Session 2420

Animations for Introductory Courses

Dr. Michael R. Gallis,
Dr. Ping Wang
Penn State Capital College, Schuylkill Campus
200 University Drive
Schuylkill Haven, PA 17972

Introduction

There are many concepts and ideas in the introductory Engineering curriculum that are difficult to convey with static diagrams. The depiction of processes, dynamic phenomena or geometric relations which depend upon a single parameter can all be enhanced with the use of computer animations. Animations can be a valuable aid to visual learners and can even provide a limited degree of interactivity. The development of computer animations (especially 3-D animations) has generally been considered expensive computationally as well as fiscally. However, with the advent of the modern PC, the computational power to render sophisticated graphics in reasonable time is readily available to most instructors. The multitude of collaborative projects on the Internet has provided an array of low- and no-cost programs that help make animation development accessible to all. *POV-Ray* is a freely available 3-D rendering package, an Internet collaborative project having over 10 years of code maturity.[1] The focus of this paper is the use of *POV-Ray* to create short animations.

A series of sample animations are presented in the next section which were created by the authors for various physics and math courses covering topics such as calculus, mechanics and electromagnetism.[2] Some of the features of *POV-Ray* that make it a natural choice for the creation of such animations are discussed. In the third section, sample code is used to illustrate how *POV-Ray*'s S*cene Description Language* is used to create the frames of animation. In the fourth section some strategies for implementing the sometimes sophisticated mathematics required for the modeling are discussed. In the fifth section some of the costs and benefits of developing the animations are discussed. Finally, some of the uses of animations as a tool in the classroom as well as in web-based ancillary materials are presented, and the prospects for student created animations as auxiliary projects are explored.

Sample Animations

The usefulness of 3-D animations in the basic engineering curriculum can best be illustrated with a few examples. These particular animations have been chosen not just because of their

curricular interest, but also to highlight some of some of the features of the *POV-Ray* rendering package.

The first example portrays the mechanics of the conical pendulum (Figure 1). Students often fail to see the need for a net force when looking at a simple diagram, particularly if they've recently had Statics. The animation helps make clear that the simple diagram is a snapshot chosen for an instant when the geometry is readily analyzed. By choosing the frames appropriately and allowing the animation to loop, the cyclical nature of the motion is easily portrayed. The controls for the media allow a small level of interactivity so that the user can drag the position bar to any frame of the animation. In this particular illustration, the Mozilla browser has been used with the Apple Quicktime plug-in. The authors have also had much success using Internet Explorer with Microsoft's Media Player plug-in. This particular animation is relatively simple, being constructed of sphere, cylinder, cone, torus and text objects from the *POV-Ray Scene Description Language*. The source code for the conical pendulum animation will be used in the following section to illustrate some of *POV-Ray*'s easy-to-use features.

