T³ Time Triangle Thing
Created by Carter Nelson

https://learn.adafruit.com/t-time-triangle-thing

Last updated on 2023-08-29 04:28:29 PM EDT
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
<td>3</td>
</tr>
<tr>
<td>• Hardware Items</td>
<td></td>
</tr>
<tr>
<td><strong>The Physics Model</strong></td>
<td>5</td>
</tr>
<tr>
<td>• LED Sand Physics</td>
<td></td>
</tr>
<tr>
<td>• Simplified Physics</td>
<td></td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>8</td>
</tr>
<tr>
<td>• Single Matrix Example</td>
<td></td>
</tr>
<tr>
<td>• Double Matrix Example</td>
<td></td>
</tr>
<tr>
<td>• Does This Work?</td>
<td></td>
</tr>
<tr>
<td><strong>Build It</strong></td>
<td>12</td>
</tr>
<tr>
<td>• Prepare the LED Matrices</td>
<td></td>
</tr>
<tr>
<td>• Wiring Diagram</td>
<td></td>
</tr>
<tr>
<td>• Making the Triangle</td>
<td></td>
</tr>
<tr>
<td><strong>Hourglass Code</strong></td>
<td>18</td>
</tr>
<tr>
<td>• Prepare the Feather Sense</td>
<td></td>
</tr>
<tr>
<td>• Hourglass Code</td>
<td></td>
</tr>
</tbody>
</table>
Overview

In this guide we'll show you how you can create a fun little physics based hourglass gadget. Hourglasses are used to track time, and this one ends up being sort of triangle shaped, so...yah, it's a Time Triangle Thing. This project was inspired by this awesome Instagram post by @david_proyectos.

We'll go over the underlying simplified physics model also. It could be a useful approach for some other gadgets. It's all written in CircuitPython.

Hardware Items

Below we link to the specific hardware items used in this guide. However, the key ingredients are:

- A reasonably powerful main board that can run CircuitPython and has I2C
- LED matrix or matrices
- Accelerometer

Therefore it's possible to come up with potentially different arrangements. For example, an accelerometer breakout could be used with a non-Feather board. Or smaller LED matrices could be used, etc. Also, the battery can be whatever size you want.
Adafruit Feather nRF52840 Sense
The Adafruit Feather Bluefruit Sense takes our popular Feather nRF52840 Express and adds a smorgasbord of sensors...
https://www.adafruit.com/product/4516

Adafruit Small 1.2" 8x8 LED Matrix w/I2C Backpack - Yellow-Green
What's better than a single LED? Lots of LEDs! A fun way to make a small display is to use an 8x8 matrix or a
https://www.adafruit.com/product/1051

Lithium Ion Polymer Battery - 3.7V 350mAh
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/2750
The physics model used here is really a simplification of the one used in the LED Matrix Sand Toy shown above. That project runs on a fairly large 64x64 LED matrix. It uses a Raspberry Pi for the processing, which gives it the computational power required. The code was written in C++, which also helps with the execution speed.

With the continued use of C++ (ala Arduino), a smaller version was even able to run on a Feather 32u4 in the Animated LED Sand project show below.
That code was ported to CircuitPython (with some effort) and the results are covered in the CircuitPython Digital Sand guide and shown below:

From that guide there is this commentary:

The first version of the python code could barely move 5 grains around in anything approaching a smooth speed. The current version does a reasonable job with 10 grains.

That was the motivation to try and see if things could be simplified even further to improve performance when using CircuitPython. First, let's review the "complex" model used on the projects mentioned above.

LED Sand Physics

Physics may be too strong a term. This is really just kinematics - how things move without worrying about the specifics of the forces involved. In our case, "force" comes from acceleration. That can be used to update the velocity of a given particle (code):

\[ v_t = v_{t-1} + a_t \]

And then the updated velocity can be used to update position (code):
The LED Matrix Sand projects above are based on these basic equations.

\[ X_t = X_{t-1} + V_t \]

The value for acceleration comes from the accelerometer, and everything else is computed from there, sand particle by sand particle, one time step at a time. The concept of terminal velocity is used as a way set a maximum upper bound on velocity. Additionally, collisions are accounted for by having particles "bounce" off each other. This is even done \textit{inelastically} \(^1\) to provide more realism.

In summary, the key items of the "complex" physics model are:

- Proper acceleration / velocity kinematics
- Enforcing terminal velocity
- Modeling inelastic collisions

All of that of course requires code and CPU cycle time to process. Which, if you have the processing power, is great. What if you don't?

