# SketchyDynamics: A Library for the Development of Physics Simulation Applications with Sketch-Based Interfaces

Abílio Costa[1], João P. Pereira[1,2]
[1] Computer Science Department, School of Engineering (ISEP), Polytechnic of Porto, R. Dr. António Bernardino de Almeida 431, Porto, Portugal
[2] Knowledge Engineering and Decision Support Group (GECAD), School of Engineering, Polytechnic of Porto, R. Dr. António Bernardino de Almeida 431, Porto, Portugal
amfcalt@gmail.com, jjp@isep.ipp.pt

***Abstract* — Sketch-based interfaces provide a powerful, natural and intuitive way for users to interact with an application. By combining a sketch-based interface with a physically simulated environment, an application offers the means for users to rapidly sketch a set of objects, like if they are doing it on piece of paper, and see how these objects behave in a simulation. In this paper we present SketchyDynamics, a library that intends to facilitate the creation of applications by rapidly providing them a sketch-based interface and physics simulation capabilities. SketchyDynamics was designed to be versatile and customizable but also simple. In fact, a simple application where the user draws objects and they are immediately simulated, colliding with each other and reacting to the specified physical forces, can be created with only 3 lines of code. In order to validate SketchyDynamics design choices, we also present some details of the usability evaluation that was conducted with a proof-of-concept prototype.**

***Keywords* — Gesture Recognition, Physics Simulation, Rigid Body Dynamics, Sketch-Based Interfaces.**

## I. INTRODUCTION

USING pen and paper to draw or sketch something in order to express an idea is very common and also very natural for us. By using this concept in user interfaces one can make the interaction process more natural and spontaneous.

In this paper we propose SketchyDynamics, a programing library to aid in the creation of applications for 2D physics simulations in which the user interacts directly with the scene using a "pen and paper" style interaction. Thus, instead of selecting from a menu which objects compose the scene to be simulated, the user can simply draw them directly into the scene. We hope that developing this library will provide a boost for developers to create new applications around this concept, be they for educational purposes, like an application used to teach physics with an interactive whiteboard, or for entertainment purposes, such as a physics-based game where the user draws parts of the scene in order to reach a goal.

The library supports three gestures to draw rigid bodies and other three to define connections between them. The first three gestures are used to produce rectangles, triangles and circles, which can be created by drawing these symbols directly. Also, the user can draw a zigzag to connect two bodies with a spring, an alpha to pin a body over another and a small circle to define a rotation axis between two bodies. Since both the circle body and the rotation axis relation use the same gesture, we only have in fact five gestures to recognize, presented in Fig. 1.

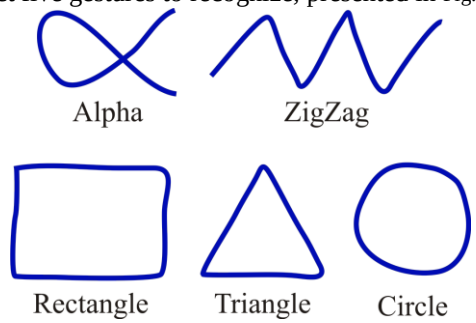


Fig. 1. Set of gestures used in our library

Although there are already several applications that combine physics simulation with a sketch-based interface, most of them have a specific scope and audience. As a library, SketchyDynamics is intended to be used in different types of applications and does not have a definite scope. We hope that our work helps developers create new and exciting applications with little effort in combining the physics simulation with the sketch-based interface.

In the next section we present an overview of the results achieved in the sketch recognition field and also works that combine sketch-based interfaces with rigid body physics simulation. Section 3 gives a little insight into a previous evaluation whose purpose was to select the sketch recognizer that best integrates with our library. In section 4 we present our library, its technical characteristics, along with its functionality. Section 5 discusses a preliminary informal evaluation and section 6 concludes this paper and presents potential future work.


DOI: 10.9781/ijimai.2013.233

## II. Related Work

This section presents some of the related work in the sketch-based interfaces domain and is divided into two subsections. The first subsection will address the work done in the sketch recognition field, while the second presents some examples of applications that result from the combination of sketch-based interfaces with rigid body physics simulation.

### *A. Sketch Recognizers*

Given the potential of automatic sketch recognition, a lot of work has been done in order to develop recognizers capable of dealing with the intrinsic ambiguity of hand-drawn sketches. Since there is a wide variety of sketch recognition algorithms, it is only natural that there's also diversity in their characteristics. Examples of these characteristics are the ability to be trained to recognize new gestures, the capacity to recognize multi-stroke gestures or the sensitivity to the gesture's orientation, scale or drawing direction.

