. Section 5 will cover integration of the system components. We analyze the overall performance in Section 6. We then highlight bottlenecks and design issues with a high impact on performance in Section 7. We conclude the paper in Section 8 and further note next steps planned to improve and evolve the system.

*e-mail: andreas.knote@uni-wuerzburg.de
†e-mail: sebastian.von.mammen@uni-wuerzburg.de

2 RELATED WORK

Merks [19] points to the importance of cell-centered models, emphasizing the view of biological systems as complex systems. Biological cells and the concept of autonomous agents both draw from high autonomy and low central control. Gene-regulatory networks are a standard approach to capture cellular behavior in cellular modeling [5]. D-VASim [1], an interactive virtual laboratory environment for the simulation and analysis of gene-regulatory networks, allows the user to load SBML [14] network descriptions and visualize and interact with the simulation state in terms of species concentrations in an environment based on LabView [21].

Physical interactions can and have been modeled in a multitude of ways. Successful models have been based on the Johnson-Kendall-Roberts (JKR) model for strong adhesion of slightly deformable soft bodies [4, 8] or mass-spring-damper (MSD) systems, including torsion coupling [7]. Here, individual cells were often represented by a single point and an explicit or implicit description of their shape. A different approach is found in the Cellular Potts Model (CPM) which is based on a grid lattice [12, 19]. Here, individual cells are comprised of multiple grid cells, and an energy function, the Hamiltonian, combines bonding energies and area or volume conservation constraints into a single equation, which controls the probability density function used for simulation of cell interactions in a Monte Carlo Simulation (MSC). Based on these mechanical interaction models, several simulation platforms have been presented. MecaCell [7] provides a framework for the integration of physical and behavioral models into a biological simulation written in C++ as a header-only library. CellSys [13] and CompuCell3D [24] offer examples for an integrated design and visualisation approach. MecaGen [6] offers a full development environment including behavioral programming through gene regulatory networks. However, these simulations are not interactive in the sense of giving the user the ability to interfere with the simulation state directly at run-time.

In the domain of molecular simulations, such as protein folding, several works on interactive simulations of the processes of interest have been published lately. Lv et al. [17] research an interactive molecular visualization system based on the Unity game engine. They created

a stand-alone viewer for displaying molecular structures, surfaces, animated electrostatic field lines and biological networks. Frey et al. [10] developed a generic system to link molecular simulations with interactive front-ends called MDdriver. Molecule position and energy data is transferred via network to the visualization engine, and user input is in return converted to physical force to be applied to the simulation. A VR prototype using VTK [23] served as a use case, where the user can interact with the molecular simulation in real time.

Considering the presented related work, our research aims for the niche between interactive molecular simulations, which offer real-time interactivity for lower-level molecular interactions, but lack higher-level cell behavior, and the existing biological simulations, which–to our knowledge–do not yet offer real-time interactivity.

3 RIS Prototype

To gather feedback from developmental biologists as early as possible during the development, an interactive, immersive prototype, using current virtual reality (VR) head-mounted displays (HMD) was implemented. Using an interactive editor, the user can recreate biological assays by placing the simulation elements (cells, diffusion grids, physical objects) into a scene and configure them. Cell behavior can be controlled by visual scripting or native code, offering full access to the simulation's features. In Figure 1, the view at runtime of the simulation is shown. The user can place and remove cells or modify density values of the diffusion simulation. Physical forces may be exerted on the cells both through the user or other simulated objects. The user can freely modify the viewport. Using a movable plane, the inside of clusters of cells can be inspected by disabling the rendering of cells above the plane. The density values created by the diffusion simulation can be visualized in a cut-through. To realize the above system, the RIS has three main requirements: (1) A sufficiently detailed simulation model, (2) the efficient handling of user input, and (3) high-quality visualizations of the simulation state. Furthermore, means to augment the display with important state information and the provisioning of appropriate user interfaces needs to be supported.

Based on the requirements, the next section will first present our physical cell and diffusion model to tackle requirement (1). Then, the integration of the simulation component with a state-of-the-art game engine is detailed, which covers requirements (2) and (3) and offers a variety of tools for UI and visual augmentation.

4 Physical Cell Model and Diffusion Simulation

Our biology-tailored RIS adapts an existing real-time physics engine to model the required system mechanics. In this section, we briefly explain the most important aspects of the adapted model. It provides the foundation for integration in the main simulation loop which will be explained in Section 5. We will first introduce our cell model based on the NVidia FleX position based dynamics solver [22] before presenting the GPU based diffusion simulation.

