Physible: Making Physics Visible
MA198 Narrative

Brian Campbell-Deem
Professor George Francis

December 15th 2015

Abstract

Physible combines four smaller, physics-related programs into one package. Each subprogram features a 3-D simulation of some physical system, which are interactive to varying degrees. Ranging from simple kinematics to quantum mechanics, Physible will demonstrate visually the mathematics behind simulations of real-world physical systems, thereby making the physics visible.

1 Introduction and Background

Physible combines four sub-programs into one main project. The user runs the program they wish to view and is prompted to give inputs which alter the system (except for the hydrogen model, which has limited user interaction possibilities). The inputs they give change the resulting animation dynamically, allowing them to experience how different parameters affect certain systems and gain a more intuitive understanding for the math behind those systems. These four sub-programs vary in conceptual complexity and are samples of the following disciplines:

- kinematics
- chaos theory
- quantum mechanics
- thermal and statistical physics

The sub-programs of my project are more surface-level explorations of the above areas in comparison to what is possible with them; they are meant to quickly show the basic properties of a system and how it changes with certain parameters.

2 Theory

2.1 Kinematics

The kinematics project is the most simple: it displays the path of motion of some thrown object under the effects of gravity. The relevant equations are derived from the assumption of constant gravitational force and the definitions of acceleration and velocity:

$$\vec{a}_y = \frac{d\vec{v}_y}{dt} = -g\hat{y}$$

$$\vec{v}_y = \frac{d\vec{y}}{dt} = \int_0^t \vec{a}_y dt = (v_{y_0} - gt)\hat{y}$$

$$\vec{y} = \int_0^t \vec{v}_y dt = (y_0 + v_{y_0}t - \frac{1}{2}gt^2)\hat{y}$$

and so given an initial velocity and position, one knows the position of the object for all times t. Because there are no forces in the $\hat{x}$ or $\hat{z}$ directions, their accelerations are zero and thus their velocities constant (until the object hits the ground).

2.2 Chaos Theory

The chaos project features the paradigm of chaos: a double-pendulum system. This system is most easily described through the approach of Lagrangian mechanics[1]. With this, one can describe the position of each pendulum as a function of its angle with respect to its pivot, and using the Euler-Lagrange equations, can solve for their motion. Figure 1 displays how the coordinates are measured.

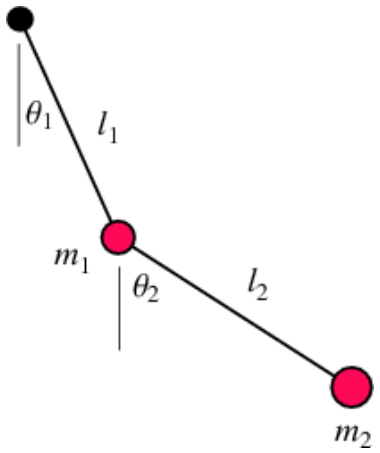

Figure 1: Coordinate measurements for the Lagrangian approach of solving the double pendulum.[1]

The Euler-Lagrange equations read

$$L(q, \dot{q}, t) \equiv T(\dot{q}, t) - U(q, t)$$
$$\frac{\partial L}{\partial q} = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}}$$

where the dot notation indicates a derivative with respect to time. As shown in Figure 1, there are two coordinates for q: θ_1 and θ_2. To solve this system, one needs the energies of each mass. This is most easily accomplished by using their x and y positions as such:

$$x_1 = l_1 \sin\theta_1$$
$$y_1 = -l_1 \cos\theta_1$$
$$x_2 = x_1 + l_2 \sin\theta_2 = l_1 \sin\theta_1 + l_2 \sin\theta_2$$
$$y_2 = y_1 - l_2 \cos\theta_2 = -l_1 \cos\theta_1 - l_2 \cos\theta_2$$

The potential energy U is simple in a gravitational field of strength g: $U = mgy$. Namely,

$$U = m_1 g y_1 + m_2 g y_2 = -m_1 g l_1 \cos\theta_1 - m_2(l_1 \cos\theta_1 + l_2 \cos\theta)$$