Rubine's recognizer [1], a trainable gesture recognizer, classifies each gesture using a linear classifier algorithm with a set of distinct features. The recognizer is very flexible since features can be easily added or removed to make the recognizer fit the application needs, as proven by Plimmer and Freeman [2]. The major limitations of Rubine's recognizer are its sensitivity to the drawing direction, scale, and orientation and inability to identify multi-stroke sketches. Pereira et al. [3] made some modifications to Rubine's recognizer in order to make the algorithm accept multi-stroke sketches, but only when drawn with a constant set of strokes, as pointed out by Stahovich [4]. Pereira et al. also present a way to make the algorithm insensitive to drawing direction.

CALI [5] is an easy to use multi-stroke recognizer that uses Fuzzy Logic and geometric features to classify gestures independently of their size or orientation. CALI divides gestures into two types: shapes and commands. Shapes can be drawn (and recognized) using solid, dashed and bold lines, while commands are only recognized with solid lines. Since CALI is not trainable, adding new gestures is not an easy task, involving analysis of which features characterize and distinguish the new gesture and hand-coding these features. To solve this limitation the authors also present a trainable recognizer but it has a lower recognition rate and requires numerous training templates for each gesture class[1].

Wobbrock et al. [6] present the $1 Recognizer which aims to be easy to understand and quick to implement. It is insensitive to scale and orientation of sketches, but sensitive to their drawing direction. One major advantage of $1 Recognizer is the simplicity to add support for new gestures, requiring only one training template per gesture class to be effective. Furthermore, the authors also explain how to make the recognizer sensitive to scale or orientation, for some or all gesture templates.

In order to solve some of the limitations of the $1 Recognizer, such as not being able to recognizing multi-stroke gestures, sensitivity to the drawing direction, and problems recognizing uni-dimensional gestures such as lines, Anthony & Wobbrock extended it and created the $N Recognizer [7]. Despite the improvements over the $1 Recognizer, $N has problems recognizing gestures made with more strokes than those used in the training templates. Also, it is not well suited to recognize "messy" gestures like a scratch-out, commonly used for erasing-like actions.

Lee et al. [8] present a trainable graph-based recognizer that is insensitive to orientation, scale and drawing direction and is able to recognize multi-stroke gestures. Since the recognizer uses statistical models to define symbols, it handles the small variations associated with hand-drawn gestures very well. Despite being a trainable recognizer, it requires all training templates of a gesture class to be drawn with a consistent drawing order or consistent orientation.

Vatavu et al. [9] present a trainable recognizer that uses elastic deformation energies to classify single-stroke gestures. The recognizer is naturally insensitive to gesture scale and orientation, since the same gesture has similar curvature functions independently of the drawing orientation or size, but is sensitive to drawing direction and starting point within the gesture.

Sezgin and Davis [10] present a multi-stroke sketch recognizer, based on Hidden Markov Models (HMM), that is capable of recognizing individual sketches in complex scenes even if the scene is not yet completed, i.e. while it is being drawn, and without the need to pre-segment it. On the other hand it can only recognize sketches in their trained orientations, thus being sensitive to orientation. Since the recognition relies on the stroke order of the trained templates, it is not well suited for domains where the stroke ordering cannot be predicted. Also, because HMMs are suited for sequences, it cannot recognize single-stroke sketches, unless they are pre-segmented.

### *B. Physics Simulation with Sketch-Based Interfaces*

The idea of using a sketch-based interface to create and manipulate a simulated scene is not something new. For example, ASSIST [11] is able to recognize sketches and convert them to mechanical objects which can then be simulated. The system recognizes circles and straight-line polygons (simple or complex) made of single or multiple strokes. The recognition is done incrementally, while the user is drawing, which makes the system feel quicker and also gives an instantaneous feedback to the user, since hand-drawn lines are converted to straight lines and colored according to the type of object recognized. When an improper interpretation of a gesture is made, the user is able to correct it using a list of alternative interpretations. In ASSIST, users can also pin one object over another with a rotational axis by drawing a small circle, or anchor objects to the background by drawing a small cross. After finishing the sketch, the user can press a "Run" button to transfer his design to a 2D mechanical simulator that runs and displays a simulation of the designed scene.

[1] A gesture class represents a unique gesture, but can be made from multiple representations of that gesture, i.e. multiple templates.

Another application, "Free-Hand Sketch Recognition for Visualizing Interactive Physics" [12] enables users to draw simple 2D objects and simulate how these objects behave in 3D. The application is able to recognize four types of objects: lines, circles, rectangles, and triangles. When the gesture cannot be recognized a small dialog is presented, requesting the user to specify the desired gesture. After creating an object, the user is able to anchor it so that it remains static during the simulation. The design process consists of three modes: the "Ink" mode where the user can draw new objects; the "Select" mode, where a circle selects the enclosed objects; and the "Erase" mode, used to remove objects. Despite the designing being done in 2D, the physics simulation is 3D and the user is able to move the camera and also move objects in 3D space.