4.1 Position Based Dynamics on GPU

To simulate the physical interactions of cells, we required an efficient physics solver capable of soft body simulation. With the FleX Unified Particle-Based Physics Simulation, NVidia provides a fast and efficient GPU-based physics simulation [18, 20, 22] for soft and rigid bodies. The position-based approach is stable and controllable compared to force-based simulations, trades efficiency for physical correctness, but achieves visually plausible results [2]. Instead of computing the change of momentum of simulated systems as a product of applied forces and numerical integration of accelerations and velocities, positions are calculated directly based on the solution to a quasi-static problem. Thus, problems such as overshooting that occur in force-based systems can be avoided. Collision constraints and penetrations can be handled easily by adjusting the simulated positions directly.

Deformable soft bodies are realized through constraints among sets of particles of clusters. The initial relative positions of the particles within these clusters define their rest positions. The deformation of a cluster is calculated from the overall change of its particles as a single transformation. Vertices of a mesh are associated with one or more clusters using the common computer graphics method of bones, or weighted summed transformations [15]. When rendering soft bodies, the mesh representation is modified using GPU-based mesh skinning [16]. Deformations that result from simulation are captured by according bone transformations. In the given implementation, this process happens at each simulation step.

4.2 Cell Model

We implemented our model following an agent-based, cell-centered [11, 19] approach. Each cell is a reactive agent with an internal state. Different types of cells with individual logic and physical state can be created, and cells can be programmed using visual scripting or C++. By controlling the relative transformations of the particles in a cluster, cell growth can be simulated by translational scaling. Furthermore, this allows realizing non-uniform scaling, such as an increase in length. We have not yet quantitatively evaluated this simulation model against empirical data, e.g. obtained by CT scan time series of developing organisms. Although we will need to rely on a different, validated physics-enabled cell model, our current model provides a baseline for the performance of GPU-based, particle-based soft body simulation in a RIS. Furthermore, it allows to create a working prototype for the evaluation of the overall system concept.

4.2.1 Cell-Cell Adhesion

In reality, cells adhere to each other using sticky proteins on their surfaces, upon which more elaborate structures may be created. We simulate these so-called adhesion junctions by partitioning the cells' surfaces into polygonal regions as seen in Figure 2. In Subfigure (2a), the truncated icosahedron underlying the partitioning can be seen. Subfigure (2b) shows the groups of particles that make up the attachment regions. Among such regions, spring joints can be created at runtime that connect the cells on their surface, as seen in Subfigure (2c) and (2d). Based on this partitioning, the k out-most particles are selected for every region, and the one closest to their common center of mass is designated as the region's pivot. The pivot represents the region for queries, e.g., regarding distances, to allow cells to attach to one another once they get close enough. To avoid degenerated attachments in certain edge cases, e.g., cells attaching to the far sides of other cells when they are overly compressed, additional constraints are enforced. When two regions are connected, only an according entry is made in the cells' list of connected regions at first. The actual spring constraints are created in each simulation step based on the stored connections, as particle indexes may change with every addition or removal of cells. An optimal spring pairing among the particles of both regions (one-to-one) is chosen the first time so that the particles of the local and remote region are paired in such a way as to minimize each pairs' initial distance. This is an attempt to minimize the introduction of unwanted additional forces created by a suboptimal pairing. Spring pairings have to be created within a GPU-CPU synchronized block, as they depend on the particle state. The spring coefficient and rest distance of the springs can be configured for each cell.

4.3 GPU-driven Diffusion Simulation

Diffusion gradients – the relative differences in the concentration of certain substances in space – play a major role in (inter-)cellular processes. We implemented an approximation of substance diffusion.