The kinetic energy may be written via its classical mechanics definition:

$$T = \frac{1}{2}m_1(\dot{x_1}^2 + \dot{y_1}^2) + \frac{1}{2}m_2(\dot{x_2}^2 + \dot{y_2}^2)$$

$$= \frac{1}{2}m_1 {l_1}^2 \dot{\theta_1}^2 + \frac{1}{2}m_2\left[{l_1}^2\dot{\theta_1}^2 + {l_2}^2\dot{\theta_2}^2 + 2l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2)\right]$$

Thus, using the above definition of the Lagrangian, the system's Lagrangian is

$$L = \frac{1}{2}(m_1 + m_2){l_1}^2\dot{\theta_1}^2 + \frac{1}{2}m_2{l_2}^2\dot{\theta_2}^2 + m_2 l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2) \tag{1}$$

Now, using this form for the Euler-Lagrange equations yields the two equations of motion for each mass:

$$(m_1+m_2)l_1\ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 \cos(\theta_1-\theta_2) + m_2 l_2 \dot{\theta_2}^2 \sin(\theta_1-\theta_2) + g(m_1+m_2)\sin\theta_1 = 0 \tag{2}$$

$$m_2 l_2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_1\theta_2) - m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 g \sin\theta_2 = 0 \tag{3}$$

The solutions to these differential equations for θ_1 and θ_2 may be found numerically in the program to display realistic behavior.

This is the method Bruce Sherwood uses in his *doublependulum.py* program; for convenience he calculates the *moments of inertia* of each bar due to their length and masses as an extra factor, but otherwise the analyses of the systems are the same.

2.3 Quantum Mechanics

The quantum mechanics project plans to demonstrate the quintessential hydrogen atom wavefunctions.
The most fundamental equation in (non-relativistic) quantum mechanics is the Schrödinger equation; once solved, you have the *wavefunction* and thereby everything there is to be known about the system[2]. The *Schrödinger equation* reads

$$i\hbar\frac{\partial}{\partial t}\Psi(\vec{r},t) = \hat{H}\Psi(\vec{r},t) \tag{4}$$

where Ψ is the wavefunction and $\hat{H}$ is the *Hamiltonian* (energy) operator

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(r,t)$$

where m is the mass of the particle, ∇^2 is the Laplace operator, and $V(r,t)$ is the (given) potential.

This differential equation may be solved by assumption of form, guessing only one of many possible solutions, and then combining these to form the general solution. Thus, assume the form

$$\Psi(\vec{r},t) = \psi(\vec{r})\phi(t)$$

Using this solution in the Schrödinger equation and then dividing by the assumed form for Ψ results in

$$i\hbar\frac{1}{\phi}\frac{d\phi}{dt} = -\frac{\hbar^2}{2m}\frac{1}{\psi}\nabla^2\psi + V$$

For this case, if one assumes the potential depends only on r (which is the case for most real situations, including the hydrogen atom) then the two equations are separated; that is, each side is a function of a different independent variable than the other side. Because r and t are independent, this implies both sides of the equation are constant; if one variable is changes and the other does not, the equality relation must still hold; the only way this can be true is if both sides are equal to some constant. The time solution may be written as

$$i\hbar\frac{1}{\phi}\frac{d\phi}{dt} = E$$

which has the solution

$$\phi(t) = e^{-iEt/\hbar} \tag{5}$$

It so happens that this constant E is actually the energy of the system, and for various Ψ_n, the nth excited state has energy E_n. The general wave function may be constructed as a sum of these solutions, called *stationary states*:

$$\Psi(\vec{r},t) = \sum c_n \psi_n e^{-iE_n t/\hbar} \tag{6}$$

Now, working on the spatial part, a potential function must first be specified; one of special interest (and the case which will be modeled by the project) is that of the Hydrogen atom, where the potential is the *Coulomb potential* between the electron and the proton nucleus, which is defined by