There are also games that take advantage of a sketch-based interface and a physics simulated environment to entertain the player. One popular example is Crayon Physics Deluxe [13], a puzzle game where the main objective is to guide a ball so that it touches all the stars in each level. Instead of controlling the ball directly, the user needs to draw objects that influence the ball, leading it to the stars. The user can draw rigid bodies with any shape and connect them with pivot points and ropes. Since the simulation is always running, sketched objects are simulated and interact with other objects right after being drawn. The game has a "children's drawing" theme, with a background that resembles a yellow paper sheet and crayon-like sketches, both characteristics that make it successfully adopt the pen-paper paradigm. Crayon Physics Deluxe also includes a level editor and an online playground, so users can create their own levels and submit them online.

## III. Sketch-Based Recognition Evaluation

Due to the high importance of having good gesture recognition, since the user must feel the interaction to be as natural and unrestrictive as drawing with a pen on a paper, the gesture recognizer used in SketchyDynamics was selected based on previous evaluation [14] [15]. The evaluation was conducted using real gesture samples drawn by 32 subjects, with a gesture set specifically arranged for our library (Fig. 1).

For the evaluation process we developed an application to collect gesture samples from the subjects, process them, and compute the recognition results. With this tool we evaluated Rubine's recognizer, CALI and the 1$ Recognizer, concluding that for our gesture set CALI achieved the highest recognition rates.

With this evaluation we were also able to improve recognition rates by tweaking the templates and the recognizer's implementation to our specific gesture set.

## IV. The SketchyDynamics Library

SketchyDynamics is a programing library that aims to simplify the implementation of 2D physics simulation applications with sketch-based interfaces. Using 2D graphics and physics simulation means that the user sketch (in 2D) produces a 2D object, which resembles the pen-paper paradigm and simplifies user interaction.

Out of the box, SketchyDynamics provides an interface for the user to interact with an application along with recognition and processing of user actions such as drawing, moving, scaling and removing rigid bodies and their joints. SketchyDynamics also deals with the physics simulation of these elements and visually represent them on the computer screen along with other user interface elements. Thus, a developer can integrate these features in an application with almost no effort.

### A. Architecture

A major concern when designing SketchyDynamics was to make it versatile, so that developers can create all kind of applications, but at the same time simple enough to enable rapid prototyping. For example, with only 3 lines of source code a developer can create a simple test application where the user can draw objects and see their simulation, while they collide with each other and react to the specified "gravitational force". With a dozen more lines the developer is able to add a background body where the user is able to attach objects, or a ground body so that drawn bodies have something to fall onto.

As stated previously, we use CALI as the gesture recognizer since it yielded the best results in our evaluations.

For the physics simulation SketchyDynamics uses the Box2D physics engine. Despite using Box2D, SketchyDynamics does not encapsulate it or hide it from the programmer. Instead programmers have access to all Box2D objects and functionality so they are able to parameterize them according to the application's needs.

Although bodies and joints are created automatically by the library when the user draws them, the application is also able to programmatically create and remove them (along with their visual representations). Furthermore, SketchyDynamics also gives the application full control over the simulation state.

To render the bodies simulated by Box2D and any other visual elements we used the OpenGL API. Despite that, SketchyDynamics was designed so that a developer can easily use another API. This is achieved by ensuring that all OpenGL-specific code is encapsulated in a few classes, thus creating a conceptual abstraction layer.

While implementing the OpenGL abstraction we took the opportunity to add some "graphics library" functionality. For example, a programmer can easily create polygons by defining their vertices and then apply geometric transformations to them, toggle their visibility on screen, among other operations, all done in an object-oriented manner. Additionally, the library provides scene query functionality and easy texture management for the developer. To render each object SketchyDynamics offers three rendering queue layers so that each individual object can be drawn on the background, on the front (as a user interface element) or in the middle of these two layers. Furthermore, the depth or order of each object inside each layer can also be specified.

Another design decision that resulted from the OpenGL abstraction was the incorporation of the window creation

process inside SketchyDynamics, thus reducing the effort on the developer's side. Moreover, SketchyDynamics delivers events received by the window, like mouse and keyboard inputs, to the application using the observer pattern, thus letting the developer take actions based on the user input.

### *B. User Interaction*

In order to best represent the pen-paper paradigm, the user interaction was designed to take advantage of systems with a touchscreen and stylus. Thus, the user only needs to press and move the stylus to interact with the system, without needing extra buttons[2]. Furthermore, no menus are used and most of the interaction is done by sliding the stylus across the screen. Although it was designed with that type of devices in mind, SketchyDynamics also works well with a traditional computer mouse.

There are two types of objects the user is able to create: bodies and joints. Bodies are rigid objects that are simulated according to physics laws while joints are used to connect bodies. Fig. 2 shows various bodies and three types of joints.

