COMPARISON OF SEVERAL PARADIGMS FOR ACCELERATING PHYSICAL SIMULATION ON A PC

Tzvetomir Ivanov Vassilev

University of Ruse
e-mails: tvassilev@uni-ruse.bg
Bulgaria

Abstract: This paper proposes three paradigms for accelerating physical cloth simulation with cloth-object and cloth-cloth collision detection for dressing virtual bodies. The first one uses parallel threads on the CPU and an image-space based collision detection accelerated on the GPU. In the second the simulation runs entirely on the GPU with compute shaders and the third uses NVidia CUDA for the cloth simulation and NVidia OptiX ray-tracing engine for collision detection. Results of the experiments, comparing the performance and advantages of the three techniques, are presented at the end of the paper.

Key words: Cloth Modelling, Collision Detection, Physical simulation, Ray-tracing.

1. INTRODUCTION

Physical simulation has been used to produce realistic animation in computer graphics for several decades now. Very often realistic garments have to be visualized on human bodies in different types of applications: computer games, garment design and electronic commerce. Garment simulation on a human body can be split into two main tasks: first implementing a suitable cloth model and second developing a collision detection (CD) and response technique, which is usually closely related to the cloth model. The essence of CD in cloth simulation on a human body is to find if a cloth vertex penetrates the body (cloth-body CD) or if there are collisions between cloth pieces (cloth-cloth CD).

This paper compares the performance of several paradigms for accelerating cloth simulation on a normal desktop or laptop computer by parallelizing the algorithms for cloth modelling and collision detection. With some modification these paradigms can be used for any simulation of elastic bodies with collisions in a scene. In all implementations the model is based on a mass-spring system with velocity modification to constrain elasticity.

In paradigm 1 the physical simulation is implemented on the CPU using separate threads to run on multiple cores. The Graphics Processing Unit (GPU) is utilized to

accelerate the collision detection using interference tests in image space. In paradigm 2 both the cloth simulation and CD are entirely implemented on the GPU using OpenGL compute shaders. In paradigm 3 the cloth simulation runs on the GPU via NVidia CUDA API and the CD is based on ray-tracing again running on the GPU via NVidia OptiX engine.

The rest of the paper is organized as follows. The next section reviews previous work on cloth simulation and CD on GPU. Section 3 describes the utilized cloth model. Section 4, 5 and 6 describe in detail the three paradigms. Section 6 gives results of the experiment and the last section concludes the paper.

2. RELATED WORK

Cloth simulation

Physical simulation of elastic objects and surfaces has been tackled by researchers since the end of 1980s. Terzopoulos et al. [1] proposed elastic deformable surfaces and utilised the finite element method and energy minimisation techniques usually used in mechanical engineering. Later other teams worked on cloth simulation using energy [2] or interacting particles [3].

A more detailed survey on physically based cloth modelling techniques can be found in several papers [4, 5].

Later some implementations appeared, based on a mass-spring system with explicit integration [6, 7], as they are simple to implement and with low computational cost. One of their main drawbacks is the super-elasticity, i.e. the computer model is much more elastic than real cloth. To overcome this Provot [6] proposed to modify cloth vertices positions and Vassilev et al. [7] velocities after each simulation step in order to constrain elasticity. Another disadvantage is due to the explicit integration, which requires small enough time step to ensure stable simulation. In order to tackle this Baraff and Witkin [8] proposed physically based model with resistance to stretching, shearing and bending with implicit integration, which allowed larger time steps and thus led to a faster stable simulation of stiff materials. The main shortcoming of implicit integration is the large systems of equations that have to be solved at each integration step and the numerical damping introduced by some popular methods like implicit Euler [9].

In the recent years some new simulation methods for deformable models based on geometry emerged. In general they are not as accurate as traditional physics based methods, but produce visually pleasing animations. Müller et al. [10] proposed meshless deformation based on shape matching. The method is fairly fast and unconditionally stable. The body is divided in overlapping regions and at each time step for each region shape matching is performed by computing a transformation (rotation and translation) matrix to match best the shape of the original region. In this way stable simulation of elastic materials is achieved. Increasing the region size allows to simulate very stiff materials.

