Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1283–1290
DOI: 10.15439/2017F191
ISSN 2300-5963 ACSIS, Vol. 11

Mobile devices' GPUs in cloth dynamics simulation

Marcin Wawrzonowski
Institute of Information Technology,
Lodz University of Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
Email: 180729@edu.p.lodz.pl

Marcin Daszuta
Institute of Information Technology,
Lodz University of Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
Email: 173059@edu.p.lodz.pl

Dominik Szajerman
Institute of Information Technology,
Lodz University of Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
Email: dominik.szajerman@p.lodz.pl

Piotr Napieralski
Institute of Information Technology,
Lodz University of Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
Email: piotr.napieralski@p.lodz.pl

***Abstract*—The realistic simulation of cloths is nowadays a key to produce good-quality, authentic graphical visualizations of various cloth, such as characters garment elements, flags or curtains. This can be computationally expensive, more and more as number of particles, which cloth is divided into, increases. The solution to this matter was to use GPU (Graphic Processing Unit) and perform all calculations on this device. On PC platform, this technique proved to be much faster than the standard CPU approach. The main purpose of this work is to check whether this solution could also be introduced on the mobile devices. In this paper, we developed fast vertex optimization methods for dynamic cloth in mobile GPU units. Additionally we develop a user interface which providing new ways of user interaction with a cloth dynamics simulation on mobile devices.**

I. Introduction

NAVIGATION, 3D models and interactive performance has significant role in computer games and other interactive graphics applications [1]. Cloth dynamics simulations are an important visual cue for creating believably objects in virtual environments. The beginnings of cloth simulation in computer graphics appeared the end of the 80's [2]. First methods employs finite differential equations for the behavior of non-rigid curves, surfaces, and solids as a function of time for elastically deformable models (Lagrange equations of motion). The next significant step was the work of Baraff and Witkin [3]. They presents fast system for enforcing constraints on individual cloth particles with an implicit integration method. Since this time many methods extends the implicit time integration of Baraff and Witkin. Eberhardt et al. [4] propose the solution of the differential equation for particle systems to be computed both correctly and very quickly. They use an IMEX method (Implicit-Explicit) to simulate draping textiles.

Parks and Forsyth [5] propose the improved Runge-Kutta method. Improvement bring some advantages for cloth simulation. Different class of methods use precomputed data. Feng et al. [6] propose hybrid method for real-time cloth animation. They use relationship between cloth deformations at two resolutions. Data transformation is trained using rotation invariant quantities extracted from the cloth models, and is independent of the simulation technique chosen for the lower resolution model with fast collision detection. Algorithm was implemented on programmable graphics hardware to achieve an overall real-time. Hahn et al. [7] propose low-dimensional linear subspace clothing simulation using adaptive bases. This was a combination of machine learning with a dynamically updated subspace basis. This approach is not fast enough for real-time applications because requires close-fitting clothing rigged to a skeleton and a set of training simulations for learning step. Gillette et al. [8] propose framework that does not require training data or a reference shape. They use a two-pass method. First pass is segmentation technique to extract spatially and temporally reliable surface motion patterns. Second pass is the detection of motion patterns to compute adaptive reference shape and a stretch tensor to dynamically generate new wrinkle geometry on the coarse cloth mesh by taking advantage of the GPU tessellation unit. There are many methods that aim for faster cloth simulation. Most of presented algorithms is suitable for the current generation of consoles and PC graphics cards [9]. Popular multi-model framework SOFA for interactive physical simulation for researchers and developers is dedicated to PC platform [10].

The main purpose of this work is to check whether this solution could also be introduced on the mobile devices. Most of them nowadays also have their own specialized GPU chips. General Purpose GPU Computing is mentioned, along with GPU framework and a comparison between it and a CPU is made, in the matter of architecture and performance. Presented implementation on mobile devices has mid-range GPU can perform very well, producing smooth animation of cloth's dense mesh, but not without a few important limitations. These include less useful API functions and shorter work time on battery as a result of intensive computations and tendency to overheating.

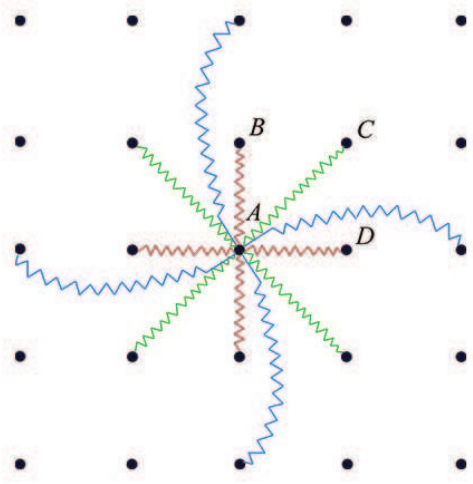

Fig. 1. Diagram of the mass model on the spring. The colors indicate all the springs involved in the vertex position calculation

II. Simulation methods

The most popular approaches for the simulation of Real-time Cloth Animation systems in computer graphics take into account discretize the cloth by a polygonal mesh. This approaches to simulating dynamic objects based on the use of forces. We can distinguish two methods of simulating these forces for the cloth simulation: "Spring Force Formulas" [11] and "Position Based Dynamics" [12].