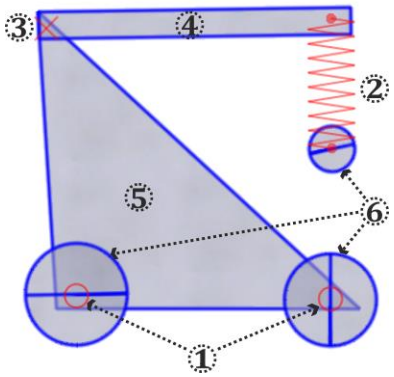


Fig. 2. Various types of joints and bodies: 1) revolute joints; 2) spring joint; 3) weld joint; 4) rectangular body; 5) triangular body; 6) circular bodies.

It is also important for the user to be able to manipulate the objects to a certain degree so SketchyDynamics lets the user change an object's position, scale, and orientation, or even delete it.

#### *1) Creating*

The creation of an object, be it a body or a joint, is done by drawing it. So, for example, if users want to create a rectangle body, they simply draw the rectangle on the screen. SketchyDynamics then recognizes the rectangle and its properties, like size and orientation, and creates the physical and visual representations of it.

SketchyDynamics supports four types of bodies: rectangles, triangles, circles and freeform bodies. When the user input is recognized as a rectangle, triangle or circle, it is represented in a beautified manner, as illustrated in Fig. 3. Otherwise, when the input is not recognized, it is interpreted as a freeform and represented in a simplified manner (with fewer vertices) for performance reasons.

[2] In a traditional mouse system this means that only the left mouse button is needed.

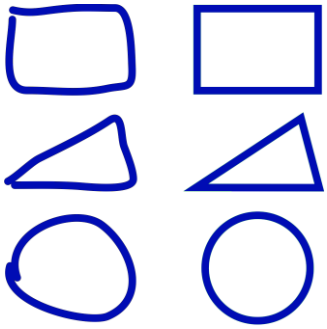

Fig. 3. Example of drawn shapes (left) and respective beautified representations (right).

The user can also connect two bodies with three different joint types: weld, revolute and spring. Weld joints connect two bodies at a specific anchor point, preventing any relative movement between them. Like weld joints, a revolute joint connects two overlapping bodies at a specific point but allows the bodies to rotate freely around that point. Spring joints try to keep a constant distance between two connected bodies, based on the distance at the time the joint was created, stretching and shrinking like a real spring.

Just like creating bodies, the creation of joints is done by drawing them. Drawing an alpha gesture over two bodies connects them with a weld joint with an anchor at the gesture's intersection, while drawing a small circle creates a revolute joint anchored at the circle's center. To create a spring joint, the user draws a zigzag gesture starting in one body and ending in another one, defining the two spring's anchor points as the start and end points of the gesture.

Regarding the visual representation of joints, the weld and revolute joints are represented by a small cross and by a small circle, respectively, on the joint anchor point while the spring joint is displayed as a zigzag line starting in one anchor point and ending on the other, stretching and shrinking subject to the distance between the bodies. The object presented in Fig. 2 was constructed using joints of the three types.

In order to better deal with the ambiguity in hand-drawn gestures, a guesses list is presented whenever the user executes a gesture. The guesses list shows all the available objects so that the user can choose an object other than the recognized one. The objects corresponding to gestures identified as matching by CALI recognizer appear bigger and first in the list, since they are the most probable choices, followed by the remaining objects. The guesses list feature can be disabled by the developer, in which case the most probable object is always selected.

Depending on the application-specific setup passed to SketchyDynamics, objects can be created while the physics simulation is in a paused state or while it is running and thus making other objects react instantly to the new object. This instantaneous simulation mode is useful for applications where the user interacts with a live environment as usually happen in games.

*2) Selecting*

For an object to be manually manipulated by the user, it first needs to be selected. When any object is selected the physics simulation is paused so that the user can easily edit it without being interrupted by other moving bodies. If the simulation was running before the selection of an object, it will resume after all objects are unselected.

Objects are selected by tapping on them with the stylus (or left-clicking them with a mouse), and can be deselected with the same action. This makes selecting multiple objects an intuitive process since users only need to keep tapping on the objects they want to select. It is also possible to unselect individual objects when there are multiple objects selected. When an object is selected, its lines assume a distinctive color, returning to the original color after being unselected. As shown in Fig. 4, this gives a clear feedback regarding the object's state. Also, tapping on an area of the screen with no objects or on an object configured as non-selectable, deselects all selected objects. Non-selectable objects are useful to create the application's scenery, which the user cannot manipulate but may be able to interact with, for example by connecting a user-made body to a scenery object.