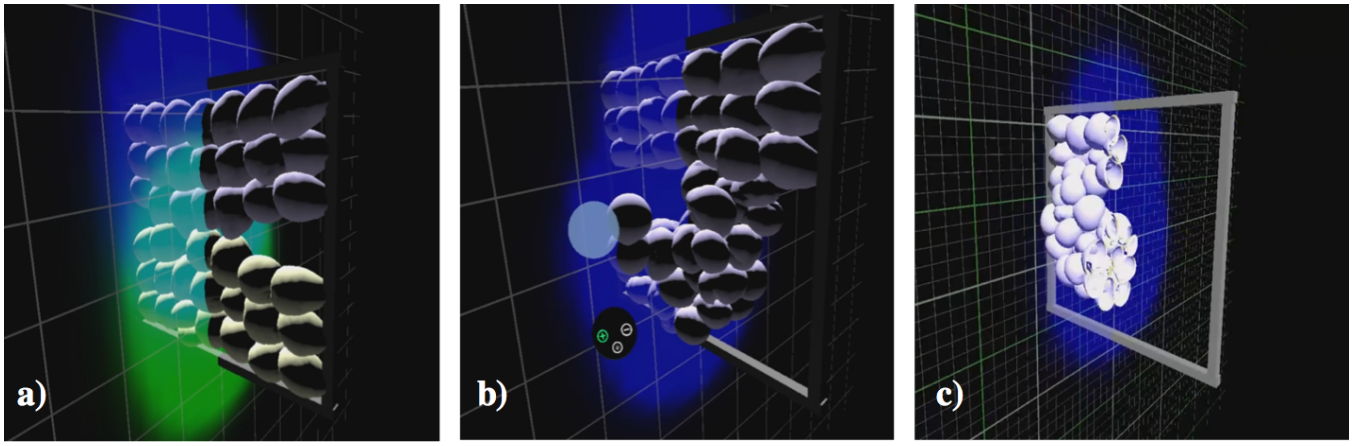

Figure 1: The RIS Prototype as seen through an HMD (field of view cropped). (a) A close-up view of the demonstration assay. A cell cluster, composed of two types of cells, is suspended in a rigid frame. Each cell type (light violet, light green) produces a different morphogenic substance. The substance densities are visualized (translucent blue and green) on a plane. The user can move the plane around by drag and drop. (b) The user can interact with the cells using a hand-held controller, visualized here by the puck-shaped object. For example, new cells can be added to the simulation. Also, forces can be exerted upon cells by holding a button and pushing the virtual controller against them. (c) The user can see the inside of a cluster of cells by defining a culling plane that can be moved and placed freely.

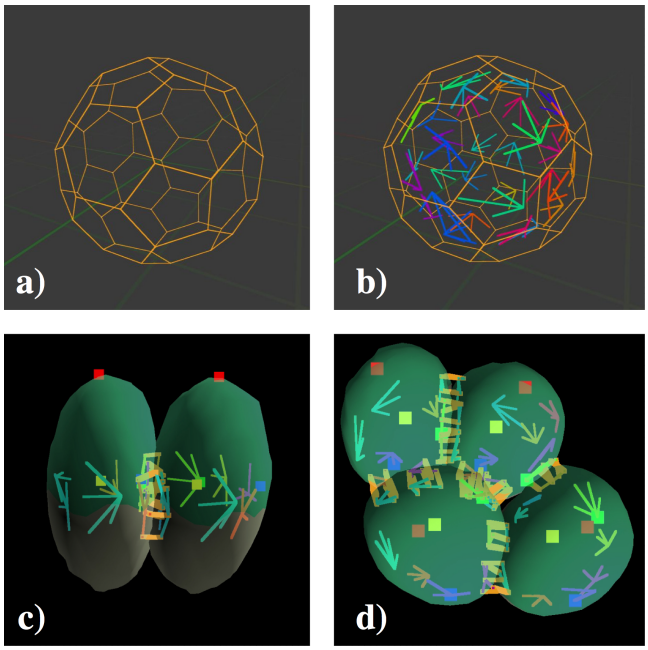

Figure 2: (a) Based on the tessellation of the surface of a spherical cell (orange), attachment regions are created by grouping the k outmost particles. (b) These regions consist of a pivot (center) and $k-1$ other particles. (here: $k=5$ particles connected by colored lines, pivot at center) (c,d) Between regions, spring constraints can be created to simulated surface-based attachments of cells. Specific regions can be activated or deactivated.

It is parallelizable and naturally adapts to a three-dimensional lattice grid.

$$\tilde{u}_{t+\Delta t} = u_t(1-\alpha) \tag{1}$$

$$u_{t+\Delta t} = \tilde{u}_{t+\Delta t} + \frac{\alpha}{\|N\|}\sum_{x\in N} u_t^x \tag{2}$$

We denote the density value at some point in the grid at time t as u_t, and the uniform flow rate between grid cells as α. The time increment between simulation steps is denoted by Δt. The neighboring density values of the point are N_t^d, where d identifies a direction (e.g., to the left, above) within the point's Moore neighborhood (i.e., its immediate neighbors including diagonals). First, the outflow from the lattice grid point is subtracted from the local density, before the influx from all neighboring cells is summed and added, according to Equations (1) and (2). To update the value of a lattice grid point according to Equation (2), only a single write operation is necessary, combined with further $n+1$ read operations on the n direct neighbor densities and the local density. A separate kernel is executed for every point on the grid, each realizing the diffusion model from Equation (2). Kernels inside a block can access a limited amount of shared memory quickly. We make use of shared memory in our implementation by first reading the value u_t of the associated grid point into a shared array. A synchronization point after the read operation makes sure that the cache is filled.

We used a double-buffered approach for the host and the device memory, so that data transfer of the last integration result can happen in parallel to the next iteration and reading access on the host does not require synchronization. To minimize the amount of data transfer from the host to the device, individual emissions of substances are condensed to total emission amounts at pairwise distinct locations. This mapping is realized using a hash map and merging sequential substance emissions to the same grid points. Thereby, instead of storing individual productions and absorption events, the net effect is calculated on the CPU immediately using a map of floats indexed by their index in the 3D grid. This map is then serialized into a list and sent to the device, where each net event is processed in parallel in a grid-stride loop before the integration step.