A. Spring Force Formula

Real-time cloth simulation for games typically uses a mass and spring system on a coarse mesh [11]. These mass and spring systems form a series of differential equations that are typically integrated using a stable integration method [13].

Real-time cloth simulation is rendered by graphical API, as a polygonal mesh with grid of vertices in 3D space. For simulations, each of these vertices had a mass and was subjected by force formulas for the displacement. In order to preserve the shape and the mesh behavior, the vertices are connected in rectangular grid, and then connected each vertex to neighboring vertex with springs. Springs has specific coefficients of elasticity and damping (Fig. 1).

There are three types of springs that appear in the presented model (Fig 1.)

- Structural springs (red) - they are used to maintain the general shape of the cloth.
- Springs for folds of the cloth (green) - they are located along the diagonal edges of the grid.
- Springs responsible for flexibility of the cloth (blue) - they protect against excessive stretching. They do not connect neighboring vertices, but follow the neighbor in the same direction.

Each type of spring can be described by other coefficients of elasticity and vibration damping, which allows to simulation of specific behavior. Figure 2 shows that the forces affect for each point of mass.

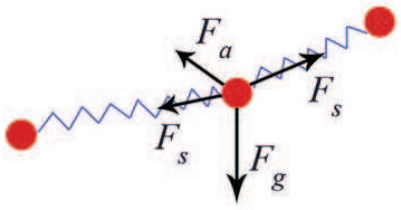

Fig. 2. Forces for a single vertex

The forces can be classified as internal and external. Gravity and collision forces are examples of external forces. Examples of internal forces are elastic forces in deformable objects or viscosity and pressure forces in fluids. To determine its value, the Hooke's Law is used to define the force of the spring and its direction and return are proportional to the pitch of the spring, ie the difference in distance between its present length and its resting length. Each vertex (i) is connected to its neighbor with 12 springs:

$$\mathbf{F_{se}} = -\sum_{j=0}^{j<12} k_s(|\mathbf{x_i} - \mathbf{x_j}| - l_{(i,j)}) \cdot \frac{\mathbf{x_i} - \mathbf{x_j}}{|\mathbf{x_i} - \mathbf{x_j}|} \ , \qquad (1)$$

where k_s - elasticity coefficient $\mathbf{x_i}$ and $\mathbf{x_j}$- Position of vertices connected by one spring $l_{(i,j)}$ - The distance between these points at relaxation vector.

Also the force of elastic vibration damping has been introduced to minimize unnecessary unrealistic vibration and risk of out of control simulation:

$$\mathbf{F_s} = \sum_{j=0}^{j<12} -k_s(|\mathbf{x_i}-\mathbf{x_j}|-l_{(i,j)}) \cdot \frac{\mathbf{x_i} - \mathbf{x_j}}{|\mathbf{x_i} - \mathbf{x_j}|} + k_d\left(\frac{|\mathbf{x_i} - \mathbf{x_j}| \cdot |\mathbf{v_i} - \mathbf{v_j}|}{l_{(i,j)}}\right) \qquad (2)$$

where k_d - vibration damping factor

These mass and spring systems form a series of differential equations that are typically integrated using a stable Verlet integration method, this method stores the velocity implicitly as the difference between the current and the last position:

$$\mathbf{x}(t + \delta t) = 2\mathbf{x}(t) - \mathbf{x}(t - \delta t) + \mathbf{a}(t)\delta t^2 \ , \qquad (3)$$

where $\mathbf{x}(t+\delta t), \mathbf{x}(t), \mathbf{x}(t-\delta t)$ - indicate the position of the vertex in the next, current, and previous simulation step. $a(t)$ - acceleration. This solution imposes an implicit calculation of the current vertex speed. This makes it necessary to provide not only the current position of each mass point, but also the location of the previous one. This increases the memory cost of the simulation to other integration techniques, but provides very fast calculations and stable results.

