National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester II, 2020/2021

Mission 2 - Contest
3D Runes

Release date: 24 January 2022
Due: 18 February 2022, 23:59

Required Files

- contest02.3-template.py
- runes.py
- graphics.py
- PyGif.py
- PIM.png

Background

You have became adept as a PIM apprentice but so are many others like yourself. With everyone attempting to prove themselves superior, it is certain unhealthy rivalry will form amongst the fresh apprentices.

But the masters have already foreseen this problem through the many generations of PIM mages they have trained. Initially masquerading as a rumour, news of the semi-annual rune conjuring contest quickly became the hottest of discussion topics.

With exquisite and intricate winning runes being displayed prominently in the grand hall and the hustle and bustle of preparation, you barely managed to get hold of a trainer to get the details. Clearly, it was not intended for all apprentices to participate but only those possessing true passion and are pure of essence. Do you have what it takes?

Task

This contest represents the 3D runes segment of the annual rune conjuring contest which you may participate in.

Being masters of rune manipulation, you are to use your creativity and design some textured and contoured runes.

You may submit up to three 3D runes in separate files. Submit your entries by writing each entry as a function in the template file provided and upload the files on Coursemology.

Please follow the following naming convention when submitting your files.
`<Name-On-Coursemology>-<Entry-Number>-3d.py` For example these 3 entries from the same person should read:
`Leong-Wai-Kay-1-3d.py`
`Leong-Wai-Kay-2-3d.py`
`Leong-Wai-Kay-3-3d.py`

Please remember to zip up all your files while uploading!

Additional instructions:

1. Ensure that your last line in each submitted file is an uncommented `stereogram`, `anaglyph` or `hollusion` function call, so that each file displays the rune when run in IDLE. If you used external images in your code, please upload the image files along with your submission.
2. You may not create your own runes and use it to generate images. You can only manipulate the runes provided in `runes.py` to generate an image consisting of these runes. (That is to say, the basic building blocks of your image should come from runes provided in `runes.py`.)
3. All functions provided in the permitted modules are allowed.
4. You may not import any other external modules beyond the following:
   (a) `graphics` (from CS1010X)
   (b) `PyGif` (from CS1010X)
   (c) `runes` (from CS1010X)
   (d) `math` (from the default Python library)

   All other modules are prohibited.
5. Your script should call the `stereogram/anaglyph/hollusion` function at the end of its running process, i.e., the assessor should be able to immediately generate the working image by running the script, without further modification to your script.
6. There is no file size limit to the file, apart from the limitation of Coursemology. However, running time to generate your image should be within a reasonable timeframe. (We should not need to wait > 10 minutes for the image to load.)
7. You are free to use external images in your program, but you should show some effort in manipulating the image.

Hint: You may want to check out the Appendices before you start working on this contest.

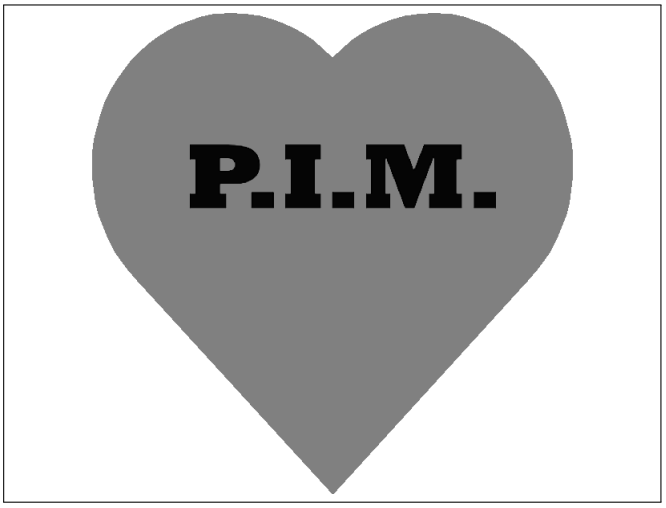
Appendix: `image_to_painter` and `function_to_painter`

In this section we provide you with two more tools that you can use to create depth map (that in turns can be used to generate stereogram). They are `image->painter` and `function->painter`.

The `image->painter` converts a given 24-bit bitmap image into a depth map painter. For example, the command:

```
pim = image_to_painter("PIM.png")
show(overlay(pim, heart_bb))
```

will produce the following depth map (notice that "P.I.M." comes from the image file `PIM.png`).



If we use `stereogram` instead of `show`, we got the stereogram shown in Figure 1. On the other hand, using `anaglyph` would get us the anaglyph shown in Figure 2. Remember that you can also use `hollusion` to create 3D runes. (A hollusion is not shown as it cannot actually be viewed in the .pdf.)

