information

MDPI

Article

A Soft Body Physics Simulator with Computational Offloading to the Cloud

Edvinas Danevičius [1], Rytis Maskeliūnas [1], Robertas Damaševičius [2,*], Dawid Połap [3] and Marcin Woźniak [3]

[1] Department of Multimedia Engineering, Kaunas University of Technology, 51368 Kaunas, Lithuania; edvinas.danevicius@ktu.edu (E.D.); rytis.maskeliunas@ktu.lt (R.M.)
[2] Department of Software Engineering, Kaunas University of Technology, 51368 Kaunas, Lithuania
[3] Institute of Mathematics, Silesian University of Technology, 44-100 Gliwice, Poland; dawid.polap@polsl.pl (D.P.); marcin.wozniak@polsl.pl (M.W.)
* Correspondence: robertas.damasevicius@ktu.lt; Tel.: +370-609-43772

Received: 26 November 2018; Accepted: 7 December 2018; Published: 11 December 2018

Abstract: We describe the gamification of a soft physics simulator. We developed a game, called Jelly Dude, that allows the player to change and modify the game engine by tinkering with various physics parameters, creating custom game levels and installing scripts. The game engine is capable of simulating soft-body physics and can display the simulation results visually in real-time. In order to ensure high quality graphics in real time, we have implemented intelligent computational offloading to the cloud using Jordan Neural Network (JNN) with a fuzzy logic scheme for short time prediction of network traffic between a client and a cloud server. The experimental results show that computation offloading allowed us to increase the speed of graphics rendering in terms of frames per second, and to improve the precision of soft body modeling in terms of the number of particles used to represent a soft body.

Keywords: soft physics; body simulation; gamification; cloud computing; computational offloading

1. Introduction

Soft body refers to an object that can deform its shape significantly in reaction to its context [1] or due to an external force, such as push, pull, tear, or bend forces [2]. Soft body simulation has been used in many fields such as medical applications [3], robotics [4], education [5], and games [6]. The topic is especially relevant for preschool education as young children often have incomplete and inaccurate understanding of "how things work" in the physical world, with only vague expectations about the behavior of real-world objects and materials. Therefore, various kinds of simulator games, which allow for mental interaction with world objects and simulation of their future state, are both popular and educationally valid among the young generation [7]. The issue is especially relevant for STEM (Science, Technology, Engineering, and Math) education, which aims to increase the attractiveness of STEM subjects and motivation of high school students to learn by employing attractive and entertaining tools such as games [8–10].

Implementing a realistic physics simulation is also important for many simulator games as it contributes to achieving higher Quality of Experience (QoE) and playability. There are different ways to evaluate and quantify the perceived QoE of the end-users, for example, by observing the players' styles of behavior and patterns of visiting game objects and levels similarly to people visiting exhibits in museums [11,12]. By providing a mathematical model of QoE such as presented in previous studies [13,14], one can solve this model for optimal values of its parameters, maximizing the desired QoE. However, here we adopt a more technical approach in cloud gaming that is linking

QoE to the key influencing factors of video graphics quality (frame rate) [15] and network quality (bandwidth, latency) [16].

While many game engines properly tackle various issues of physics simulation, some specific physics of materials are often omitted from game mechanics (e.g., game objects composed by liquids or amorphous jelly-like bodies), and as a result, are unsatisfactorily rendered by game engines. This limitation may hinder game developers when creating highly realistic video games and game mechanics. To improve immersion into a game, the physics laws simulated by the game engine should realistically simulate the laws of physics that the game needs. In order to ensure computing of the physics simulation as fast as possible, simplifying the fundamental mathematical model of the modeled physics behavior is often the solution. Nevertheless, the simulated bodies should have simulated behavior that closely matches expected behavior [17]. The development of realistic physics engines is one of the key technologies named as prerequisite requirements for mobile games [18,19], especially Virtual Reality (VR) and Augmented Reality (AR) games, which require high resolution and low latency of graphics rendering [20,21].

Soft bodies are usually represented as systems consisting of a large number of minute particles connected by flexible spring joints adhering to the laws of kinematics. One of the solutions is using pre-computed data-driven models for rendering physics-based deformable scenes to achieve fast real-time implementation [22]. An example is using the Navier-Stokes equations for modeling soft bodies as compressible fluid encaged in a mesh [1]. Wyvill et al. [23] combine particle systems with implicit functions to model soft bodies. Terzopoulos and Fleischer [24] modeled the behaviors of a deformable body by configurations of plastic, viscous, and elastic components. James and Pai [25] employ linear elasticity methods and use a boundary element method (BEM) solver to achieve desired behavior in real-time. Miller [26] animates the deformable bodies of worms and snakes using a lattice of mass elements linked with springs. Matyka and Ollila [27] adopt the Clausius-Clapeyron state equation for modeling pressure forces affecting a 3D mesh object and apply Newton's second law to achieve realistic simulation of the behavior of a soft object. Kenwright [28] uses strain models for simulating deformations of rigid objects. McAdams et al. [29] demonstrate surface deformation on hexahedral lattices for a geometric skeleton-based rigging, i.e., inserting a skeleton in a surface mesh and defining transformations on the mesh vertices. Martin et al. [30] unified the modeling of 3D objects by using the elaston (which are the specific rules of integration for volumetric objects) representation. Wu et al. [31] solve subspace deformations and motions of rigid and soft objects using linear solve and cubature optimization schemes to compute the elastic forces and their Jacobian matrices.