B. Position Based Dynamics

The model based on the position and the mass model on the spring have a common part - it is the calculation of shifts caused by gravitational forces and air resistance by the Verlet integration. The shifts resulting from the external forces are

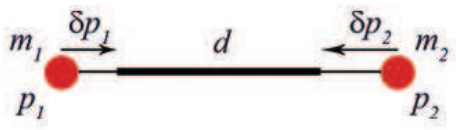

Fig. 3. Diagram of operation of the limiters between two points of mass

called predicted shifts. Each vertex of the grid is described, apart from mass, position and velocity, also by the so-called limiters set. Each of them is defined by a certain function $C_j : R^{3n_j} \rightarrow R$ Set of indices $\{i_1, \ldots, i_{n_j}\}, i_k \in [1, \ldots, N]$ i- stiffness parameter, $k \in [0 \ldots 1]$. The limiter may be of the type of equality, which means that its limitation is fulfilled when $C_j(x_{i_1}, \ldots, x_{i_{n_j}}) = 0$. It can also be the type of unevenness, with the condition $C_j(x_{i_1}, \ldots, x_{i_{n_j}}) \geq 0$. In this case only the first type stops will be considered. The key element of course is the function C_j, which defines how the predicted position will be improved, where this improvement depends - that is, the behavior of the cloth.

The basic type of limiter is the stretch limiter. It is defines the overall shape and proper behavior of the cloth. Its function is:

$$C(\mathbf{p_1}, \mathbf{p_2}) = |\mathbf{p_1} - \mathbf{p_2}| - d \ . \tag{4}$$

where $\mathbf{p_1}$ and $\mathbf{p_2}$ are the positions of the considered vertices, and d - the initial distance between them.

Li et al [14] propose function solution $C_j(x_{i_1}, \ldots, x_{i_{n_j}})$:

$$s = \frac{C_j(\mathbf{p_{i_1}}, \ldots, \mathbf{p_{i_{n_j}}})}{\sum_j w_j |\nabla_{\mathbf{p_j}} C_j(\mathbf{p_{i_1}}, \ldots, \mathbf{p_{i_{n_j}}})|^2} \ , \tag{5}$$

where:

$$\delta \mathbf{p_i} = -s w_i \nabla_{\mathbf{p_i}} C_j(\mathbf{p_{i_1}}, \ldots, \mathbf{p_{i_{n_j}}}) \ . \tag{6}$$

where w_i - inverse mass of vertex. This two simulation methods, it should be noted that each of them has its pros and cons. The greatest advantage of the spring mass model is its ease of simplicity and ease of implementation. It is easy to imagine a cloth as a collection of vertices connected by elastic springs, whose elastic forces are calculated using the simple laws of physics. Certainly the biggest advantage of a position-based model is the performance advantage. It results from the lack of need to use numerical integration. The cloth behavior is not determined by the set of resilient forces, and the limiters immediately modify the position. This allows for significant computational savings. In case of a spring mass model, these calculations can not be avoided for each of the springs. For more accurate results, more complex integration methods should be used. This leads to a decrease in productivity.

C. Improved Position-Based Method

Considering that the displacement is directly proportional to weight, it is easy to consider that - if the mass of the particle is infinite, the offset will be equal to zero. When function $C_j(x_{i_1}, \ldots, x_{i_{n_j}})$ will be replaced by $C(p_1, p_2 = |p_1 - p_2| - d$, we can get the following stretch limiter:

$$\delta \mathbf{p_1} = -\frac{w_1}{w_1 + w2}(|\mathbf{p_1} - \mathbf{p_2}| - d)\frac{\mathbf{p_1} - \mathbf{p_2}}{|\mathbf{p_1} - \mathbf{p_2}|} \ , \tag{7}$$

$$\delta \mathbf{p_2} = \frac{w_2}{w_1 + w2}(|\mathbf{p_1} - \mathbf{p_2}| - d)\frac{\mathbf{p_1} - \mathbf{p_2}}{|\mathbf{p_1} - \mathbf{p_2}|} \ . \tag{8}$$

As with the spring mass model, the 'force' of the limiter depends on the difference between the current distance between the mass points and the resting distance. The coefficient of elasticity is like the stiffness parameter multiplied by offset (result from the projection). For k equal 0, the delimiter will not be taken into account at all. For k equal 1 the point never changes its initial position.

In the presented method there were delimiter of bending. This method uses other collision detections. Most of the methods are based on a baseline approach where stretchers are used, taking into account only vertices located in the neighborhood of a given point. Experiments have shown that the effect similar to the use of bending delimiters can be achieved by increasing the set of considered vertices by one more position from the mesh. This is not the exact like method of bending deflection, where we adjust the angle between the triangles, but still gives the correct visual effect with better performance. The presented solution include bounding spheres and AABB in the case of external collision and the bounding spheres in the internal collision.

III. A CPU-GPU FOR REAL-TIME CLOTHING ANIMATION

Optimizing graphics performance for GPU vs. CPU are quite different. The CPU has too many vertices to process. Rendering is not a problem on the GPU or the CPU, there may be an issue for physics of cloth (dynamic forces). It is quite important to get a good performance on mobile GPUs. Mobile GPUs are less powerful like low-end PC GPUs. CPU commonly has 4 to 8 fast, flexible cores, GPU's has massive parallelism (Fig. 4). This highly parallel architecture is the reason that a GPU can quickly process large number of data (dynamic cloth simulation).