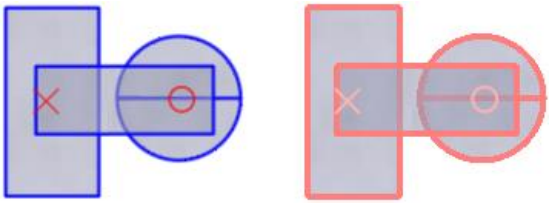

Fig. 4. Set of objects in unselected (left) and selected (right) states

When there are multiple bodies connected by joints and one of them is selected, all the other connected bodies are automatically selected, as long as they are selectable objects. This feature was introduced in order to improve the usability of the system, since we found that when multiple bodies are connected the user typically wants to manipulate them as a whole.

*3) Moving*

A selected body or joint can be moved by pressing over it and dragging the stylus. The object will move in sync with the stylus as long as the user keeps it pressed on the screen.

When there are multiple objects selected they all move in a synchronized manner, regardless of which object was pressed by the stylus.

*4) Scaling and Rotating*

Scaling and rotation of bodies is done simultaneously in a single action. As the action to move an object, scaling and rotation is done by pressing and dragging the stylus, but instead of pressing inside the selected body, the user needs to press outside it. As the user drags the stylus, the selected bodies scale and rotate based on the stylus initial and current positions. Only bodies can be rotated or scaled, so this operation is not applicable to joints.

The scale factor is calculated based on the current distance from the stylus position to the body center and the initial distance (before dragging the stylus). Regarding rotation, it is done based on the angle between two imaginary lines: the line from the current stylus position to the body's center, and the initial line (before dragging the stylus). Thus, moving the stylus closer or farther from the body scales it while moving the stylus around the body rotates it.

When multiple bodies are selected, they are all subject to the same rotation and scaling factor, but instead of using the body's center point as the reference point, the geometric average of all individual center points is used.

In order to aid the user during a scaling and rotation operation, SketchyDynamics displays a rectangle enclosing the selected objects, which rotates and scales along with them. Also, a small circle is displayed on the center reference point, along with a line connecting that point to the mouse cursor, so that the user can clearly perceive the operation being done. These visual cues are displayed in Fig. 5.

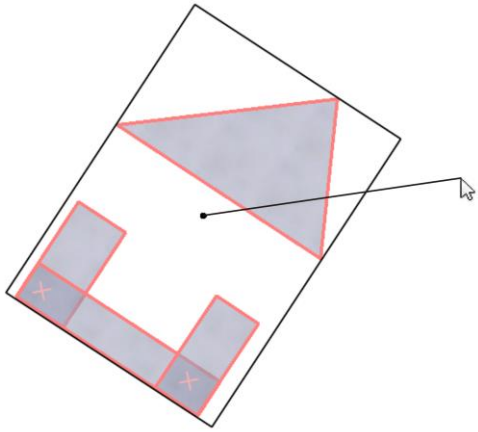

Fig. 5. Set of objects being subject to simultaneous rotation and scaling operations

*5) Removing*

Since removing objects is an important operation that contributes to user's creative freedom, it was designed to be simple, intuitive, and to have a low impact on the user's cognitive load. In fact, removing an object is a just special case of moving it.

When an object starts being moved by the user, a large rectangle with a trash bin icon slides down from the top of the screen, sliding back up and off-screen when the object cease to be moved. If the stylus enters the trash bin area while moving any object, the trash bin icon turns red. If the user lifts the stylus while on this rectangle, all the selected objects are removed. Fig. 6 shows the trash bin area in context of a simple, almost empty, application, and also the trash bin icon representations before and after the stylus drags an object onto it. We choose to keep this area hidden unless the user starts moving objects to improve the use of screen real estate, since objects can only be deleted when they are being moved by the user.

Joints can also be removed by simply being moved outside any of the two bodies they connect, without the need to move them to the trash bin rectangular area, although the trash bin works for joints too.

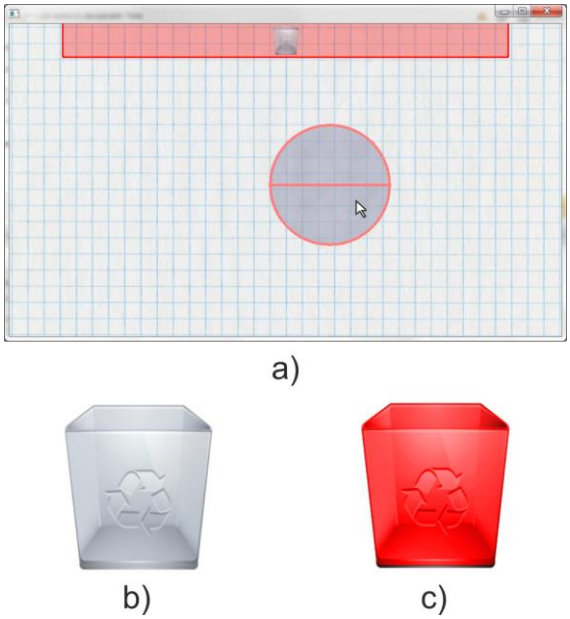


Fig. 6. a) simple application showing the trash bin area in context; b) trash bin icon in its normal state; c) trash bin icon when an object is dragged inside the trash area.

