National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester II, 2022/2023

Contest 2.3
3D Runes

Release date: 26th January 2023
Due: 5th February 2023, 23:59

Required Files

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

Background

You have become 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/contoured runes. Simply define your runes in the template functions provided.

You may submit up to three 3D runes. Submit your entries by including the code for each rune in the corresponding template function. If you are submitting less than three runes, leave the extra functions empty.

Just remember to submit and finalise your submission when you are done!

Additional notes: Please submit your runes in the order of Entry 1, 2, 3, if you are submitting less than 3 runes. Please do not submit duplicate runes. You are also not allowed to use external images in your entry.

Warning! The **runes.py** and **graphics.py** files are not to be edited. As such, you are **only** allowed to use runes that are already made available in the **runes.py** file. In particular, you are not allowed to define your own runes to accommodate for colour.

You should simply return the runes instead of showing the runes. Here's an example.

WRONG

```
def rune_entry_1():
    your_rune = ...
    return anaglyph(your_rune)
choice1 = None
```

CORRECT

```
def rune_entry_1():
    your_rune = ...
    return your_rune # Return the rune instead of showing it!

# Replace None with anaglyph/hollusion/stereogram
# Do not edit the choice variable name!
choice1 = anaglyph # Put the function here, without any brackets
```

Hint: You may want to check out the Appendices before you start working on this contest, including those on other rune-related missions or side quests.

Viewing your rune:

To view your rune, individually uncomment the line that calls your rune function. For example, if you want to view your rune in Entry 1, uncomment the line

```
anaglyph(rune_entry_1())
#hollusion(rune_entry_1())
#stereogram(rune_entry_1())
```

Warning! Do not uncomment more than one rune at a time, if you do you might not see the rune you want to see. If you want to view your rune in Entry 2, uncomment the statement for Entry 2 and comment Entry 1.

Viewing Your Rune as a file:

[Optional] For advanced users only! Often times if your rune is extra large, our graphics module will not be able to save an image file that matches what you see on your screen. In this case you might want to download your rune as a file to check what we will be uploading. To do this, simply uncomment the respective following lines to the end of your rune function:

```
save_anaglyph('rune_entry_1')
#save_hollusion('rune_entry_1')
#save_stereogram('rune_entry_1')
```

You will have to delete the previous copy of the image if you want to save a new copy of the image again.

Appendix: `function_to_painter`

In this section we provide you with a tool that you can use to create depth maps (that in turn can be used to generate stereograms). This tool is `function->painter`.

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 $x^2 + y^2 = a^2$).

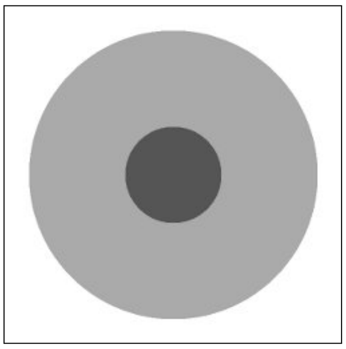
Note that radius1 should be < than radius2. Can you see why? How would you modify this such that the requirement is no longer necessary?

```python
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.