Bender et al. [9] used a shape matching technique on two types of regions to model cloth. The 2D triangular areas model resistance to stretching and shearing, while the edges increase the stretching stiffness of the simulated material. However, their model cannot naturally represent resistance to bending with shape matching. Therefore they had to introduce a quadratic bending term, however it does not guarantee unconditional stability in the time integration anymore. In addition, in order to increase the stretching and shearing stiffness of the cloth model, they presented a multi-resolution shape matching which further increases the computational complexity and thus reduces the overall performance.

In the last years position-based dynamics [11, 12, 13] has become popular in the computer graphics applications. Unlike physical simulation these methods compute the positions at each simulation step directly without storing forces and velocities, using external forces and a nonlinear system of constraints. The position-based approaches are stable and controllable and are well-suited for interactive environments. However, they are generally not as accurate as physics-based methods but provide visually pleasing results. In addition, if the nonlinear system of constraints to be solved at each iteration step becomes too big, this will slow down the simulation. So, their main application areas are virtual reality, computer games and special effects in movies and advertisements.

Tang et al [14] implemented the mass-spring cloth model of Provot entirely on the GPU. They do not mention how they handle the super elasticity, but their approach can use both explicit and implicit integration, however the time step of implicit integration is not larger in order to handle collisions of complex models.

The system, used in all the three paradigms, is based on physically based mass-spring simulation, which manipulates velocities to solve a system of constraints about super elasticity of the cloth and the collision response.

Collision detection

Usually the CD is very computationally complex, and as a result its impact on the overall simulation speed in graphics applications is substantial. So, significant research has been carried out on the topic by the computer graphics community. This resulted in a wide variety of methods and solutions. Initially, most of them ran on the CPU only. Later approaches involved the GPU to produce collision depth maps, but without direct GPU programming. In recent approaches it is common for the GPU to take part in or entirely solve the CD problem. Due to GPU's stream processing oriented architecture it may not always be feasible to implement the entire CD algorithm to run on the GPU only. Thus, hybrid solutions that involve both the CPU and GPU in the CD process are quite common.

The CD techniques can be divided in two major groups: based on geometrical interference tests in object-space and based on depth buffer interference tests in image-space.

Most common object-space techniques use bounding volumes [15]. They partition space in a set of boxes or spheres, which allow fast intersection tests.

Bounding Volume Hierarchies (BVHs) place these volumes into tree structures to accelerate searching even further. Larsson and Akenine-Moller [16] built a BVH with dynamically resizing existing Axis Aligned Bounding Boxes (AABBs). Another commonly used structure is the distance or radial fields, which store the shortest distances to a surface in a Cartesian grid [17].

In 2003 Knott et al [18] presented an image-space based CD algorithm utilizing the GPU. The GPU is used implicitly via hardware frame buffer operations, which implement ray-casting in order to detect static interference between solid objects in the scene. Kim et al [19] developed hybrid parallel continuous CD method that takes advantage of hybrid multi-core architectures – using the general-purpose CPUs to perform BVH traversal and culling while GPUs are used to perform the actual tests for collisions which require solving cubic equations. Another hybrid approach is proposed by Georgii et al [20]. Unlike the work of Kim et al the GPU is used to locate couples of possibly colliding polygons via ray tracing and depth peeling techniques. After that the actual tests for exact interaction are performed on the CPU. Tang et al [14] implemented a BHV CD technique entirely on the GPU. Hermann et al [21] proposed ray-traced collision detection for deformable bodies. A ray is shot from each surface vertex in the direction of the inward normal. A collision is detected when the first intersection belongs to an inward surface triangle of another body. This CD algorithm does not always work well for cloth simulation.