It is possible use a parallel GPU computation to obtain density gradients for all points in the grid. The complete gradient data for a diffusion grid lattice increases the transfer load by 300%, as three additional values, one vector per point, need to be moved for each

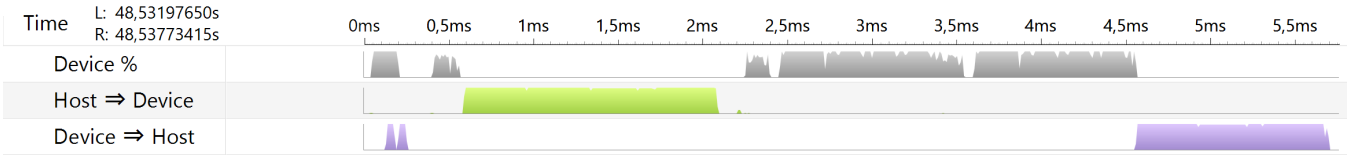

Figure 3: CUDA Benchmarking Results: 49 Cells, 2 diffusion grids (54x54x54), no diffusion simulation substepping. Between 0ms and 0.25ms, the diffusion simulation fully occupies the GPU (grey). The data transfer back to the host (violet) and computation of the two grids overlaps. Next, the FleX simulation is executed. First, modified state data is pushed to the GPU (green) before the solver computation is executed. The resulting state data is then pulled from the GPU.

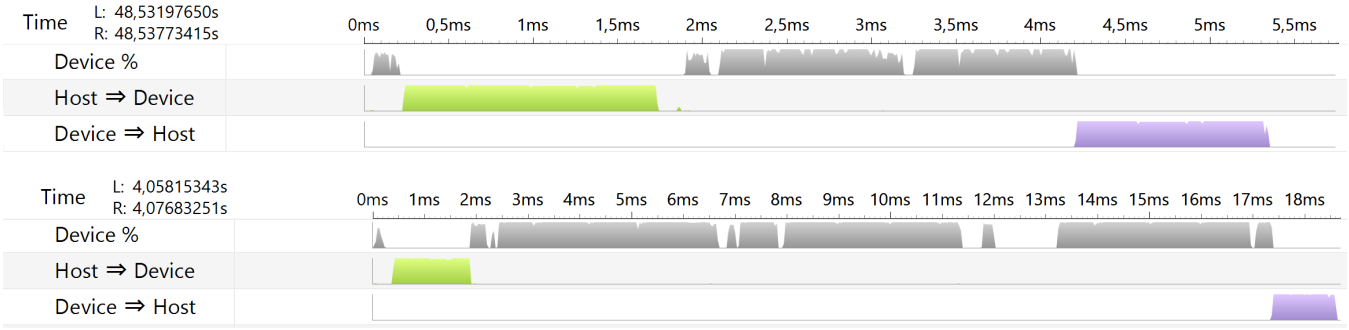

Figure 4: CUDA Benchmarking Results: Flex solver configured for 300000 Particles. 49 vs. 1150 Cells, each comprised of 255 particles. Data transfer times (green, violet) depend on the maximum number of particles and are almost independent of the active particles. Computation time varies with the number of particles (grey).

lattice grid coordinate. Thus, using the GPU to calculate the gradients for all points can be enabled, but by default, gradients are calculated on the CPU.

5 Integration with the Game Engine

We implemented our first RIS prototype relying on UE4 which offers source code access free of charge below certain revenue thresholds [9]. It provides high-quality 3D graphics and easy handling of user input, including spatial controllers common to current virtual reality (VR) gear. NVidia maintains a fork of UE4 that integrates the FleX solver tightly with the engine infrastructure, e.g., by providing dynamic soft body rendering and mesh-particle collisions [22]. Minor adjustments were done to the FleX integration to accommodate dynamic changes at runtime to the underlying constraints that define the shapes of the soft bodies. The interface to define spring constraints was also extended to allow bulk processing new spring definitions as created by surface attachment described above.

5.1 Asynchronous Execution and Synchronization

The following system components were most important for our prototype: The game engine itself, including functionality for graphics, input, audio, the FleX solver, the logic processing of the individual cells, and the diffusion simulation.

In the current implementation, these subsystems are synchronized once during each iteration of the game engine loop. The main loop is partially parallelized internally (i.e., during physics calculations), but can be considered to be executed by a single thread if we only observe modifications of entities through scripting or user input. As, by default, all tick methods on game objects are executed without concurrency, programmatic interaction with other entities or the game world, in general, does not require synchronization.