Figure 1: *Stereogram created from overlaying "P.I.M." (generated by* `image_to_painter`*) and* `heart_bb`*.*

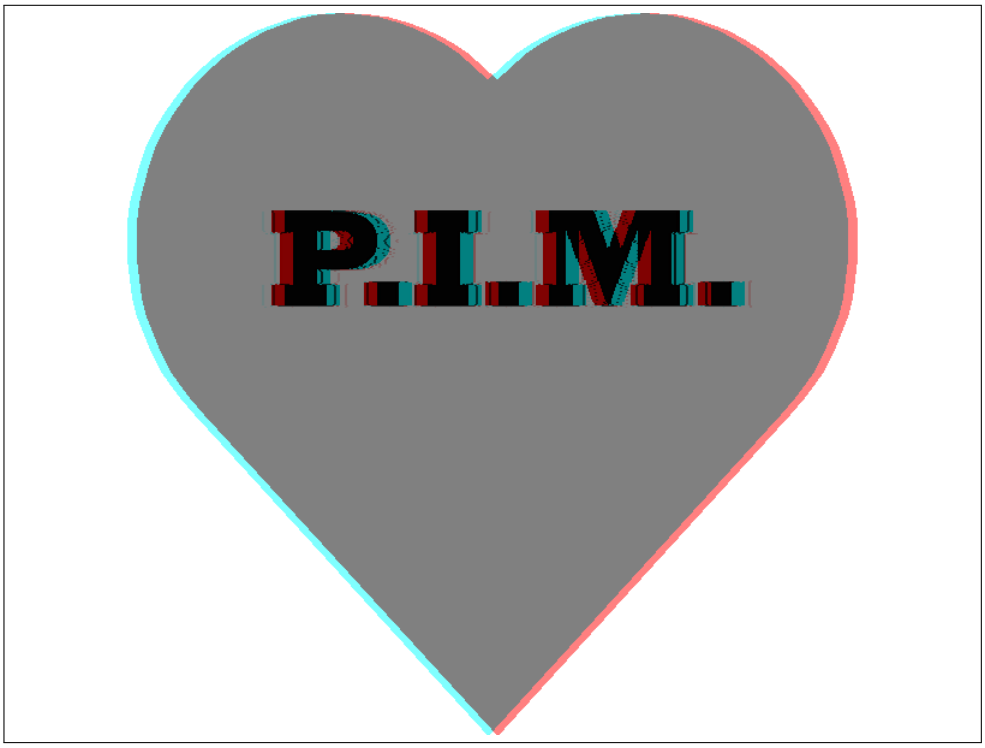


Figure 2: *Anaglyph created from overlaying "P.I.M." (generated by* `image_to_painter`*) and* `heart_bb`*.*

Note: `image_to_painter` expects the input image to be 600×600 pixels in size. Bigger

images may be cropped. Also, loading the image might take quite some time, please be patient. (:

Lastly, we shall introduce `function_to_painter`. As mentioned in Appendix A, a depth map can be seen as a visualization of a z-function. A z-function is a function that, given a point (x, y) will return the depth of the object at that point z, $0 \leq z \leq 1$. `function->painter` accepts such z-function and convert it to a depth map painter. Note that the passed function must take two parameters x and y, $0 \leq x, y \leq 600$ (you might ask: "Why 600?" Can you guess why? See footnote for answer[1]). `function_to_painter` samples the given z-function at intervals of x and y (you may use this fact to aid you in creating the z-function). The following example shows a combination advanced techniques that may help in creating a z-function easily. While the example simply creates two concentric circles, it involves translation of the origin (to the point $(300, 300)$) and using comparison operator to easily specify a range of (x, y) that returns the same value. The code should be self-documenting enough that you should be able to read it easily (knowing that equation of a circle centred at the origin with radius a is given by $x^2 + y^2 = a^2$).

Note that radius1 should be $<$ than radius2. Can you see why? How would you modify this such that the requirement is no longer necessary?

```
def create_conc_circle_zf(radius1, depth1, radius2, depth2):
    def square(x):
        return x * x

    a1_sq = square(radius1)
    a2_sq = square(radius2)
    def helper(x, y):
        d_sq = square(x - 300) + square(y - 300)
        if d_sq < a1_sq:
             return depth1
        elif d_sq < a2_sq:
             return depth2
        else:
             return 1
    return helper

show(function_to_painter(create_conc_circle_zf(90, 1/3, 270, 2/3)))
```

This will result in the following depth map. Try it for yourself!

[1]The reason is that the viewport height is 600 pixels; most of the patterns in quilts are generated taking this into accounts. If we were to sample less, the image would be more grainy. Also, a square painter is easier to do than a rectangle.