Development of such experiments requires "Application of Experimental Test". Setting goals and objectives for experiment accomplishes key objectives. First task, is presentation of two models of textile simulation. It is important to compare them in terms of performance, stability and visual effect.

Performance is understood as the time for calculate one step of simulation. The application informs the user about it by displaying the relevant information in a textual form. As for the next two factors, it is best to evaluate the cloth visually simulation - visualization. For this purpose, the program draws it in 3D space. The key issue here is the interaction with other 3D objects. The purpose of this paper is also to compare

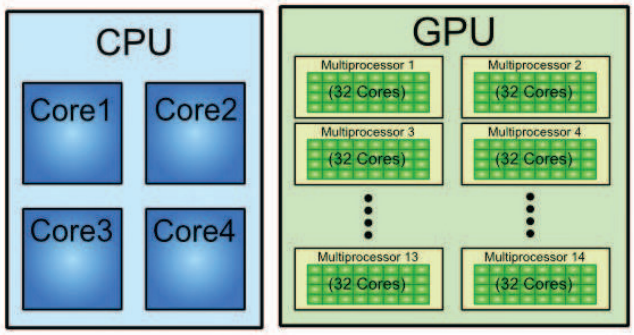

Fig. 4. Typical CPU's architecture vs. a typical GPU's architecture (source http://blog.goldenhelix.com/)

the speed of simulation calculations of real-time cloth models simulation on CPUs and GPUs, and to examine the difference in GPU performance of the mobile device and the GPU of the PC. For the first case, at each step, the appropriate GPU assignments should be assigned to the vertices of the vertices that contain the pre-generated data needed for the simulation. Then set graphics library to the computation program mode, set all homogeneous variables, bound buffers homogeneous, and run the transform join. Case for the CPU is much simpler as all data and arrays have already been initialized in the process described in the previous section and simulation can start immediately. The situation is complicated when using multithreading. In this method, four working threads were broken up, because the device has four physical processing units. The division algorithm is simple - the number of vertices is complemented by the number divisible by 4 and divides it into four equal ranges, with the first three being considered, and the last one being equal to the number of other vertices. Mutexes and thread counters have been used for synchronization. Work threads manage the main thread. Each of the former is slumbered into the mutex until the simulator function is called. Then they wake up and start calculating. After they have finished raising the counter and waiting for the next mutex. The main thread at this time waits until the counter reaches the required value and unlocks the next calculation step.

The process was divided into three separate stages. The first step is to calculate the movement of the cloth according to the accepted simulation model. The second step is solving collisions and applying cloth movement resulting from user interaction. The third stage is the conversion of normal vectors. After the first two steps, the input and output data identifiers are exchanged. For both implementations on the CPU, after completing the processing step, you still need to submit new position and vector data for normal vertices to the GPU, so that they can be drawn.

All possible data that does not need to be recalculated at each step is calculated during the initialization of the simulator, and the results are simply passed to the corresponding functions during the program run. This is perfect for the GPU programming methodology. This solution minimizing the number of conditional statements and avoiding unnecessary calculations that are repeatedly performed. Each vertex will be assigned a list of identifiers and multipliers that are 1 when the neighbor exists or 0 if it is not, and in this case the calculated force or displacement does not take part in further processing. That also eliminates the need for conditional commands, which further improves performance. Each vertex has the following attributes:

- position (16 Bytes),
- texture coordinate (8 Bytes),
- normal vector (16 Bytes),
- color (16 Bytes),
- centrobaric coordinate (16 Bytes),
- index (4 Bytes).

Simulation of clothes requires the definition of a large number of parameters. Initially they are initialized on the CPU side. Some of them may be different for each vertex, so they are passed to the GPU in the form of array attribute values.

IV. User interface for interaction with a cloth dynamics simulation on mobile devices

Very important for real-time visualization is the ability to interact by the user with cloth by Graphical User Interface (GUI). User can easily to work with software and collect data for the test results. The program can draw two-dimensional GUI elements in the screen space, such as text dynamic fields, real-time animation and buttons. User input requires different handling in a mobile application like addition to the on-screen input methods. The application design assumes that the user must be able to reposition, rotate and zoom the camera, reset the simulation and modify its parameters, change the object display mode and interact with the cloth in two ways. The first way is to move the object to collide with the clothes. The second way is to move the clothes with finger movements. It is also required to inform the user about the speed at which the simulation is running and what parameters it currently has and what type it is (Fig. 5).

There are two ways for interaction with a cloth by the user. By moving an object (sphere or cuboid) with which it collides, or by means of a touch screen. In the first case, the effects are applied when solving external collisions. For the second method, special calculations need to be made to know which vertices need to be further shifted to which direction and to what extent.