$$V_H = \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2}{r} = -\frac{1}{4\pi\epsilon_0}\frac{e^2}{r}$$

where e is the *elementary charge* and ϵ_0 is the *permittivity of free space*. It is natural to use spherical coordinates for this system[1]; by using the spherical form of the Laplace operator, the Schrödinger equation reads

$$-\frac{\hbar^2}{2m}\left[\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\psi}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\psi}{\partial\theta} + \frac{1}{r^2\sin^2\theta}\left(\frac{\partial^2\psi}{\partial\phi^2}\right)\right]\right] + V\psi = E\psi$$

As before, the attempt to solve this comes via separation of variables, using a spacial wavefunction in the form $\psi(r,\theta,\phi) = R(r)Y(\theta,\phi)$. If this form is used and the result divided by ψ, as well as multiplying by a factor $-2mr^2/\hbar^2$, it produces

$$\left\{\frac{1}{R}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) - \frac{2mr^2}{\hbar^2}\big[V(r) - E\big]\right\} + \frac{1}{Y}\left\{\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial Y}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial\phi^2}\right\} = 0$$

The first term is a function only of r while the second is a function of only θ and ϕ. Like previously, these must each be constants, which are chosen to be written as such:

$$\frac{1}{R}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) - \frac{2mr^2}{\hbar^2}\big[V(r) - E\big] = l(l+1)$$

$$\frac{1}{Y}\left\{\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial Y}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial\phi^2}\right\} = -l(l+1)$$

To solve the angular equation first, again try separation of variables with $Y(\theta,\phi) = \Theta(\theta)\Phi(\phi)$; using this form and dividing by Y gives

$$\left\{\frac{1}{\Theta}\left[\sin\theta\frac{d}{d\theta}\left(\sin\theta\frac{d\Theta}{d\theta}\right)\right] + l(l+1)\sin^2\theta\right\} + \frac{1}{\Phi}\frac{d^2\Phi}{d\phi^2} = 0$$

[1] the convention for θ and ϕ are exchanged in standard physics notation in comparison to standard math notation: here ϕ is the azimuthal (horizontal) angle and θ the polar angle (measured from the z-axis .

again indicating two individual pieces which are functions of only one variable, implying they are constant; this time they are set to be

$$\frac{1}{\Theta}\left[\sin\theta\frac{d}{d\theta}\left(\sin\theta\frac{d\Theta}{d\theta}\right)\right]+l(l+1)\sin^2\theta=m^2$$

$$\frac{1}{\Phi}\frac{d^2\Phi}{d\phi^2}=-m^2$$

The ϕ equation is simply solved:

$$\Phi(\phi)=e^{im\phi} \tag{7}$$

Where m may run negative to cover all possible solutions. The requirement that $\Phi(\phi+2\pi)=\Phi(\phi)$ is periodic limits m to integer values.

The θ equation has been solved to have the solution

$$\Theta(\theta)=AP_l^m(\cos\theta) \tag{8}$$

Where P_l^m is the associated Legendre function

$$P_l^m(x)\equiv(1-x^2)^{|m|/2}\Big(\frac{d}{dx}\Big)^{|m|}P_l(x)$$

where $P_l(x)$ is the lth Legendre polynomial

$$P_l(x)\equiv\frac{1}{2^l l!}\Big(\frac{d}{dx}\Big)^l(x^2-l)^l$$

both of which are, in this case, functions of $\cos\theta$ i.e. $x=\cos\theta$. There are two other solutions (because the differential equation is second order) however they are unphysical, and thus ignored. Because the total probability must be 1, there is a need to normalize the wavefunction, which is done conveniently for each piece separately. Doing this results in the final answer for Y:

$$Y_l^m(\theta,\phi)=\epsilon\sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}e^{im\phi}P_l^m(\cos\theta) \tag{9}$$

where $\epsilon=(-1)^m$ for $m\geq 0$ and $\epsilon=1$ for $m\leq 0$. These are notably the **spherical harmonics**.