3. CLOTH MODEL

The cloth model utilised in all the three paradigms is based on the mass-spring model proposed by Provot [6] and modified by Vassilev et al [7]. The elastic model of a cloth piece is a rectangular topology mesh of $m \times n$ mass points, linked to each other by massless springs of natural length greater than zero. There are three different types of springs: stretch, shear, and bend, which implement resistance to stretching, shearing and bending.

For each cloth vertex the positions, velocities, and accelerations of the i-th mass point at time t are denoted: $\mathbf{p}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{a}_i(t)$, $i=1,\ldots, m \times n$. The system is governed by Newton's law:

$$\mathbf{a}_i(t) = \mathbf{f}_i(t) / m_i \tag{1}$$

where m_i is the mass of point $\mathbf{p}_i$ and $\mathbf{f}_i$ is the resulting force applied at that point. The force $\mathbf{f}_i$ is the sum of the internal and external forces.

The internal forces are due to the tensions of the springs. Utilising the Hook's law the overall internal force applied at point $\mathbf{p}_i$ is the sum of the forces caused by all springs linking this point to its neighbours:

$$\mathbf{f}_{int\,i}(t) = -\sum_j k_{ij} \Delta \mathbf{l}_{ij} \tag{2}$$

where k_{ij} is a stiffness coefficient of the spring linking $\mathbf{p}_i$ and $\mathbf{p}_j$ and $\Delta \mathbf{l}_{ij}$ is the elongation of the same spring related to its natural length.

The external forces in this cloth simulation are three types: viscous damping ($-c\ \mathbf{v}_i(t)$), gravity and seaming forces applied to the edges to be stitched together.

$$\mathbf{f}_{ext\ i}(t) = -c\ \mathbf{v}_i(t) + \mathbf{f}_{gr} + \mathbf{f}_{seam\ i}(t) \quad (3)$$

The above equations allow to compute the force $\mathbf{f}_i(t)$ applied on the cloth vertex $\mathbf{p}_i$ at any time t. In the original method of Provot and Vassilev et al the fundamental equations of Newtonian dynamics are integrated over time by explicit Euler integration.

Another possibility is to use Verlet integration with half-step velocity like this

$$\begin{aligned}
&\mathbf{a}_i(t) = \frac{1}{m_i}\mathbf{f}_i(t)\\
&\mathbf{v}_i\left(t + \tfrac{1}{2}\Delta t\right) = \mathbf{v}_i(t) + \frac{1}{2}\mathbf{a}_i(t)\Delta t\\
&\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t) + \mathbf{v}_i\left(t + \tfrac{1}{2}\Delta t\right)\Delta t\\
&\mathbf{a}_i(t + \Delta t) = \frac{1}{m_i}\mathbf{f}_i(t + \Delta t)\\
&\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i\left(t + \tfrac{1}{2}\Delta t\right) + \frac{1}{2}\mathbf{a}_i(t + \Delta t)\Delta t
\end{aligned} \quad (4)$$

This makes it possible to take larger time steps and as a result the simulation should converge faster. The cloth simulation program is implemented as a NVIDIA CUDA kernel and runs entirely on the GPU.

4. PARADIGM 1

In paradigm 1 the computations are accelerated by parallelizing the algorithm using several threads on a multicore CPU. The OpenMP API is utilised to implement the parallelism. The GPU is used implicitly for an image-space based collision detection approach. Using this technique it is possible to find a collision only by comparing the depth value of the garment vertex with the according depth information of the body stored in the depth maps. The graphics hardware is also used to generate the information needed for collision response, that is the normal vectors of each body point. This can be done by encoding vector co-ordinates (x, y, z) as colour values (R, G, B). Depth and normal maps are created using two parallel projections: one of the front and one of the back of the body (or object). For rendering the maps two orthographic cameras are placed at the centre of the front and the back face of the body's bounding box (BB), then the images are read back to the main memory, where the collision test is performed on the CPU. To increase the accuracy of the depth values, the camera far clipping plane is set to the far face of the BB and the near clipping plane is set to near face of the BB. Both cameras point at the centre of the BB. The depth maps are used to detect collisions and the normal maps to get the normal vector at the collision point, which is needed for the collision response. In case of a static human body the maps are generated only once before the simulation. In case of a moving body the maps have to be generated at each

animation step, although if the body movements are known, they can be pre-computed. The main drawback of this approach is that it cannot handle collision detection if there are occlusions, e.g. an arm is in front or at the back of the torso.

