Developing user-controlled animations with Actionscripts 3.0 to enhance student learning of engineering dynamics

Oai Ha & Ning Fang

Utah State University
Logan, Utah, United States of America

ABSTRACT: Computer animation (CA) is increasingly being employed in engineering education. The user-controlled animations allow users (students) to manipulate input variables and observe how the output numerical results change simultaneously and visually with animations. This article describes how ActionScript 3.0 was used to create user-controlled computer animations to improve student learning of complex concepts and spatially dependent motions in engineering dynamics, a foundational course in many engineering disciplines, such as mechanical, aerospace and civil engineering. Three representative computer animations are provided for demonstration purposes, including: a) general plane motion with a crank and slider mechanism; b) general plane motion and the principle of conservation of energy with a yoyo; and c) moment of inertia and the principle of conservation with inclined planes. These CA modules were implemented in a recent semester for students at Utah State University to learn rigid body dynamics. The results of a questionnaire survey administrated at the end of the semester are reported at the end of this article.

Keywords: User-controlled animations, ActionScript 3.0, engineering dynamics, visualisation

INTRODUCTION

Engineering Dynamics (ED) is a foundational course in many engineering disciplines, such as mechanical, aerospace, and civil engineering. This course requires students to have strong abstract thinking and visualisation skills to deal with complex concepts and spatially dependent motions. With the ever-increasing use of computers in education, many ED instructors have employed computer animation (CA) in teaching and learning. It has been reported that CA improved student learning outcomes, helped students gain interactive learning experiences, and increased student enjoyment and motivation to learn [1]. In order to make learning with animation more effective, it is important to provide students (i.e. users) with control over the animation and the pace of the animated learning content [2]. In CA modules, the parameters to control animation can be employed as input variables for the problems associated with animation. The user-controlled animations allow users to manipulate input variables and observe how the output numerical results change simultaneously and visually with animations.

AUTHORING TOOLS FOR DEVELOPING COMPUTER ANIMATION

As CA modules for engineering courses usually contain a mixture of text, mathematical equations, graphics, animation and video/audio content, the use of authoring tools in CA development has become more popular. An authoring tool is defined as any software or collection of software components that help developers write multimedia applications that are playable by most Web browser plug-ins, e.g. Java applet, Shockwave or Flash players [3]. These plug-ins are software components that add a new function to a Web browser to view a specific video file format. They are provided virtually free and are constantly updated in the most popular Web browsers, such as Internet Explorer, Firefox, Chrome and Safari. It was found that 75% of CA modules or programs made use of the availability of software plug-ins that were pre-installed (and constantly updated) in most popular Web browsers and provided wide accessibility to students. Recently, the trend of using free Web browser plug-ins to display CA modules has been increasing. According to a 2011 Millward Brown survey in mature markets, including the USA, Australia, New Zealand and several other countries, Flash player and Java plug-ins were available in 99% and 73%, of Internet-enabled PCs, respectively [1]. Adobe projects that the number of addressable devices with Flash technology will be over one billion by the end of 2015 [4].

Adobe Flash Professional is a popular authoring tool for creating vector graphics, animation and game applications in the form of Shockwave Flash (SWF) files that can be viewed in Flash player and common Web browsers. Animation can