## V. USABILITY EVALUATION

In order to validate SketchyDynamics' features and also to better understand what needs improvement, we conducted a usability evaluation session that was attended by 8 subjects (2 females and 6 males), comprising students, teachers and researchers from the Computer Science field. During the session, participants experienced SketchyDynamics' functionalities using a traditional mouse but also using an interactive display with a stylus (Wacom Cintiq 15X).

Using a prototype application developed with SketchyDynamics, each subject performed an efficiency test by creating a complex scene[3], consisting of 17 bodies and 11 joints (Fig. 7). Before beginning the execution of the efficiency test, 5 subjects had a few minutes to experiment with the prototype. Also, during the test, the session coordinator clarified doubts raised by each of the 5 subjects. Regarding the remaining 3 subjects, they executed the test in a slightly different manner: they all done the test simultaneously, using only one computer; the experience was timed from the moment they had contact with the prototype; and had no help from the session coordinator. With this group we hope to evaluate the usability of SketchyDynamics when users are in a more adverse situation: for example, when they have no access to touchscreen and stylus, and/or have no time to get familiar with the application.

Considering the complexity of the scene to reproduce along with the inexperience of the subjects with the SketchyDynamics library prototype, the results of the efficiency tests are very encouraging. The first 5 subjects completed the test on an average of 9 minutes and 12 seconds, with a standard deviation of 3 minutes and 34 seconds.

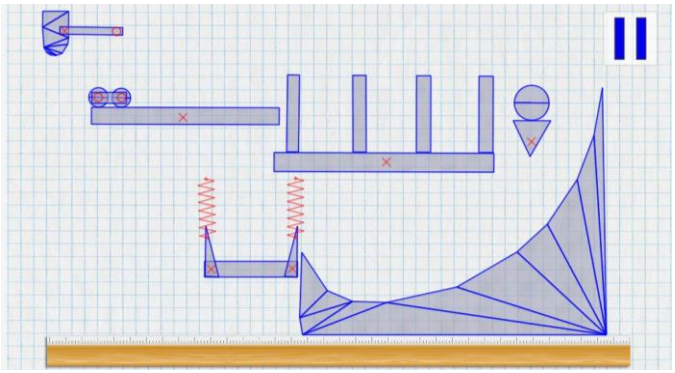

Fig. 7. Scene reproduced by subjects during the efficiency test (the ruler, at the bottom, along with the pause indicator, at the top-right corner, are part of the prototype and not user-made objects)

Regarding the remaining 3 subjects, who performed the test together, it took them about 24 minutes to complete the test, which we consider to be a positive result since these 24 minutes include the time they spent learning how to use the system and discovering its functionalities. Fig. 8 presents the time taken by each subject to complete the efficiency test. Note that since subjects 6, 7 and 8 executed the test together, their results are unified.

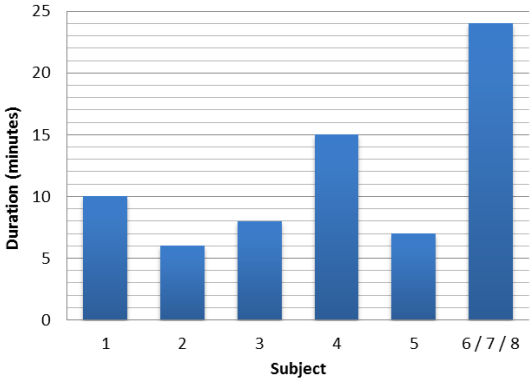


Fig. 8. Time spent per subject in the efficiency test

After the efficiency test, each subject filled out a survey form regarding their experience with the prototype. All the questions in the survey achieved average results above 1 point, in a scale from -3 (awful) to +3 (excellent), where 0 represents a neutral response, showing that SketchyDynamics pleased the users and is on the right track.

In order to know if the selected gestures were successful, one section in the survey asked about the suitability of each gesture in the creation process. As shown in Fig. 9, the average results for the majority of the gestures were equal or above 2 points, except for the gesture used to create weld joints. This lower result can be explained by the difficulty to draw an alpha gesture using a traditional computer mouse.

