CEIG - Spanish Computer Graphics Conference (2014)
Pere-Pau Vázquez and Adolfo Muñoz (Editors)

CLISKA - An application for physics education in secondary school

C. Marín-Lora[1] and I. García-Fernández[2]

[1]ETSE-UV, Valencia, Spain
[2]Departament d'Informàtica, ETSE-UV, Valencia, Spain

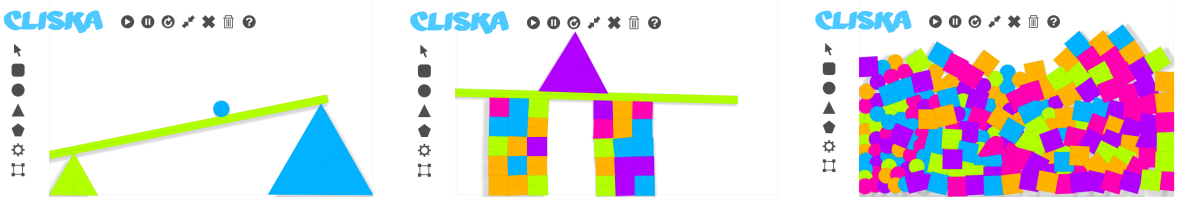

Figure 1: *Three simulation scenes of the application CLISKA, presented in this paper.*

Abstract

In this paper we present an application for teaching Physics courses in secondary education. It is a web application, based on HTML5 and on JavaScript. We rely on the graphical capabilities of this technology, including rendering on 2D canvas context and SVG rendering, and we develop a framework for simulating physics behavior. We review other applications with similar features, explaining the motivations of the current development. The system architecture and the software design are discussed and the simulation techniques are explained. We finish with a discussion on some possible examples of use for the application and on the future development plans.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications— K.3.8 [Computers and education]: Computer Uses in Education—Computer-assisted instruction

1. Introduction

Education of Physics courses in primary and secondary education comprises many challenges. Educators face lack of motivation and the difficulty of giving the students the intuition behind the mathematics that formalize the physics concepts. New paradigms, such as constructivism, potentiate learning by doing as a way to effective learning, in opposition to lecture-based education, in which the student is a mere spectator.

The incorporation of new technologies in the classroom has made possible to provide the students with an active learning environment. In the case of Physics discipline, by means of computer simulation the students can interact with virtual experiments and observe the reaction to different actions.

In this work we present CLISKA, a web-based application for the interactive practice of Physics in the classroom. The application simulates two dimensional rigid and deformable objects, with collision detection and reaction. It has been developed with HTML5 and JavaScript and it provides an adequate web-based framework for interactive simulation. It has a dedicated physics engine and some additional developments, such as a user interaction module for the HTML5 Canvas element.

The paper is structured as follows. In Section 2 we motivate our work and review other applications that can be used in the same context. Section 3 describes the general system architecture and Sections 4 to 7 describe in more detail de different modules that compose the application. Section 8 presents some possible usages in the classroom for teaching Physics. In Section 9 we discuss the capabilities and limi-

DOI: 10.2312/ceig.20141114

tations of the software. Finally, in Section 10 we give some concluding remarks and present the future plans for the application.

2. Previous Work

For decades, traditional education has considered the teacher as the main performer in the process of learning. According to Nogueira Gustavino, in Spain *the educational paradigm focuses on the teacher, that is, in what the teacher wants to instruct as basic knowledge, which prevails over what student should learn, and [...] in which it takes precedence the learning by force of all the contents [...] over its effective comprehension by the students* [Nog11].

This approach, however, is suffering from a deep methodological reformation, with the introduction of the constructivist paradigm. This paradigm shifts the main action in education from *teaching* to *learning*, and the goal of the teacher is to facilitate experiences to the students who build knowledge by means of reflection and action [Kol84, Nog11].

In this context, tools that help the teacher to provide the students with the possibility of experimenting are necessary. Nowadays, widespread of Information and Communication Technologies is a great ally for this goal. Several projects have been developed in the past to build different applications aimed to learning by experimenting, not only in physics, but also in other disciplines [RAJC09, Bod09, SPB12, LZM12]. In this section we review the most relevant applications that can be found for helping Physics education.

One of the core components of an application that involves physics simulations is a physics engine that computes the evolution of virtual objects. There are different physics engines available for JavaScript that can be used for this purpose. We also discuss them and their main features.