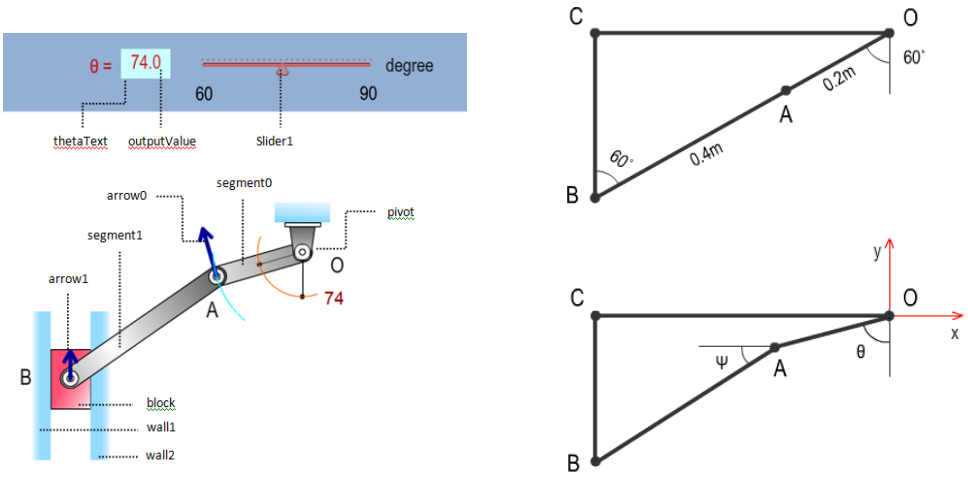
be created in a Flash Professional environment by setting up an animation's contents on the stage and manipulating them via layers and frames in the Timeline. This method is used by many developers and does not require ActionScript. ActionScript 3.0 (AS3) is an object-oriented programming language and can be used in a Flash Professional environment to create Flash application files or class files. AS3 can also be used in a development environment, such as Flex SDK or Flash Builder. The benefits of using ActionScript to develop CA modules include: a) allowing the developers to animate more complex motions; and b) animating with ActionScript gives developers certain flexibility in manipulating variables and provides users with some types of control over animation.

This article describes how ActionScript 3.0 was used to create user-controlled computer animations to improve student learning of complex concepts and spatially dependent motions in engineering dynamics. For demonstration purposes, three representative computer animations are provided, including: a) general plane motion with a crank and slider mechanism; b) general plane motion and the principle of conservation of energy with a yoyo; and c) moment of inertia and the principle of conservation with inclined planes. It should be noted that a detailed discussion of the instructional design and learning strategies associated with the use of CA is beyond the scope of this article.

GENERAL PLANE MOTION WITH A CRANK AND SLIDER MECHANISM

In most textbooks, general plane motion is broken into two separate processes and depicted by two static pictures: one for showing the translation of an object from the initial to the *pre-final* position, and the other for showing the rotation of the object from the *pre-final* to the final position. In reality, however, the translation and rotation take place simultaneously. Therefore, students are required to integrate the two motions in their minds and sometimes infer the combined motion of the object at the intermediate positions. These cognitive processes quickly overload a student's working memory capacity and reduce the cognitive resources for other learning tasks [5]. The computer animation of general plane motion gains a big advantage over static pictures by showing explicitly dynamic motions over time and space. Students are not required to infer and combine both translational and rotational motions in their minds.

Figure 1a shows the animation of a crank and slider mechanism. This mechanism is mainly used to convert rotary motion to a reciprocating motion, or vice versa. The mechanism consists of two links, OA (*segment0*) and AB (*segment1*) and a sliding block B. Link OA has only rotational motion, sliding block B has only translational motion and link OA has general plane motion. Link OA's angular position is labeled θ, and link AB's angular position is labeled ψ. The angular velocity of link OA is given as ω_{OA}. Students are asked to find the velocity of block B, called V_B, given the length of the two links, the angular velocity ω_{OA}, and an initial angular position θ of 60°.



a) b)

Figure 1: a) Components of a crank and slider mechanism; and b) its simplified diagram.

From Figure 1b, the following expressions for ω_{AB} and V_B can be derived:

$$\tan\Psi = (B_y - A_y)/(B_x - A_x) \quad (1)$$

$$\omega_{AB} = (0.6 * \cos\theta)/(-0.4 * \cos\Psi) \quad (2)$$

$$V_B = 0.6 * \sin\theta - \omega_{AB} * (-0.4 * \sin\Psi) \quad (3)$$

All components of this mechanism were created by using their corresponding classes. A class in object-oriented programming is a blueprint or template or set of instructions to build a specific type of object [6]. All objects built from a specific class share the same properties and behaviours. For example, the objects *segment0* and *segment1* were built from the class Segment; the object *arrow0* and *arrow1* were built from the class Arrows; the object *block* was built from

the class Block, and so on. Some of the classes used in this animation were adapted from Peters' works [7] and their original source codes can be downloaded from the publisher's Web site [8]. These *.as* files are not excutable, but indispensable during the compilation the main *fla* file.