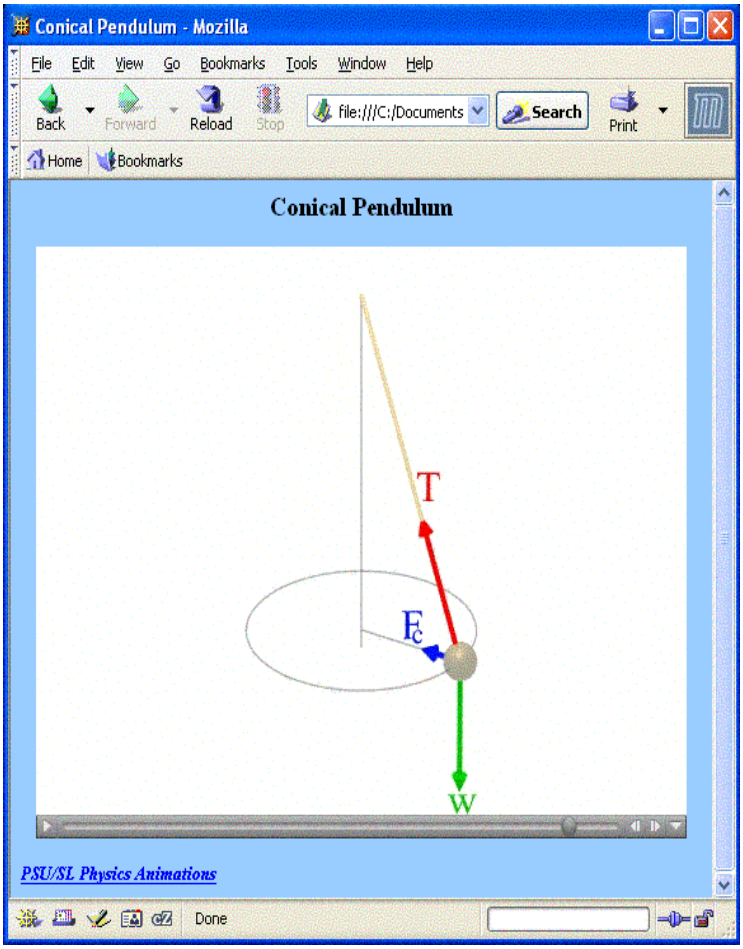

Figure 1: Conical Pendulum Animation Embedded in a Web Page

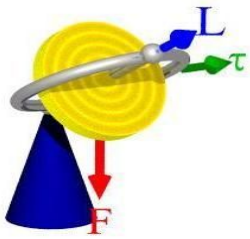

Figure 2: Gyroscope Precession

The second example (Figure 2) portrays the precession of a toy gyroscope. The gyroscope is a more complex object than the conical pendulum, but still uses the simple geometric shapes built into *POV-Ray*. No attempt was made to include the actual rapid rotation of the gyroscope's disk since the frame rate used would not be able to capture such motion. Instead, a circular texture was used for the disk to help provide appropriate visual clues.

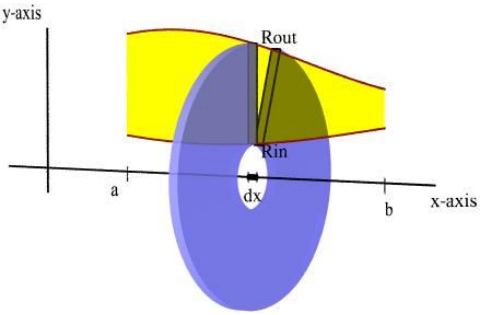

Figure 3: Volume of Rotation

The third example is from calculus and portrays the process for using the washer method for a volume of rotation. The first part of the animation shows a sweeping out of the infinitesimal washer whose radii are determined by the two bounding functions (shown in Figure 3), which provides a visual motivation for the integrand. The second shows the accumulating volume as many such washers are added together. This makes use of transparent pigmentation, which allows the details of one geometric aspect to be seen through foreground objects. Parametric curves describing the bounding functions are created with a series of short cylinders. The bounded area between the two functions are created with the polygon object.

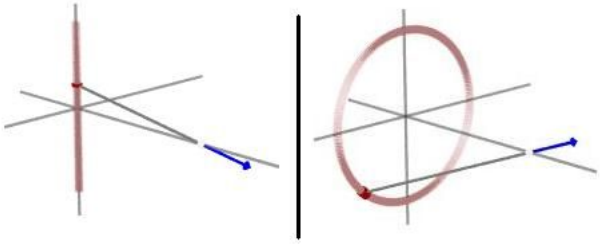

Figure 4: Electric Fields from Continuous Distributions

The next pair of animations (sample frames are depicted in Figure 4) are from the Electromagnetism curriculum. The animations illustrate the net electric field for two continuous

charge distributions: a line of charge and a ring of charge. The animations show the geometry of the contributions to the electric field at a particular field point due to each piece of charge, and then shows the result as the net electric field is accumulated over all contributions. By using the media controls, the viewer can interactively explore the symmetry and geometric details for these charge configurations. The *POV-Ray* code for these animations was designed for the general case of any curved line of charge; the parametric description of the curve is easily modified and the net electric field is evaluated numerically within the code. A fair amount of numerical calculations can be done within the *Scene Description Language*; however, users should be aware of limitations on accuracy and the potential cost in calculation and rendering times (in terms of numerical calculations, *POV-Ray*'s speed and accuracy might be comparable to a simple BASIC interpreter).