Soft physics is rarely used in computer gaming due to high computational requirements, especially on mobile platforms. As a result, this feature of the game engine is quite unique and can attract a lot of attention. Bullet [32], an open-source physics simulation library, has added support to soft bodies. CryEngine 3, a popular game engine, has a soft-body physics simulator implemented by Rigs of Rods (BeamNG). Digital Molecular Matter (DMM) is based on a finite element method (FEM), as described in [33], which is very efficient and achieves realistic results while providing support for many platforms. The Sulfur physics engine [17] uses the Verlet integrator with particles systems to model soft and rigid objects, and also implements collision management. Chrono [34], a multi-physics dynamics engine, also has support for modeling of fluid–solid interaction (FSI) as well as for linear and nonlinear finite element analysis (FEA). SpriteBuilder [35], a 2D game development environment for iOS, which among other features also provides support for soft-body physics.

This paper focuses on cloud-based gamification of an aesthetically pleasing soft body deformation model for real-time interactive environments. We describe the implementation of soft body physics in the game as well as describe the computation offloading management of computation-intensive functions for achieving higher performance.

2. Methods and Materials

2.1. Soft Body Physics

Soft bodies are usually modeled as a collection of elastically linked small particles that are characterized by mass, position, and velocity. The constraints linking two particles connected by an elastic joint can be described mathematically by Hooke's law with an inclusion of the damping force as follows:

$$F(t, x(t), \dot{x}(t)) = -k(|d| - l_0)\vec{d} + b(\dot{x}(t) - \dot{x}_p(t)) \quad (1)$$

where t is time; l_0 is the length of the link; k is the elasticity constant; b is the damping constant, d is the variation in the length of the link; $\dot{x}_p(t)$ is the velocity of the counterpart particle linked by the joint.

Such a model is easy to calculate and represent in time, however, it provides an approximation of the true physics only to a certain level of precision.

2.2. Game Implementation

The game physics is based on the Java's JBox2D library. This library has a lot of functions for physics simulations with hard bodies, but it does not directly support the simulation of soft bodies. The Jelly Dude soft body modeling is based on JBox2D constant-length springs. See an example of the spring-mass model of a soft body presented in Figure 1.

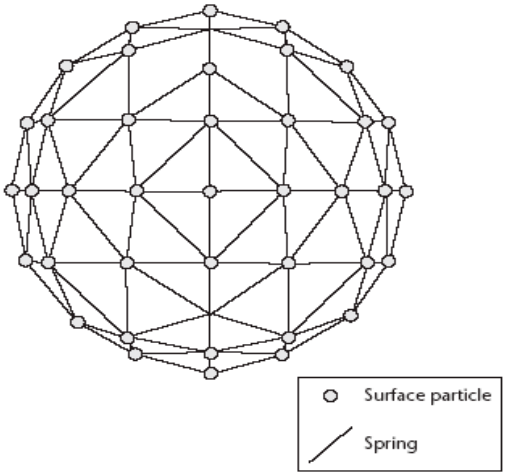
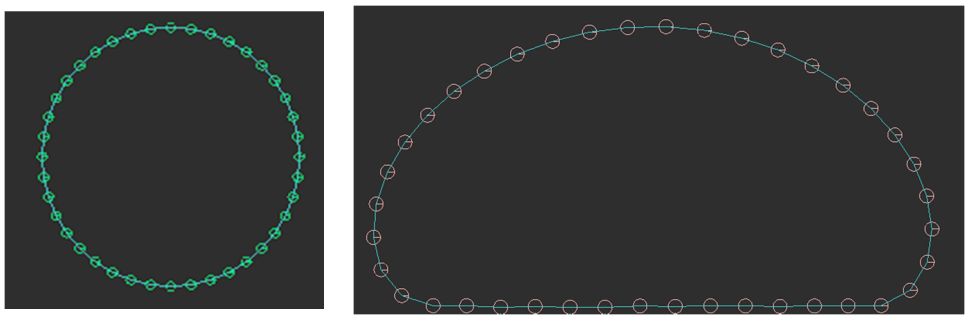


Figure 1. A model of spring-mass system with particles joined by links.

The game uses two different strategies to simulate soft bodies: a constant area strategy and a spring-loaded strategy. A constant area strategy simulates fairly realistic soft physics. It is realized by combining a large number of small particles with flexible springs, and then according to the total area and perimeter of the body, each particle of the body is updated to maintain a constant overall area. The operation of this strategy is presented in Algorithm 1 and is visualized in Figure 2.



(**a**) (**b**)

Figure 2. (**a**) Soft body in the beginning; (**b**) soft body after several steps of Algorithm 1.