On the stage, the authors put a scrollbar (*Slider1*) and a dynamic text box (*thetaText*) to display the scrollbar's current value (*outputValue*). This scrollbar is used to control the animation of the *segment0* (Figure 1a) starting from the original angle of 60° to its final position of 90° with respect to the *y* axis. Then, the authors used the *addChild()* method to add components of the mechanism to the display list according to the following chain of order: *pivot, segment0, segment1, block, wall1,* and *wall2*. For example, the following codes add *pivot, segment0* and *segment1* to the display list:

```
pivot = new Pivot(26,18,0x66CCFF);
addChild(pivot);
    pivot.x = anchorX;
    pivot.y = anchorY;
    segment0 = new Segment(segment0Length,segmentWidth, true,0x999999);
addChild(segment0);
    segment0.x = pivot.getPin().x;
    segment0.y = pivot.getPin().y;
    segment0.rotation = 150; // rotate 150 degree clocks-wise to align it to the angle of 60 degree from vertical
    segment1 = new Segment(segment1Length,segmentWidth, false,0x666666);
addChild(segment1);
    segment1.x = segment0.getPin().x;
    segment1.y = segment0.getPin().y;
    segment1.rotation = 150;
     arrow0 = new Arrows(0,0,50);
addChild(arrow0);
    arrow0.x = segment0.getPin().x;
    arrow0.y = segment0.getPin().y;
    arrow0.alpha = 0;
    arrow1 = new Arrows(0,0,50);
addChild(arrow1);
    arrow1.x = segment1.getPin().x;
    arrow1.y = segment1.getPin().y;
    arrow1.alpha = 0;
```

In the above code, the *Pivot()* function accepts three parameters: pivot width and pivot height in *Number* data type, and colour in *Unit* data type. The *Segment()* function needs four parameters: segment width and segment height in *Number* data type, centre line in *Boolean* data type to show the centre line of the segment (true) or not (false). The *Arrows()* function shows the vector *arrow0*, at the temporary point (0, 0) with the length of 50 pixels. The vector is also temporarily hidden (*arrow0.alpha = 0*) because the mechanism has not moved yet.

Then, its real location is determined by assigning its coordinates to the end point of *segment0* (*arrow0.x = segment0.getPin().x; arrow0.y = segment0.getPin().y*). Its real orientation will be changed according to the user's input. The code for this control is presented below in an IF statement.

The *anchorX* and *anchorY* were the coordinates where the authors wished to put the pivot. Once the pivot is added on the stage, the positions of other components in the chain depend on the preceding ones thanks to the *getPin()* method embedded in each class. For example, the coordinates of *segment0* depend on the pivot's location (*segment0.x = pivot.getPin().x*; and *segment0.y = pivot.getPin().y*) and the coordinates of *segment1* depend on the *segment0*'s location (*segment1.x = segment0.getPin().x*; and *segment1.y = segment0.getPin().y*). The animation was initiated whenever the users manipulate *Slider1*. An event handling function will *listen* to any move of *Slider1* and call another function *onChange()* which calculates the relative location of each component of the mechanism depending on the output value of *Slider1*. Details of the function *onChange()* are given below:

```
Slider1.addEventListener(Event.ENTER_FRAME,onChange);
function onChange(event:Event):void
{
    outputValue = Slider1.value;
    thetaText.text = outputValue.toFixed(1);
    Bx = -0.52;
    By= -Math.sqrt(0.16 -Math.pow((-0.52+0.2*Math.sin((outputValue)*Math.PI/180)),2))
            - 0.2*Math.cos((outputValue)*Math.PI/180);
    Ax = -0.2*Math.sin((outputValue)*Math.PI/180);
    Ay = -0.2*Math.cos((outputValue)*Math.PI/180);
```

```
    Chi = Math.atan((By - Ay)/(Bx-Ax));
    Rx = (-0.4 * Math.cos(Chi));
    Ry = (-0.4 * Math.sin(Chi));
    OmegaAB = (0.6*Math.cos((outputValue)*Math.PI/180))/Ry;
    Vb = 0.6*Math.sin((outputValue)*Math.PI/180) - OmegaAB*Rx;

    segment0.rotation = outputValue + 90;
    segment1.x = segment0.getPin().x;
    segment1.y = segment0.getPin().y;
    arrow0.x = segment0.getPin().x;
    arrow0.y = segment0.getPin().y;
    segment1.rotation = 180 - Chi * 180 / Math.PI;
    block.x = segment1.getPin().x;
    block.y = segment1.getPin().y;
    arrow1.x = segment1.getPin().x;
    arrow1.y = segment1.getPin().y;
    prevX = curX;
    prevY = curY;
    curX = segment0.getPin().x;
    curY = segment0.getPin().y;
}
```