For solving the radial equation, it is convenient to make the substitution $u(r)\equiv rR(r)$ (and thus $dR/dr=rd^2u/dr^2$; doing so and entering in the Coulomb potential for the hydrogen atom gives

$$-\frac{\hbar^2}{2m}\frac{d^2u}{dr^2}+\left[-\frac{e^2}{4\pi\epsilon_0}\frac{1}{r}+\frac{\hbar^2}{2m}\frac{l(l+1)}{r^2}\right]u=Eu$$

This is solved by looking at the asymptotic behaviors of u and introducing some extra functions to create the in-between behavior. Introduce $\kappa\equiv\frac{\sqrt{-2mE}}{\hbar}$ (which is real, since bound state energies are negative and the electron is bound)

and $\rho \equiv \kappa r$ and $\rho_0 \equiv \frac{me^2}{2\pi\epsilon_0\hbar^2\kappa}$. Now introduce the "helper" function $v(\rho)$ and find that the radial equation now reads

$$\rho\frac{d^2v}{d\rho^2} + 2(l+1-p)\frac{dv}{d\rho} + \big[\rho_0 - 2(l+1)\big]v = 0$$

By expressing $v(\rho)$ as a power series and using the derivatives, one finds that the coefficients are determined by a recursive formula. It is also found that the series must terminate after some maximal index number j_{max} because if it didn't it would produce unphysical results ($u(\rho)$ as an exponential of ρ). Defining the principal quantum number $n \equiv (j_{max} + l + 1)$ determines ρ_0 and thereby the energies:

$$E_n = -\left[\frac{m}{2\hbar^2}\left(\frac{e^2}{4\pi\epsilon_0}\right)^2\right]\frac{1}{n^2} \tag{10}$$

and finally the radial function

$$R_{nl}(r) = \frac{1}{r}\rho^{l+1}e^{-\rho}v(p) \tag{11}$$

where $v(p)$ is a polynomial of degree j_{max} whose coefficients are determined by the recursive formula

$$c_{j+1} = \frac{2(j+l+1-n)}{(j+1)(j+2l+2)}c_j$$

and thus the final solution for the constituent pieces of the stationary states of the wavefunction are found. Because they are orthogonal and complete, being careful of normalization one may use linear combinations of them to build whatever wavefunction they like, in a manner exactly like Fourier analysis. More explicitly, the spacial term is

$$\psi_{nlm} - \sqrt{\Big(\frac{2}{na_0}\Big)^3\frac{(n-l-1)!}{2n[(n+l)!]^3}}e^{-r/na}\Big(\frac{2r}{na_0}\Big)^l\Big[L^{2l+1}_{n-l-1}(\frac{2r}{na_0})\Big]Y^m_l(\theta,\phi) \tag{12}$$

where

$$L^p_{q-p}(x) \equiv (-l)^p\Big(\frac{d}{dx}\Big)^p L_q(x)$$

$$L_q(x) \equiv e^x\Big(\frac{d}{dx}\Big)^q(e^{-x}x^q)$$

and a_0 is the *Bohr radius* $a_0 = (r\pi\epsilon_0\hbar^2)/(me^2)$.

2.4 Thermal and Statistical Physics

The thermal and statistical physics project simulates a gas in a box i.e. the interaction of many particles in an enclosed space.
The goal of these areas of physics is to model the behavior of large collections of particles (order 10^{20} or larger). Because of the nature of the program, most equations won't hold (as well) because they are made with approximations based on the fact that the number of particles N is very large. However, one relevant equation is the simple yet fundamental equation for conservation of momentum

$$\sum_{i}^{N} \vec{p}_{i,i} = \sum_{j}^{N} \vec{p}_{j,f} \text{ when } \frac{d\vec{P}}{dt} = 0 \tag{13}$$

