Jonathan San Miguel
Physics 273

Physics Honors Project: 2D Particle Wave Simulation Using a Spring Lattice

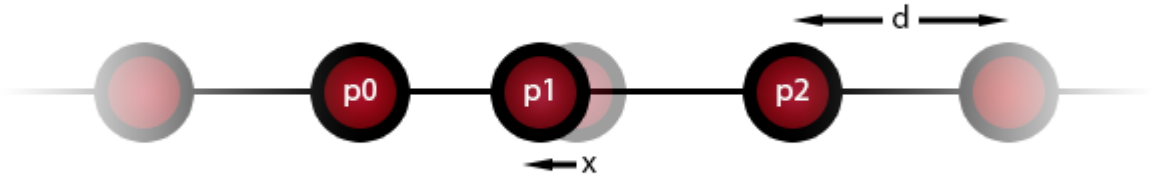
Introduction

In this project, I have created a wave simulator that uses individual particles, each connected to its neighbors by spring forces. The particles are arranged in a grid with four connections, which as we will see increases the error in the program but also improves computational efficiency. A compromise between analytical and numerical analysis, this program allows the user to get a good qualitative view of waves in a two-dimensional medium, similar to a ripple tank.

Derivation and Analytical Cases

Spring lattice models, along with finite-element analysis, are one of the two main methods for representing wave transmission. Finite-element analysis has long been valued for computational efficiency, but spring lattices are simpler and in some cases more accurate, especially with regards to sound waves travelling through crystalline solids (Holt-Phoenix). The main fields I found literature on spring-lattice models in were seismology and material sciences.

It is clear that an initial displacement in a spring lattice will move in a wave pattern.



In the diagram above, a displacement in p_1 creates force on not only p_1, but p_2 and p_0 as well. If they are connected by ideal springs with constant k, the force on all three will be kx, in different directions. With p_0 and p_2 attempting to move back to equilibrium, the distance between those particles and their neighbors will no longer be d as well, and the process will repeat. The equation of motion for p_1 of mass M (and any other particle p_n) is:

$$\frac{d^2p_0}{dt^2} = -\frac{k}{M}(p_2 - p_1) - \frac{k}{M}(p_0 - p_1)$$

which simplifies to:

$$\frac{d^2p_0}{dt^2} = -\frac{k}{M}(p_2 + p_0 - 2p_1)$$

The resemblance to the equation of motion for the loaded string is clear, which has a continuum limit that reduces to the wave equation.

In Two Dimensions

Working with a lattice in two dimensions is a little bit trickier. Now particles have two indices $p_{m,n}$. For a particle p_1 displaced from its normal position d meters from p_2, then the force of p_2 on p_1 is now:

$$F_{p1,p2} = \frac{k}{M}\left|d - |p_2 - p_1|\right|\hat{\imath}$$

Where k is the spring constant, d is the normal length of the spring, and $\hat{\imath}$ is a vector pointing in the direction of the displacement between the two particles. This equation is more complicated because now any particle can potentially be anywhere. It accounts for rotations that the one-dimensional version does not. If, however, the particles are expected to only make small oscillations about their starting positions, an approximation can be made:

$$F_{p1,p2} = \frac{k}{M}(de_i - (p_2 - p_1))$$

Where e_i is the *standard basis vector* pointing in the direction of the original displacement between the two springs. This greatly simplifies calculations, allowing combinations of three particles in one equation, similar to the one-dimensional case.

