Physical Simulation Related Exercises for the Education in the STEM Field – Approaches Based on the Physolator Framework

Dirk Eisenbiegler[1*], Dietmar Gruber[2], Thomas Jörg[2]

[1]University of Furtwangen, Germany; *dirk.eisenbiegler@hs-furtwangen.de
[2]Hector Seminar, Germany

SNE 27(1), 2017, 45 - 52; DOI: 10.11128/sne.27.en.10367
Received: February 20, 2017 (Selected ASIM STS 2016 Postconf. Publ.), Accepted: March 10, 2017
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. This paper presents different kinds of approaches towards using physical simulation based exercises for early teaching of STEM topics at school[1]. The approaches presented in this paper are based on the Physolator physics simulation framework. This paper analyzes to which degree these approaches can be used to meet given teaching goals.

Introduction

In our world, computer based physical simulations are omnipresent. They are used for science and engineering as well as in computer games and in animations for the movie industry. Professionals working with computer based physical simulations are aware of the fact that it takes good skills at least in the following fields for producing such computer based applications: physical modeling, geometry, numerical mathematics, programming and graphics programming. The significant skills for producing physical simulations are all located in the STEM fields[1]. Work in this field is interdisciplinary. It takes physicists, mathematicians and computer scientist working together in such projects.

The guiding questions of this paper: Could physical simulation be used in early teaching of mathematics, physics and computer science at school? Could this lead to more application oriented understanding of the different topics in mathematics, physics and computer science? Could this strengthen an interdisciplinary thinking for these domains?

1 Teaching Physics and Physical Simulation

There are numerous publications dealing with the question on how to improve teaching in science and especially in physics [7,8,9]. In a physics lecture, students shall learn the theory of physics and they shall learn how apply this knowledge to real world scenarios. Experiments play an important role when teaching physics. Experiments are used to confirm theoretical models. On the other hand side, experiments are stimulated by theory. Students should learn, how theoretical models are used to describe nature and how experiments in nature are used to verify theoretical models.

Physical simulation is a supplement to theory and experiment. Working with physical models means applying theoretical knowledge [6]. Just like real world experiments, physical simulations are motivated by theory. The students have to analyze the simulation runs to see if or if not the physical simulation confirms their expectations about the real world behavior. Physical simulations can never be a substitute for experiments. Physical simulations are used for verification and clarification: Is the physical simulation, which is based on the theoretical formulas, consistent with the real world observations from experiments?

A learner who is starting to set up his first physical simulation is confronted mainly with two different challenges: Understanding the physical model and understanding the way of implementing the physical model using a programming language.

[1] STEM = science, technology, engineering, and mathematics

The following aspects have to be considered when learning programming: learn the syntax of a programming language, learn the fundamental, imperative concepts of programming and optionally learn the concepts of object orientated programming such as classes and instances, inheritance and dynamic binding. From a didactic point of view it is important to keep the different aspects of programming as separated as possible and to teach them in a step-by-step manner.

The approaches presented in this paper are based on the Physolator framework (see www.physolator.de). Physolator uses the Java programming language. Java is taught in many secondary schools. Frameworks such as Mathematica or Matlab could also be used for physical simulations. However, they come with their own programming language and the student has to learn this extra programming language to get started with such frameworks. The Physolator is Java based. Students do not have to learn an extra programming language before they get started with physical simulations.

Applying Java in the context of physical simulations helps students to acquire a deeper understanding of Java since implementing a physical simulation is an exercise of already taught programming lessons.

Furthermore, Physolator is able to encapsulate the higher level OOP parts of Java, therefore entanglements of too many didactic aspects can be avoided effectively.

An exercise on the beginner level should put a focus on one topic only: One exercise for learning the meaning of gravity by playing with a given model, another exercise for learning how to build a physical model, another exercise for learning how to build graphics components. Each of these topics should first be learned in an isolated way. When for example building a first physical model in a science class, students should put a focus on understanding the underlying physical context, and the corresponding formulas. Students should not at the same time have to deal with the complexity of modern object oriented programming or with numerical mathematics or with graphics programming.

2 The Physolator Framework

Programming a physical simulation from scratch is considered to be very challenging. The Physolator is designed for physical simulations at the beginner's level. Physical systems are implemented as Java programs.