2.1. Applications for Teaching Physics Courses

One of the most popular educational applications for learning physics is Algodoo [Bod09, Alg13]. It features a set of systems that can be simulated within an open world environment. It has a community of users that share simulation scenarios. Algodoo can be run on Windows desktop computers, on MacOS desktop computers and on Apple iPad devices. The software cannot be run on other operating systems or through the web.

Phyzicle Sandbox an a 2D interactive physics application that runs on mobile devices [Ago12]. The software has versions that can run on Android OS, Apple iOS and BlackBerry OS. The graphic design is simple and visual, following what is sought in our work. Among the physical properties that it can handle, the application can simulate both rigid bodies and fluids, but it cannot simulate deformable bodies. During the simulation, it is possible to use the mobile device accelerometers. However the set of parameters that the user can adjust for the scene and the bodies is very limited. The software cannot be run on a desktop computer or through the web. Moreover, the development of the application seems to be discontinued, as it has had no updates since 2012.

Yenka is a suite of tools for teaching science in the classroom [Cro]. It has several modules for experimenting with physics, mathematics, chemistry or programming. The library of predefined scenarios is wide and well documented. However it has several drawbacks. The user interface is complex, it is not very intuitive, and edition of new scenes is also complicated. Yenka is only provided as a desktop application, and does not have mobile device version. Moreover, it is sold under a license which, depending on the number of students and the number of modules, which are sold separately, can cost a few thousand Euro. This contrasts with the rest of applications discussed here, which are free.

SketchyPhysics is a physics plug-in for the 3D modeller Google Sketchup [Ske09]. The physics engine is based on the Newton Game Dynamics library [JS11], an open source physics library. This library provides support for scene management, collision detection and dynamics behaviour with a deterministic solver. The plug-in is not specifically intended for setting up simple scenarios to teach physics, but as a module to introduce physics in the modeller. There is a set of examples uploaded by users that can be accessed from the project page. In order to run this plug-in it is necessary to use Sketchup, which runs on Mac and Windows desktop computers. The project seems to be discontinued since September 2010.

Apart from these standalone applications, we can find several web sites that present small plug-ins or web applications to exemplify or experiment with different physics concepts and principles [Ló14, Nat13]. However, their degree of interactivity varies among them, and none of them presents the possibility of building custom complex scenarios or of mixing different types of objects or physics principles.

2.2. Physics Libraries in JavaScript

As the capabilities and flexibility of web based applications increase, new libraries are developed to add different features to this technology. Among these libraries we can find several modules that provide the programmer with a framework for physics simulation.

Newton is a JavaScript library for simulating particle physics, based on de Newton's laws [Lof13]. It can simulate the application of forces and the definition of some constraints on the particles. Particles can be grouped to form what they call *bodies*, which are logical entities. However, it does not consider actual rigid body simulation.

The library Verlet.js [Sub13] features a simple physics engine that simulates particles, a reduced set of solid shapes

and cloth. Collision detection and some restrictions on particles are also available. Another library with a similar set of features is Matter.js [Bru14].

Apart from these JavaScript Libraries, there exist some other libraries that are built in different languages but have JavaScript ports. Box2D [Cat13] is currently one of the most extended 2D physics engines. It considers rigid body with contact and friction and includes a complete set of kinematic constraints and force actuators. Box2D is written in C++. Nape [Del12] is another 2D rigid body library, written in Haxe. It simulates contacts and constraints and it provides a way of defining new kinematic constraints without modifying source code.

The use of one of these physics engines was considered during the development of our project. However, support for deformable bodies is very limited, and it has to be done by tweaking rigid constraints. Moreover, none of them is able to simulate fluids or thermodynamics experiments. For this reason we have decided to develop our own physics module for the simulator.

3. Application Description

Our intent is to create a physics simulator that can run configurable scenes in an open world where a student can interact and play around with objects and different physical properties.

We take advantage of the HTML5 standard to build a platform independent application. It can be installed on any desktop computer with a compatible browser or, depending on the availability of Internet access, can be run online. Also, we chose to develop a web application instead of developing the tool for mobile devices, because in many cases it would require the users to make an investment to purchase the terminal, while a PC room is available at most Spanish schools.

For the first version of the tool, which is the version described here, we have developed rigid and deformable body dynamics, with collision detection. An important requirement considered is to be able to build different scenarios and save them to disk, so that the teacher can set up problems and examples adequate to her didactic interests.