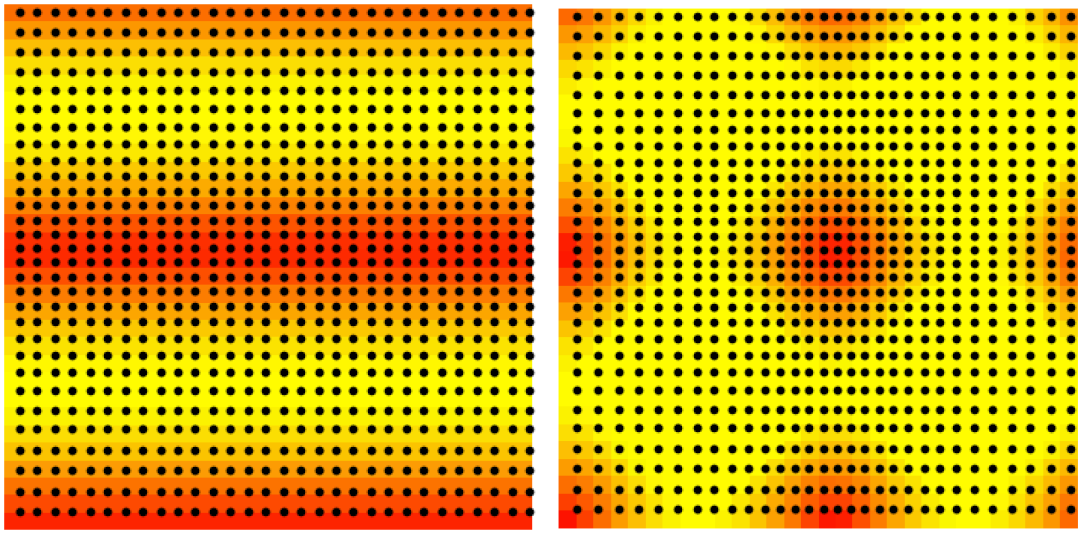
$$\frac{d^2p_{m,n}}{dt^2} = \frac{k}{M}\left(p_{m+1,n} + p_{m-1,n} - 2p_{m,n}\right) + \frac{k}{M}\left(p_{m,n+1} + p_{m,n-1} - 2p_{m,n}\right)$$

Not only does this more than double computational efficiency by getting rid of the normalizing vector operations, but it also creates a differential equation with useful information about the properties of the system.

Longitudinal and Transverse Waves

A possible plane wave solution for the above equation is:

$$p_{m,n} = (m + e^{i(c_x m - \omega t)}, n + e^{i(c_y n - \omega t)})$$



Two examples of movement with these conditions are shown above, with a color plot similar to the one used in the program behind the particles themselves. The longitudinal properties of this wave can be clearly seen. It can also be seen from the equation that this wave is a travelling wave that will sum to create a standing wave. This standing wave is a normal mode at frequency ω. In the literature, such waves for 2D systems are usually referred to as phonons (Kittel). Phonons represent a more general type of wavelike behavior in any kind of system composed of multiple particles.

To solve for ω, we substitute these conditions back into the equation of motion, getting two equivalent equations for both components of the vector. One is shown for simplicity:

$$\omega^2 e^{i(c_x m - \omega t)} = \frac{k}{M}(e^{i(c_x(m+1)-\omega t)} + e^{i(c_x(m-1)-\omega t)} - 2e^{i(c_x m - \omega t)}$$

The time dependence cancels, leaving:

$$\omega^2 = \frac{k}{M}e^{ic_x} + \frac{k}{M}e^{-ic_x} - 2\frac{k}{M}$$

and

$$\omega^2 = \frac{k}{M}e^{ic_y} + \frac{k}{M}e^{-ic_y} - 2\frac{k}{M}$$

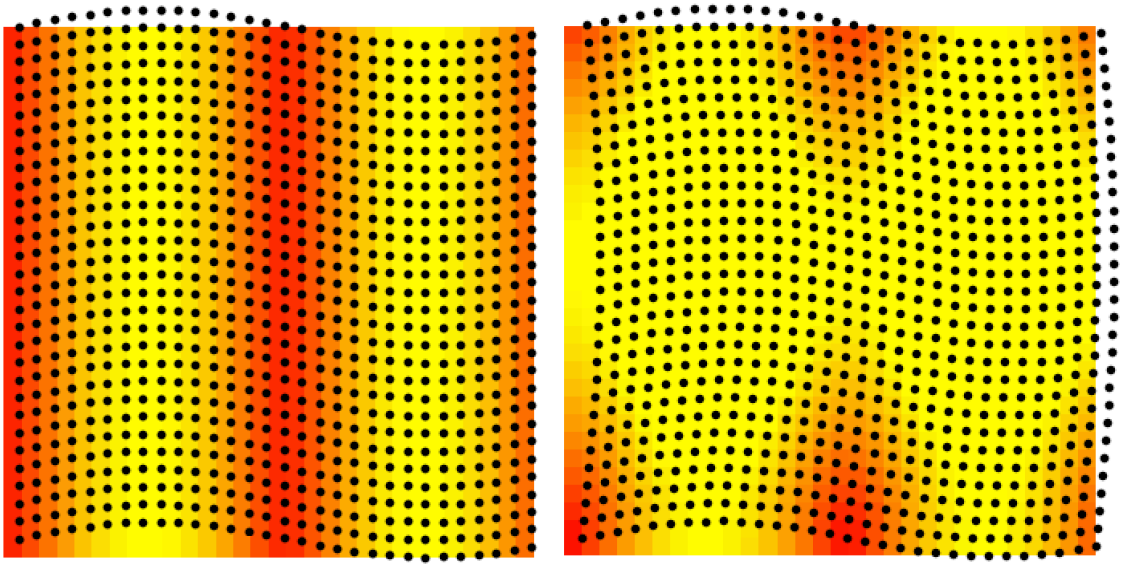
Adding them together, and using some trig identities we get:

$$\omega = \sqrt{\frac{k}{M}(\cos c_x + \cos c_y - 2)}$$

This shows that c_x and c_y are independent. This is also the dispersion relation of the longitudinal wave.

Another important phonon in this system will be transverse motion, where displacement is perpendicular to the wave propagation. The equation for this wave would be:

$$p_{m,n} = (m + e^{i(c_y n - \omega t)}, n + e^{i(c_x m - \omega t)})$$



Above two such transverse waves are shown. Substituting into the equation of motion and solving for ω, we find:

$$\omega^2 e^{i(c_x m-\omega t)} = \frac{k}{M}(e^{i(c_y(n+1)-\omega t)} + e^{i(c_y(n-1)-\omega t)} - 2e^{i(c_y n-\omega t)})$$

and

$$\omega^2 e^{i(c_y n-\omega t)} = \frac{k}{M}(e^{i(c_x(m+1)-\omega t)} + e^{i(c_x(m-1)-\omega t)} - 2e^{i(c_x m-\omega t)})$$

These equations need to be factored to cancel out the time dependence.

$$\omega^2 e^{i(c_x m-\omega t)} = \frac{k}{M} e^{i(c_y n-\omega t)}(e^{ic_y} + e^{-ic_y} - 2)$$

$$\omega^2 e^{i(c_y n-\omega t)} = \frac{k}{M} e^{i(c_x m-\omega t)}(e^{ic_x} + e^{-ic_x} - 2)$$

Then, solving for ω^2:

$$\omega^2 = \frac{k}{M} e^{i(c_y n-c_x m)}(e^{ic_y} + e^{-ic_y} - 2)$$

$$\omega^2 = \frac{k}{M} e^{i(c_x m-c_y n)}(e^{ic_x} + e^{-ic_x} - 2)$$

So:

$$\omega^4 = \frac{k}{M}(e^{ic_x} + e^{-ic_x} - 2)(e^{ic_y} + e^{-ic_y} - 2)$$

$$\omega = \sqrt[4]{2\frac{k}{M}(\cos c_x - 1)(\cos c_y - 1)}$$

This value is the dispersion relation for a transverse wave. The group velocities for both of these waves can be found by taking their partial derivatives with respect to c_x and c_y. For the longitudinal wave:

$$\frac{\partial \omega}{\partial c_x} = -\frac{\sin c_x}{2\sqrt{\frac{k}{M}(\cos c_x + \cos c_y - 2)}}$$

and

$$\frac{\partial \omega}{\partial c_y} = -\frac{\sin c_y}{2\sqrt{\frac{k}{M}(\cos c_x + \cos c_y - 2)}}$$