Collision response

If a collision was found, a resultant velocity is computed as shown in Figure 1. Let **v** be the velocity of cloth vertex **p** and **n** is the normal vector at the collision point taken from the normal map. The velocity **v** is split in its two components: normal ($\mathbf{v}_n$) and tangent ($\mathbf{v}_t$). The cloth new velocity is computed as proposed in [7]

$$\mathbf{v}_{res} = C_{fric}\mathbf{v}_t - C_{refl}\mathbf{v}_n, \quad (1)$$

where C_{fric} is a friction coefficient, C_{refl} is a reflection coefficient, which depend on the material. In this way friction between the cloth and the object can be simulated.

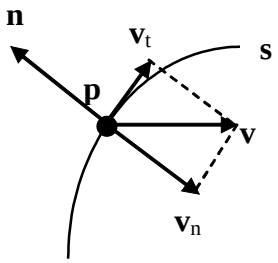

Figure 1. Computing new velocity as a collision response

One integration step of the proposed cloth simulation on the CPU can be described with the pseudo code in Algorithm 1.

Algorithm 1: Cloth simulation on multicore CPU

```
Parallelize For each spring
  Compute internal forces
  Add forces to 2 end mass points
Endfor
Parallelize For each mass point
  Add external forces
Endfor
Parallelize For each mass point
  Compute velocity
  Do collision detection
Endfor
Parallelize For each spring
  Correct velocities for over-elongated springs
  and for collision response
Endfor
Parallelize For each mass point
  Compute new positions
Endfor
```

5. PARADIGM 2

In this paradigm the whole cloth simulation is implemented on the graphics processor. There are several possibilities for GPU general purpose programming: OpenGL shading language (GLSL), NVidia CUDA (however this will limit the simulation only for NVidia processors), OpenCL, etc. Paradigm 2 uses GLSL, as the visualization is in OpenGL and this makes data exchange between the simulation and visualization modules easier and faster. In addition, with the compute shaders and shader-storage (SS) buffers in OpenGL 4.3 and above the programming is as flexible as in OpenCL and CUDA. The collisions are detected again using the image space approach with depth and normal maps, but they are not read back in the main memory, as checks are performed on the GPU. The algorithm of one iteration step is shown below.

Algorithm 2: Cloth simulation on the GPU with compute shaders

```
Parallelize For each mass point
  For each spring connected to this mass
     Compute and sum internal forces
  Endfor
  Add external forces
  Add joint forces
  Compute velocity
  Do collision detection and response
  Correct velocity for over-elongated springs
  Compute new position
Endfor
```

6. PARADIGM 3

As already mentioned the main drawback of the collision detection approach in paradigms 1 and 2 is that it cannot handle collisions if there are occluded objects, e.g. an arm is in front or at the back of the torso. In order to overcome this problem, this paradigm uses ray-tracing to detect collisions, which is implemented with NVidia OptiX engine, as described in [22].