We start describing the basic features of the application and the overall system architecture. Later, in the next sections, we review more deeply some of the modules that form the application.

3.1. The Application

When the application is loaded on the web browser, it will be displayed in its document area. The user will find different controls and an empty space, that contains the simulation world.

Using the controls, the user can create objects of different basic geometric shapes, place them in the scene and give them different physical properties. When the application starts, the simulation is paused, so no action is taken. At any time the user can run the simulation and during this time the user can also interact with the objects in the scene and remove or add new objects.

3.2. System Architecture

The system proposed consist of the following modules: a simulation engine, an graphical user interface (GUI) and a render engine. In addition, an export module lets the user save an scene in xml format. The different modules have been developed with HTML5 and JavaScript and they run on the side of the client during the execution of he application. Next, we present a brief description of the subsystems.

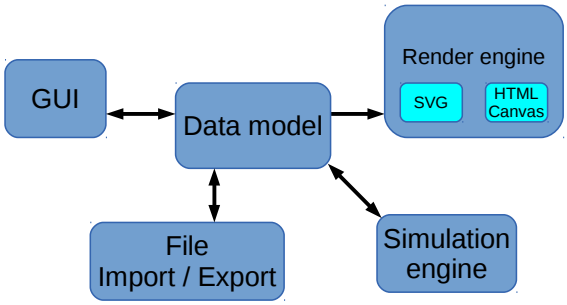

Figure 2: *Overall system architecture*

The user interface uses HTML5 and CSS3, as they are standard and most web browsers will be able to display it and to perform the desired functionality. For the same reason, the framework that creates and controls the scenes, as well as the interaction between the elements that compose them, has been developed using JavaScript.

For displaying the simulation we have developed two display engines. A display framework that uses Scalable Vector Graphics and a display framework that uses the HTML5 Canvas element.

Finally, an export module is under development to let the user save the scenes to a text file. This module makes use of an XML format to store the description of the scene for later usage.

4. Graphical User Interface

Our Graphical User Interface (GUI) has been defined with a modular design and can be modified according to the user preferences. The main element of the UI is the scene area, which covers the whole document space in the browser. This area contains both the simulation and the user controls.

In the basic setup, it shows a side toolbar with main system functionalities: new objects, erase objects or access to

scene settings, among other options. In the upper part, a *simulation control* bar lets the user start, stop and pause the simulation. Figure 3 shows an image of the initial setup of the application GUI. As we said previously, the GUI has been designed with modularity and configurability in mind. Every toolbar and panel can be dragged and dropped on any location of the scene area.

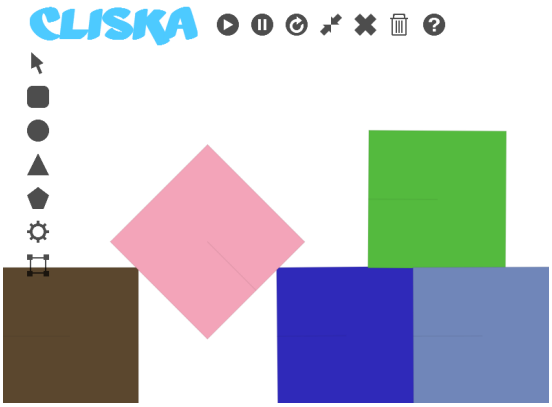

Figure 3: *An screen shot of the application GUI in its basic configuration. Users can redistribute the UI elements to their convenience.*

A scene is formed by one or more geometric figures that represent solid objects. These objects can be added and removed to the scene. Using the left mouse button, any object can be dragged and placed on any location of the scene. Right mouse click will show the object's properties panel, where the user can view, modify or reset the physical properties of the object. During the simulation, the mouse can also be used to apply forces to objects, by dragging after clicking on an object.

A debug mode can be enabled that shows real-time information about the whole scene, the selected object and additional information about the system state or the simulation control.

5. Render engine

For the graphical display of the scene, the render module takes the information from all the elements in the simulation and displays them according to their properties. The use of WebGL [Khr13] was considered, as it is efficient and powerful. However, finally it was discarded due to compatibility issues with many browsers, specially on mobile devices. Finally, two render systems have been built for rendering purposes. The main render system, which is the default option in the application, is based on the HTML5 Canvas element [W3C13a], which works generating bitmaps. A secondary render system uses the Scalable Vector Graphics (SVG) image format [W3C11].

Next we describe the advantages and disadvantages of both approaches, and justify the decision of building a second render engine.