An IF statement compares the current coordinates of point A with its coordinates in the previous time frame and determines the directions and magnitudes of velocity vectors. This IF statement is inserted in the *onChange()* function:

```
if ((prevY>curY)||(sliderValue==90))
    {
        arrow0.alpha = 1; // when the mechanism is in motion, the velocity vector is shown
        arrow0.rotation = outputValue -180;
        arrow1 = new Arrows(0,0,int(Vb*Vscale));//Vscale adjusts true value Vb to draw it on the stage
        addChild(arrow1);
        arrow1.x = segment1.getPin().x;
        arrow1.y = segment1.getPin().y;
        arrow1.rotation = -90;
    }
    else if (prevY < curY)
    {
        arrow0.alpha = 1;
        arrow0.rotation = outputValue;
        arrow1 = new Arrows(0,0,int(Vb*Vscale));
        addChild(arrow1);
        arrow1.alpha = 1;
        arrow1.x = segment1.getPin().x;
        arrow1.y = segment1.getPin().y;
        arrow1.rotation = 90;
        if (sliderValue == 60)
            {
            arrow1.alpha = 0;
            arrow0.alpha = 0;
            }
     }
```

In summary, in this animation, the developers and users can control the variable *outputValue* to move the short segment (*segment0*) and, then, control the motion of the whole mechanism.

GENERAL PLANE MOTION AND THE PRINCIPLE OF CONSERVATION OF ENERGY WITH A YOYO

The motion of a yoyo can be used to learn engineering dynamics problems involving general plane motion and the principle of conservation of energy. The animation requires two objects: string and yoyo (Figure 2a). To create the string object, the authors drew a vertical line and converted it into the movie clip symbol (*string_mc*). Then, they placed an instance of this symbol (also *string_mc*) on the stage. The registration point of the *string_mc* symbol was set at the bottom of the string (Figure 2b). They created the yoyo object by drawing a circle (the four coloured pies and its centre are optional) and, then, converted it into the movie clip symbol yoyo with the option box *Exported for ActionScript* checked. Its registration point was set at the centre of the symbol (Figure 2b). Motions of the string and yoyo objects are best animated by using the tween class. After putting the instance *string_mc* on the stage and adding the yoyo object to

the display list by using the *addChild()* method, the authors used the *new* operator to create a new instance of the tween class. Following is an example:

```
var myYoYo:YoYo = new YoYo ;
addChild(myYoYo):
var tweenY:Tween = new Tween (myYoYo,"y",Strong.easeOut, originalY, originalY + 270, 2,true);
var tweenRotation:Tween = new Tween(myYoYo,"rotation",Regular.easeOut,0, - myAngle, 2,true);
var string:Tween = new Tween (string_mc,"y", Strong.easeOut, string_mc.y, string_mc.y  + 270, 2, true);
```