Algorithm 1

1. Create small solid particles and link each particle to its neighbor particles using a soft spring.
2. Calculate the initial body area S_0.
3. For each subsequent step, perform the following:

    a. Calculate the physical perimeter P.
    b. Calculate the total area of the body.
    c. Find the sum of the extensions E by subtracting the current area S of the initial area S_0. This difference E is divided in half and is also divided by the perimeter P_i.
    d. Calculate the expansion vector Di for each pair of vertices, by multiplying the sum of the normal vectors of summands of N_i and $N_i + 1$ of the vertex pairs from the expansion factor E.
    e. The tip of the body $V_i + 1$ is the vector of the expansion of the vector D_i, which is added to its vector.

Using this strategy, after the first iterations of the algorithm the shape of the soft body changes, as is shown in Figure 2. After completing several steps of Algorithm 1, the shape of the composite soft body begins to change while maintaining its area (Figure 3).

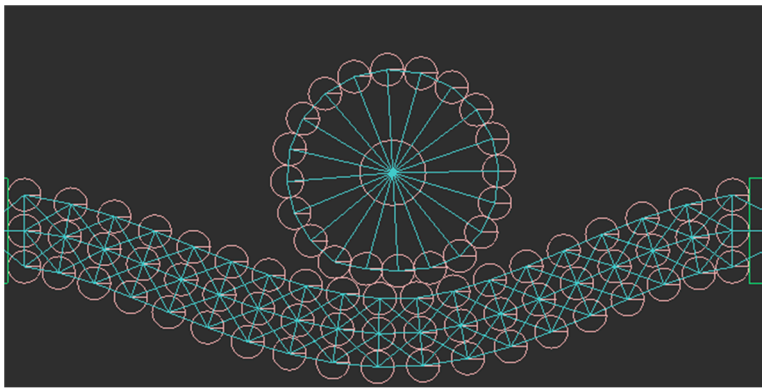


Figure 3. An example of a deformed polygon-shaped body and a circle-shaped body.

If this strategy is applied to the ultra-small bodies, it is evident that they are not in place (slightly moving), since in each iteration it is modified by the position of their constituent particles and the movement is visible due to the accumulation of computation error. Also, such calculations are quite expensive in terms of computing resources required, due to the large amount of calculations for each algorithm step.

The spring jointing strategy is based on a similar principle to that of a constant area strategy (combining the body with soft springs). However, this strategy does not add any additional calculations

to maintain the body shape. It is realized using only standard JBox2D functions. Its implementation is presented in Algorithm 2.

Algorithm 2

1. Create an initial body shape from small particles.
2. Fill the inside of the body evenly with small particles.
3. Connect each body component with its neighbors using flexible springs.
4. Perform physics simulation of the internal particles of the body.

This method makes it easy to create polygons and circles. Because the simulation of these types of bodies is relatively cheap in terms of computing resources, it is convenient to use these bodies as objects of the game level, as they can be created in large quantities. Also, this type of body is easy to combine with other soft and hard bodies, thus creating more complex shapes. Examples of sketches for objects that use this strategy are shown in Figure 3.

2.3. Game Architecture

The game is implemented in Java 8. The libGDX library was selected for game development. This library has particularly accelerated the tasks of graphing and creating user interfaces. Using this library, it was easy to create a simple object drawing system and handle player input. However, developing a fluid drawing system required lower-level drawing interfaces that this library does not have. In order to realize the fluid drawing system, a lower-level Lightweight Java Game Library LWJGL provided a software interface. The JBox2D Library was chosen to realize simulation of soft bodies. This library supports the simulation of various fluids but does not have the ability to simulate the physics of soft bodies.

The gdx-holo user interface resource library was used, which is based on the user interface style used by the Android platform. This library has textures for all user interface elements. Game levels were created without focusing on a particular style. We wanted to have a great variety of levels with different styles. For some game levels, our own set of textures was used. However, for the majority of levels, a free set of game textures was used. The game engine consists of classes that are essential for the program's operation. The activity diagram of the game engine is given in Figure 4.

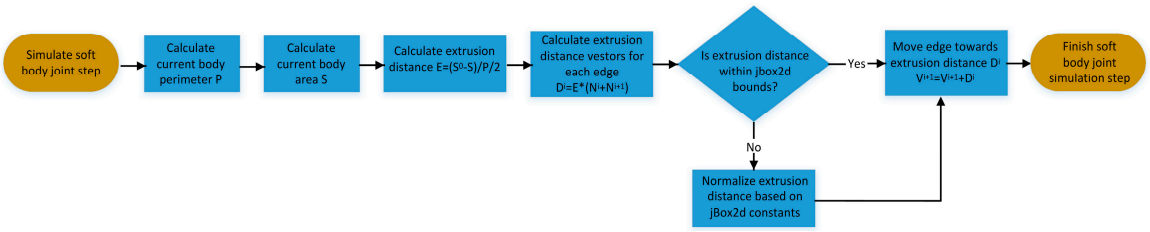
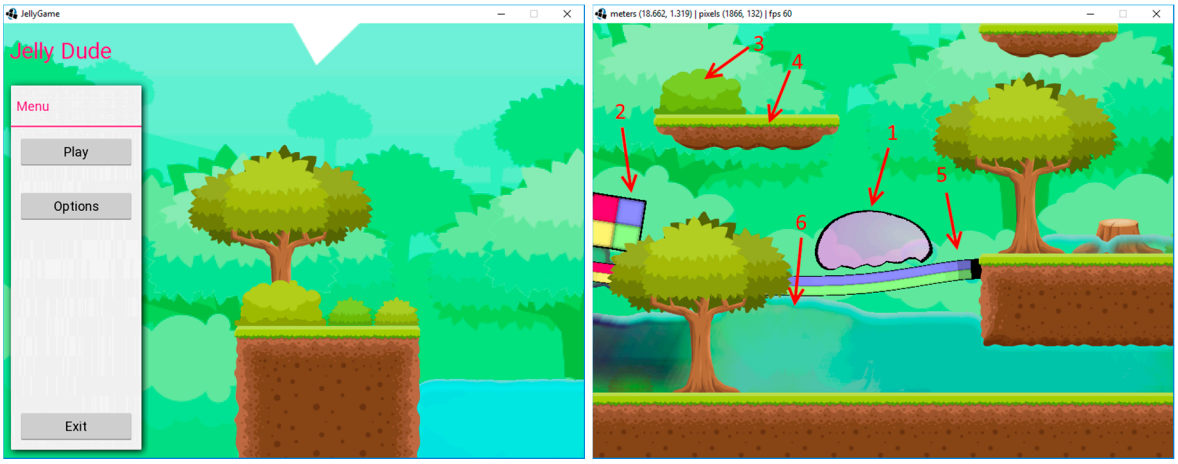