In ray-tracing as many rays as the number of buffer pixels are cast from the camera to a rectangle of pixels situated behind the scene. Each ray is traced and if an intersection is found with an object, its material shader is called. The material shader is a CUDA program that computes the colour of the pixel, where the ray hit the object surface. In this way the output RGB buffer is filled with RGB colour values. As OptiX programs are executed on the GPU, the computations for each ray are performed in parallel, which increases the performance. On the latest NVidia RTX cards this is hardware accelerated,

Ray-tracing can be utilised for CD as follows. A collision buffer of quadruplets XYZW is created. The size of the buffer is equal to the number of cloth vertices. For

each cloth vertex a collision ray is constructed (R_1 in figure 2), which starts from the vertex and its direction is along the velocity of the cloth mass point. The length of the ray is set to some reasonable tolerance value, in our case 3 mm. The longer the ray, the slower the simulation, as intersection with more objects in the scene is sought. The ray is traced and if an intersection with another object (or cloth surface) is found, the collision material shader is called and it writes the following values in the output buffer:

- XYZ: the coordinates of the normal vector of the object or cloth surface at the intersection point. The normal vectors at the cloth vertices are computed by the simulation and for the rigid objects in the scene they are pre-computed.
- W: the distance from that vertex to the object, computed by the ray-tracing engine.

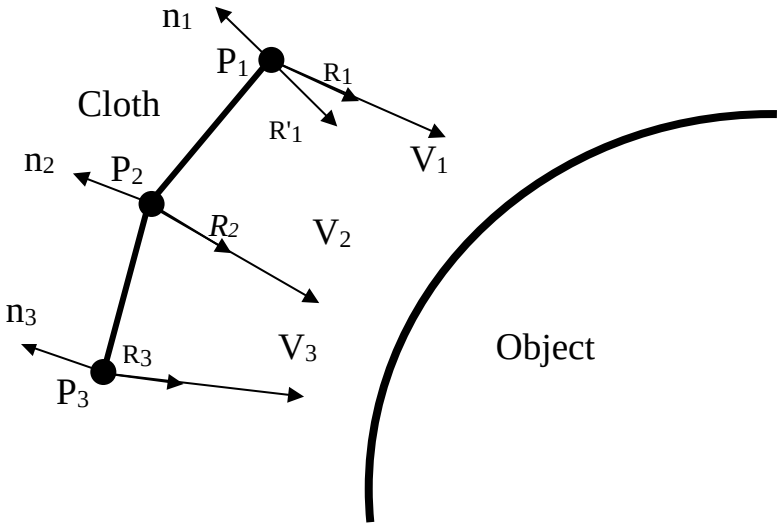

Figure 2. Cloth vertices and their velocity and normal vectors

If no collision is found for the first, then a second ray is cast (R'_1 in figure 2), which starts from the cloth vertex and its direction is opposite to the normal vector of the cloth surface at that vertex. The ray is traced again as explained above.

When simulating layers of cloth, collisions of vertices on lower layers with back faces on upper layers are possible to occur. In such cases the surface normal vector at the intersection point on the back face has to be inverted. This can be detected by calculating the dot product of the ray and the surface normal.

The cloth model is implemented in NVidia CUDA as it is easier to communicate with OptiX, which also uses CUDA. The algorithm is shown below.

Algorithm 3. Cloth simulation with ray-tracing for CD

```
forall vertices do
   compute force f_i
   compute velocity v_i and position p_i
endfor
Detect Collisions using ray-tracing
loop Iteration times
   forall vertices modify v_i to respond to collisions
   forall vertices modify v_i to correct over-elongation
endloop
forall vertices modify v_i to respond to collisions
forall vertices compute new position p_i
```

5. RESULTS

The algorithms were implemented in MS Visual C++ under Windows 10 and were tested on a laptop with a 2.2 GHz Intel core-i7 processor, 16 GB RAM and NVidia GeForce RTX 2070 graphics card with hardware ray-tracing acceleration using OptiX ray-tracing engine 6.5.

Figure 3 shows a pair of jeans dressed on a virtual body. The simulation converges in about 0.4 seconds with 500 iterations for paradigm 2 and 3.