which says that when there are no external forces ($\vec{F}_{net} \equiv d\vec{P}/dt$, where P is the total momentum), the momentum of the system will be conserved. This applies to the collision of particles, meaning that their total momentum remains unchanged by a collision (Newton's Third Law says no net force is exerted on either due to each other). Thus, though they bounce in different directions, knowing their momenta must have the same vector sum before and after allows us to solve for their final individual momenta. Also necessary (if the given parameters do not suffice) is conservation of energy:

$$\sum_{i}^{N} \frac{1}{2} m_i v_{i,i}^2 = \sum_{j}^{N} \frac{1}{2} m_j v_{j,f}^2 \tag{14}$$

which says that the total (kinetic, as there is no potential) energy of the system remains constant. This holds true as long as the collisions are elastic (no internal force or energy changes to hold them together) and no net force acts on the system. It is somewhat redundant, as because $\vec{p} = m\vec{v}$ for particles of constant mass, $E = \frac{p^2}{2m}$, so if momentum is conserved then energy would be as well. However the extra equation constrains more parameters, should they be needed, to solve for the final state of the system after a collision.
By choosing to modify Bruce Sherwood's *gas.py* an inherent included section of math comes from the *Maxwell velocity distribution* which describes the probability distribution of velocities for ideal gases obeying Maxwell-Boltzmann statistics[3]. From standard thermodynamics, the probability of finding a particle in a given state is proportional to its *Boltzmann factor* $e^{-E/kT}$ where E is the energy of that state, k is the Boltzmann constant, and T is the absolute temperature. For a particle moving at a speed v the kinetic energy will mean the probability density of the velocities will be proportional to $e^{-mv^2/2kT}$.
To evaluate the true factor, *velocity space* is introduced and imagined as a sphere on which a given speed v lives with radius v. A larger speed corresponds to a larger sphere and thereby more possible velocity vectors; a speculative claim then is that the probability distribution is also proportional to the surface area of the velocity sphere, namely its surface area $4\pi v^2$.

Taken together, the form of the distribution is

$$D(v) = C(4\pi v^2)(e^{-mv^2/2kT})$$

for some constant C. This constant may be solved by the same probability analysis as in the quantum section: the distribution must integrate to 1. The integral to be solved then is

$$1 = 4\pi C \int_0^\infty v^2 e^{-mv^2/2kT} dv$$

Solving for C gives the *Maxwell velocity distribution*:

$$D(v) = \left(\frac{m}{2\pi kT}\right)^{3/2} 4\pi v^2 e^{-mv^2/2kT} \tag{15}$$

3 Goals (Pre-completion)

The overall project should be self-contained in the sense that the user may navigate the programs and back through some "hub" without explicitly having to run separate files. Each sub-project should be interactive and display a physical system in some realistic capacity. Below are the individual goals for each sub-project:

3.1 Kinematics

- Allow the user to pick mass, radius, velocity (or choose them to be random) for the launched object.
- Draw the launched object in a field and show the path of motion it travels
- Possibly add external forces e.g. winds, drags that will alter the motion in some more complex manner

3.2 Chaos Theory

- Allow the user to enter masses and lengths of the pendula, as well as initial angles.
- Go beyond the double pendulum into a triple pendulum and possibly even an n-pendulum system.

3.3 Quantum Mechanics

- Allow the user to enter the quantum numbers to view different eigenstates of the hydrogen atom.
- Display either the wave function or the probability density for each state

3.4 Thermal and Statistical Physics

- Have realistic particle bounce which obeys the laws of physics
- Allow user some input such as average velocity of particles or number of particles
- Implement some method of displaying average characteristics of the particles, which is the point of this discipline

4 Methods (Pre-completion)

This program will be written entirely in Python using primarily the VPython library for the graphics and some of the included mathematical libraries such as NumPy. The files will be separate but navigable via either the terminal or (more preferably) the actual window in which the simulation is displayed, so that running the main file will allow you to examine all of the files.
VPython has plenty of resources available to make these projects possible; the ease of creating mutable objects leaves most of the work into simply efficiently coding the math in. For example, the double pendulum program is made much simpler through frames and $.rotate(\theta)$ methods. VPython was built for physics simulations, which means it is the perfect platform for my project.