Figure 4. Activity diagram of the game.

2.4. Game Interface and Main Elements

When starting the game, the player will initially see the main menu (Figure 5a). From here, they can change the settings, start playing the game, or quit the game. Selecting the game level starts the game right away. An example of a game level is shown in Figure 5b. It features both solid immovable bodies as well as demonstrates the behavior of soft bodies influenced by interaction with the player.



(**a**) (**b**)

Figure 5. (**a**) The main menu of the game (left); (**b**) elements visible during the game (right). (1) Player-operated character. (2) Soft bodies to which you can stick with the main character. (3) Background objects that cannot be touched. (4) Solid immovable objects and obstacles. (5) Soft bridges to which one can stick with the main character. (6) Simulated water.

2.5. Computational Offloading Model and Offloading Decision Scheme

Cloud offloading is the employment of physically remote computing resources to perform the computation and energy-demanding functions of the system with data and results sent over the network [36]. The principle challenges of implementing efficient computation offloading concern the high volatility of internet networks, especially mobile networks. An important parameter influencing the effectiveness of cloud-offloading is network delay. If delay is large, the offloading strategy may not succeed. Therefore, constant monitoring of network connection to the cloud server and cloud response time is important. In fact, one needs to predict the cloud server response time in short-time in order to decide whether to offload computations or not. Such short-time forecasting (from ms to seconds) is useful for dynamic task scheduling and can be employed to boost the Quality of Service (QoS) and Quality of Experience (QoE) characteristics and playability of the game by adapting computation offloading mechanisms to mitigate network traffic volatility.

To improve the performance of the developed game we followed the computational model introduced in [37] and the offloading decision scheme suggested in [38]. We considered that a local machine has a computing task $J(w, S_m)$ that can be performed locally or remotely via computation offloading in a cloud, where w denotes the size of input data (e.g., program code or parameters that are needed for a task) used in the task, J and S_l is the computation speed of the local device. For computing locally, a local device performs its task J locally, and the execution time of the task J is $t_{loc} = w/S_l$. For computing on the cloud, a device offloads its computation task J to the cloud. Then the computation time is given as $t_{off} = w/S_s$, here S_s is the processing speed of the cloud server, and the sending time is defined as $t_{send} = d_i/B$, where d_i is the input/output data and B is the network bandwidth. Then the total time of offloading and executing the remote task is $d_i/B + w/S_s$. We can partition all tasks of a simulation program as a part that is running locally and another part that is offloaded to a cloud. We formulate the requirement for increasing computing performance as follows. If $d_i/B + w/S_s < w/S_l$, we can assume that offloading will increase the performance of computing.

2.6. Implementation of Intelligent Cloud Offloading

Here we adopted the opportunistic computation offloading approach suggested in [39]. To manage the cloud offloading efficiently, the game architecture was supplemented with mechanisms for the monitoring offloading process and computation task scheduling to implement adaptive cloud offloading optimizations in a straightforward manner. The Intelligent Offloading Management Module

(IOMM) (see Figure 6) manages the CPU-intensive computations by offloading the computation-costly computation for cloud computing. It consists of an Artificial Neural Network (ANN) that analyzes network latency and predicts future network latency. Based on the prediction results, the decision is taken whether to compute locally or offload computations to a cloud.

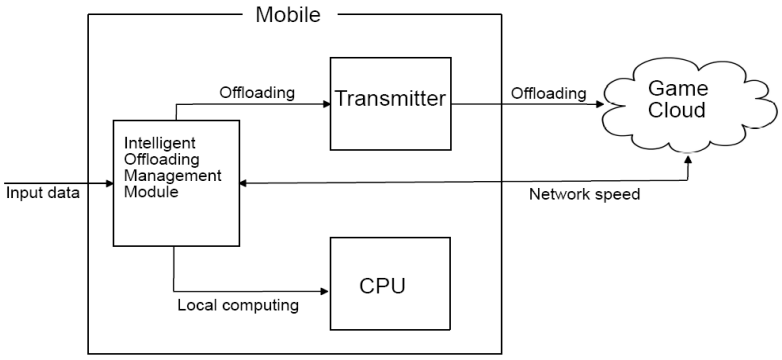


Figure 6. Intelligent offloading management.