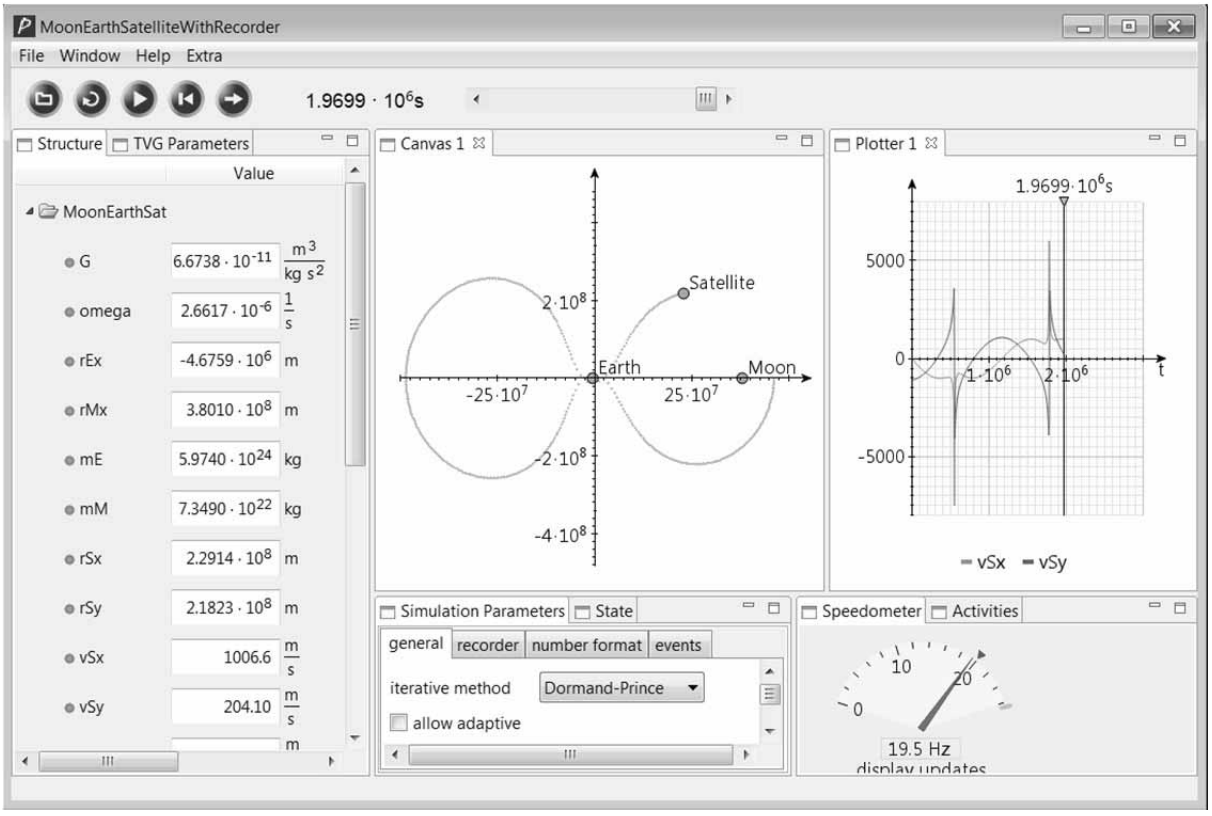

Figure 1: The Physolator Framework.

To build your own physical system up and running, you write a piece of Java code, load it to the Physolator framework and then start the simulation by pressing the start button inside the Physolator. The Physolator framework is based on ODE solvers. From the Physolator's perspective, physical values are initial value problems and it uses ODE solvers for executing the simulations. Besides the ODE solvers, the Physolator framework also supports an event oriented programming style for simulating physical events such as collisions.

Figure 1 shows a snapshot of the Physolator framework. A physical system with a satellite revolving around moon and earth has already been loaded. Physical variables are always displayed on the left. Their values change during simulation. On the middle there is a graphical representation of the physical system and on the right there is a plotter for displaying the function graphs of selected variables. With the round buttons on the top you can load physical systems, start and stop them. During simulation time the user may also interfere and change the variables values manually. Editing the variables immediately changes the state of the system and its graphical representation.