[3] A video demonstrating the creation of such scene can be found at http://youtu.be/1niigTt_m_I

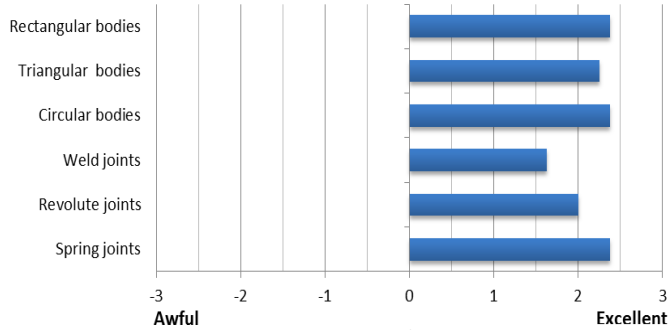


Fig. 9. Average results on the suitability of each gesture in the creation process

Regarding the object transformation process, we found the results to be very positive (Fig. 10), since the only action that achieved an average score lower than 2 points was the continuous selection of multiple objects. By observing the subjects during the interaction with the prototype, it was evident that the action to select multiple objects caused some trouble, since it conflicts with the usual experience users have with computer applications. While in most applications a click over an object selects it and deselects any other object that was previously selected, in SketchyDynamics clicking over an object selects it but does not deselects the remaining objects. As a result of this conflict, participants would misguidedly apply transformations on objects that they thought to be deselected. Despite that, the overall opinion of the participants in relation to the object transformation process was very good, with an average score greater than 2 points.

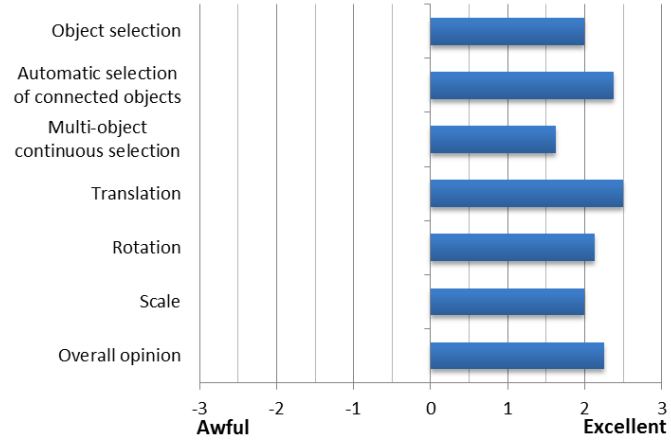


Fig. 10. Average results on the object transformation process

Although subjects found that it was useful to remove a joint by simply displacing it out of the bodies it connects, the results presented in Fig. 11, despite being very encouraging, show that there is still some room for improvement in regards to the object removal process. One of the criticisms mentioned by several subjects was the impossibility to remove and object by pressing the "Delete" key. In fact, this is a feature that is present in most computer applications for the operation of removing or deleting an object.

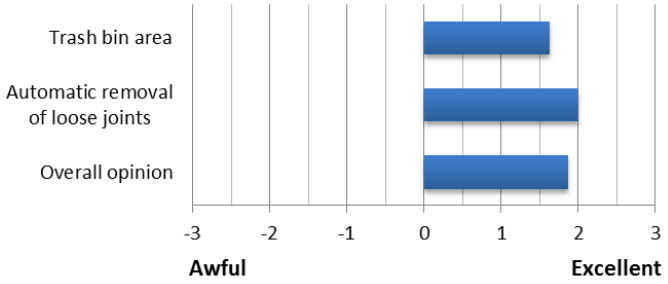


Fig. 11. Average results on the object removal process

Regarding the overall perception of SketchyDynamics, the results showed that subjects feel that it is easy to use and is also adequate for creating physically simulated scenes (Fig. 12). Concerning the stimulus, which achieved a lower result, certain participants demonstrated frustration when using the stylus, due to hardware problems. Also, some participants complained about the impossibility to undo operations. In relation to flexibility, participants have suggested that SketchyDynamics should support a larger number of object types.

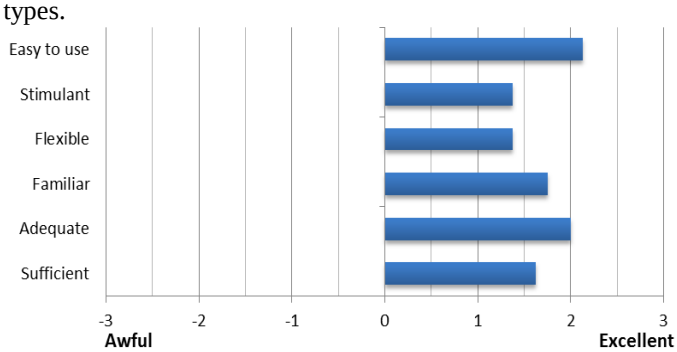


Fig. 12. Average overall results on SketchyDynamics' functionalities

In addition to these questions, the survey also inquired subjects about the interaction devices, the arrangement of the user interface, and also about the manipulation of the simulation. Further discussion on the usability evaluation and also on the SketchyDynamics library can be found on [15].

## VI. Conclusions

We have presented a library capable of speeding up the development of applications by providing developers a sketch-based interface combined with physics simulation. The library also provides facilities in managing the graphical side of the application and dealing with user input.