5.1. SVG

The advantage of using SVG is that the output is rendered from a vector graphics description. Most modern web browsers are capable of rendering such graphics in a native way, which results in a smooth render, regardless the resolution of the display and the scale of the image. Moreover, the standard [W3C11] provides an API that includes many functions for user interaction, including the possibility of using timers. This makes SVG a very good option both from the quality and from the programming perspective.

However, once the state of the scene is processed by the application and an SVG description is generated, this new representation of the scene has to be processed again by the web browser to render the final image. For this reason, when simulating complex scenes with many objects the SVG-based render module can show a low frame rate. This is not adequate for educational purposes, specially when interaction is one of the main features of the application.

5.2. HTML5 Canvas

In contrast with SVG, which is a graphics description based on geometric primitives, the Canvas element of the HTML5 recommendation [W3C13a] only provides the programmer with a bitmap image and an API to draw pixels on it. Although the image quality can be poorer than the result achieved with vector graphics, this approach is much faster than the use of an SVG description of the scene. This higher performance makes it a good option for generating raster graphics in real time.

However, it lacks an API for any user interaction or for animation events. For this reason it was necessary to build an API on top of the standard HTML5 Canvas to extend it to have timer events and user action events, for interactive scene manipulation.

The first extension to Canvas that has been implemented is the decision whether a point is inside a polygon, described by its vertexes. This first extension has been later used for other features.

Other methods that have been implemented have been selection and dragging of objects. This basic feature is not available in the Canvas element, as it is bitmap based.

Another feature that can be found in the SVG display that is not present in the Canvas element is the use of timers that fire events at fixed intervals. For this reason, our Canvas based display uses the `requestAnimationFrame()` method [W3C13b]. Another method, `setInterval()`, which calls a function after a fixed time interval, seemed the

natural option, but was discarded as it caused flickering in the display refresh.

Motivated by the lower quality of the Canvas display, compared to SVG display, we have implemented a dropped shadow which improves the aesthetic result of the geometric shapes. Despite the simplicity of this improvement, it produces a noticeably better result, specially when objects are in movement.

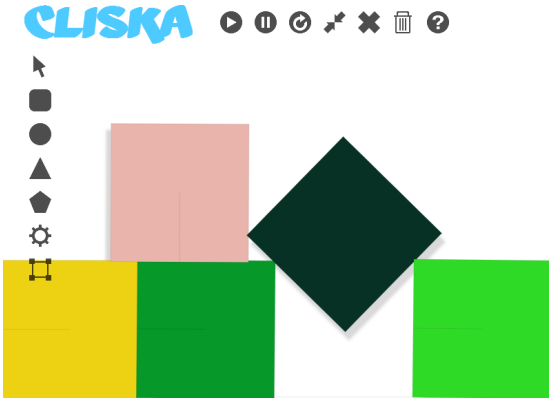

Figure 4: *In order to improve the visual result when using Canvas-based viewer the objects drop a soft shadow.*

6. Simulation Engine

The core module of the application is the simulation engine. It is a set of routines that compute the evolution of the scene according to physics laws using numerical methods. Our scene will consist of an empty space in which the user can place objects of different types. The types of physical objects that can be created are the following:

- Particles: point objects that have mass but do not have spatial extent.
- Rigid bodies: rigid objects with geometries and associated materials. We can vary their position and orientation. Their geometry is fixed.
- Deformable bodies: As rigid bodies, we can vary their position and orientation. In addition, their geometry is flexible.
- Force Fields: They are values of force acting on a given region of space.

In addition to the previous types of objects, currently some other models are being developed to simulate:

- Fluids: They are formed by particles. With no geometry, they adapt to the shape of the container.
- Constraints: Physics limitations. They limit the range of movement and force some dynamics behaviours on the objects. Some of them are springs and joints. They reduce the degrees of freedom of a system.

6.1. Data Model

Our simulation is based on a simplified scene graph, with a root node representing the virtual world and nodes that are inserted in the graph when they are created. We consider no hierarchy among the physics objects, and we use a list structure to store all the systems in the scene.

The basic entity of our model is an abstract object that has a position and orientation, a geometry and a material description. The material description includes the mass and inertia, if applicable, and other properties that can be used for drawing the object or for its collision properties. Our geometry description is based on vertexes; the geometry of any object is defined as a list of vertexes to build closed polygons.