Figure 3. A pair of jeans on a virtual body

Figure 4 compares the simulation speed of the three paradigms for the number of cloth vertices changing from 3000 to 15000, which is sufficient for cloth simulation of virtual try on. As expected Paradigm 1 with cloth simulation running

on the CPU as parallel threads is the slowest. It produces relatively reasonable results for a small number of vertices (3000-4000), but the time increases linearly with increasing the number of vertices. Due to the highly parallel architecture of the NVidia GPU the performance of the Paradigm 2 and 3 almost does not depend on number of cloth mass points. Paradigm 2 is faster, but as above mentioned its collision detection approach doesn't work when there are occluding objects in the scene. Paradigm 3 is slightly slower than 2, but the collision detection algorithm is more general and can be used for both cloth-body and cloth-cloth collision handling.

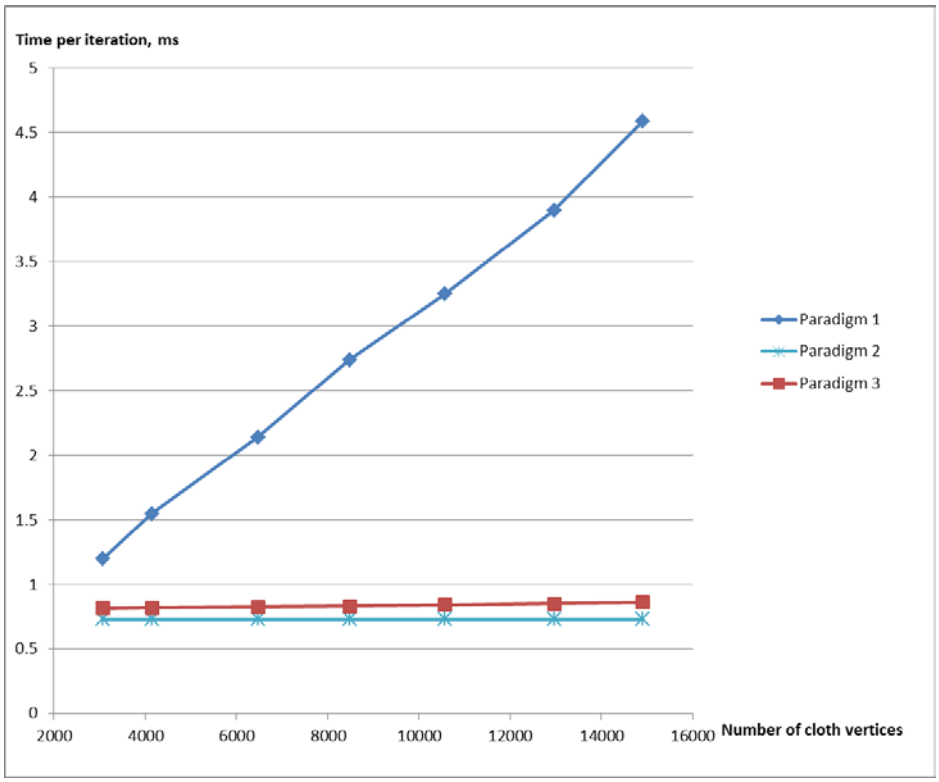

Figure 4. Number per iteration versus number of cloth vertices

4. CONCLUSION AND FUTURE WORK

This paper presented three techniques for accelerating a physical cloth simulation with collision detection and response on modern personal computer and compared their performance and advantages. The following conclusions can be drawn:

- Paradigm 2, for which the simulation runs entirely on the GPU using compute shaders and image-space approach for collision detection, is the fastest, however it cannot be used when there are occlusions in the scene, as this will produce wrong depth maps;

- Due to the hardware accelerated ray-tracing on the modern NVidia RTX GPU and their OptiX engine, ray-tracing can be used for real-time collision detection, so Paradigm 3 also produces very good results, although it is a bit slower than Paradigm 2.
- Paradigm 1 can also be used for real-time simulation, but for relatively small number of cloth mass points: about 3000-4000.

REFERENCES

[1] Terzopoulos D., J. Platt, A.Barr, K. Fleischer. Elastically deformable models. *ACM Proceedings of SIGGRAPH* 21 (4), 1987, pp. 205–214.