The only input data are two two-dimensional vectors, called "touch vectors". One specifies the place on the screen where user touched the screen, and the second is the direction in which users finger moves. They were expressed in screen space. In order to make a vertex translation, important is a vector position in that space. It is obtained by multiplying it successively by world matrices, view matrix and projection matrix, and dividing the result by component w. In this way, a vertex vector with components is obtained in the range $< -1, 1 >$, same as the touch vector. Next, using the Gaussian

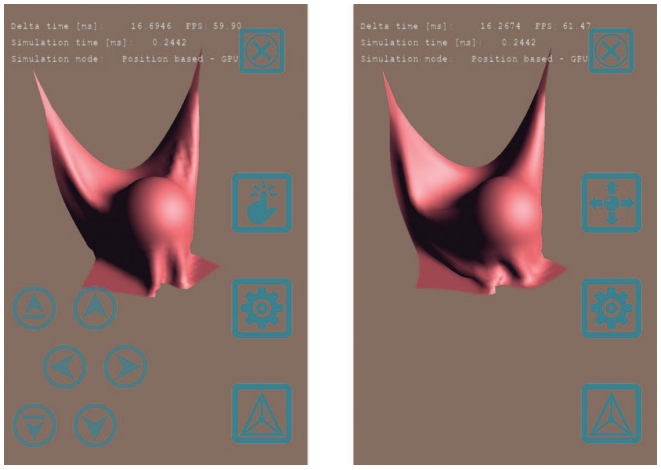

Fig. 5. Interactive Graphical User Interface and results of the simulation in real-time

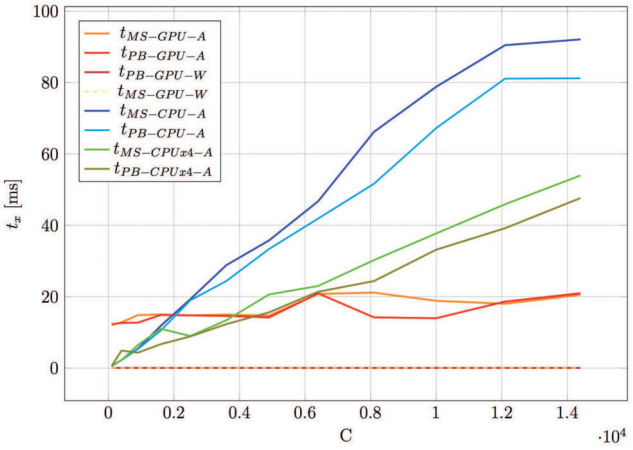

Fig. 6. Graph of time dependence on the number of vertices.

formula, the c coefficient is calculated to determine how far the translation will take place. It is directly proportional to the distance of the vertex position from the touch point:

$$c = Ae^{\frac{(\mathbf{p}t_x - \mathbf{p}i_x)^2 + (\mathbf{p}t_y - \mathbf{p}i_y)^2}{2\sigma}}, \quad (9)$$

where A and σ are top-defined constants and they are respectively 200 and 300, while $\mathbf{p}_t$ are the position of the touch, $\mathbf{p}_i$ of the vertex.

Once it have moved, it have to express them back in the model coordinates. This is multiplied by the inverse of the projection, view, and world matrix. At the end, it simply added offset vector to current position.

V. Results

The execution time is understood as the time it takes to process one full step of a clothes simulation. Expressed in milliseconds. This is the most important benchmark because it tells how much computing takes on the hardware, how large a percentage of the total engine work is and, if the simulator is fluid.

The effect for execution time has number of processed data, like density of the cloth mesh, and the selected implementation. These relationships are presented in tables and graphs, separately for each method and implementation. It was assumed that:

1) C - number of all vertices.
2) MS-GPU-A - Spring mass model, GPU implementation, Android platform.
3) PB-GPU-A - Item based model, GPU implementation, Android platform.
4) MS-GPU-W - Spring mass model, GPU implementation, Windows platform.
5) PB-GPU-W - Item based model, GPU implementation, Windows platform.
6) MS-CPU-A - Weight model on the spring, CPU implementation, Android platform.
7) PB-CPU-A - Position based model, CPU implementation, Android platform.
8) MS-CPUx4-A - Spring mass model, CPU implementation (4 working threads), Android platform.
9) PB-CPUx4-A - Position based model, CPU implementation (4 working threads), Android platform.

The graph shows a great performance advantage of methods implemented on the GPU. In the case of Android, the calculation time is almost constant regardless of the number of vertices of the cloth. Minor fluctuations are mainly due to measurement error (in the order of several ms). A slight increase in processing time at the final test phase may not result from the same computational overhead as with the increasing temperature of the device and the consequent gradual decrease in performance by the operating system. The inability to obtain a calculation time of less than 12-15ms is probably due to the fact that vertical synchronization is enforced by the implementation of transformational feedback in the Adreno graphics card driver. As it might expect, the GPU version on the PC platform is much more efficient. In this case, the difference is almost 300 times. Interestingly, the vertical sync problem does not occur here, although the processing time also remains constant.