A rigid body has a geometric description, that determines the inertia tensor. On the contrary, a particle has neither a geometric description nor an orientation, but just mass and a position, and also lacks an inertia tensor. Deformable objects are modelled by means of the Mass-Spring Model, so they are described as sets of particles, with springs that link them.

For the implementation of the simulation step, every model has its own numerical integrator that is run in the update process. This integrator is implemented as a method of the corresponding class in JavaScript.

Although fluid simulation is not yet complete, the description of a fluid object is straightforward if a Lagrangian approach is used. Our implementation will be based on the SPH model [MCG03] that will be implemented as an additional particle system, with the corresponding update process.

6.2. Collisions

Collision detection and reaction is a key feature in simulation of physics based models. An adequate response to contacts between two bodies is necessary for a realistic behaviour.

In our application, we perform collision detection in two steps, following the classical approach. First, a broad phase is performed in order to discard any pair of objects that are too far apart to be colliding. By doing this step, we avoid computing a huge number of collision detection tests [Eri05]. For this phase, we use axis aligned bounding boxes and discard any pair of objects whose bounding boxes do not overlap.

All the pairs of objects that have not been discarded are considered to be potentially colliding. Then, we compute a narrow phase collision detection to determine if they are actually colliding and, in case of collision, we determine a contact point and a normal direction [Eri05]. Taking advantage of the geometric description of the shapes, based on a list of vertexes, this narrow phase is done by detecting if any vertex

of one geometry is inside the other geometry, and contact is always considered as a vertex-edge contact (see Figure 5).

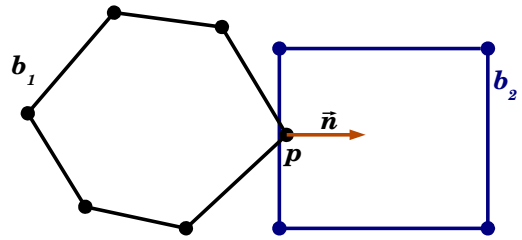

Figure 5: *Every object is described as a list of vertexes. Collision detection is performed checking if any vertex of body* $\mathbf{b}_1$ *is inside the shape of body* $\mathbf{b}_2$.

For every contact, an impulse is computed so that the relative velocity of bodies at contact point is not pointing in the direction that causes interpenetration. If interpenetration has already happened, it is corrected before the simulation runs the next step. The impulse and a friction force are computed according to [ESHD05]. In Figure 6 a set of shapes has been dropped and they have formed an object stack.

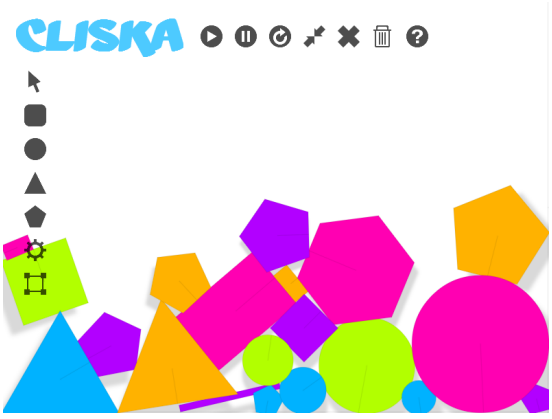

Figure 6: *An object stacking formed by objects of different shapes.*

Prior to clearing the forces after the simulation step, for every object we check if both velocity and applied forces are smaller than given thresholds. If that is the case, the object is put to sleep, and it is not integrated in the next simulation step. This strategy, used by many physics engine, helps saving computations and also prevents object drift.

6.3. Implementation of the Simulation Step

The simulation step is launched as an event during the drawing process. Before drawing a scene, the simulation loop is called to compute the new state. When called, it performs a series of substeps, ending with a new set of positions and velocities for all the objects in the scene [RD11]. The steps that are executed during the simulation update are the following:

1. Collision detection
2. Inter-penetration correction
3. Force computations
4. Numerical integration of velocities and positions
5. Impulse computation and application

The equations of motion of the different objects depend on their type. However in most cases they reduce to the Second Newton Law [ESHD05]. The numerical integration of the system is done using a Verlet scheme with a fixed time step [HNW93,ESHD05]. Depending on the frame rate of the drawing process, several integration steps can be performed to guarantee a real-time behaviour and a stable simulation.

7. Scene Export

An export module is necessary to save scenes for later usage. In order to use this feature, the user can set up a scene and let it evolve until the desired state, e.g., a rest state. Then, with the simulation in pause, the scene can be saved to file.