For the transverse wave:

$$\frac{\partial \omega}{\partial c_x} = (-\frac{k}{2M}\big(\cos c_y - 1\big)\sin c_x)(2\frac{k}{M}(\cos c_x - 1)\big(\cos c_y - 1\big))^{-3/4}$$

$$\frac{\partial \omega}{\partial c_y} = (-\frac{k}{2M}(\cos c_x - 1)\sin c_y)(2\frac{k}{M}(\cos c_x - 1)\big(\cos c_y - 1\big))^{-3/4}$$

Why are these given in components? Because the spring is connected in 4 ways, most of the motions of the particles will be in the *x* and *y* directions. Furthermore, any transverse or longitudinal solution will just be a superposition of 2 waves, and the *x* movement will be independent of the *y* movement.

Note: I've provided the Mathematica file that I used to show these analytical solutions along with my numerical model. It's called analytical.nb. In it, you can look

at a variety of transverse and longitudinal waves, with different initial conditions. Conditions that violate Hooke's Law can be created by changing the **w** parameter – the natural frequency. They are still waves, however, just with different dispersion relations.

Building the Model

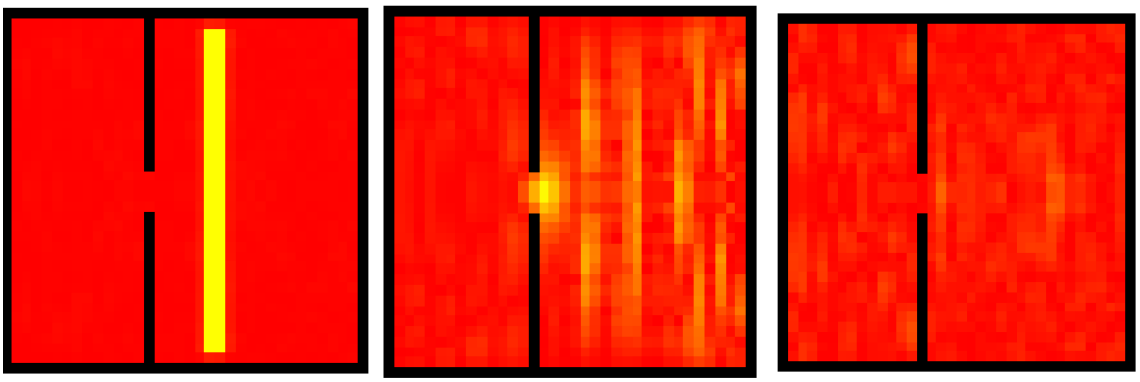
While the analytical equations present a concise and accurate picture, they are also very incomplete. A grid of particles with sides n in length will generally have n^2 normal modes, not just these two. In addition, many important wave forms, like radial waves, represent vibration at different frequencies and consequently aren't normal modes at all. While any conditions can be expressed as a sum of normal modes, it might be difficult or even impossible to find exact values.

Another issue is boundary conditions. Simply putting four walls on the outside of the grid makes finding normal modes extremely difficult (there will be $4n$ boundary conditions). A slit or point in the middle of the plane might make solving the differential equation impossible.

Numerical models can help solve all of these issues by taking simple principals and generalizing them to more complicated global behavior.

While how the model works has been mostly explained, the limitations of numerical approximation have created a few unique properties.

Damping



Simulation at 0 seconds, 20 seconds, and 1 minute, 30 seconds, with damping term .03

Because the model is based on small time steps instead of actual differential equations, the error magnifies as time goes on. Eventually, the amplitude of oscillations will increase, and after some time all the particles will leave the ripple tank entirely. This is clearly not the intended behavior.

By adding damping, the system becomes stable in the long run. Unfortunately, it also makes the dispersion relation extremely complicated, creating smaller waves as can be seen in the second image. In general, any discrete waveform in this simulation will decay into smaller ones at different wave numbers.

Creating Wave Forms

Instead of creating simple waves in this program, the functions (**wavePlane** and **waveRadial**) create pulses. Creating correctly shaped pulses was largely guesswork, and reliant on the appearance of the simulation.