**Simplified Physics**

So are all those kinematic details necessary? To provide realistic motion, yes. However, to actually appreciate the effects of that realism, the simulation needs to play out on a reasonably sized display, like a 64x64 matrix. For a smaller display, like an 8x8, there really isn't enough space for the realism to be seen. So what if we make some fairly sweeping and hand wavy simplifications?

- Just move one pixel in direction of current acceleration
- Attempt diagonal move for any collision

And that's it.
Move one pixel in direction of current acceleration - one of the red arrows.

For diagonal collisions, attempt to move left/right.

Will that work? Well...we weren't sure either. So we coded it up and tried it out. Let's see how it works with some basic breadboard examples.

Examples

Each code example is broken into two general parts:

- matrixsand.py - This contains a class which has the physics engine. You can't use it stand alone, you need to use it your application code. Also, you generally don't change anything in here.
- code.py - This is your application code. This is what you write. It relies on matrix sand.py.

Both files should be placed in the CIRCUITPY folder. Clicking the Download Project Bundle link for the code listing below will provide a .zip file that contains everything needed. Not only the two .py files, but also any require libraries - to be placed in CIRCUITPY/lib.
Single Matrix Example

Let’s start simple and just use a single 8x8 LED matrix. Wire up a matrix with address set to the default 0x70 as shown:

Click the Download Project Bundle button and save the .zip file. Then from the zip file, copy code.py and matrixsand.py to the CIRCUITPY folder. Also copy any libraries from the zip to CIRCUITPY/lib.

```python
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_lsm6ds.lsm6ds33
from adafruit_ht16k33 import matrix
import matrixsand

DELAY = 0.00 # add some delay if you want

# setup i2c
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C() # For using the built-in STEMMA QT connector on a microcontroller

# the accelo
accelo = adafruit_lsm6ds.lsm6ds33.LSM6DS33(i2c)

# the matrix
matrix = matrix.Matrix8x8(i2c, 0x70)

# the sand
sand = matrixsand.MatrixSand(8, 8)

# simple helper
def update_matrix():
    for x in range(8):
        for y in range(8):
            matrix[x,y] = sand[x,y]

# add some initial sand
for sx in range(4):
    for sy in range(4):
        sand[sx, sy] = 1
update_matrix()

# loop forever
while True:
```

©Adafruit Industries
```python
# read accelo
ax, ay, az = accelo.acceleration

# rotate coord sys
xx = ay
yy = ax
zz = az

# iterate the sand
updated = sand.iterate((xx, yy, zz))

# update matrix if needed
if updated:
    update_matrix()

# sleep
time.sleep(DELAY)
```

Once the code files and libraries have been copied over, the example should run. Now move the breadboard around and watch the grains of sand go!

Double Matrix Example

OK, now let’s try two 8x8 matrices to create a 16x8 area. Since the class that takes care of the physics is general purpose, we can set it up for different sizes. So adapting the code for different sizes is easy.

Wire up the two matrices as shown below, with address 0x71 on the left and address 0x70 on the right.
Same as above, click the Download Project Bundle button to get the .zip and copy the files as needed to the CIRCUITPY folder.

```python
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_lsm6ds.lsm6ds33
from adafruit_ht16k33 import matrix
import matrixsand

DELAY = 0.00  # add some delay if you want

# setup i2c
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller

# the accelo
accelo = adafruit_lsm6ds.lsm6ds33.LSM6DS33(i2c)

# the matrix
matrix1 = matrix.Matrix8x8(i2c, 0x70)
matrix2 = matrix.Matrix8x8(i2c, 0x71)

# the sand
sand = matrixsand.MatrixSand(8, 16)

# simple helper
def update_matrix():
    for x in range(8):
        for y in range(16):
            if y < 8:
                matrix1[x, y] = sand[x, y]
            else:
                matrix2[x, y-8] = sand[x, y]

# add some initial sand
for sx in range(4):
    for sy in range(4):
        sand[sx, sy] = 1

update_matrix()

# loop forever
while True:
    # read accelo
    ax, ay, az = accelo.acceleration

    # rotate coord sys
    xx = ay
    yy = ax
```
zz = az

# iterate the sand
updated = sand.iterate((xx, yy, zz))

# update matrix if needed
if updated:
  update_matrix()

# sleep
time.sleep(Delay)

Now move the breadboard around and watch the grains of sand go!

Does This Work?

So...does this approach work? Well, as you can see from running the examples above, it's not perfect. Things move in sort of a klunky fashion and they can stack up in odd ways. Also, the lack of true kinematics seems to start to become apparent with 16 pixels to move.

But it does provide a general sense of the actual physics. It's reasonably fun to move the examples above around and watch the "sand" move about. So, sure, it works. Just not perfectly. But it's a good option for smaller displays where the full physics are not necessarily warranted.

Let's now use it to make the Time Triangle Thing.