The Physolator has been designed for modular style of implementing physical systems. With the Physolator one can build a set of basic physical components. Using this set of physical components, one can build more complex components by just joining together the basic components. Also the graphical components for visualizing the simulation results as well as the numerical procedures are developed independently. For every exercise the instructors provide the students with a set of base components: physical components, graphics components and numerical procedures. During the exercise, the students have to build a physical system on top of this infrastructure. They can focus their work on a very specific task. The given infrastructure of components should cover all the aspects the students should not have to take care off.

This paper presents different categories of exercises related to physical simulation with a focus on exercises at the beginner level. The different categories represent different categories of didactic concepts.

With the Physolator one could also define exercises for advanced students such as programming numerical algorithms. Such advanced tasks, however, are not part of the scope of this paper. This paper limits itself to exercises that are well suited for students at school level or at the beginner level at a university.

2.1 Category 1: Experimentation with a given physical system

Traditionally, physical experiments are based on mechanical or electrical devices. During the exercise, the students build an apparatus by assembling these devices and then execute different runs with varying parameters and conditions. Such experiments pursue the following teaching goals:

- Give the students a practical experience of the theoretical physical concepts presented during the theoretical parts of the lecture (e.g. gravitation, friction...).
- Make sure, the students can apply theoretical concepts to the real world: relationship between variables and formulas in the theoretical world and observations in the real world.
- Explain to the students the meaning of physical models computer based simulations as a part of the scientific research process of physicists.

Readymade computer simulations can be a complement for such real experiments. For good real world examples you need the right devices. Some of them are costly and assembling an apparatus is time consuming. Some experiments that would be useful to for a better understanding of the physical domain, simply cannot be run in a classroom. One can simulate the orbit of a satellite on the computer, but not in a classroom.

The moon-earth-satellite example is a physical system that is well suited exercises of this category. Exercises: Give the satellite the right initial position and speed and observe the path of its movement! Try find an initial position and speed so the satellite runs on a closed orbit! During simulation time observe the relevant forces and accelerations: gravity, Coriolis force, centrifugal force!

In this kind of experiment, the student loads a readymade physical system to the Physolator. In this virtual experiment interacting with the physical system means ‘playing’ with the physical variables and observing the impact on the behavior of the physical system.

2.2 Category 2: Building a physical model

In this kind of experiments, mathematical formulas shall be used to build a model of a simple physical system. The students shall write down the physical variables and physical formulas in Java notation and then load and start their physical system.

The learning tasks to be pursued with this kind of exercises are:

- Learn, how to describe a consistent physical model using physical variables, formulas, and derivation relationships and provide the physical system with an initial state.
- Understand the relationship between formulas defining the behavior and temporal progressions of physical systems following the rules defined by these formulas.
- Learn how to define physical models and learn about the limitations of physical models.

In this kind of exercise, students write Java program code. The programming language Java, however, is used in a very limited way. In this context it is only used for writing down physical variables and formulas. The entire program code only consists of Java variable declarations and value assignments to variables. The Java variables correspond to physical variables. Java variables are a means for representing physical variables in a computer. Variable assignments are a means for representing formulas. The variable assignment in Java assigns a value to the variable. The value is defined by a mathematical expression. The mathematical expression represents the formula.

The following program code gives an example for such a physical system.

```
public class SpringMassPendulum
           extends PhysicalSystem {
  @V(unit = "m")
  public double p0 = 1;
  @V(unit = "m")
  public double g = -9.81;
  @V(unit = "kg")
  public double m = 0.03;
  @V(unit = "N/m")
  public double D = 2;
  @V(unit = "")
  public double k = 0.05;
  @V(unit = "N")
  public double F;
  @V(unit = "m", derivative = "v")
  public double x = 0;
  @V(unit = "m/s", derivative = "a")
  public double v;
  @V(unit = "m/s^2")
  public double a;
  public void f(double t, double h) {
    F = g * m + D * (p0 - x) - k * v;
    a = F / m;
  }
  public void initPlotterDescriptors(
          PlotterParameters r) {
    r.add("x,v,F", 5, -7, 7);
  }
}
```

The program code above defines a spring mass pendulum with damping. The mass *m*, the pivot position *p0*, the earth acceleration *g*, the spring constant *D* and the coefficient of friction *k* are given constants. The position of the point mass *x* and its actual velocity *v* define the state of the physical system. Annotations *@V* are used for attaching physical units to the variables and for defining derivation relationships. In this case, *v* is the first derivative of *x* and *a* is the first derivative of *v*. The formulas are defined inside method *f*. These formulas define the actual force and the actual acceleration.