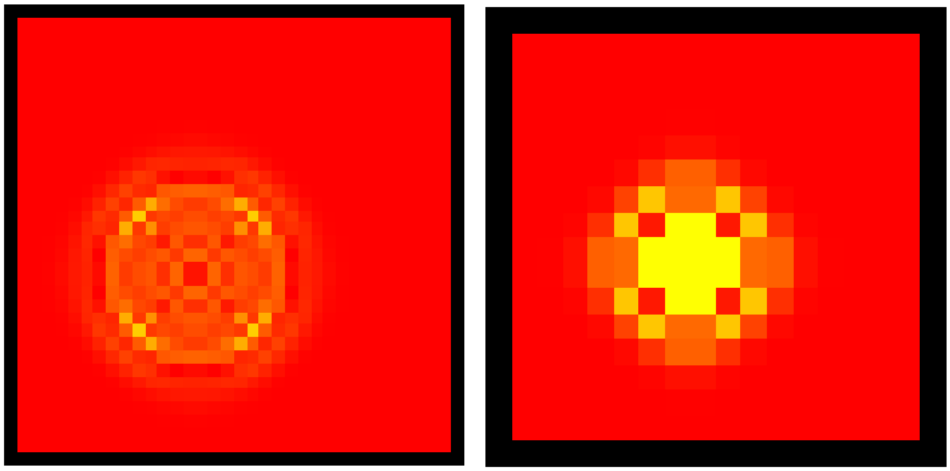
For the radial wave, I found that pulling 12 nearby particles all into their center of mass created a good circle shape when the particles were pushed outwards. The plane wave was a little more difficult, and I'm still not totally satisfied with it. The boundaries usually cause decay along the edges of the wave, which rapidly makes it unrecognizable. I found that creating a transverse wave with a relatively large amplitude was the best solution to this, as it allowed the main waveform to be seen even as the surroundings became chaotic.

Both types of waves do not form single wave forms, instead decaying into multiple peaks. This is due to the non-linear dispersion relation of the medium.

Error and Computing Efficiency

All of the parameters in the function – **t, k, b,** and **size** – can be related with error. **t**, and **k** are both proportionally correlated with error – that is, they decrease the precision as they get larger. In both cases, larger numbers will be added every step. **b** is mainly for reducing the error propagation, and not the error in each step. Without a small **b**, the position values will eventually approach infinity as the error magnifies, but with **b** they go to 0. Once **b** is added, however, it reduces the amount added every step, increasing the precision somewhat.

It is difficult to evaluate the impact of **size** besides qualitatively. It is clear from only a small glance, however, that a larger resolution creates much clearer images.



Radial waves at size = 36 and size = 18.

Size also has an inverse proportionality with computational efficiency. Using the framerate (amount of steps per second), we can calculate this empirically.

Size	Average Frame Rate (frames per second)
10	33
20	27
30	17
40	13
50	9
60	7

Numerical Details

This section attempts to explain how the numerical model is precisely carried out by all of the functions in the program. It also details how the functions can be modified to add further functionality.

The first two functions, **xIn** and **vIn**, both create a **size**x**size** matrix of coordinate values that will later be modified successively every time step. **xIn** is initialized to contain all the positions within the range of the simulation and **vIn** is initialized to all values of 0.

setAcc, and **aList** are where the main work is done. **setAcc** calculates the force on one particle from its neighbors, by applying force from the particle above, below, left, and right. Then, **aList** applies **setAcc** to every point, generating a matrix of acceleration values. The most important part of **aList** is that it only applies it to a set of values **crds**, thereby creating the boundary conditions. This is because the particles not in **crds** will remain motionless and will act as walls.