The implementation of the CPU is a separate issue. It can be seen that the processing time increases linearly with the number of vertices and very quickly reaches values for nice image. Only for the low density of the grid has the advantage over the GPU, due to the problem mentioned above. It can also be seen that a decrease in performance for implementation with 4 threads of work is about twice less than in the case of a sequential approach.

For GPUs, no significant difference in performance was made between simulation methods, although on a CPU, the position model achieved for large numbers of vertices was slightly better than its rivals. The second most important problem of the simulation is its instability, understood as

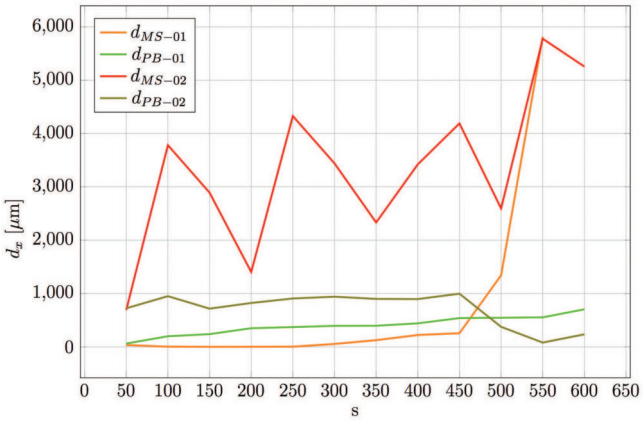

Fig. 7. Diagram of vibration dependence from stiffness coefficient.

the tendency for the cloth to fall into uncontrolled vibration, which in turn can lead to an "explosion". Even if this does not happen, continuous movements of the system result in unrealistic visual effects. This phenomenon is therefore very undesirable and often forces you to restart the simulator. One of the peaks in the middle of the cloth was selected for testing, and its vibration at rest was examined, ie the mean difference between the present and the previous position at each simulation step. Measurements were made for different stiffness coefficients, and then presented this relationship in the form of tables and graphs. Two methods were used for each method, including different masses, gravitational forces, attenuation coefficients, and mesh density. The state of rest is defined as the state in which the cloth has fallen freely from horizontal to vertical, suspended at two points, and ceased to move. It is worth recalling that for a position-based model the stiffness parameter (s) was scaled accordingly to fit within the required range [0, 1], and carried the same effect as its equivalent in the spring mass model. The test platform is a mobile version of the application, with an implementation on the GPU.

The main difference between spring and position-based mass models is that in the first case, for the first attempt, the lowest oscillation was recorded from the beginning, but it is growing rapidly with the increase of the stiffness parameter, at its highest value, leading to the "explosion" of the simulation. As for the second approach, large oscillations can be observed practically regardless of the flexibility of the cloth, suggesting that mesh compaction also has a significant impact on vibration. They were present for practically the entire time of the simulation. Still moving small distortions are very detrimental to visual reception and in any practical application would be unacceptable. Tests have shown that position-based models are exceptionally stable - oscillations are sometimes slightly larger than rivals, but in both trials they remained steady, regardless of the increase in the stiffness parameter or the number of vertices. The second test showed, however, that for a little elasticity and a dense mesh, the cloth begins to fall into uncontrolled collisions with itself. It is so rigid that, with the proper alignment of the masses, lead to the "hovering on itself" and the immobilization in the air, in fact ignoring the force of gravity. Strong waving occurred mainly in the red rectangle, and the middle area of the sample was left at rest. The last criterion is simply the degree to which the behavior and appearance of the simulated cloth reflects reality. This identifier is completely subjective, but one can clearly see the direct proportional relationship between quality and mesh density. A small number of vertices physically does not allow for the generation of realistic wrinkles or folds, so characteristic elements of cloth animation. For each simulation model, screenshots showing the "visual effect" dependence will be presented on the various parameters and in particular on the grid density. The test platform is the mobile version of the application. Similarities, however, end when they compare the parameters used to achieve similar effects - they are completely different. Undoubtedly, a positioning model generates a stiffer cloth than its rival. Sometimes this results in the above errors. The velocity of the cloth itself itself is also important - it should fall off and react to interactions with moving objects as quickly as in reality. In spite of their anomalies and the difficulty of obtaining a suitably flexible model, the denser spring mass method gives better visual results. On the other hand, the position-based approach is much easier to adjust flexibility and greater stability, but there may be problems with setting the appropriate animation speed. Fixed the δt parameter sent to the simulator. In both methods, it is easier to select the parameters for the desired behavior, with fewer nodes having a mesh.