5 Implementation and Narrative

Having finished my project, there are of course parts of my original plan (included in the Goals section) which I didn't get to; nonetheless I believe I have achieved my chief goal of creating a heuristic set of physics programs which display how certain physical systems behave. The main piece of the project that I was unable to achieve was tying the programs all together through one *main.py* which would allow easy navigation (as opposed to separately running each file manually). I believe the ability is there, however the methods I attempted proved faulty and easily broken, and this was quickly scrapped in favor of completing the "meat" of the project in time. Additionally, due to time constraints I had to modify two already-written programs as opposed to writing them myself; while this is bad for my own practice, it saved a lot of time and allowed me to use two already excellent programs.

5.1 kinematics.py

The *kinematics.py* program begins by asking the users for a y-component of velocity. The $x-$ and $z-$components are generated randomly to create a dynamic effect, but because the analysis of motion is chiefly based on the y coordinate of the ball, the camera is initialized to always be perpendicular to the plane

of motion. Once the user enters their value, the animation loop begins running; the sphere is located on "grass" surrounded by a "sky", both for visual effect. The animation loop updates the position of the sphere using an integration approximation based on the kinematics equations described in the theory section. While the ball is moving, another window is additionally updated; this window contains three plots of magnitude vs. time; the magnitudes graphed are the ball's y-components of acceleration, velocity, and displacement. The graphs are updated in real time with the ball animation, giving another, more mathematical view of what is happening in the system. Once the ball contacts the ground, the animation loop ceases and the user is prompted with the choice of running the program again. Because the entire program runs in a loop, if the user chooses 'yes' the windows are closed and reset, and the next iteration continues.

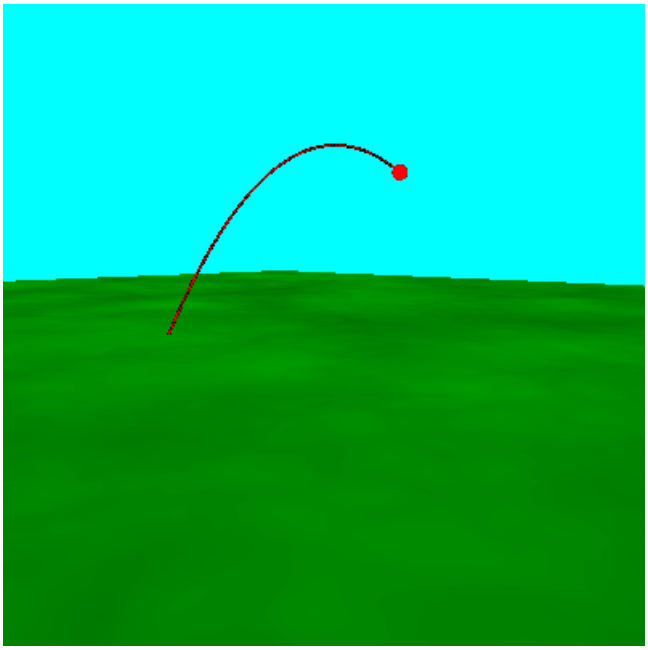

Figure 2: *kinematics.py*

5.2 doublependulum.py

For *doublependulum.py*, most of the work was already completed by Bruce Sherwood. It initiates several bars into frames with each other, and then through an animation loop it does an approximation with extremely small time steps to simulate a solution to the differential equations governing the motion. Because of the complexity of the system, not only must the positions and velocities be updated in time, but even the accelerations; this extremely fine dependence on

time is exactly what gives the system its chaotic behavior. Using these accelerations and velocities, it rotates the frame positions accordingly to simulate the motion of a double-pendulum. For the portion I wrote, I essentially copied the code to execute again in a second window; for this other system I have a small offset, entered by the user, which slightly alters the starting position in comparison to the "original" double-pendulum. This slight offset quickly leads to erratically different behaviors, whose difference is easily distinguished in the side-by-side view. I also coded in a sphere to the bottom of the lower pendulum, which moves along with the system (outside of the frame) allowing a trail of the traveled path to be drawn, should the user indicate they want it to appear. When the program first runs, the user is also prompted to enter two masses for each pendulum, giving them an extra parameter whose effect they may explore.