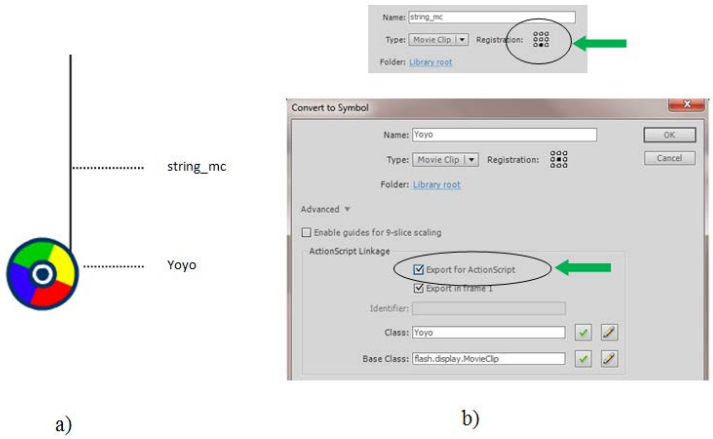


Figure 2: a) String and yoyo; and b) registration points and instance names of movie clip symbols.

The above tween commands come with several parameters. The first parameter is the object to be animated (*myYoYo*). The second parameter is the name of the property, which will be animated. The properties of a display object can be *x, y, alpha, width, height, xscale, yscale, rotation*, etc. The third parameter is the easing effect, which will determine how the tween animation will run. The Adobe *fl.transitions.easing* class lists 19 easing effects used to mimic the motions of an object moving with a variable acceleration [9]. The *Elastic.easeOut* and *Strong.easeOut* parameters render motions that are very close to the motion of a yoyo in action. The fourth and fifth parameters are the starting and ending position of the yoyo. They must be specified as numbers. The sixth parameter (number 2) is the duration of the tween. And lastly, the last parameter specifies the unit of the fifth parameter as the second (true) or frame (false). In this case, the duration of the tween is two seconds. For the rotation of the yoyo object, the fourth and the fifth parameters are starting (0°) and ending angles (*myAngle*) in degree. A negative value indicates that the rotation is counter clockwise.

In summary, in this animation, the developers and users can control the following variables to change the ways the yo-yo run: *height* and *width* (or the radius of yo-yo).

MOMENT OF INERTIA AND THE PRINCIPLE OF CONSERVATION OF ENERGY WITH INCLINED PLANES

In this section, the authors describe the animation of two objects rolling down without slipping, on two inclined planes. The problem is set up as follows. The spring is initially un-stretched and connected to objects A and B. Object A is a thin hoop and object B is a solid cylinder (Figure 3a). The spring constant k = 10 N/m. Both objects have the same radius r = 0.05 m, the same mass m = 2 kg, and are released from the same height h = 0.4 m. The question is to determine the speeds of the mass centres of objects A and B when they reach the bottom.

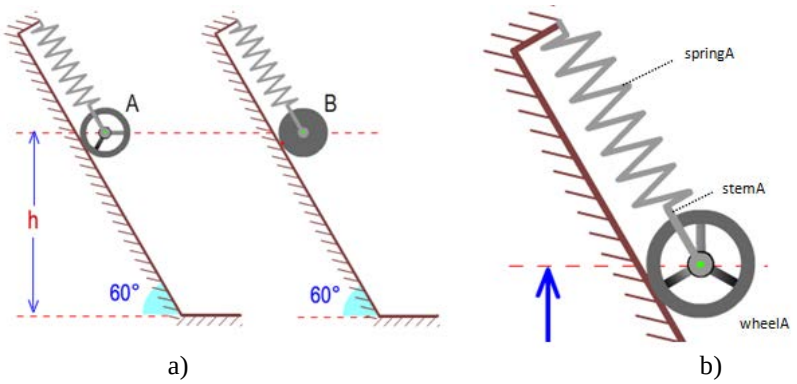


Figure 3: a) Thin hoop and solid cylinder on inclined planes; and b) components to be animated.