In the case of a small number of edges, inaccurate collision detection between cloth and cuboid can be observed. This is not a rule, as the problem also occurs for denser nets. Here, however, there is also a lack of friction force implementation, which causes the tops to slide over the straight walls of the object, stretching the cloth and creating larger holes in the breakthrough. For the surrounding sphere, due to its uneven shape, the problem of breakage is not present. Exceptions are fast-moving objects that can simply jump through the cloth, in one step of calculations, in front of her, and then in the next. A continuous collision detection method, more complex mathematically but eliminating such phenomena, should be used.

VI. Discussions

A test application was created, one of its main purposes being the visualisation of two selected simulation methods – mass-spring and position-based model. It was equally important to show cloth's collisions between other objects in scene and itself. The user is allowed to set various parameters that influence the simulation, such as the aforementioned method type, mesh density and dimensions or elasticity coefficient. He can also impact the movement of the cloth, swiping his finger along the device's touch screen, which is something unique to the mobile platform. To fully measure every important factor

of the simulation, its three implementations were created – one using GPU for computing and the other two using GPU, in sequential and multi-threaded approach. To have a comparison between mobile and PC platform, a PC version of the application was created, both similar and sharing as much code with each other as possible. Both methods bring similar results, with a very small victory of the positioning model in CPU implementations. It was not possible to accurately examine differences in GPUs as the volume of homogeneous buffers did not allow the cloth to produce so many vertexes that performance time increased beyond 20 ms. Given the similar level of complexity of the code itself, it should be assumed that it would also be small. During testing, it was noted that a significant portion of the computation time was occupied by a fragment of the algorithm responsible for solving the collision. This may be due to the fact of using conditional statements in code executed on the GPU. More objects in the scene would certainly be associated with a deeper optimization of the issue, for example by limiting the number of potential entities that may come into contact with the cloth at the CPU level.

Both simulation models are characterized by a certain parameter-dependent instability, but it is much higher in the case of the spring mass model. The fact is that the composition of the formula on which the force acting on the vertex is calculated is the component responsible for vibration damping, and the user can adjust its coefficient. This method is characterized by an increase in net oscillation with an increase in the coefficient of elasticity. They have the form of small but fast vibrations on the entire surface of the fabric. With the turn for large numbers of edges, it takes a lot of rigidity to maintain the right shape, which further increases the problem. Large oscillations seem to keep them constant, but with any sudden change of position of vertices, such as in collisions, they can lead to a rapid 'burst' of simulation, which in practice is unacceptable. In the case of the position-based model, also the relationship between the increase in mesh density and its rigidity was observed, and loss of stability. The vibrations here are much slower and have a delicate, uncontrolled ripple, which is much less noticeable to the user. A big plus is the absence of an "explosion" effect, regardless of the parameters set. This effect was achieved through a kind of implementation trick - the position of the vertex transmitted to the calculator function of the limiter is updated only in the context of adjacent neighbors. The disadvantage of the position-based model in the present implementation is the tendency to fabric block itself on high elasticity.

Both methods of cloth simulation generate the desired visual effect, ie realistic folds and wrinkles of the fabric and its characteristic positioning on the object. Their quality is minimal for the position-based model. There are no minor vibrations there and more responsive to changes in stiffness coefficient. It should be noted that for example, for games in many cases there is no need for detailed mapping of fabric details, these can be obtained using normal maps. The two discussed methods have a faithful reproduction of this aspect even for a small number of edges. With the turn, when considering dense grids, there is a problem with the speed of animation. A high number of vertices requires a sufficiently high stiffness factor, this slows down fabric shifting, especially in a position-based model. The solution could be a more accurate matching of coefficients or an increase in the δt parameter, unchanged in simulation. Improvements to the situation can also be achieved by setting other stiffness parameters for each of the groups of springs or stops (ie parallel to the edge of the fabric, lying diagonally and such as the first, but located one position further). The collision detection method has proved to be a major disadvantage in the visual effects issue. It does not satisfactorily resolve internal collisions, and external collision errors are often encountered, eg when the fabric falls on the cuboid. To fix the problem, a different technique would have to be implemented. However, it would definitely entail the loss in performance and the most demanding computational component of the simulation.

The tests clearly indicate the winner of this performance comparison. GPUs are many times faster than CPUs when calculating issues that can be processed in parallel, and that is exactly what the problem is. Spreading the cloth overheads to individual GPUs is an intuitive and efficient solution despite the redundancy. Regardless of the amount of data, the recorded speed of performance turns out to be the same, which can not be said for CPU implementations, where it decreases linearly. Split into working threads increases it twice, which a little improves the situation, but in the case of detailed fabrics and so the performance is too low. With the turn on the GPU, there was a limitation by the transformational feedback of the rendered frames in one second to the value that matched the refresh of the screen. This is a defect that does not allow full evaluation of the performance and in some cases blocks the full speed of the application. The problem might be solving a change of test equipment to another, or using another GPGPU computation API.