The main principle of the simulation, then, is adding matrices. With the acceleration calculated, all that is left is to progress in time. The formula for each step then is:

$$V = V + t \cdot A$$

and

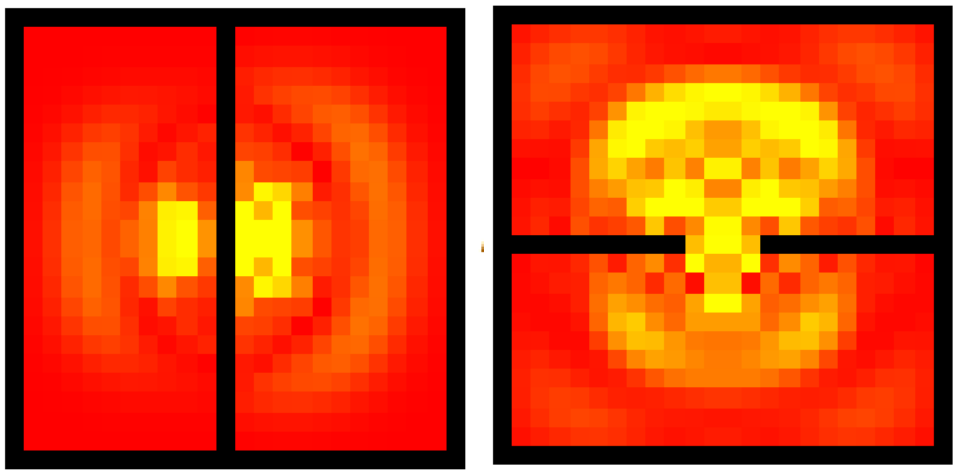
$$X = X + t \cdot V$$

Where A is the matrix of acceleration values, V of velocity, and X of position. t is a scalar value representing the time step. Understanding this allows you to customize two important things: initial conditions and boundaries. By changing initial conditions, a huge variety of different shaped waves can be created. These can be put into the simulation by using them in the line where **xList** is declared and passing in **xIn** as a parameter. Any custom function that creates a matrix of the right size will work.

It is important that these initial conditions do not alter the walls. That is the purpose of the **replace** function, which is used by both **wavePlane** and **waveRadial** and should be used in creating any custom starting conditions.

The boundary conditions are applied in the line where **coordsList** is declared. This is a little bit more finicky – the edges of the simulation are required to be walls. This is because all moving particles need 4 neighbors, an important fact of

the optimization. Below are two different boundary conditions, **divider** and **slit** (with 24x24 particles and a radial wave):

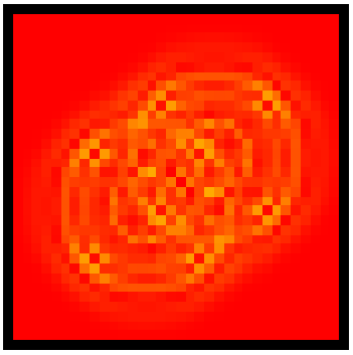


Applications: Some Interesting Starting Conditions

Below are a few examples of conditions that may be difficult or even impossible to model analytically, but can be progressed in time in my numerical model by simply changing a few parameters. Below each heading is the code that will be pasted into the *DynamicModule* function, replacing the lines where **coordsList** and **xList** are declared. All pictures are shown on a 36 by 36 grid. With a differently sized grid, the relative positions of the particles (and therefore the waves) might differ.

1. Two Radial Waves

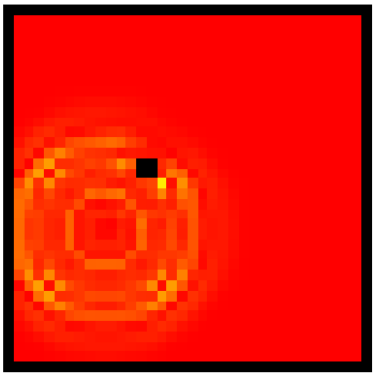
```
coordsList = basicWalls[size];
xList = waveRadialDouble[xIn[size],{15.5,15.5},{20.5,22.5},coordsList];
```



Radial waves are difficult to model in a square grid, requiring two-dimensional Fourier series or transforms (Stanford). However, pinching a roughly circular group of particles together creates a visually identical pulse quite well. In the above image, the superposition of the two waves can also be clearly seen.