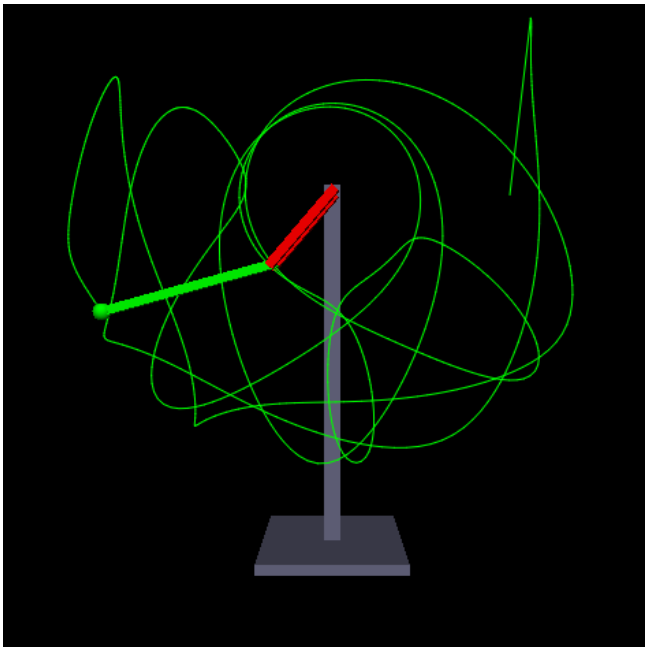

Figure 3: *doublependulum.py*

5.3 hydrogen.py

hydrogen.py simulates thousands of electron measurements across an ensemble of hydrogen atoms, and represents them as points in a single hydrogen atom (i.e. around a proton). The proton is represented by a red sphere, and the scene is scaled to be in multiples of the fundamental Bohr radius, which is the most probable radius for the electron. Depending on the amount of points to be displayed at one time (the default is 4000) an array of hidden points is initialized. Then, in the animation loop, a random point around the proton is chosen

based on the probability density for the ground state electron. To realize this, I employed a brute-force method I devised to weight the included *random()* function to align with the probability density. I integrated the probability density in small steps (usually $a_0/10$) to find the percentage of time the electron would be found there, and then a random seed is compared to that range. If the seed is too large (indicating the electron is farther out), it moves on to check to see if it lies in the next area of integration. The probability density mathetmatically extends to infinity, but for practical purposes I had the possible radii cease at $5a_0$, which totaled to 99.7% of the total 100% probability. To conserve probability, I evenly distributed the remained 0.3% into the original ranges to keep the distributions correct relative to one another. The angular coordinates are chosen randomly, evenly, as the probability density has no angular dependence. Once the coordinates are chosen, it moves a point to that point and loops again to do the same for the next point; once it hits the end of the array, it flips back to 0 to run through the points again, thereby always retaining the same amount of points on the screen at once. The animation rate is also set very high so a point doesn't stay on the screen for more than half a second; combined with the high number of points, this creates a cloud-like distribution reminiscent of the familiar electron-cloud image. The density gradient allows the user to see the most probable radii and how it quickly drops off with increasing radius.
An additional static graph is created that shows the wavefunction, probability density, and probability plots for the ground state.