In Figure 5, the synchronization between systems and components during one iteration are visualized. The FleX solver and the diffusion simulation run on the device and are synchronized with the main engine thread to modify the respective simulation state. At the beginning of an engine loop update, the logic update of the cells

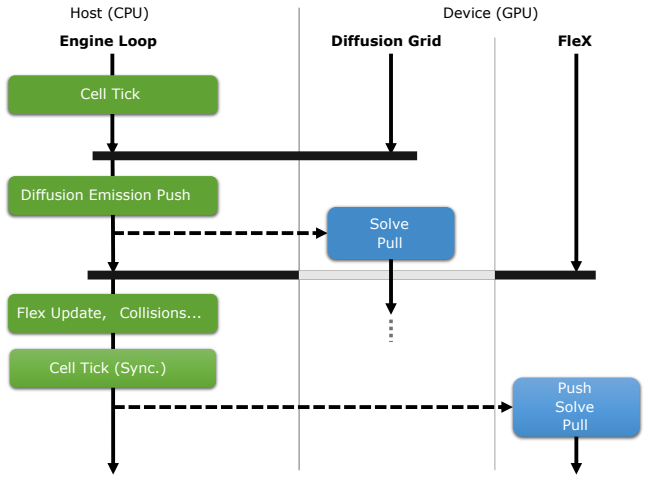

Figure 5: Synchronization barriers and high-level execution steps between the engine loop, the diffusion grid simulation, and the FleX simulation. Outside of the synchronized regions between a synchronization barrier (thick black bars) and the asynchronous calls (dashed arrows), execution of the engine loop on the host and the physics and diffusion simulation on the device are executing in parallel. For FleX, the complete data transfer is asynchronous, and no read access is possible outside of the region. Thus an additional cell logic call (Cell Tick (Sync.)) was introduced inside the safe region.

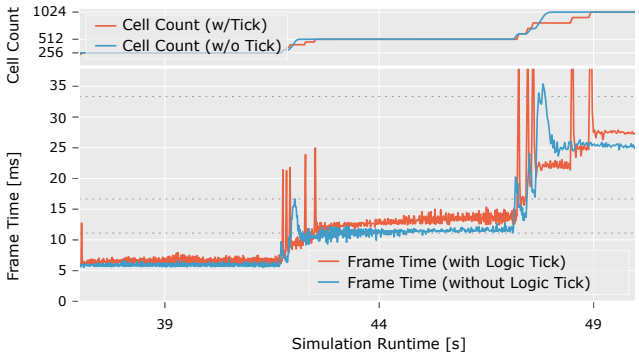

Figure 6: These bottom-line performance charts show the frame times of a simulation with increasing numbers of cells which only feature physics, but no biological logic. The red graph depicts the overhead incurred when the scripting system for cell behavior is enabled.

is executed. Any outstanding GPU work for diffusion or physics simulation can be processed in parallel. At the first synchronization barrier the main loop will block until the latest diffusion simulation data was pulled from the device and made available for reading access during the next update cycle. Any diffusion emissions generated during the current simulation will be merged and sent to the device in a batch to avoid performance degradation through multiple small data accesses. The preparation of the emission events and the swapping of the data pointers are the only non-negligible blocking calls. On the device, the previous output data is updated with the new emissions, and the pointers for in- and output data are swapped. Then the density kernels and, if enabled, gradient kernels are launched, and finally the transfer of data to the host is scheduled. The solver and the subsequent data readback are scheduled and executed in parallel, ending the synchronized region. The data of the diffusion simulation is pulled from the device asynchronously to the main engine loop.

The FleX synchronization barrier asserts that the transfer of the latest FleX simulation state data finished. The FleX data cannot safely be accessed outside this synchronized region, except for a small subset (e.g., cell positions), which is duplicated. The logic of cells can conceptually be separated into parts that modify the physics state or require full access to the data of all particles in the simulation. Accordingly, two tick methods were implemented for cells, one that is executed during the main tick of the engine (pre-physics) and one that is executed during the synchronized phase of the FleX solver and the engine loop. The spring constraints are cleared and recreated from the individual cells' attachment data to account for indexing changes or removed particles. Then, the pushing of the modified FleX state data, the solver step, and the readback of the result is scheduled, and the synchronized region ends. When the simulation is executed on the GPU, there is no further explicit coordination between FleX and OpenGL. CUDA devices are able to schedule multiple processes on the GPU to overlap. However the high number of particles and the diffusion grid points in our simulation usually occupies the GPU completely.