[2] Carignan M., Y. Yang, N. M. Thalmann, D. Thalmann Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics Proceedings, Annual Conference Series*, 1992, pp. 99–104.

[3] Eberhardt B., A. Weber, W. Strasser. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications* **5** (vol.16), 1996, pp. 52–59.

[4] Magnenat-Thalmann N., P. Volino. From early draping to haute couture models: 20 years of research. *The Visual Computer*, 21, 2005, pp. 506–519

[5] Nealen A., M. Müller, R. Keiser, E. Boxerman, M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, **4** (vol.25), 2006, pp. 809–836.

[6] Provot X. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. *Proceedings of Graphics Interface*, 1995, pp. 141–155.

[7] Vassilev T.I., B. Spanlang, Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum,* **3** (vol. 20), 2001, pp 260–267.

[8] Baraff D., A. Witkin. Large steps in cloth simulation. *Computer Graphics Proceedings, Annual Conference Series*, 1998, pp. 43–54.

[9] Bender J., D. Weber, R. Diziol. Fast and stable cloth simulation based on multi-resolution shape matching, *Computers & Graphics,* **8** (vol. 37), 2013, pp. 945-954.

[10] Müller M, B. Heidelberger, M. Teschner, M. Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics*, **3** (vol. 24), 2005, pp. 471–478.

[11] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, Position based dynamics, *J. Visual Commun. Image Representation*, **2** (vol. 18), Apr. 2007, pp. 109–118.

[12] Bender J., M. Müller, M. A. Otaduy, and M. Teschner. Position based methods for the simulation of solid objects in computer graphics, in *Proc. Eurographics*, 2013.

[13] Bender J., M. Müller, M. A. Otaduy, M. Teschner, and M.Macklin. A survey on position-based simulation methods in computer graphics, *Computer Graphics Forum*, **6** (vol. 33), 2013, pp. 228–251

[14] Tang M., R. Tong, R. Narain, C. Meng, D. Manocha. A GPU-based Streaming Algorithm for High-Resolution Cloth Simulation, *Pacific Graphics*, **7** (vol 32), 20113.

[15] Ericson C., *Real-Time Collision Detection*, Boca Raton, FL, USA: CRC Press, 2004.

[16] Larsson T., T. Akenine-Moller, A dynamic bounding volume hierarchy for generalized collision detection, *Computers & Graphics*, **3** (vol. 30), 2006, pp. 451–460.

[17] Friston S., A. Steed, Real-Time Collision Detection for Deformable Characters with Radial Fields, *IEEE Transactions on Visualization and Computer Graphics*, **8** (vol. 25), 2019, pp. 2611-2622.

[18] Knott, D., D. K. Pai (2003) CInDeR - Collision and Interference Detection in Real-time using Graphics Hardware. *Proceedings of Graphics Interface*, Halifax, Nova Scotia, June 2003, pp. 73-80.

[19] Kim, D., J.P. Heo, J. Huh, J. Kim, S. Yoon. HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs. *Computer Graphics Forum* (Pacific Graphics), 2009.

[20] Georgii J., J. Krüger, R. Westermann. Interactive GPU-based Collision Detection. *IADIS Computer Graphics and Visualization*, 2007.

[21] Hermann E., F. Faure, B. Raffin. Ray-traced collision detection for deformable bodies. *3rd International Conference on Computer Graphics Theory and Applications*, GRAPP Funchal, Madeira, Portugal, 2008, pp. 293–299.

[22] Vassilev, T.I. Collision Detection for Cloth Simulation Using Ray-Tracing on the GPU. *International Journal on Information Technology and Security*, **4**, 2012, pp. 3-12.

Information about the authors:

Prof. Tzvetomir I Vassilev, PhD, Department of Informatics and Information Technologies, University of Ruse, Bulgaria, Phone: +359 82 888475.

Manuscript received on 08 July 2020