All this does not change the fact that CPU implementation also has its uses and advantages. It is necessary to use it if device does not support OpenGL ES 3.0 or any specialized API such as OpenCL. OpenGL ES is a flavor of the OpenGL specification intended for embedded devices. It may be that it would have a performance advantage when the test platform had a very low-level GPU. It can also be used with certainty when the data set is only a very small number of vertexes, or if you have decided to do animation only in 2D space. It should also be noted that fabric simulation is much easier to implement on the CPU, as it does not require a deeper knowledge of the graphical API or GPGPU, and the creation of fairly complex buffering, homogeneous variables, programs, and transformational feedback.

The performance of mobile devices in this issue will be many times lower than that of PCs. Creation of two versions of the application, one on the Android platform, the other on the Windows platform, confirmed this assumption. The speed difference is about 300 times, the problem with the number of rendered frames per second disappears in the PC add-on. As far as how a GPU smartphone can seamlessly animate a very

dense mesh fabric that is sufficient to reproduce most details, this platform is harnessing a number of significant issues.

The first is the repeated lack of textured buffers on the part of the device, which limits the maximum possible quality. Cloth simulation heavily utilizes hardware capabilities, leading to overheating of the device. This leads to performance degradation by the operating system, which, in turn, results in significantly longer processing times and has had an impact on test results.

It is proved that the cloth simulation can be implemented on mobile devices and the mid-range GPU can perform very well, producing smooth animation of fabric's dense mesh, but not without a few important limitations. These include less useful API functions and shorter work time on battery as a result of intensive computations and tendency to overheating.

VII. Conclusions and Future work

This paper presented a technique for efficient fabric simulation in the real-time on a mobile device. The experience has shown that mobile devices can be used for real-time simulation of cloth animation with fast vertex optimization methods in mobile GPU units.

Tests have shown that while the GPUs of mobile devices are slightly slower than PC's ones, the relationship between processing speed on CPU and GPU remains similar. The GPU in both cases is significantly faster than the CPU built into the same machine.

Although technical aspects of User Interface have been created, they still require UX testing and further development.

References

[1] Wojciechowski, A. Camera navigation support in a virtual environment. Bulletin of the Polish Academy of Sciences-Technical Sciences 61, 871-884 (2013).

[2] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. SIGGRAPH Comput. Graph. 21, 4 (August 1987), 205-214

[3] David Baraff and Andrew Witkin. 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.

[4] B. Eberhardt and O. Etzmuß and M.Hauth. 2000 Implicit-Explicit Schemes for Fast Animation with Particle Systems. Computer Animation and Simulation 2000: Proceedings of the Eurographics Workshop in Interlaken, Switzerland, August 21-22, 2000, Springer Vienna, 137-151

[5] Hu X., Wei L., Li D. (2007) A Modified Numerical Integration Method for Deformable Object Animation. In: Park JW., Kim T.G., Kim YB. (eds) AsiaSim 2007. AsiaSim 2007. Communications in Computer and Information Science, vol 5. Springer, Berlin, Heidelberg

[6] Wei-Wen Feng, Yizhou Yu, and Byung-Uck Kim. 2010. A deformation transformer for real-time cloth animation. In ACM SIGGRAPH 2010 papers (SIGGRAPH '10), Hugues Hoppe (Ed.). ACM, New York, NY, USA, Article 108, 9 pages

[7] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. 2014. Subspace clothing simulation using adaptive bases. ACM Trans. Graph. 33, 4, Article 105 (July 2014), 9 pages.

[8] Russell Gillette, Craig Peters, Nicholas Vining, Essex Edwards, and Alla Sheffer. 2015. Real-time dynamic wrinkling of coarse animated cloth. In Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15). ACM, New York, NY, USA, 17-26.

[9] Wojciechowski, A., Gałaj, T. GPU Assisted Self-Collisions of Cloths. Journal of Applied Computer Science 24, 39-54 (2016).

[10] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, et al.. SOFA: A Multi-Model Framework for Interactive Physical Simulation. Yohan Payan. Soft Tissue Biomechanical Modeling for Computer Assisted Surgery, 11, Springer, pp 283-321, 2012, Studies in Mechanobiology, Tissue Engineering and Biomaterials, 978-3-642-29013-8. <10.1007/8415_2012_125>

[11] Lander, J. 1999. Devil in the blue-faceted dress: Real-time cloth animation. Game Developer Magazine (May)

[12] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. J. Vis. Comun. Image Represent. 18, 2 (April 2007), 109-118.

[13] Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F. O'Brien. 2010. Example-based wrinkle synthesis for clothing animation. In ACM SIGGRAPH 2010 papers (SIGGRAPH '10), Hugues Hoppe (Ed.). ACM, New York, NY, USA, Article 107, 8 pages

[14] H. Li, Y. Wan and G. Ma, A CPU-GPU hybrid computing framework for real-time clothing animation, 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, Beijing, 2011, pp. 391-396.