This kind of Java program represents a physical system. The students learn, that this kind of notation is used for writing down physical variables, formulas and derivation relationships. At that time, the students do not necessarily have to understand Java. There are no concepts being used that go beyond physical variables and formulas – no control structures, no parameters, no methods, no exception handling etc..

The initPlotterDescriptors method declares, that *x*, *v* and *F* shall be plotted during simulation. Figure 2 shows the result of the simulation run.

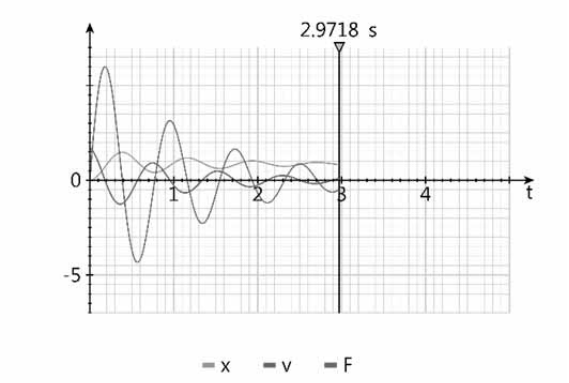

Figure 2: Spring mass pendulum with damping.

2.3 Category 3: Building a physical system by composing given physical components

In this kind of experiment, physical systems are build up by composing given physical components. The following program code describes a double pendulum with two point masses connected to a pivot point via two springs (see Figure 3).

In the first the basic variables are declared and they are initialized with appropriate values. You do not really need to have any Java knowledge to understand, that this piece of code creates a vector g representing the earth acceleration, one pivot point, two springs and two point masses. In the second part (the constructor), these physical components are connected to each other. The first spring refers to the pivot and the first point mass, the first point mass refers to the first and the second spring, the second spring refers to both point masses and the second point mass refers to the second spring.

```
public class SpringDoublePendulum2D
     extends PhysicalSystem {
  public Vector2D g =
      new Vector2D(0, -9.81);
  public Vector2D pivot1 =
    new Vector2D(4, 8);
  public Spring2D spring1 =
    new Spring2D(25.5, 1, 1e5);
  public Spring2D spring2 =
    new Spring2D(25.5, 1, 1e5);
  public PointMass2D pm1 =
    new PointMass2D(3.5, 4, 0.5, 1, 0.5, g);
  public PointMass2D pm2 =
    new PointMass2D(5.5, 1.5, 0.2, 0,0.5, g);
  public SpringDoublePendulum2D() {
    spring1.r1 = pivot1;
    spring1.r2 = pm1.r;
    spring2.r1 = pm1.r;
    spring2.r2 = pm2.r;
    pm1.springs =
      new Spring2D[] { spring1, spring2 };
    pm2.springs = new Spring2D[] { spring2 };
  }
}
```

This piece of program code represents a physical system that is ready to be loaded and run. Be aware, that this physical system does not contain any formula. All you have to do is create such components and connect them with one another. The physical formulas are inside these components. This is why a spring 'knows', how to calculate its force as soon as it is connected with two endpoints. If a point mass is connected to one or several springs, then the point mass 'knows', that the forces from the springs have to be applied to the point mass.

Exercises from this category are similar to category 1. The students should learn about specific physical phenomena. Other than in category 1, the physical system is not ready made, but the students can compose them by themselves.

Thereby, they can also vary the model and build their own physical model. Example: Build a chain of point masses interconnected with springs, stimulate the first point mass and see how a wave moves through the physical system.