On the stage, the authors set up two systems of an inclined plane problem. Each system consists of a spring, a stem, and a wheel, which are converted to movie clip symbols to facilitate the animation (Figure 3b). Tween motions are applied

to each component. For instance, the tween motion of the wheel A includes translational tweens on *x* and *y* directions and a rotational tween for the rotation of the wheel. Following are the codes:

```
var tweenWheelAx:Tween = new Tween(wheelA,"x",Regular.easeOut,startAx,finishAx,timeA,true);
var tweenWheelAy:Tween = new Tween(wheelA,"y",Regular.easeOut, startAy,finishAy,timeA,true);
var tweenWheelARot:Tween = new Tween(wheelA,"rotation",Regular.easeOut,0,450,timeA,true);
var tweenStemAx:Tween = new Tween(stemA,"x",Regular.easeOut,671,bottomX,timeA,true);
var tweenStemAy:Tween = new Tween(stemA,"y",Regular.easeOut,300,bottomY,timeA,true);
var tweenSpringAScaleY:Tween = new Tween(springA,"scaleY",Regular.easeOut,1,2.853,timeA,true);
```

As with the case of yoyo animation, the tween commands use the third parameter to determine how the objects move. The authors used the *Regular.easeOut* parameter to mimic the spring-like action. The sixth parameter *timeA* is the duration of the tween. *TimeA* can be a true time for wheel A travelling on the inclined plane given that its translational velocity, its diameter, and the travel distance are known. However, for the purpose of illustration, *timeA* can be any number, which is large enough, so that the motion is discernible by learners. Remember, the use of a mathematical model can be ignored for many animations. In some cases, the value of *timeA* can be artificially exaggerated and controlled by users via dials, buttons, input fields or scrollbars to make the motion to become much slower and facilitate student learning.

Note that, with the given facts in the problem statement, the velocity of wheel B (Equation (5)) is always approximately 1.15 times faster than that of wheel A (Equation (4)).

$$v_A = \sqrt{g(h-r) - \frac{2k(h-r)^2}{3m}} \tag{4}$$

$$v_B = \sqrt{\frac{4}{3}\left[g(h-r) - \frac{2k(h-r)^2}{3m}\right]} \tag{5}$$

From the facts about wheel A's diameter and the length of its trajectory on the stage, the authors determined the total rotation angle of wheel A, which was from 0° to 450° in our case. The last parameter is set as *true* because the authors use the second as the unit for *timeA*. The springs A and B are two instances of the movie clip symbol *coil* that was previously created and put in the library. On the stage, the two instances are rotated 30° on a vertical axis, so that they can lay down on inclined planes. To animate the stretching of springs in 2D, the authors apply tween motion for *scaleY* to make the spring longer in the Y direction. This technique keeps the overall shape and the width of the spring in the *x* direction intact. The only disadvantage of this technique is that the line stroke of the spring wire looks bolder when it is stretched. However, with a small expansion of *scaleY*, the distortion can be unnoticeable.

In summary, in this animation, the developers and users can control the following variables to change the ways two objects run down an inclined plane: the radius and mass of the objects, the height of the inclined plane, and the time (*timeA*) for the objects moving down the ramp.

STUDENT FEEDBACK

During a recent semester, the authors developed and implemented a set of CA modules for students to learn rigid body dynamics. These CA modules also included the ones that employed user-controlled animations described in this article. The students were exposed to each CA module after they finished the lesson from regular in-class instruction on the same topic introduced in the CA modules. At the end of the semester, survey questions were administrated as shown in Table 1. These questions used a five-point Likert scale, ranging from *1 = strongly disagree* to *5 = strongly agree*. Except for student responses connecting the usage of CA modules and students' motivation, most students indicated that the CA modules increased 1) their confidence for learning engineering dynamics (53%); 2) their conceptual understanding of rigid body dynamics (59.4%); and 3) their procedural skills for solving rigid body dynamics problems (55%).

Table 1: The results of student surveys.