Here we applied the neuro-fuzzy approach [40] for prediction (see Figure 7). The combination of ANNs and fuzzy reasoning by the neuro-fuzzy approach utilizes the advantages of both approaches simultaneously. The flexibility of fuzzy logic and the learnability of neural networks provides more power and adaptability in modeling the client-to-server connection traffic characteristics.

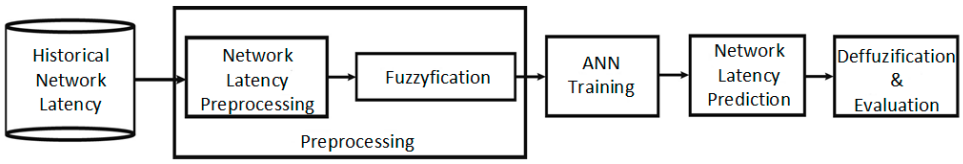


Figure 7. Summary of the neurofuzzy approach.

For short-term network latency prediction, we used the Jordan neural network (JNN) [41] and the Real Time Recurrent Network (RTRN), in which the outputs are delayed and sent back to the network input. Figure 8 illustrates the architecture of the JNN. The network inputs are the delayed signal of the process $u(k-1)$, the output signal of the model, $y(k)$, which are fed back to a delay unit. The JNN has K hidden neurons with transfer function $\varphi : R \rightarrow R$ and one linear output neuron, which can be described by the first-order system as follows:

$$y(k) = f(u(k-1), y(k-1)) \tag{2}$$

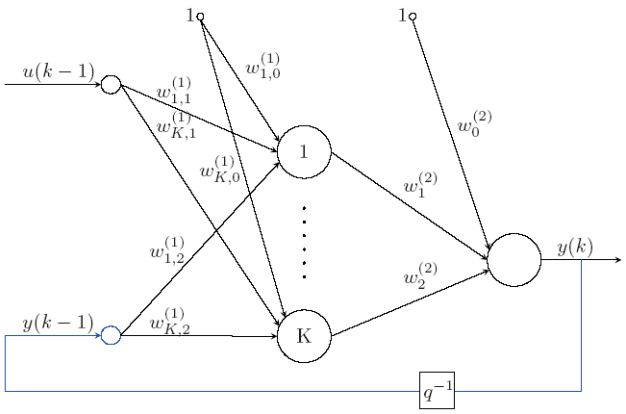


Figure 8. Architecture of Jordan neural network.

Let $w_{i,j}^{(1)}$ (for $i = 0, \ldots, K$, $j = 0, \ldots, 2$) and $w_i^{(2)}$ (for $i = 0, \ldots, K$) be the weights of the first layer and the second layer of the neural network, respectively. The network output can be written as.

$$y(k) = w_0^2 + \sum_{i=0}^{K} w_i^{(2)} \varphi(z_i(k)) \tag{3}$$

where $z_i(k) = w_{i,0}^{(1)} + w_{i,1}^{(1)} u(k-1) + w_{i,2}^{(1)} y(k-1)$ is the sum of inputs of the i-th hidden node.

The output of the neural network is used to make network delay predictions using the fuzzy logic [42]. A fuzzy subset A of X is a set defined by a membership function $f_A(x)$, which maps an element x in X to a real number in $[0, 1]$. The value of $f_A(x)$ is the membership value of x in X. As a membership function, we use the probability density function (PDF) of Weibull probability distribution $X \sim W(k, \eta)$, $x > 0$, $k > 0$, $\eta > 0$, which is often used for survival and failure analysis, and is characterized by the following PDF:

$$p(x) = \frac{k}{\eta} \left(\frac{x}{\eta} \right)^{k-1} e^{-\left(\frac{x}{\eta}\right)^k} \tag{4}$$

The result of defuzzification is the value of x that corresponds to the maximum value of $p(x)$.

3. Results

The simulations were implemented on a desktop computer with 4.3 GHz Intel i9-7900x CPU and NVIDIA GeForce 1080 GPU. Experiments were performed by setting the screen resolution to 1024×768 and measuring the frame rate per second (FPS). The game speed was measured in FPS (how many times the game logic was played per second) and the game's charge rate. If FPS is less than 60, the game loses accuracy and gradually becomes difficult to control. To get results, one of the game levels was fetched and the minimum, maximum, and average counts per second were calculated. Also, before loading the game, the game loading rate was counted. By loading the game level, the virtual camera moved randomly in all directions. This process took ~5 min. The test results for frames per second (FPS) are presented in Table 1.

Table 1. Summarized results of local computation vs. offloading to cloud.

Computation Scheme	Load Time (s)	FPS (min)	FPS (max)	FPS (mean)
Local computation	4	14	138	40
Offloading (cloud)	2.9	192	656	277

A low level testing game was prepared. By activating the test level, the amount of soft body particles used for simulation was gradually increased. During the test, the average number of FPS were registered depending upon the number of particles. The results are presented in Figure 9. It is evident that both faster and slower devices worked, with a relatively good scale of up to 3000 particles. When testing the game on a slow device, it started to cling to ~4000 particles and later the game became difficult to control. From these results we can conclude that the level of the game should be limited to 3000 particles used for soft body simulation.