2. Radial Wave with Small Barrier

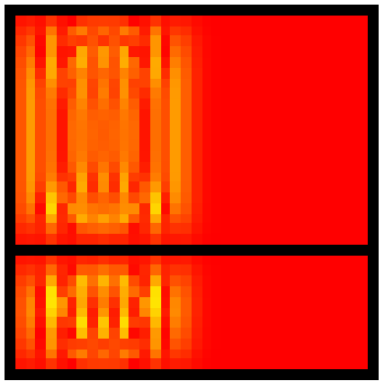
```
coordsList = point[size,20,14];
xList = waveRadial[xIn[size],14.5,10.5,coordsList];
```



In this example, a small obstacle is created that blocks the wave's progress. The wake of the object will roughly correspond to the angle that it subtends when the wave first collides with it. The wave is also reflected off the boundary at the walls.

3. Plane Wave with Divider

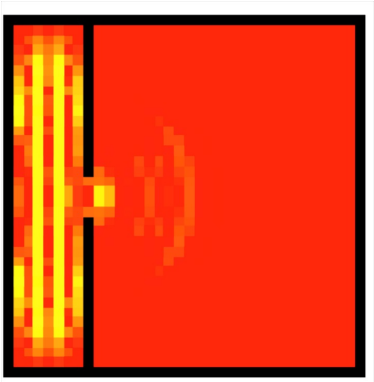
```
coordsList = divider[size,13,y];
xList = wavePlane[xIn[size],x,10,coordsList];
```



The above waves emerge from a single, transverse displacement in the y axis. The dispersion relation can clearly be seen from the fact that the wave has split up into several distinct higher-*k* peaks, as well as the effects of the boundary conditions. While the wave rapidly becomes chaotic, most of the oscillation is still in perpendicular to the initial direction of propagation.

4. Plane Wave with Slit

```
coordsList = slit[size,9,x,4];
xList = wavePlane[xIn[size],x,5,coordsList];
```



This example shows that plane waves are essentially composed of several radial waves, in many ways modeling the classical diffraction experiments. Whenever a wave passes through the slit, it becomes circular.

Conclusions

Unlike 1-dimensional systems, 2-dimensional waves have both many possible behaviors and many possible approaches to model them. In this paper, I have looked at just a few: first, an analytical model of transverse and longitudinal travelling waves, and second, a numerical model that allows for many different waveforms.

Each approach had distinct strengths and weaknesses. In the analytical model, precise quantities could easily be found for basic features: the dispersion relations and frequencies of travelling waves. However, it lacked boundary conditions and more complex motion.

The numerical model, on the other hand, theoretically allowed for any set of initial conditions. Unfortunately, several limitations were forced upon it by both computational restrictions and error propagation. The biggest among these were damping and boundary conditions on all four borders.

The numerical model also made it difficult to recover data. One of the reasons models like this one are often used for specific real-life systems is because there is no real way to mathematically evaluate their error. The best method is to compare them with whatever they are supposed to model.

Qualitatively, though, the advantages of the numerical model are quite clear. As a demonstrational tool, it excels in making clear principles like interference, diffraction, and reflection by animating them in real time. Because of this, a ripple tank is perhaps the best analogy; it is a visual aid for problems that can be quite complicated mathematically but may have very distinctive emergent behaviors.

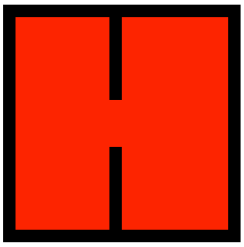
Appendix: User Guide

The program requires Wolfram Mathematica to run. The simulation.nb file it comes in provides the source code as well as the final simulator. Little Mathematica

knowledge is required to run the program, except the order in which to evaluate the functions, which is provided here. The keyboard shortcut Shift+Enter to evaluate is useful here.