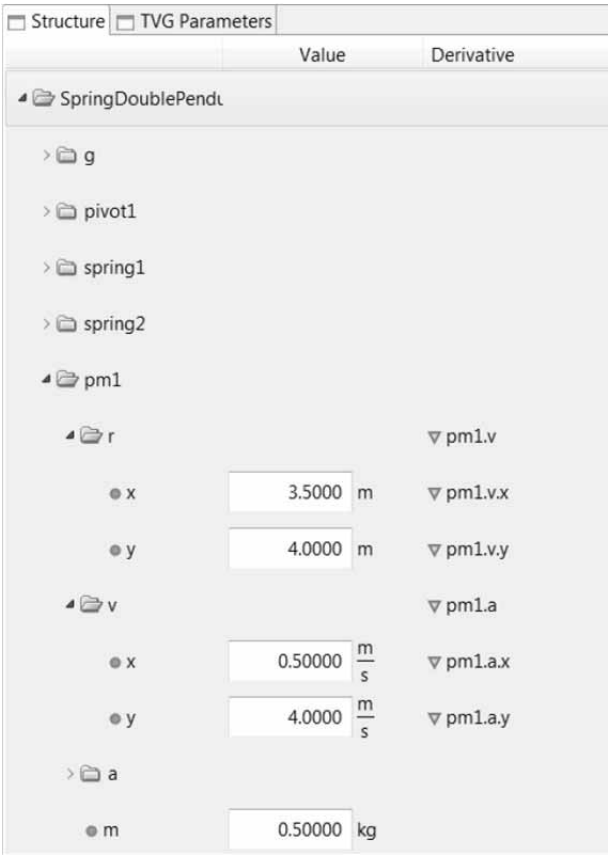

Figure 3: Structure of spring mass double pendulum.

Basically, this example uses object oriented programming techniques. The students, however, do not necessarily have to understand the underlying concepts. The program code is used as a specific kind of notation for describing physical components and the relationships between these components. This is how the students get used to object oriented modeling in an application oriented fashion – without yet knowing the underlying object oriented concepts such as classes, instances, constructors, and inheritance. After loading the physical system, the structure of this physical system with its hierarchy of components and subcomponents and its derivation relationships is visually represented in the Physolator framework (see Figure 3).

For such systems, one may also provide graphics components that automatically visually represent the state of the system on the screen. *MechanicsTVG* is a generic 2D graphics component for visualizing simple mechanical systems. Adding the following program code to the above physical system results in an additional graphics component drawing the point masses and springs on the screen (see figure 4).

```
public void initGraphicsComponents(
  GraphicsComponents g, Structure s,
  Recorder r, SimulationParameters sp) {
    g.addTVG(new MechanicsTVG(this, s, r));
}
```

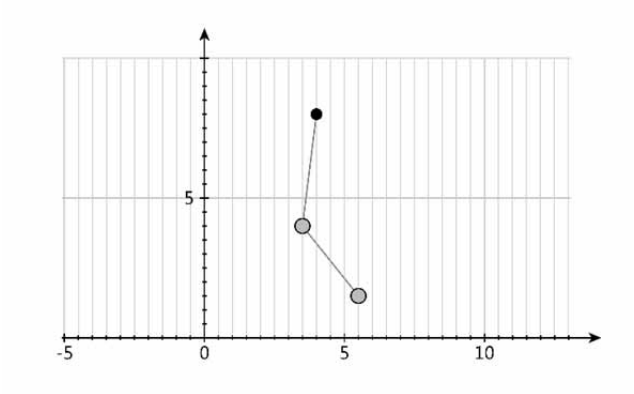

Figure 4: Spring mass double pendulum.

2.4 Category 4: Graphics programming

Simple two dimensional graphics programming is well suited for programming exercises at the beginner's level. When the program draws lines and circles on the screen, students get an immediate visual feedback, showing if the program is doing what it is supposed to do. At the same time, the students also have to deal with geometry in an applied manner.

For a simple two dimensional graphics programming, you would not necessarily need the Physolator framework. If your students, however, have already built a physical model such as the examples from category 2 or 3, then it makes perfect sense to define an exercise, where the students program a graphics component that graphically displays the state of the physical system.

During a physical simulation, the graphics are in motion. The graphic represents the physical system state and the physical system state changes during time. Time dependent graphics are nothing but movies.

With the Physolator, you can also produce movies without having to deal with any physics: Build an empty physical system, let this physical system load a graphics component and from inside the graphics component access the actual value of the simulation time *t*.