The performance of the IOMM was evaluated using the real-world network traffic data collected during the game sessions on 24 September 2018, from 9.00 a.m. to 15.30 p.m. The client of the game was physically located in Kaunas (Lithuania), while the cloud server was setup in a physically remote location at Gliwice (Poland). The logarithmic transformation was applied to network round-trip time (RTT) data at t before the training of the neural network. For learning, the network RTT at $t + 1$ was binned and the accuracy calculated as the number of correctly predicted bins to which the network RTT value had been assigned. Using such a procedure, we achieved a true recognition rate of 84.1%. The example of true vs. predicted values is shown in Figure 10.

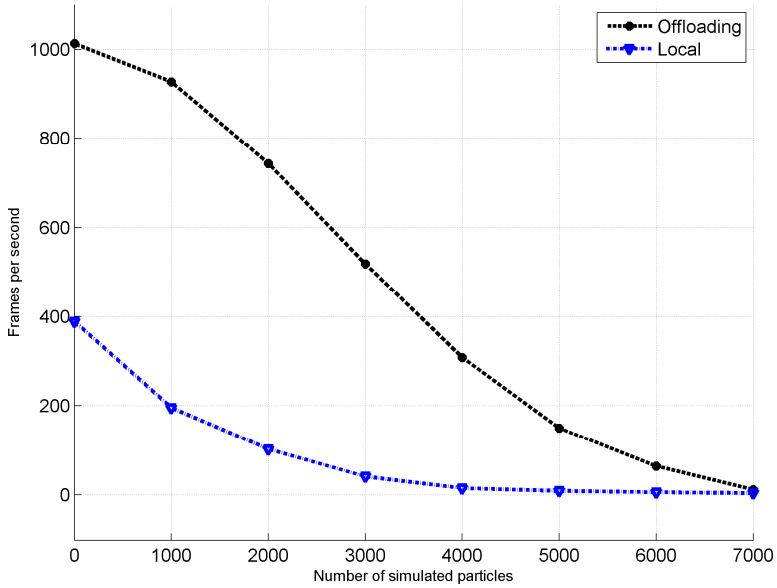


Figure 9. Number of frames per second rendered vs. the number of simulated particles in soft body.

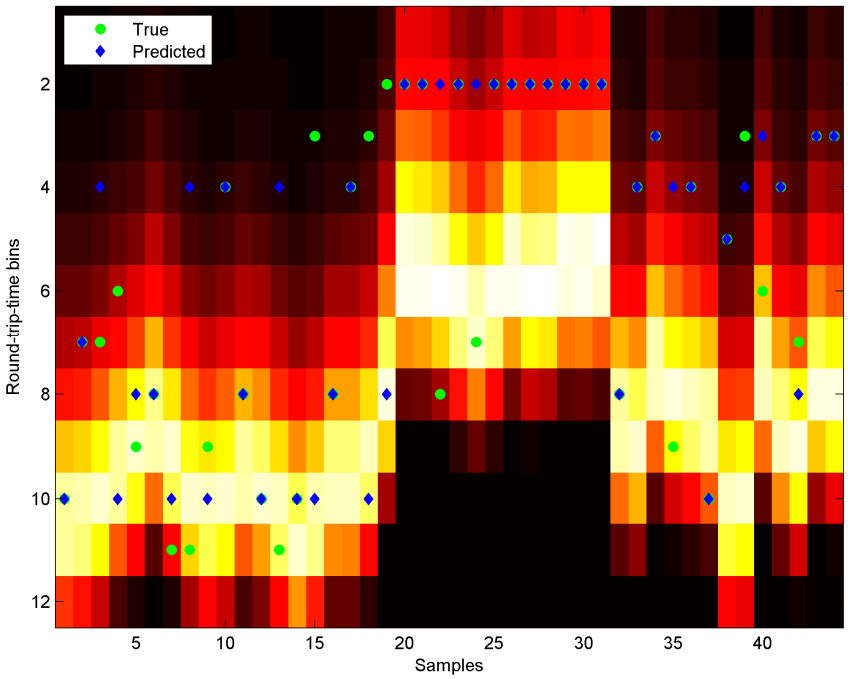


Figure 10. Prediction of network latency by neuro-fuzzy approach: the color of the background indicates the value returned by the fuzzy membership function assigned to each bin.

4. Discussion and Conclusions

We have developed a two-dimensional graphics game for real-time simulation of simple soft objects. The game demonstrates the simulation of soft physics laws and allows users to control a game character that consists of soft bodies. The game realistically demonstrates the behavior of objects, obeying the soft physics laws applied on simple soft objects, such as circle- and polygon-shaped bodies made of amorphous jelly-like materials, and could be used for educational purposes or as an example of physics law gamification. The limitations of our soft body modeling approach include the lack of collision modeling, which has not been implemented so far. To improve the performance characteristics of the game, we have demonstrated the computational offloading of a game to a cloud

server. The offloading decision is managed by an intelligent offloading management module consisting of a Jordan neural network and fuzzy logic, which analyzes the client-to-cloud connection round-trip time (RTT) and takes the computation offloading decisions in real time. We have achieved 84.1% accuracy (true recognition rate) of RTT prediction, which allowed us to improve game performance in terms of frames per second as well as to improve soft body modeling accuracy in terms of the number of body particles, which can be used for modeling.