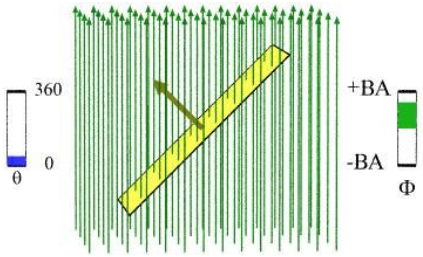

Figure 5: Magnetic Flux

Figure 5 shows an animation also from the Electromagnetism curriculum, and illustrates some of the geometrical ingredients in calculating magnetic flux. The slide control of the media player allows the viewer to interactively manipulate the orientation of the area. This example also demonstrates one of the powerful features of *POV-Ray*: the electric field lines were iteratively placed instances of a single composite object representing a field line.

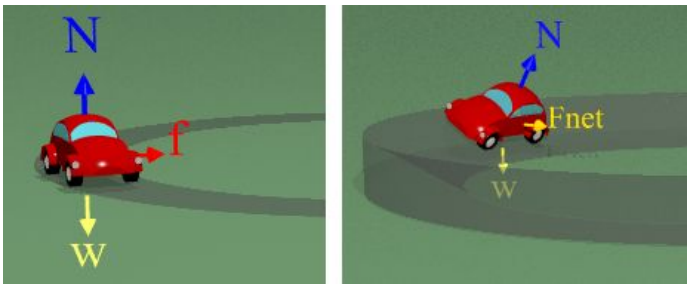

Figure 6: Mechanics of an Automobile on a Curved Road

The next pair of animations are from Mechanics, and illustrate the forces on an automobile during a turn. Figure 6 shows frames from two separate animations; the first shows the role of friction in an unbanked curve while the second shows the role of the normal force of the road on a frictionless banked curve. With the exception of the automobile, the animations are relatively simple. Objects as complex as the automobile do take an additional investment of time, but can readily be recycled. In this case, the object's source code has been placed in a separate include file. As shown in Figure 7, the automobile itself is comprised of separately textured and pigmented objects created from *POV-Ray*'s *Constructive Solid Geometry* (CSG) operations

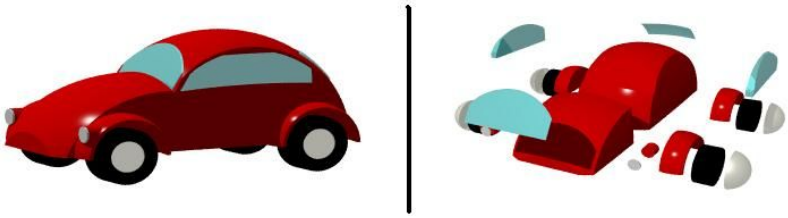

Figure 7: Automobile Parts

(union, intersection, difference, and merge). There are several shareware and freeware modeling programs to facilitate the creation of complex objects for *POV-Ray*. The *POV-Ray* web site contains links to numerous resources.

Another pair of animations which use the automobile object illustrate the requirements for a car to coast through a vertical circle, as well as the consequences for the vehicle when it does not start with sufficient initial speed (shown in Figure 8). While it would be possible to model the motion and perform the relevant numerical calculations within *POV-Ray*, it was expedient to use another program to actually calculate the position and orientation of the object as a function of time. For this purpose the 2-D simulation program *Interactive Physics*[3] was used for physical modeling. The relevant data was exported to a text file which was later read by *POV-Ray* during the rendering process. This technique also allowed the additional simulation of some nontrivial mechanics; the automobile actually bounces and flips after it hits the ground, a type of motion easily modeled with the Interactive Physics program. While this "grand finale" is not essential to the points the animation is intended to illustrate, it does provide some drama which students seem to appreciate.