The Physolator framework also supports three dimensional graphics based on OpenGL. Three dimensional graphics programming is far more challenging. In such an exercise the students have to learn, how to define a 3D environment with a given camera position and camera direction, certain sources of light, 3D objects of certain shapes and a certain reflection behavior of their surfaces, fog, etc..

2.5 Category 5: Object oriented programming

Teaching object oriented programming concepts is not easy. The teaching goals of such lectures are:

- Learn the basic language constructs and concepts of an object oriented programming language: classes, instance, constructors, encapsulation and inheritance.
- Learn, how to apply these language constructs for developing complex software system and build a software structure that is designed for reusability.

Object oriented programming pays out when developing complex software. Unfortunately, in a programming class the time for the practical exercises is very limited and this is why in such exercises usually only small pieces of program code are produced. For small sized problems, it is hard to explain, that object oriented techniques are superior to the quick and dirty approach without a welldefined object oriented structure.

In a category 5 exercise, students shall use object oriented programming techniques to build their own physical components and use them within physical systems. Before starting with a category 5 exercise, students should first do some category 3 exercises. In a category 3 exercise, students have learn, how to build a physical system by composing given physical components. In a category 3 exercise, the students are using the notations from object oriented programming without necessarily understanding, that this program code is about object oriented programming and that the program code deals with classes, uses constructors and creates instances of classes.

As a preparation for a category 5 exercise, students have to learn the meaning of these language concepts. Then the students shall use these concepts to build their own physical components and use them inside physical systems.

An exercise from this category could ask the students to build the physical components from scratch. As an example, the program code for point masses and springs is easy to implement. Examples with point masses and springs can be found in [4]. This book also thoroughly discusses different kinds of modeling techniques using these examples. Other examples for physical components and the object oriented approaches being used to implement them, can be found in [3] and [5].

An object oriented modeling exercise does not necessarily have to start from scratch. Inheritance can also start with a given example. Sample-Scenario: Build a physical system which is based on the following physical system, but replace the graphics component with your own graphics component and use different simulation parameters. Another example, where inheritance is used on the level of physical components: Build a physical system, that is based on the string double pendulum example from category 3, but replace the given linear springs (Hooke's law) with nonlinear plate springs. The following program code uses inheritance to define *PlateString2D* a son class of *Spring2D*. Due to the fact, that *PlateString2D* inherits from *Spring2D*, one can modify the spring double pendulum code by replacing all occurrences of *Spring2D* by *PlateString2D* and then run the same example with plate springs.

The program code below adds the relevant parameters of a plate springs (*F0*, *p* and *h*) and uses overwriting to provide the class with a new implementation of method computeF in order to define the physical behavior of the plate spring.

```
public class PlateSpring2D
                      extends Spring2D {
  public double F0;
  public double p;
  public double h;
  public PlateSpring2D(double F0, double p,
        double h) {
    this.F0 = F0;
    this.p = p;
    this.h = h;
  }
  public double computeF(double distance) {
    double sigma = distance / h;
    return F0 * sigma *
        ((1 - sigma) * (1 - 0.5 * sigma) *
        Math.pow(h / p, 2) + 1);
  }
}
```

2.6 Category 6: Learning basics about physical simulation

In the exercises from the previous categories, students run physical simulations without necessarily understanding, how physical simulations are executed. There are quite some facts, that students could learn about physical simulations and there are exercises for deepening the understanding. This kind of knowledge is not only applicable to physical simulation, but also gives the students a better understanding about the core principles of computer games and computer based animations.

The teaching goals to be pursued in this category of exercises.

- Understand, that a physical simulation is executed in a time discrete manner.
- Learn, that a numerical simulation has limited accuracy. Understand, that a smaller step width results in a more precise simulation run, but result in a higher computational effort.
- Learn, that there some physical systems such as a simple trajectory, where there are algebraic ways to describe the behavior of a physical system with respect to time (closed solutions). In most other cases, the computer based simulations have to be used.
- Learn about different kinds of numerical procedures: fixed step width vs. flexible step width, single step vs. multi-step procedures.