When the user decides to save the scene, the different data structures that contain the scene information are transferred to a new data structure using the JavaScript Data Object Model [W3C09]. Then, this structure is saved to disk in a XML file. When the user needs to restore the simulation state, the XML file is loaded and the scene is created with the data stored therein.

This module is currently under development. When it is fully operative, this module will let the users build a library of scenes and examples that will be helpful for the educational purpose of the application. Moreover, when the module is complete, the application will include a sample library of scenes corresponding to concepts taught at different primary and secondary school courses.

8. Usage Examples

The application has been designed to have a flexible behaviour, so that it has very few constraints in what regards its usage. Users, either professors or students, can define their own work-flow to fit their needs. Next we present a proposal of a set of possible usage scenarios.

A first possible usage of the application is as a tool to be used by the professor during lectures to support theoretical concepts. Together with the explanation of a physics principle, the professor can launch the application with an scene that presents one or more examples of the concept that is being introduced. The application can also be used as a virtual physics laboratory. The professor can propose activities and a set of questions to the students, and the students must use the application to reproduce the experiments and answer

the questionnaire. Another possible use is to compare typical physics exercises related to mechanics with the results obtained in the simulator. The students would solve a set of exercises on paper, using calculus, and then would simulate a set of scenes that reproduce the same exercises to observe and manipulate the results.

Next we present a small set of sample exercises that can be developed using CLISKA, for the courses on Science (*Ciencias Naturales*) and Physics (*Física*). They have been designed considering the contents that are taught in secondary school in Spain, according to the applicable regulations [Min07]. For every exercise we present its goal, the scenario that is used and a description of the activity.

Ballistic shot. The goal of this exercise is to put into practice the scientific method.

Scenario. A single object is placed at one side of the scenario. The student sets the starting velocity in magnitude and angle using the object's properties.

Exercise. The student has to set a hypothesis about the reach of the shot. Then, by experimentation, she has to test the hypothesis and validate or discard it. The exercise can be followed by an analytical approach, to compare with the experimental results. An additional concept that is involved is the usage of proper units of measure.

Use of a balance. The goal of this exercise is to practice the usage of measurement devices.

Scenario. A simple balance is built using bodies and linkages. Several small bodies are also set as masses.

Exercise. The student creates a body and sets its mass using the properties dialogue. Then, the body must be weighted using the balance, by placing it on one plate and moving the masses and placing them in the opposite plate. An additional concepts that are involved are the lever and the units of measure.

Coordinate system. The goal of this exercise is to practice the use of Cartesian coordinates.

Scenario. An empty scenario is used.

Exercise. The students are given a picture with some objects placed on a rectangular area. Using a ruler, they have to measure their coordinates from reference system. Then, they have to create objects in the application and place them in the corresponding locations.

Inclined plane. The goal of this exercise is to understand force decomposition and the Newton Laws.

Scenario. A box is placed on an inclined plane without friction.

Exercise. The student predicts analytically the time that the box needs to reach the end of the plane and compares it with experimental results for different angle values. Forces are drawn during the exercise. The students will also practice the scientific method and the experimental process.

Inclined plane with friction. The goal of this exercise is to understand the concept of friction.

Scenario. A box is placed on an inclined plane with friction.

Exercise. After studying the concept of friction the student hypothesize the behaviour of the box on the plane. Then they simulate the scenario for different plane angles and friction values, comparing the results with the theoretical prediction. Later, the experiment is repeated using a sphere. The effect of friction on the rolling movement of the sphere is discussed and tested experimentally. The students will also practice the scientific method.

Harmonic oscillator. The goal of this exercise is to understand Hooke's law and periodic movement.

Scenario. A single object is placed attached to a spring.

Exercise. The student has to displace the object away from the rest position and let it oscillate. The students can be requested to measure the period of oscillation manually and to relate it with the spring constant. In addition, the concepts of kinetic and potential energy are also involved.

This is just a sample of some of the activities that can be carried out using the simulator. Moreover, in the future, the application will be extended to consider additional concepts, related to thermodynamics, electricity or fluids, among others. Then, it will be possible to extend the set of exercises using new objects and practising new course contents.

9. Results

The application that is presented in this paper is a small simulator for educational purposes that runs on a web browser. A running version of the system can be found at `http://www.cliska.com`. We have tested the application in the following web browsers: Microsoft IE, Firefox, Chrome, Safari and Opera, in their desktop versions.