Survey questions	Student responses (n = 54 - 69)
Q1: Do you agree with the statement: *Overall, these modules increase my confidence for learning engineering dynamics?*	53% agree or strongly agree Median (*IQR*) = 4 (3,4); n = 68
Q2: Do you agree with the statement: *Overall, these modules increase my motivation for learning engineering dynamics?*	38.2% agree or strongly agree Median (*IQR*) = 3 (2,4); n = 68
Q3: Do you agree with the statement: *Overall, these modules increase my conceptual understanding of rigid-body dynamics problems?*	59.4% agree or strongly agree Median (*IQR*) = 4 (3-4); n = 54
Q4: Do you agree with the statement: *Overall, these modules increase my procedural skills of solving rigid-body dynamics problems?*	55% agree or strongly agree Median (*IQR*) = 4 (3-4); n = 69

Note: IQR = Interquartile range

Representative students' comments are provided as follows:

- *I really struggled with learning these sections in class, and the simulations clearly showed how these concepts were applied. I liked the simulations because they illustrated the changes in instantaneous centre and the velocity which were a couple concepts that I had a hard time with.*
- *They were easy to follow and learn from. I liked how you could change certain factors in the problem like mass, or radius and to see how those factors would change the outcome of the problem.*
- *Being able to visualize what was going on in the problem and gaining a visual of what I was solving for was incredibly helpful in increasing my understanding of the concepts behind each problem.*
- *I am able to better see the relations of variables to each other to which helps me see the big picture.*
- *The ability to change parameters like mass and velocity helped me understand how mass moment of inertia changed how the rigid body reacted.*

CONCLUSIONS

Along with the ever-increasing use of computers in education, there is a growing trend of using computer animation to improve student visualisation and understanding of abstract concepts and complex motion. In engineering education, computer animation provides students with external and dynamic visualisations, which is a big advantage over text and static graphics. While most computer animations can be created in the Flash Professional environment, creating animations with AS3 provides instructional developers with the flexibility in manipulating variables, gives users control over animation and make the learning with animation more effective.

ACKNOWLEDGEMENT

This material is based upon work supported by the US National Science Foundation under Grant No. DUE 1122654.

REFERENCES

1. Ha, O. and Fang, N., Computer simulation and animation in engineering mechanics: a critical review and analysis. *Proc. American Society for Engng. Educ. Annual Conf. & Exposition*, Atlanta, GA (2012).
2. Ayres, P., Kalyuga, S., Marcus, N. and Sweller, J., The conditions under which instructional animation may be effective. *Proc. Inter. Workshop and Mini-conf.*, Open University of the Netherlands (2005).
3. World Wide Web Consortium, Authoring Tools, Social Media (2015). 11 June 2014, http://www.w3.org/standards/agents/authoring
4. Adobe, Statistics (2015), 14 June 2015, http://www.adobe.com/products/flashruntimes/statistics.html
5. Kozhevnikov, M., Motes, M.A. and Hegarty, M., Spatial visualization in physics problem solving. *Cognitive Science*, 31, **4**, 549-579 (2007).
6. Yaiser, M., Object-oriented programming concepts: objects and classes (2011), 14 June 2015, http://www.adobe.com/devnet/actionscript/learning/oop-concepts/objects-and-classes.html.
7. Peters, K., *Foundation Actionscript 3.0 Animation: Making Things Move!* California: Apress (2007).
8. Peters, K., Source code/Downloads (2007), 4 March 2014, http://www.apress.com/9781590595183.
9. Adobe, Package fl.transitions.easing (2014), 11 June 2015, http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/fl/transitions/easing/package-detail.html.

BIOGRAPHIES

Oai Ha has a PhD in Engineering Education from Utah State University, an MS in mechanical engineering from California Polytechnic State University at San Luis Obispo, and a BS in mechanical engineering from University of Technology in Ho Chi Minh City, Vietnam. His research interests include mechatronics, automation control, virtual reality, spatial visualisation abilities, and cognitive processes in engineering design and problem solving.

Ning Fang is a Professor in the Department of Engineering Education in the College of Engineering at Utah State University, USA. He has taught a variety of courses at both graduate and undergraduate levels, such as engineering dynamics, metal machining, and design for manufacturing. His areas of interest include computer-assisted instructional technology, curricular reform in engineering education, the modelling and optimisation of manufacturing processes, and lean product design. He earned his PhD, MS and BS degrees in mechanical engineering and is the author of more than 60 technical papers published in refereed international journals and conference proceedings.