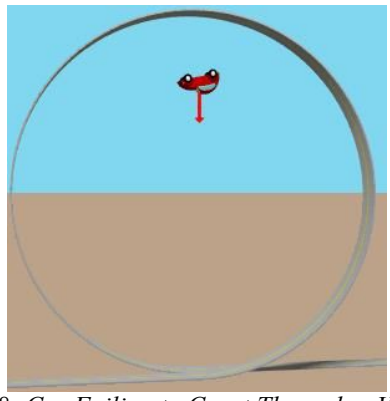

Figure 8: Car Failing to Coast Through a Vertical Loop

Building Animations

The production of an animation takes several stages: the writing and debugging of the source files, the rendering of the animation frames, the creation of the media file containing the animation and finally the playback of the animation. A basic illustration of the process is provided in Figure 9, although there will often be test rendering of selected animation frames as the code develops. For the remainder of this section sample code from the conical pendulum

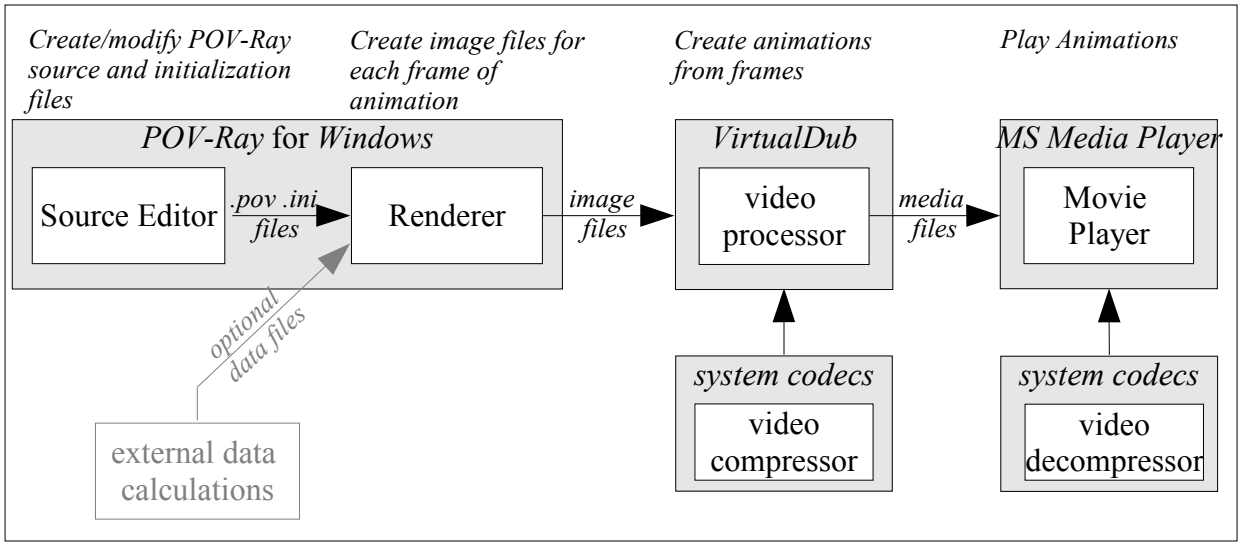

Figure 9: Schematic Diagram of Software Used to Create Animations

example will be used to illustrate the construction process for animation.