After opening the program, you will see all of the code. Scrolling down will reveal a fully functional simulator:

```
In[16]:= DynamicModule[{t = .05, k = 10, b = .08, r = .25, size = 20},
          coordsList = slit[size, 10, x, 4];
          xList = wavePlane[xIn[size], x, 4, coordsList];
          vList = vIn[size];
          frameCount = 0;
          time = 1;
          DR = Raster[Table[{If[MemberQ[coordsList, {i, j}, 2], 1, 0], Norm[{i, j} - xList[[i, j]]] * 1.5, 0},
              {i, 1, size}, {j, 1, size}]];
          {DynamicWrapper[clock, c = Clock[Infinity]],
           Dynamic[vList = vList + t * aList[k, b, xList, vList, coordsList, size];
            xList = xList + t * vList;
            frameCount++;
            If[c > time, time++;
             fps = frameCount;
             frameCount = 0];
            Table[DR[[1, i, j, 2]] = Norm[{i, j} - xList[[i, j]]] * 1.5, {i, 1, size}, {j, 1, size}];
            Dynamic[Graphics[DR]]], Dynamic[fps]}, SaveDefinitions → True]
```

Out[16]= {clock,  , 14}

Outlined in blue is the main body of code that runs the simulator (this organizes all of the functions that calculate the individual values), and in red is the graphical representation of the simulator itself. The black pixels are the walls, and the red pixels are the medium in which waves propagate. The number on the right is frames per second.

If you want to change the conditions, you will have to change the area in blue. In order to do this, first select all and hit Shift+Enter to evaluate everything. Then, if you want to change a parameter, change the number, and hit Shift+Enter again. The main parameters can be seen in the first line of the function:

```
DynamicModule[{t = .1, k = 6, b = .05, r = .25, size = 20},
```

t represents the time step. Velocity and position are calculated numerically, by multiplying the instantaneous acceleration (for velocity) and velocity (for position) by this value. **k** represents the spring constant over mass, or the natural frequency squared. **b** represents the damping term. This cannot be set to 0, though it can be made very small. **size** is how many particles there are on each side. Changing size is a little different from changing the other parameters – changing it and evaluating while the function is running will cause Mathematica to crash. In order to stop evaluation, go under the *Evaluation* menu item and select *Dynamic Updating Enabled*. Then change the size value, evaluate, and turn on dynamic updating again. A few additional parameters exist, the starting conditions of the simulation:

```
coordsList = basicWalls[size];
xList = wavePlane[xIn[size], x, 5, coordsList];
vList = vIn[size];
```

These parameters are actually lists of points. In this specific example, **wavePlane** creates a 2 dimensional plane wave parallel to the **x** axis that originates at y=**4**. **vIn** is the default velocity list that sets the velocity of all particles to 0. **coordsList** specifies the boundary conditions: **basicWalls** is simply 4 walls at the edges of the screen. More complicated boundary conditions can be selected, such as **slit**.

Bibliography

Holt-Phoenix, Marianne S. "Wave Propagation in Finite Element and Mass-Spring Dashpot Models." June 2006. dSpace@MIT. 7 December 2013 <http://dspace.mit.edu/bitstream/handle/1721.1/35683/76882884.pdf?sequence=1>.

Kittel. "Physics 575 Lecture 26 and 27: Phonons." 2010. Oregon State Physics Department. 8 December 2013 <http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CFAQFjAE&url=http%3A%2F%2Fwww.physics.oregonstate.edu%2F~tate%2FCOURSES%2Fph575%2Flib%2Fexe%2Ffetch.php%3Fmedia%3Dl24.pdf&ei=_bOkUvv9AcysATDnIDIBg&usg=AFQjCNGgSKl4nR2thBeMtoIAa6CrhI3Jww&sig2=cKIhbQdyf-dLRoXBErmV-Q&bvm=bv.57752919,d.cWc>.

"n-Dimensional Fourier Transform." Stanford Engineering Everywhere. Stanford University. 18 December 2013 <http://see.stanford.edu/materials/lsoftaee261/chap8.pdf>.
(see page 359)

Osharovich, G. and M. Ayzenberg-Stepanenko. "On Resonant Waves in Lattices." ArXiv. Cornell University. 7 December 2013 <http://arxiv.org/pdf/1108.0357.pdf>.

Suzuki. "Lecture Note on Solid State Physics: Lattice Waves." 6 December 2007. University of Binghampton. 7 December 2013 <http://www2.binghamton.edu/physics/docs/lattice-waves.pdf>.