6 Results

To assess the performance of the system in more detail, we will first look at the overall achievable frame times in relation to the number of cells active in a synthetic simulation setup. Then, we will analyze the performance of FleX and the diffusion simulation on the CUDA device. Based on this, bottlenecks will be pointed out. The timings were obtained using NVidia NSight CUDA profiler, the UnrealEngine profiler, an NVidia GTX 1080 and an Intel i7-6700k 4GHz, VisualStudio 2015 and the customized UnrealEngine 4.12.5 with FleX.

6.1 System Performance

We benchmarked our system using a simplified "test"-cell with no active logic or biological behavior to examine the bottom line performance.

6.1.1 Overall Performance

In Figure 6, obtained frame rates using test cells in an otherwise empty simulation are shown. Frame times depend on the number of cells active in the simulation. For 256 cells, 5ms of frame time remain available for additional logic computation while staying below 11ms (90Hz) in total. Such setups would be well-suited for VR applications. At 512 cells, 16.6ms (60Hz) for non-VR applications can be maintained. At 1024 cells, frame times around 27ms (30Hz) are suitable only for less demanding applications. The spikes in frame times are a result of the instantiation of new cells in the simulation. There is a noticeable difference between the simulation runs depending on whether cell logic ticks are enabled. As there is no additional logic executed (no actual logic is implemented), the spikes likely stem from an overhead in UE when instantiating new entities which make use of the scripting system.

6.1.2 CUDA Performance

Our model currently uses 255 particles per cell. After configuring FleX for 300k particles to accommodate a maximum of 1150 cells, about 17 MiB of data needed to be copied to and from the device at each frame. The data comprises, among other, the complete state data of the simulated particles (position, velocity, mass). Constraints, such as springs, are pushed to the device, but not read back (this functionality was lacking in the used FleX release). Particle data is allocated a-priori on the GPU and remains constant in size. Constraint data is dynamically added and removed.

A complete GPU execution timeline can be seen in Figure 4. First, the diffusion grid calculation is executed, which takes about 200 μs, including data transfer to and from the GPU. Subsequently, the FleX simulation is executed, beginning and ending with a relatively large amount of data transfer. The transfer costs are fixed and independent of the actual number of cells, compare Figure 3. For 1150 cells, the time required for executing the physics solver on the GPU is about 11ms. The execution time dependends on the number of particles and constraints that are active in the simulation.

Multiple diffusion grids can be placed in the simulation. As the number of grid points usually exceeds the number of available threads, and one thread currently updates one grid point, the GPU is fully utilized when performing the diffusion simulation. Thus, only data transfers overlap with the diffusion kernel execution. In Figures 3 and 7, the execution timings for simulations with two diffusion grid instances can be seen. Measured kernel execution times per substep are 125µs and data transfer times 48µs (615 KiB). If the grid resolution is increased to 93x93x93, measured kernel execution times are 650µs per substep and data transfer times 245µs (3.1 Mib) (not depicted). In Figure 7, the amount of data transfer saved by only transferring net changes is shown (448 bytes vs. 615 KiB).

7 Discussion

The integration with existing engine technologies can reduce development efforts significantly, especially in domains related to visualization and the capture of user input. For small numbers of cells, data transfer times between the GPU and the host dominate the execution time. However, CPU logic – which was not benchmarked here, as it is highly dependent on the implementation and the type of behavior to be realized – is performance intensive and not easy to parallelize using the standard features of UE4 without duplicating data structures, as most API methods are not thread-safe. Reusing

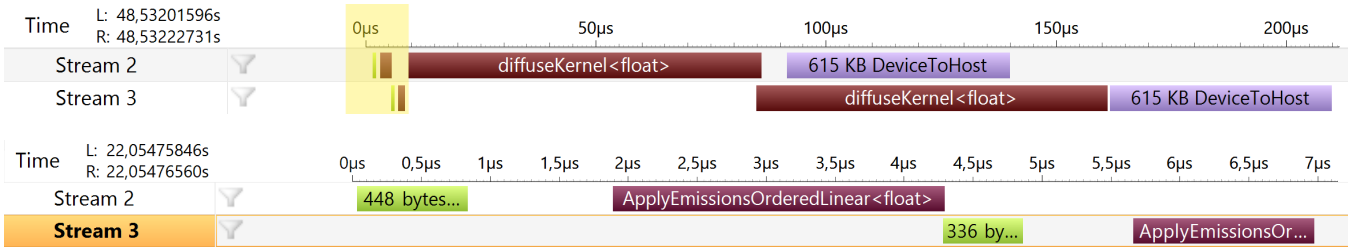

Figure 7: CUDA Benchmarking Results: 2 diffusion grids (54x54x54). Top: A complete update on the GPU including all data transfers. Bottom: Cropped to host-device transfer at the beginning of the diffusion grid update. To minimize data transfer between host and device, only delta changes (e.g., emissions from cells or user input) are sent to the device. Notice the time scale (μs).

the implementation of FleX with UE4 was chosen as it allowed to quickly test the modeling, configuration, and overall feasibility of the interactive cell simulation. However, the tight integration with the engine loop limits scalability. Although the actual computation of FleX is asynchronous to the main loop, it is synchronized every tick. As the benchmarks showed, even at just 1k cells, the physics simulation time alone would exceed 11ms, making it unsuitable for VR applications as long as the simulation-visualization lockstep is maintained.