We have done some tests on an AMD Turion X2 Ultra Dual-Core, Mobile, ZM-82, with 4GB RAM and the Chrome web browser. In Figure 1 three of these tests are shown. From left to right, a ball rolling down an inclined plane, a block construction castle and a pool full of objects. For these tests we have evaluated the average number of contacts processed during a time step and the average frame rate (rounded to integer value) for the two render methods. The frame rate was synced to a maximum of 60Hz. In Table 1 we show the results for these tests. The pool example has been run with two values for the number of objects.

The results indicate that the application can run at interactive frame rates for scenes up to about 50 objects with

Scene	Objects	Avg. Cols.	fps^c	fps^S
Inclined Plane	4	7	60	60
Blocks Castle	32	85	60	55
Pool 1	130	250	40	15
Pool 2	200	630	15	1

Table 1: *Results of tests with different scenes. Columns show the number of objects in the scene, the average number of collisions per simulation step and the average frame rate for both the canvas (fps^c) and the SVG (fps^S) render engines.*

both render engines. This is a satisfactory result, considering that most scenes in an educational context can involve about tens of objects. In addition, the table shows that the implementation of the canvas render method has extended the range of scenarios that can be simulated. From the table we can also assume that the frame rate degrades as the number of collisions increase. This result is reasonable, as collision detection involves a considerable number of computations. However, more tests need to be done to decide if the computations involved in every module are the expected ones, and what are the possible bottlenecks.

The application currently has the limitation that cannot run on a mobile device or tablet, as it cannot handle properly the events related to gestures. Also, the application does not interpret mobile gestures that can be an alternative to typical desktop events such as, right click. We have also found some difficulties on some of the browsers tested. While Chrome and Firefox run the application without remarkable issues, the `requestAnimationFrame()` method is only implemented in the newest versions of Safari, IE and Opera. Versions of IE prior to 9 do not have Canvas and SVG implemented at all, and Safari shows refresh rate problems even in its newest version.

Regarding the physics engine, the main limitation found in the working features affects collision response. We have detected that when there is a large number of bodies in contact, the impulses that are computed cannot prevent completely the intersection of objects. This issue can be observed in some of the pictures presented.

10. Conclusions and future work

In this paper we have presented CLISKA, a web application for education of physics courses in primary and secondary school. It is a tool that helps the comprehension of physics laws by experimentation and reproduction of examples.

The system has been developed in a modular manner, so that the display of the system is independent of the physics engine. Moreover, the display module has been implemented using two different approaches, that can be selected by the user; a vector representation and a bitmap representation of the scene. The former provides a higher quality image, which requires more processing, while the latter provides a faster representation, adequate for complex scenes.

The application is highly flexible, as the starting point is an empty scene that the user can arbitrarily populate with objects. For this reason, the system can be used for a wide range of situations that can arise in typical education activities.

The project is still work in progress, as there are many additional features and physics properties that can be added. Currently the export module is being developed to complete the capability of saving and loading scenes. This will also allow to create a library of pre-defined scenes and examples.

The physics engine is capable of simulating rigid and flexible bodies and particle systems. A fluid dynamics module is being developed using the SPH discretization. Moreover, we have plans to develop modules to simulate concepts related to thermodynamics, electricity and electronics, optics or chemistry, among others.

Compatibility with old versions of browsers is not actually an issue, as both Firefox and Chrome are free and can be updated with no cost. However, some work needs to be done to guarantee compatibility with some web browsers, specially to handle properly events generated in mobile versions. In the long term, the application could be rebuilt to run natively on mobile devices and to use the different sensors they carry, such as accelerometers or clinometers.

References

[Ago12] AGOP SHIRINIAN: Phyzicle sandbox. Software package, 2012. Accessed March 2014. URL: https://play.google.com/store/apps/details?id=com.nullular.phyzicle. 2

[Alg13] ALGORYX: Algodoo. Software package, 2013. Accessed March 2014. URL: http://www.algodoo.com/. 2

[Bod09] BODIN M.: Creative interactive environment for doing physics. In *MPTL 14 International Workshop on Multimedia in Physics Teaching and Learning, 23-25 September 2009, University of Udine, Italy* (2009). 2

[Bru14] BRUMMITT L.: Matter.js. HTML5 JavaScript physics engine. Software package, 2014. Accessed May 2014. URL: http://brm.io/matter-js/. 3