To facilitate further testing and improvement of the Jelly Dude game, the source code of the game has been made available on GitHub (https://github.com/Edvinas01/jelly-dude), and a set of promotional videos have been launched on YouTube (https://www.youtube.com/playlist?list=PL5No3sH5hcxuFBLlb_7-5g6G21-uueiQf).

Author Contributions: Formal analysis, M.W.; investigation, E.D.; software, E.D.; validation, D.P.; writing—original draft, R.D.; writing—review & editing, R.M.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare that there is no conflict of interests regarding publication of this paper.

References

1. Nixon, D.; Lobb, R. A Fluid-Based Soft-Object Model. *IEEE Comput. Graph. Appl.* **2002**, *22*, 68–75. [CrossRef]
2. Song, M.; Grogo, P. A framework for dynamic deformation of uniform elastic two-layer 2D and 3D objects in OpenGL. In Proceedings of the 2008 C3S2E Conference (C3S2E '08), Montreal, QC, Canada, 12–13 May 2008; pp. 145–158. [CrossRef]
3. Nebel, J.-C. Soft Tissue Modelling from 3D Scanned Data. In *Deformable Avatars*; Springer: New York, NY, USA, 2001; pp. 85–97.
4. Kriegman, S.; Cappelle, C.; Corucci, F.; Bernatskiy, A.; Cheney, N.; Bongard, J.C. Simulating the evolution of soft and rigid-body robots. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17), Berlin, Germany, 15–19 July 2017.
5. Panthalookaran, V. Gamification of physics themes to nurture engineering professional and life skills. In Proceedings of the IEEE Global Engineering Education Conference (EDUCON), Tenerife, Spain, 17–20 April 2018; pp. 931–939. [CrossRef]
6. Ullman, T.D.; Spelke, E.; Battaglia, P.; Tenenbaum, J.B. Mind Games: Game Engines as an Architecture for Intuitive Physics. *Trends Cognitive Sci.* **2017**, *21*, 649–665. [CrossRef] [PubMed]
7. Pérez, F.Q. Applying of gamification tools in physics and chemistry of secondary education. *Opcion* **2016**, *32*, 327–348.
8. Tembo, T.T.; Lee, C. Using 2D simulation applications to motivate students to learn STEAM. In Proceedings of the 25th International Conference on Computers in Education: Technology and Innovation: Computer-Based Educational Systems for the 21st Century (ICCE 2017), Christchurch, New Zealand, 4–8 December 2017; pp. 403–409.
9. Oyesiku, D.; Adewumi, A.; Misra, S.; Ahuja, R.; Damasevicius, R.; Maskeliunas, R. An Educational Math Game for High School Students in Sub-Saharan Africa. In *Communications in Computer and Information Science*; Springer: Cham, Switzerland, 2018; pp. 228–238.
10. Raziunaite, P.; Miliunaite, A.; Maskeliunas, R.; Damasevicius, R.; Sidekerskiene, T.; Narkeviciene, B. Designing an educational music game for digital game based learning: A Lithuanian case study. In Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 21–25 May 2018; pp. 800–805. [CrossRef]
11. Chittaro, L.; Ieronutti, L. A visual tool for tracing users' behavior in Virtual Environments. In Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04), Gallipoli, Italy, 25–28 May 2004; pp. 40–47. [CrossRef]
12. Sookhanaphibarn, K.; Thawonmas, R. A Movement Data Analysis and Synthesis Tool for Museum Visitors' Behaviors. In Proceedings of the 10th Pacific Rim Conference on Multimedia (PCM 2009), Bangkok, Thailand, 15–18 December 2009; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5879, pp. 144–154. [CrossRef]