The spikes in frame times when instantiating large numbers of new cells might be mitigated by pre-allocating cells ahead of the simulation or potentially continuously during simulation runtime. Currently, this would have to be done for each potential cell type individually, as the cells are subclasses of a common cell class and, as such, do not necessarily share a common memory layout. The FleX representation of a cell, which is comparatively light weight in terms of memory consumption, barely contributes to the spikes.

As depicted in Figure 5, a decision was made to separate logic from physics updates and partially synchronize the processes. The current exemplary simulations were not extensive enough to show wether the resulting implementation bias qualitatively reduces the usefulness of the simulation.

Better decoupling of the engine loop and the simulation loop is required. It is important to maintain a clean separation between the visualization and interaction components and the cell simulation engine. This might also benefit the maintainability and reusability of the individual components. Increasing the simulation tick time and, optionally, using a smoothing algorithm to cover up visual artifacts, could allow the interactive exploration of rather demanding (regarding computational time per simulation step) scenarios, as long as an appropriate visual representation can be found. To this end, an abstraction layer similar to MDDriver [10] seems a reasonable approach. Here, only essential information about the position of individual atoms and energy fields is transmitted, allowing to reconstruct the full visual image based on predefined visual representations. Obviously, this allows decoupling the simulation and rendering loops. In the long run, research into the integration with decentralized simulations might be worthwhile by spreading the computational load on multiple machines.

The diffusion simulation currently operates in a brute-force manner. As no sparse representation is used, every grid point has to be calculated by and the complete state data pulled from the GPU. Detection of empty regions, where all densities including the marginal areas could considerably reduce the transfer and computation times. Furthermore, the medium is treated as being static, without any macroscopic flow taking place. Further analysis needs to be done to decide on whether simulating this aspect is really necessary.

Regarding the visualization of morphogen densities, using cutting planes is merely a stop-gap solution that should be replaced by volume rendering [3], providing higher visual fidelity and a more fine-grained spatial representation.

The use of game engine technology, and especially a physics engine with its focus on visual plausibility, naturally raises concerns on applicability of the results. For this specific prototype, we were motivated to create a simulation of sufficient scale, including surface-based mechanical interactions of deformable cells. The system has not yet been quantitatively evaluated against empirical biological data or models. The exploitation of the GPU for this task showed great potential, and the availability of a functional soft body physics simulation enabled the creation of an interaction prototype to explore future research directions. The authors are fully aware of the limited applicability of the quantitative results. We will explore the integration of proven simulation models into interactive systems in the future. However, providing a simulator that integrates high level behavior through a phenomenological cell model, mechanical interactions and chemical messaging was well received by empiricists and modelers specialized in biological development at several opportunities of live testing. These informal evaluations make a case for the further research in more realistic real-time suitable simulation model.

8 Conclusion

We presented our current prototype RIS for biologists which realizes an immersive simulator for the study of morphological development. In particular, we provided an overview and elaborated on the technical details of model adaptation and integration in the context of the UE4 platform. The system integrates several key components that are required to support the research of the domain experts in developmental biology. These include a mechanical cell model that incorporates physicality and surface interactions between cells and a diffusion simulation that models the signaling systems of biological cells.

There are obvious shortcomings of the current approach: We made various design decisions to integrate and adapt existing sub-engines for the biological simulation, and even at a lightweight simulation of cells only driven by physics and not also by higher level biological logic, interactive frame rates cannot be maintained well with rising numbers of cells. However, we have identified several means to mitigate this challenge. First, the physics simulation, selected for its anticipated speed and flexibility, will need additional scrutiny. The diffusion simulation provides a substantial speed-up over similar CPU based solutions, but is limited in its accuracy and does not scale well. Most importantly, the current approach exposed the tight coupling of simulation and visualization loop, making it hard to keep frame times stable in demanding simulation setups.

Despite these necessary improvements, feedback by domain experts has been very encouraging. We plan on introducing high-level behavior definition based on gene regulatory networks to better represent current research and modeling approaches. Efforts are being made to adapt the cell model to the simulation of craniofacial development in cooperation with biologists, incorporating and evaluating the proposed system changes. Furthermore, the planned integration

of existing simulation models into a RIS will provide apt testing grounds for interaction principles, visualization, and the applicability of interactive simulation to biological research.