POV-Ray comes with a built-in editor which provides many of the desirable features expected from a programing text editor. Much of the code can easily be created by selecting items from drop-down menus and modifying the resulting code by hand. Most of the examples begin with the following:

```
// ==== Standard POV-Ray Includes ====
#include "colors.inc"    // Standard Color definitions
#include "textures.inc"          // Standard Texture definitions
#include "functions.inc"  // internal functions usable in user
#include "metals.inc"
global_settings{
      assumed_gamma 1
      ambient_light rgb <1, 1, 1>
}
background  {<1,1,1>}
#declare lclock=clock+ .0;          //local clock variable
```

The first three include files can be obtained from within *POV-Ray*'s editor by selecting Insert→Header→Standard Includes. The additional file "metals.inc" provides a selection of metallic finishes. Two parameters are set within the global settings statement. The assumed gamma correction of 1 seems best suited for most animations. The ambient_light parameter specifies that there is effectively uniform light coming from all directions. In this case, the color is specified by the red-green-blue (rgb) color vector to be white. This type of lighting scheme is convenient, so that it is not necessary to specify individual light sources for the scene to be rendered. The background is also specified to be white, tantamount to having a plain white "sky". Finally, a local clock variable is declared, which in this case is offset from the *POV-Ray*

clock variable by zero. When rendering single frames, the system clock variable is zero, so the offset can be modified for test rendering of selected frames by changing the offset for the lclock variable.

The next segment of code sets up some of the "physical" parameters for the animation, where the origin is at the center of the "orbit" of the pendulum:

```
#declare pendulum_angle=lclock*360;     //where in the orbit the pendulum is now
#declare stringlength = 50;
#declare dangle = 20;                   //angle from vertical of pendulum string
#declare pendulum_radius=stringlength*sin(radians(dangle));  //radius of pendulum "orbit"
#declare cone_height=stringlength*cos(radians(dangle));      //height of cone
#declare ball_radius=.05*stringlength;
#declare x_ball= pendulum_radius*cos(radians(pendulum_angle));
#declare z_ball= pendulum_radius*sin(radians(pendulum_angle));
#declare ball_pos=x_ball*x+z_ball*z;
```

Note how the ball's radius is specified in terms of the string length. It is useful to set the dimensions of objects relative to each other so that tweaking the scale of one object allows other objects to scale along without extensive parameter revision. The last line specifies the position of the pendulum *as a vector*. The system variables *x*, *y* and *z* are unit vectors (i.e. *x* = <1,0,0> in *POV-Ray*'s vector notation). It is also important to remember that *POV-Ray* uses a left-handed coordinate system, and that built-in vector functions can yield surprises if this fact is forgotten. The easiest way to visualize *POV-Ray*'s coordinate system is: *x* is positive towards the right of the screen, *y* is positive upwards and *z* is positive into the screen.

The pendulum and the chain are declared simply by:

```
sphere{0, ball_radius
      translate ball_pos
      texture{T_Brass_3B  }
 }
cylinder{cone_height*y,ball_pos,.125*ball_radius
      texture {T_Gold_1B }
}
```

The chain is a cylinder which extends from the top of the cone down to the current position of the pendulum. The brass and gold textures are defined in the "metals.inc" include file. The lines in the animation that illustrate the geometry are created with the following code:

```
//geometry
union{
      torus { pendulum_radius, .05*ball_radius
```

```
        }
        cylinder{
            -ball_radius*y,cone_height*y,.05*ball_radius
        }
        cylinder{
            ball_pos,0*y,.05*ball_radius
        }
        pigment{ color rgbt <.1,.1,.1,.6> }
        finish {ambient 1 } no_shadow no_reflection
    }
```

The objects are joined with a CSG union operation, which allows a single specification of the texture for all the components. The color is specified by a 4-component vector specifying the rgb color components along with a transparency level. The finish has only one specification setting the level of reflected ambient light to 1, which in turn allows the ambient_light specification to provide maximal lighting (and eliminate the need to add additional "light sources" for most objects). The no_shadow and no_reflection prevent artifacts that aren't part of the physical situation from appearing in the rendered image (for example, reflections of elements intended to mark up and label the illustration should not show up in shiny surfaces such as the metallic pendulum).