13. Tsiropoulou, E.E.; Thanou, A.; Papavassiliou, S. Modelling museum visitors' Quality of Experience. In Proceedings of the 2016 11th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), Thessaloniki, Greece, 20–21 October 2016; pp. 77–82. [CrossRef]
14. Tsiropoulou, E.E.; Thanou, A.; Papavassiliou, S. Quality of Experience-based museum touring: A human in the loop approach. *Soc. Netw. Anal. Min.* **2017**, *7*, 33. [CrossRef]
15. Slivar, I.; Suznjevic, M.; Skorin-Kapov, L. Game Categorization for Deriving QoE-Driven Video Encoding Configuration Strategies for Cloud Gaming. *ACM Trans. Multimedia Comput. Commun. Appl.* **2018**, *14*, 1–24. [CrossRef]
16. Jarschel, M.; Schlosser, D.; Scheuring, S.; Hoßfeld, T. Gaming in the clouds: QoE and the users' perspective. *Math. Comput. Model.* **2013**, *57*, 2883–2894. [CrossRef]
17. Maggiorini, D.; Ripamonti, L.A.; Sauro, F. Unifying Rigid and Soft Bodies Representation: The Sulfur Physics Engine. *Int. J. Comput. Games Technol.* **2014**, *2014*. [CrossRef]
18. Xie, J. The research on mobile game engine. In Proceedings of the International Conference on Image Analysis and Signal Processing, Wuhan, China, 21–23 October 2011; pp. 635–639. [CrossRef]
19. Ali, Z.; Usman, M. A framework for game engine selection for gamification and serious games. In Proceedings of the FTC 2016—Future Technologies Conference, San Francisco, CA, USA, 6–7 December 2016; pp. 1199–1207. [CrossRef]
20. Połap, D.; Kęsik, K.; Książek, K.; Woźniak, M. Obstacle Detection as a Safety Alert in Augmented Reality Models by the Use of Deep Learning Techniques. *Sensors* **2017**, *17*, 2803. [CrossRef] [PubMed]
21. Buzys, R.; Maskeliūnas, R.; Damaševičius, R.; Sidekerskienė, T.; Woźniak, M.; Wei, W. Cloudification of Virtual Reality Gliding Simulation Game. *Information* **2018**, *9*, 293. [CrossRef]
22. James, D.L.; Fatahalian, K. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph. (TOG)* **2003**, *22*, 879–887. [CrossRef]
23. Wyvill, G.; McPheeters, C.; Wyvill, B. Data Structures for Soft Objects. *Vis. Comput.* **1986**, *2*, 227–234. [CrossRef]
24. Terzopoulos, D.; Fleischer, K. Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *ACM Comput. Graph.* **1988**, *22*, 269–278. [CrossRef]
25. James, D.; Pai, D. ArtDefo—Accurate Real Time Deformable Objects. In Proceedings of the Computer Graphics (Proc. Siggraph 99), Los Angeles, CA, USA, 8–13 August 1999; ACM Press: New York, NY, USA, 1999; pp. 65–72.
26. Miller, G. The Motion Dynamics of Snakes and Worms. *ACM Comput. Graph.* **1988**, *22*, 169–177. [CrossRef]
27. Matyka, M.; Ollila, M. Pressure Model of Soft Body Simulation. In Proceedings of the Annual SIGRAD Conference, Umeå, Sweden, 20–21 November 2003; pp. 29–32.
28. Kenwright, B. Scalable Real-Time Vehicle Deformation for Interactive Environments. Available online: http://www.xbdev.net/misc_demos/demos/vehicle_deformation/paper.pdf (accessed on 9 December 2018).
29. McAdams, A.; Zhu, Y.; Selle, A.; Empey, M.; Tamstorf, R.; Teran, J.; Sifakis, E. Efficient elasticity for character skinning with contact and collisions. In Proceedings of the ACM SIGGRAPH '11, Vancouver, BC, Canada, 7–11 August 2011; pp. 37:1–37:12.
30. Martin, S.; Thomaszewski, B.; Grinspun, E.; Gross, M. Example-based elastic materials. *ACM Trans. Graph.* **2011**, *30*, 72:1–72:8. [CrossRef]
31. Wu, X.; Mukherjee, R.; Wang, H. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Trans. Graph.* **2015**, *34*, 241. [CrossRef]
32. Coumans, E. Bullet physics simulation. In Proceedings of the ACM SIGGRAPH 2015 Courses—SIGGRAPH '15, Los Angeles, CA, USA, 9–13 August 2015; ACM Press: New York, NY, USA, 2015.
33. Parker, E.G.; O'Brien, J.F. Real-time deformation and fracture in a game environment. In Proceedings of the SCA '09 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, New Orleans, LA, USA, 1–2 August 2009; pp. 165–175. [CrossRef]
34. Tasora, A.; Serban, R.; Mazhar, H.; Pazouki, A.; Melanz, D.; Fleischmann, J.; Negrut, D. Chrono: An Open Source Multi-physics Dynamics Engine. In *Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2016; pp. 19–49. [CrossRef]
35. Itterheim, S. Soft-Body Physics. In *Learn SpriteBuilder for iOS Game Development*; Apress: Berkeley, CA, USA, 2014.

36. Kwon, Y.-W.; Tilevich, E. Facilitating the Implementation of Adaptive Cloud Offloading to Improve the Energy Efficiency of Mobile Applications. In Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems, Florence, Italy, 16–17 May 2015; pp. 94–104. [CrossRef]
37. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
38. Gu, F.; Niu, J.; Qi, Z.; Atiquzzaman, M. Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions. *J. Netw. Comput. Appl.* **2018**, *119*, 83–96. [CrossRef]
39. Li, W.; You, X.; Jiang, Y.; Yang, J.; Hu, L. Opportunistic computing offloading in edge clouds. *J. Parallel Distrib. Comput.* **2019**, *123*, 69–76. [CrossRef]
40. Georgy, M.E.; Chang, L.-M.; Zhang, L. Prediction of Engineering Performance: A Neurofuzzy Approach. *J. Constr. Eng. Manag.* **2005**, *131*, 548–557. [CrossRef]
41. Wysocki, A.; Lawrynczuk, M. Jordan neural network for modelling and predictive control of dynamic systems. In Proceedings of the 2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, 24–27 August 2015. [CrossRef]
42. Zadeh, L.A. Fuzzy sets. *Inf. Control* **1965**, *8*, 338–353. [CrossRef]