In an effort to make the library suitable for the widest range of applications we are working on adding more functionality into it, such as a new rope-like joint.

One useful feature would be the ability to select an individual body from a set of connected bodies and transform it using the joint anchor point as a reference. This poses some design problems since an object can have multiple joints (which one should be used?). The problem further increases if there is more than one selected object. Before implementation, further study on how to overcome these problems is needed.

Another interesting feature would be the existence of object hierarchies, in which transformations applied to one object are propagated onto its child objects, but not the opposite. The construction of this hierarchy could be based on the depth of the objects.

As noticed during the usability evaluation, implementing common functionalities such as clipboard to duplicate objects and undo/redo capabilities is extremely important to improve the system's usability and reduce user's frustration

Another requested feature is the ability to perform a scale or rotation operation individually. A possible and familiar solution would be the use of a modifier key to restrict the action to a single operation. Every time this key is pressed, the system could check if the mouse movement was mainly radial or tangential, doing only a scale or rotation operation, respectively. This concept could also be applied to restrict the movement of objects to horizontal, vertical and 45 degree translations.

Nevertheless, we think that current state of SketchyDynamics already enables it to be integrated and used to develop exciting applications.

## REFERENCES


[1] Rubine, D.: Specifying Gestures by Example. SIGGRAPH Computer Graphics, Volume 25 Issue 4, 329 -337 (1991)

[2] Plimmer, B., Freeman, I.: A toolkit approach to sketched diagram recognition. Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI.but not as we know it (BCS-HCI '07) 1, 205-213 (2007)

[3] Pereira, J., Branco, V., Jorge, J., Silva, N., Cardoso, T., Ferreira, F.: Cascading recognizers for ambiguous calligraphic interaction. Eurographics Workshop on Sketch-Based Interfaces and Modeling (2004)

[4] Stahovich, T.: Pen-based Interfaces for Engineering and Education. Sketch-based Interfaces and Modeling, 119-152 (2011)

[5] Fonseca, M., Pimentel, C., Jorge, J.: CALI: An online scribble recognizer for calligraphic interfaces. AAAI Spring Symposium on Sketch Understanding, 51-58 (2002)

[6] Wobbrock, J., Wilson, A., Li, Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. 20th annual ACM symposium on User interface software and technology (UIST '07), 159-168 (2007)

[7] Anthony, L., Wobbrock, J.: A lightweight multistroke recognizer for user interface prototypes. Graphics Interface 2010 (GI '10), 245-252 (2010)

[8] Lee, W., Kara, L., Stahovich, T.: An efficient graph-based recognizer for hand-drawn symbols. Computers & Graphics 31, 554-567 (2007)

[9] Vatavu, R.-D., Grisoni, L., Pentiuc, S.-G.: Gesture Recognition Based on Elastic Deformation Energies. Gesture-Based Human-Computer Interaction and Simulation 5085, 1-12. (2009)

[10] Sezgin, T., Davis, R.: HMM-based efficient sketch recognition. 10th international conference on Intelligent user interfaces (IUI '05), 281-283 (2005)

[11] Alvarado, C., Davis, R.: Resolving Ambiguities to Create a Natural Computer-Based Sketching. Proceedings of IJCAI-2001, 1365-1371 (2001)

[12] Kamel, H., Shonoda, M., Refeet, M., Nabil, R.: Free-Hand Sketch Recognition For Visualizing Interactive Physics. (Accessed 2012) Available at: http://code.google.com/p/sketch-recognition-simulation-tool

[13] Purho, P.: Crayon Physics Deluxe. (Accessed 2012) Available at: http://crayonphysics.com

[14] Costa, A., Pereira, J.: SketchTester: Analysis and Evaluation of Calligraphic Gesture Recognizers. 20º Encontro Português de Computação Gráfica (20ºEPCG) (2012)

[15] Costa, A.: SketchyDynamics: Apoio à Produção de Sistemas Baseados em Interfaces Caligráficas para a Simulação da Dinâmica de Corpos Rígidos (M.S. thesis). School of Engineering (ISEP), Polytechnic of Porto, Portugal (2012)



**Abílio Costa** received his MSc in Computer Science Engineering at the School of Engineering, Polytechnic of Porto (ISEP-IPP), in 2012, where he also accomplished his graduation. Currently working on the graphics and rendering engine of a new product at NDrive, he has experience in user interfaces and computer graphics.

**João P. Pereira** earned his BSc, MSc and PhD in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto (FEUP).
He is currently teaching at the Computer Science Department of the School of Engineering, Polytechnic of Porto (ISEP-IPP), and researching at the Knowledge Engineering and Decision Support Research Center (GECAD). His main areas of interest are Computer Graphics and Human-Computer Interaction.