The vector indicating the tension in the pendulum chain and the accompanying symbols are created by:

```
    //vectors
    #declare vec_length = .4*stringlength;
    #declare head_length = .05*stringlength;
    #declare vec_radius = .15*ball_radius;
    #declare head_radius =3*vec_radius;
    #declare vdir=cone_height*y-ball_pos;
    #declare vdir=vdir/vlength(vdir); //unit vector from ball to fixed end of string
    union{
        cylinder{0,(vec_length-head_length)*vdir,vec_radius}
        cone{(vec_length-head_length)*vdir,head_radius,vec_length*vdir,0}
        text {
            ttf "Times" "T" .005, 0
            scale 6
            translate 1.1*vec_length*vdir
        }
        translate ball_pos
        pigment{ color rgb <1,0,0> }
        finish { ambient 1 } no_shadow no_reflection
    }
```

The combination of cylinder and cone create the vector, and the label "T" is created as a text object (any *TrueType* font may be used). Similar code is used to create the vectors and labellings for weight and net centripetal force.

The final important parts of the scene's code is

```
camera {
   location 3*<0, stringlength*3 ,-6*stringlength>
   look_at  <0, cone_height*.3, 0>
   angle 5.5
}
light_source { <0,+80,-40> color White}
```

The camera specifies the characteristics by which the scene is "viewed". The location specifies the apparent coordinates of the observer, generally chosen to be at a distance that is large compared to the dimensions of the object(s) being viewed to prevent apparent visual distortion of the object(s) when rendered. The look_at parameter specifies the focal point (as a vector) of the rendered scene, and is used to center the scene in the rendered image. The angle parameter specifies the field of view of the rendered image, chosen so that the scene fills the rendered frame. An additional point light source was included for this animation to help give the pendulum its metallic sheen.

The scene file in and of itself will only produce a single frame when run from the *POV-Ray* console. A relatively easy way to set the animation parameters is with an initialization file, which can be run from *POV-Ray* as if it were a scene description file. The conical pendulum initialization (.ini) file, which has been adapted from the *POV-Ray* sample file, contains the following:

```
Antialias=on
Antialias_Threshold=0.2
Antialias_Depth=3
Input_File_Name=conical_pendulum_rebuilt.pov
Initial_Frame=1
Final_Frame=80
Initial_Clock=0
Final_Clock=1
Cyclic_Animation=on
Pause_when_Done=off
```

The anti-aliasing parameters help prevent straight lines from rendering with a "staircase" or jagged appearance, which generally improves the overall appearance of the rendered image. The Input_File_Name identifies the source file to be rendered for the animation, and the output file

name can be specified in a similar fashion (the animation frame number is appended to the file name automatically as well as the appropriate file extension). The next four lines are used to specify the animation parameters with a great deal of flexibility. The Cyclic_Animation parameter automatically adjusts the clock so that the resulting animation will loop smoothly. Alternatively, one could specify an "extra" frame for the final frame (which would be identical to the first frame in a complete cycle) and then simply discard that final frame when combining the frames for the animation.

POV-Ray does not actually create a finished animation; it creates separate files of high quality images for each frame. Thus additional programs are required to complete the creation of the animation by "stitching" the image frames together and compressing the results in a single animation file of reasonable size. The authors have had a fair amount of success creating Audio Video Interleave (.avi) files with the free program *VirtualDub*,[4] which can make use of any video compressor (codec) resident on the computer used to create the animation file. The decompresser codec must be also be present on any computer used to view the animation. Thus the availability of codecs to both the animation creator and the intended audience plays a large role in selecting which codec to use for a particular animation. The quality and size of the resulting animation will also play a role in selecting an appropriate codec. In order to make the animations as portable as possible, the authors have extensively used the somewhat dated *Cinepak*[5] compressor from *Radius*, which seems to be included with most versions of Microsoft Windows. In the absence of compression, a several-second animation can end up being tens of megabytes in size, while the compressed version can be as small as a fraction of a megabyte (although a few megabytes are perhaps more typical for most of the animations produced by the authors).