[Cat13] CATTO E.: Box2d. a 2d physics engine for games. Software package, 2013. Accessed May 2014. URL: http://box2d.org/. 3

[Cro] CROCODILE CLIPS LTD.: Yenca. Software package. Accessed March 2014. URL: http://www.yenka.com/. 2

[Del12] DELTODESCO L.: Nape physics engine. Software package, 2012. Accessed May 2014. URL: http://napephys.com/. 3

[Eri05] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2005. 5

[ESHD05] ERLEBEN K., SPORRING J., HENRIKSEN K., DOHLMANN H.: *Physics-based animation*. Charles River Media, 2005. 6

[HNW93] HAIRER N., NORSET S. P., WANNER G.: *Solving ordinary differential equations 1: Nonstiff problems*. Springer, Berlin, 1993. 6

[JS11] JEREZ J., SUERO A.: Newton physics engine. Software package, 2011. Accessed March 2014. URL: http://newtondynamics.com/. 2

[Khr13] KHRONOS GROUP: Webgl specification, version 1.0. Web page, 2013. Accessed May 2014. URL: https://www.khronos.org/registry/webgl/specs/1.0/. 4

[Kol84] KOLB D. A.: *Experiential Learning-Experience as the Source of Learning and Development*. Prentice-Hall, Nova Jersey, 1984. 2

[Lof13] LOFTIS H.: Newton. a playful, particle-based physics engine. Software package, 2013. Accessed May 2014. URL: http://hunterloftis.github.io/newton/. 2

[LZM12] LEWIS M. S., ZHAO J., MONTCLARE J. K.: Development and implementation of high school chemistry modules using touch-screen technologies. *Journal of Chemical Education 89*, 8 (2012), 1012–1018. 2

[Ló14] LÓPEZ J.: La web de física. Web Page, 2014. Accessed March 2014. URL: http://www.lawebdefisica.com/nivel/secundaria.php. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2003), Eurographics Association, pp. 154–159. 5

[Min07] MINISTERIO DE EDUCACIÓN Y CIENCIA: Real decreto 1631/2006, de 29 de diciembre, por el que se establecen las enseñanzas mínimas correspondientes a la educación secundaria obligatoria. *Boletín Oficial del Estado*, 5 (2007), 677–773. 7

[Nat13] NATIONAL TAIWAN NORMAL UNIVERSITY: Ntnujava Virtual Physics Laboratory. Web Page, 2013. Accessed March 2014. URL: http://www.phy.ntnu.edu.tw/ntnujava/. 2

[Nog11] NOGUEIRA GUSTAVINO M.: El constructivismo como base teórica del nuevo método docente y su proyección en los estudios de derecho del trabajo. *Revista General de Derecho del Trabajo y de la Seguridad Social 27* (2011). 2

[RAJC09] ROVELO G., ABAD F., JUAN M. C., CAMAHORT E.: Sistema de realidad aumentada para enseñanza de geometría. In *CEIG09: Congreso Español de Informática Gráfica* (2009), pp. 27–36. 2

[RD11] RAMTAL D., DOBRE A.: *Physics for Flash Games, Animation, and Simulations*. Friendsof, Apress, 2011. 6

[Ske09] SKETCHYPHYSICS: Sketchyphysics. Software package, 2009. Accessed March 2014. URL: http://sketchyphysics.wikia.com/wiki/SketchyPhysicsWiki. 2

[SPB12] SAMPEDRO F., PUIG A., BENSENY A.: Modular design of graph theory based software for scientific applications and education. In *CEIG12: Congreso Español de Informática Gráfica* (2012), pp. 167–167. 2

[Sub13] SUB PROTOCOL: Verlet.js. Software package, 2013. Accessed May 2014. URL: http://subprotocol.com/verlet-js/. 2

[W3C09] W3CONSORTIUM: JavaScript and HTML DOM Reference. Web Page, 2009. Accessed March 2014. URL: http://www.w3.org/DOM/. 6

[W3C11] W3CONSORTIUM: Scalable vector graphics (SVG). Web Page, 2011. Accessed March 2014. URL: http://www.w3.org/Graphics/SVG/. 4

[W3C13a] W3CONSORTIUM: HTML canvas 2D context, level 2. Web Page, 2013. Accessed March 2014. URL: http://www.w3.org/TR/2dcontext2/. 4

[W3C13b] W3CONSORTIUM: Timing control for script-based animations. Web Page, 2013. Accessed March 2013. URL: http://www.w3.org/TR/animation-timing/. 4