Build It

Here are the details for building the Time Triangle Thing.
Prepare the LED Matrices

Follow this guide for the general assembly of the 8x8 LED matrices:

LED Matrix Assembly

You'll also need to change the I2C address of one of the matrices. See this part of the guide:

Changing I2C Address

and solder the A0 jumper on one of the LEDs matrices to set the address to 0x71. That way we will end up with these addresses for our matrices:

- Matrix 1 = 0x70 (default, no solder)
- Matrix 2 = 0x71

Wiring Diagram

Here's how everything gets wired together. The 0x70 address matrix in on top and the 0x71 address matrix in on the bottom. Also reference the photos below in the general build.
Making the Triangle

Let's use some paper craft skills to make our Time Triangle Thing. You'll need a piece of cardboard that is 9" x 6". The bottom of an Adafruit shipping box can be used, which is what is shown here. But any scrap cardboard should do.

Gather the items you'll need:

- A source for the cardboard
- A pencil and a ruler
- Something to cut with
- Some tape
Cut the cardboard into a 9" wide by 6" tall rectangle.
Mark vertical lines at 3", 6", and 7.5" from the left edge.
Mark a horizontal line 3" up from the bottom as shown in the far right area.

Place the components down and use them to mark the cut outs.
Use the LEDs themselves (not the PCB edges) to mark square cut outs.
Use the Feather to mark cut outs for the headers to poke through.

This is how the markings should end up looking.
The small circle by the Feather is for passing through the battery cable.

Neatness counts for the LED cut outs. So take your time with those.
CAREFULLY cut out the areas.

Bend the cardboard at the 3" and 6" lines using the edge of the ruler to help make it nice and sharp.

It should look something like this.
Tape the components in place. Note the orientation of the LED matrices. The top one should be 0x70 and the bottom one should be 0x71. The Feather is on the outside and pokes through the slots.

Wire everything together. Consult the wiring diagram.

Now close everything up into the eponymous triangle.
Tape along the edge to close it up.

DONE!

Now move on to loading the code...

Hourglass Code

Now let's get the Feather setup with CircuitPython, the necessary libraries, and the hourglass code.

Prepare the Feather Sense

Follow this guide for setting up CircuitPython on the Feather nRF52840 Sense:
Hourglass Code

Click the Download Project Bundle button in the code listing below to get a zip file with the project code and necessary libraries. Copy code.py and matrixsand.py to the CIRCUITPY folder.

Also copy the libraries to CIRCUITPY/lib. For quick reference, the CIRCUITPY/lib folder should have these contents (having more is OK):

![File list](image)

Here is the main code listing for code.py. The matrixsand.py file will be found in the zip, along with everything else. Click the Download Project Bundle to get the zip.

```python
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_lsm6ds.lsm6ds33
from adafruit_ht16k33 import matrix
import matrixsand

DELAY = 0.05 # overall update rate

# setup i2c
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller

# the accelo
accelo = adafruit_lsm6ds.lsm6ds33.LSM6DS33(i2c)

# the matrices
m1 = matrix.Matrix8x8(i2c, 0x70)
m2 = matrix.Matrix8x8(i2c, 0x71)

# the sand
sand1 = matrixsand.MatrixSand(8, 8)
sand2 = matrixsand.MatrixSand(8, 8)

# simple helper
def update_matrix(m, s):
    for x in range(8):
        for y in range(8):
            m[x,y] = s[x,y]

# fill up some sand
for sx in range(8):
    for sy in range(8):
        sand1[sx, sy] = True
sand1[0,0] = sand1[0,1] = sand1[1,0] = False
```
sand1[0,2] = sand1[1,1] = sand1[2,0] = False
update_matrix(m1, sand1)
update_matrix(m2, sand2)
updated1 = updated2 = False

while True:
    # read accelo
    ax, ay, az = accelo.acceleration
    # rotate coords
    xx = -ax - ay
    yy = -ax + ay
    zz = az

    # move grain of sand from upper to lower?
    if yy > 0 and sand1[7,7] and not sand2[0,0] and not updated2:
        sand1[7,7] = False
        sand2[0,0] = True
        updated1 = updated2 = True

    # move grain of sand from lower to upper?
    elif yy <= 0 and sand2[0,0] and not sand1[7,7] and not updated1:
        sand2[0,0] = False
        sand1[7,7] = True
        updated1 = updated2 = True

    # nope, just a regular update
    else:
        updated1 = sand1.iterate((xx, yy, zz))
        updated2 = sand2.iterate((xx, yy, zz))

    # update matrices if needed
    if updated1:
        update_matrix(m1, sand1)
    if updated2:
        update_matrix(m2, sand2)

time.sleep(DELAY)