Strategies for Implementing Model Calculations in *POV-Ray*

The modeling calculations in the sample code for the conical pendulum were extremely simple: a straightforward calculation of geometrical aspects with some time-dependent rotation at a fixed angular velocity. The sophistication of required calculations can vary quite a bit depending upon what is to be portrayed in the animation. Even very complex calculations can be completed using *POV-Ray's Scene Description Language*. As an extreme example, the authors have implemented the mathematics of differential geometry in two dimensions using indexed arrays of functions, numerical estimation of derivatives and numerical evaluation of integrals in order to illustrate properties of geodesics in various curved geometries. Scratch text files can be used to read and write temporary states of dynamics to aid frame-to-frame calculations.

Clearly, much can be accomplished within the framework of *POV-Ray*; however, there are many reasons one might consider doing some or all of the calculation externally and then importing the results. The creation of the animation of the automobile failing to coast through a vertical loop provides an example where it was easier to evaluate dynamical calculations using software more suited (and, at the time, more familiar to the authors) for mechanical simulations. *POV-Ray*'s ability to read text files allows a great deal of flexibility in using external applications to perform mathematical calculations.

In some cases, the authors have found it necessary to import *parts* of modeling calculations. In order to create animations illustrating the vibrational modes of circular drum heads, it was necessary to use Bessel functions (not implemented in *POV-Ray*) in conjunction with *POV-Ray'*s isosurface object. To accomplish this, *Mathematica*[6] was used to create a formatted tabulated set of values for several Bessel functions, which was then copied into a source file as part of a definition for a *POV-Ray* spline definition. The smooth interpolation of the spline allowed the depiction of the surface vibration modes.

There is often a conflict between the purists' desire to "get it right" and the pragmatists' desire to "get it done". When creating the code to describe a particular system, there may be trade-offs given time constraints versus the desire to be accurate or to create generalized code for possible recycling in future projects. Sometimes this results in the need to "cheat". As an example, one animation depicted a ball rolling without slipping down a straight ramp into a brief curved section and finally rolling onto a horizontal surface. Modeling the constant acceleration down the straight ramp and the rolling at constant speed on the level section was simple, but it would have been nontrivial create a physically accurate model of the motion along the curved section. The "cheat" consisted of having the ball's speed through the curved section be constant which, because the curved section was comparatively small, produced no discernible artifacts in the animation. Because the "cheat" allowed much simpler modeling and code, the animation was completed in time for the lecture for which it was intended.

Costs and Benefits

The primary cost in developing short illustrative animations is the time of the creator of the animation. Part of that time can be attributed to the "learning curve" of the software, which for *POV-Ray* is fairly shallow. Most of the animations the authors have developed have taken anywhere from a few to tens of hours to develop the code. The actual rendering of each image frame can take from seconds to several minutes depending upon the complexity of the image, which for a 100 to 200 frame animation can result in a total rendering time of several minutes to many hours. Fortunately, this stage is automated so that the creator of the animation need not attend to the process. The conversion of the image files to a single animation media file generally takes just a few minutes.

The other resources required to create the animations are not expensive. Aside from computer operating systems, the software used to create the animations are all free and freely available on the Internet. The hardware required to run these programs are probably typical of the computing facilities available to almost any faculty in a technical field. The authors have used computers ranging from a 400 Mhz Pentium II with 128 MB of RAM running Windows 98 to a 3 Ghz Pentium IV with 512 MB of RAM running Windows XP Pro. Clearly a faster computer with more RAM will render images faster, but satisfactory performance can be obtained with almost any relatively modern desktop computer.

In an attempt to assess recent efforts in curriculum enhancement, students currently enrolled in their first semester of calculus-based physics were polled via an anonymous mid-semester survey on a number of technology-based resources. The following questions and responses provide some valuable feedback:

How helpful are the computer animations presented in class in helping you understand the material?

A great deal	54% (7)
A fair amount	38% (5)
Some	8% (1)
A little	0% (0)
None	0% (0)

How often do you access the animations available from the Schuylkill Physics web server?