The exercises from this category shall deepen the understanding about this domain. The Physolator framework provides several simulation parameters. The simulation parameters allow the user to choose, how the simulation is carried out. Among others, the user may choose a step width and the user may also choose among different kinds of numerical procedures (ODE solvers) such as Runge-Kutta, Adam-Bashforth, Cash-Karp and Dormand-Prince. In an exercise, students could be asked to load a given physical system and then find the right simulation parameter settings so the simulation runs with a highest possible accuracy and with a minimum of computational effort (CPU time consumption).

The Moon-Earth-Satellite from figure 1 would be well suited for this purpose – but one could also use any other physical system. In this exercise, students have to choose the right numerical procedure and the right step width. The computational effort for every simulation run can be monitored using the Physolator's built in monitoring tools (see Figure 5).

Students working on this exercise will quickly realize, that the accuracy and the computational effort are depending on the underlying numerical procedure as well as the step width. They will first have to produce a very precise result using very small step widths thus decreasing the truncation error. Too small step widths, however, lead to an increase of the round-off error. By working on such an exercise, the student get an awareness of the fact, that simulations are run in a time discrete manner and that the total error increases the longer the simulation runs. In a follow-up lecture one could explain the underlying theoretical problems.

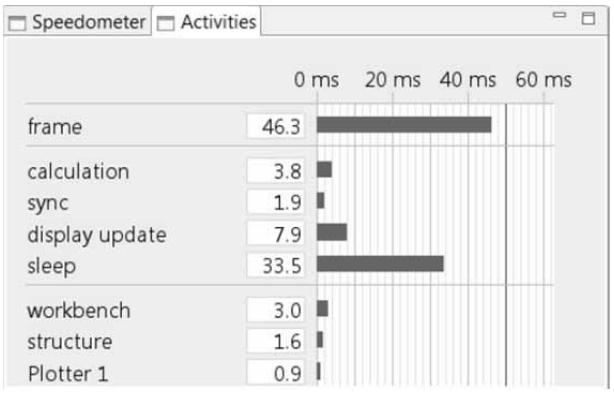

Figure 5: Performance monitor.

A variation of this kind of exercise would work with a physical system, where a direct, algebraic solution exists. Examples for such physical systems: simple trajectory, damped point-spring-pendulum. Other than in the previous type of exercise, a precise solution is given and therefore it is easy to compute the error at any time. Working with such an example would also deepen the awareness, that at least for some physical systems, the behavior of physical systems can be described using closed equations and a physical simulation is not necessarily required in these cases.

3 Summary and Conclusion

This paper has presented different categories of exercises related to physical simulation and it has been explained, how the Physolator simulation framework can be used as an infrastructure for such exercises.

It has been shown, that the different types of exercises pursue different kinds of teaching goals. All of the teaching goals for such exercises are in the STEM field. In many exercises, several STEM qualifications are needed: physics, programming, mathematics. These qualifications have to be combined when working on the exercise. This is why physical simulation is a domain, where students not only acquire knowledge from different STEM fields, but also learn, how to work in an interdisciplinary manner and combine these skills.

References

[1] Eisenbiegler D. “The Software Architecture of the Physolator – a Physical Simulation Framework”, MSAM 2015, Atlantis Press, pp. 61-64.

[2] Eisenbiegler D. “Object Oriented Modeling and Simulation with the Physolator – Getting Started”, https://opus.hs-furtwangen.de/frontdoor/index/index/docId/614, 2016.

[3] Eisenbiegler D. “A Generic Particle Modeling Library for Fluid Simulation”, AMSM 2016, Atlantis Press.

[4] Eisenbiegler D.“Objektorientierte Model-lierung und Simulation physikalischer Systeme mit dem Physolator”, BoD Norderstedt, 2015.

[5] Eisenbiegler D. “An Object Oriented Library for Acoustics Simulation Based on the Physolator Simulation Framework”, CMSAM 2016, DEStech Publications.

[6] Bloom B. “Taxonomy of Educational Objectives”, Pearson Eduction, 1984

[7] Pintó R, Couso D (editors). “Contributions from Science Education Research”, Springer, 2014

[8] Papadouris N, Hadjigeorgiou A, Constantinou C (editors). “Insights from Research in Science Teaching and Learning”, Springer, 2013.

[9] Mikelskis-Seifert S, et.al.. “Physik-Methodik”, Cornelson Stuttgart, 2010.