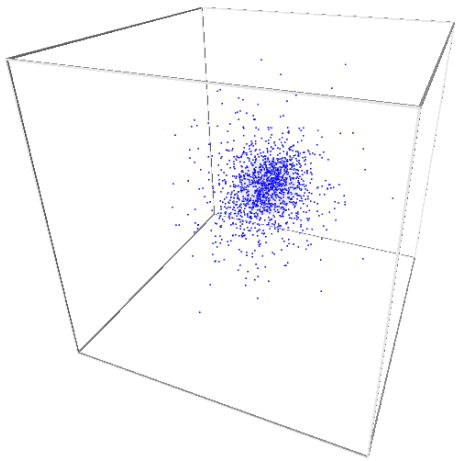

Figure 4: *hydrogen.py*

5.4 gas.py

Here again most of the work is done by Sherwood. In a rather brilliant method, he initializes the particles in an array, and then using a separate array of their momenta, updates their positions all at once. After doing so, a method he wrote deduces any colliding pairs, and consequently causes them to bounce. Because of the efficiency of his method, there can be several hundred animated particles bouncing at once with no noticeable lag. The collision check also keeps the particles confined to the drawn box they reside in; a special particle is colored yellow, as opposed to cyan, and has an ephemeral trail following it, allowing the user to easily follow the motion of a single particle through the box.
Sherwood also implemented a bin-sort method, which creates bins of velocity ranges, and counts the particles that reside in each range. These bins update in real time and are plotted against the Maxwell velocity distribution, demonstrating that distribution equilibrates to follow its prediction nearly exactly.
There wasn't a great deal for me to change, so I added user interactivity to choose how many particles are displayed, allowing them to see how as N grows the equations begin to become better approximations for the system more and more quickly. I additionally added a graph which updates, once per second, the average collisions per second, demonstrating that they begin high and eventually relax to some value. Unfortunately, I couldn't find good (fitting) theory that described this situation well, and the length of time it takes to relax to a steady value implies that N is not large enough for such accuracy as it is for the Maxwell velocity distribution. Nonetheless, I left the graph in with no theory comparison to demonstrate a separate type of equilibrium that the particles achieve.

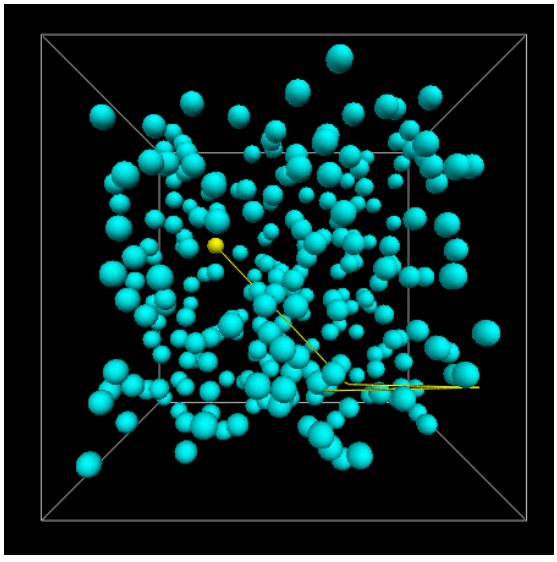

Figure 5: *gas.py*

6 Conclusion and What Comes Next

With my project finished, though I didn't achieve every goal I set out to do, I'm nonetheless very happy with the results. I wrote two physics programs from scratch which I believe model real systems in an understandable manner, as well as modifying two existing programs to make them more catered to my goal. Each project still retains huge possibility for exploration: the kinematics example can include more real-world effects such as drag and bounce; the hydrogen atom has an infinite number of states to explore beyond the ground, and no real-world energy perturbations were taken into account; etc. Even outside of the physical systems, the subprograms have yet to be tied together into one truly cohesive program, and each still has minor bugs which could be fixed. Though I'm finished with the project, there are still huge areas to improve upon here–with the range of possibilities, it wouldn't be difficult to continue *Physible* as another project.

References

[1] http://scienceworld.wolfram.com/physics/DoublePendulum.html

[2] Griffiths, David J. "Chapter 4: Quantum Mechanics in Three Dimensions." Introduction to Quantum Mechanics. 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005. 131-57. Print.

[3] Schroeder, Daniel V. An Introduction to Thermal Physics. San Francisco, CA: Addison Wesley, 2000. Print.