Very often	0% (0)
Often	15% (2)
Sometimes	31% (4)
A little	54% (7)
Never	0% (0)

How useful are the Physics 211 web-based quizzes in allowing you to develop an understanding of the material?

A great deal	77% (10)
A fair amount	15% (2)
Some	8% (1)
A little	0% (0)
None	0% (0)

The first set of responses indicate that the students generally find that the animations can be an asset to understanding the materials. The second set of responses suggests that more can be done to help the students make use of the fact that the illustrative animations are available outside of the classroom experience. This is especially important because it is *outside* of the classroom where most of the students' learning takes place. The third question and set of responses regarding web-based quizzes come from other information gathered in the survey, and are included here for two reasons. First, they provide an example comparison of the students'

evaluation of the animations to other resources supplied for this course and indicate that the animations are valuable, but not necessarily the most valuable resource available. Second, it provides an indication of one avenue for getting the students to access the animation resources to a greater degree: more tightly couple the web-based quizzes to other web accessible resources (eg., via links from quiz questions directly to helpful materials). This is particularly important for freshman level courses where students have not yet developed all the skills that carry them through their university experience.

Comments and Conclusions

The authors have used animations extensively in lectures as a tool to help portray a wide variety of concepts in the introductory engineering curriculum, particularly physics and math. The animations and other curricular materials are available to the students via the Internet, to allow the students to have some self-directed access to these resources. The animations have provided a valuable visual aid that transcends the limitations of static diagrams.

The animation examples that have been discussed here demonstrate that it is possible to create high quality 3-D animations with relatively modest resources. Programs such as *POV-Ray* and *VirtualDub* provide free access to sophisticated 3-D graphics rendering that could easily cost thousands of dollars commercially. The evolution of the modern desktop computer has provided a platform capable of the intensive calculations required for 3-D graphics, allowing users to generate animations in tolerably short periods of time. *POV-Ray*'s geometric and object-oriented framework, along with excellent documentation, provides an extremely shallow learning curve for new users. Thus, the ability to create 3-D animations for curricular materials should be within reach of anyone within the science and engineering education community.

There are many aspects of the development of animations that could easily provide opportunities for engineering and science students. The transitions from model concepts to mathematical abstraction to code certainly allows the students to more deeply delve into how solutions to problems are found in the modern technological world. The construction of complex *POV-Ray* objects, perhaps using CAD-like programs such as Moray[7] (shareware) can allow students' creative and artistic design skills to be honed. Because *POV-Ray* is so intuitive to anyone comfortable with basic programming and geometry, engineering students should be able to build upon sample code to develop simple animations as part of class projects.

In conclusion, animations can be useful in a wide variety of circumstances related to the engineering curriculum. They provide a powerful mechanism for the illustration and interactive exploration of a range of concepts. The creation of animations itself can be an enriching process, and, perhaps most importantly, the ability to create animations is accessible to all.

References

1 http://www.povray.org/
2 The authors' physics and math animations can be found at:
http://phys23p.sl.psu.edu/phys_anim/Phys_anim.htm
and
http://phys23p.sl.psu.edu/~mrg3/mathanim/vmath.html
3 http://www.interactivephysics.com/
4 http://www.virtualdub.org/
5 http://www.cinepak.com/begin.html
6 http://www.wolfram.com/
7 http://www.stmuc.com/moray/

Biographical Information

DR. MICHAEL R. GALLIS is an assistant professor of physics at the Penn State Schuylkill Campus where he teaches the introductory calculus-based physics courses for engineering and science students. He has been actively developing computer-based curricular resources for his classes for much of his academic career.

DR PING WANG is an associate professor of mathematics at the Penn State Schuylkill Campus where he teaches various mathematics courses for engineering and science students. He has been actively developing computer-based curricular resources for his classes and has been collaborating on mathematics animation projects with Dr. Gallis for the past few years.