REFERENCES

[1] H. Baig and J. Madsen. D-vasim: an interactive virtual laboratory environment for the simulation and analysis of genetic circuits. *Bioinformatics*, 33(2):297–299, Sep 2016.

[2] J. Bender, M. Müller, and M. Macklin. Position-Based Simulation Methods In Computer Graphics. In *EUROGRAPHICS 2015 Tutorials*. Eurographics Association, 2015.

[3] J. Beyer, M. Hadwiger, and H. Pfister. A Survey of GPU-Based Large-Scale Volume Visualization. In R. Borgo, R. Maciejewski, and I. Viola, eds., *EuroVis - STARs*. The Eurographics Association, 2014.

[4] Y.-S. Chu, S. Dufour, J. P. Thiery, E. Perez, and F. Pincet. Johnson-kendall-roberts theory applied to living cells. *Physical Review Letters*, 94(2), Jan 2005.

[5] E. Davidson and M. Levin. Gene regulatory networks. *Proceedings of the National Academy of Sciences*, 102(14):4935–4935, Apr 2005.

[6] J. Delile, M. Herrmann, N. Peyriéras, and R. Doursat. A cell-based computational model of early embryogenesis coupling mechanical behaviour and gene regulation. *Nature Communications*, 8:13929, Jan 2017.

[7] J. Disset, S. Cussat-Blanc, and Y. Duthen. MecaCell: An Open-source Efficient Cellular Physics Engine. In *07/20/2015-07/24/2015*, p. 67. The MIT Press, 2015. doi: 10.7551/978-0-262-33027-5-ch014

[8] Y. M. Efremov, D. V. Bagrov, M. P. Kirpichnikov, and K. V. Shaitan. Application of the Johnson–Kendall–Roberts model in AFM-based mechanical measurements on cells and gel. *Colloids and Surfaces B: Biointerfaces*, 134:131–139, 2015. doi: 10.1016/j.colsurfb.2015.06.044

[9] Epic Games Inc. UnrealEngine 4. `https://docs.unrealengine.com/latest/INT/`, last retrieved: 10.1.2018, January 2018.

[10] N. Férey, O. Delalande, G. Grasseau, and M. Baaden. From interactive to immersive molecular dynamics. In *Proceedings of the Fifth Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2008, Grenoble, France, 2008.*, pp. 89–96, 2008.

[11] A. Gelfand. The Biology of Interacting Things: The Intuitive Power of Agent-Based Models: Biomedical applications of ABMs are taking off. *Biomedical Computation Review*, 2013.

[12] J. A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Phys. Rev. E*, 47:2128–2154, Mar 1993.

[13] S. Hoehme and D. Drasdo. A Cell-Based Simulation Software for Multi-Cellular Systems. *Bioinformatics*, 26(20):2641–2642, 2010. doi: 10.1093/bioinformatics/btq437

[14] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, , the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, and et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003.

[15] A. Jacobson, Z. Deng, L. Kavan, and J. Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014.

[16] D. L. James and C. D. Twigg. Skinning mesh animations. In *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 399–407. ACM, 2005.

[17] Z. Lv, A. Tek, F. Da Silva, C. Empereur-mot, M. Chavent, and M. Baaden. Game on, science - how video game technology may help biologists tackle visualization challenges. *PLoS ONE*, 8(3):e57990, Mar 2013.

[18] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Trans. Graph.*, 33(4):153:1–153:12, July 2014. doi: 10.1145/2601097.2601152

[19] R. M. H. Merks and J. A. Glazier. A cell-centered approach to developmental biology. *Physica A: Statistical Mechanics and its Applications*, 352(1):113–130, 2005. doi: 10.1016/j.physa.2004.12.028

[20] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. In *Proceedings of the Third Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2006, Madrid, Spain, 2006.*, pp. 71–80, 2006. doi: 10.2312/PE/vriphys/vriphys06/071-080

[21] National Instruments. Labview. `http://www.ni.com/en-us/shop/labview.html`, last retrieved: 6.2.2018, 2018.

[22] NVidia Corporation. NVidia Flex Physics. `https://developer.nvidia.com/flex`, last retrieved: 10.1.2018, 2018.

[23] W. Schroeder, K. Martin, and B. Lorensen. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition*. Kitware, 2006.

[24] M. H. Swat, G. L. Thomas, J. M. Belmonte, A. Shirinifard, D. Hmeljak, and J. A. Glazier. Chapter 13 - Multi-Scale Modeling of Tissues Using CompuCell3D. In A. R. A. Arkin and A. P., eds., *Methods In Cell Biology : Computational Methods In Cell Biology*, vol. Volume 110, pp. 325–366. Academic Press, 2012. doi: 10.1016/B978-0-12-388403-9.00013-8