National University of Singapore
School of Computing
CS1101S: Programming Methodology (JavaScript)
Semester I, 2012/2013

Contest 3-3
3D Runes

Start date: 23 August 2012
Due: 3 September 2012, 23:59

Readings:

- Textbook Sections 1.1.1 to 1.1.4

Background:

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

But the masters have already foreseen this problem through the many generations of JFDI knights they had 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 an instructor to get the details. Clearly, it was not intended for all initiates to participate but only those possessing true passion and are pure of essence. Are you?

Task 1:

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 3 separate 3D runes. Each rune should be submitted in its own source file such that by running the JavaScript code in the file with its respective HTML file, the rune will be created in the viewport.

More information on the contest, such as judging criteria, will be provided as the due date approches.

Additional Instructions: The goal of this contest is to build cool 3D runes. Your submission must either be a stereogram, anaglyph or hollusion. Please state clearly in your submission which of these three forms your submission is meant to be.

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

Task Files

- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/runes.js
- contest_3-3_1.html
- **contest_3-3_1.js**

Submission

To submit your work to the Academy, click the upload button on the mission page and upload your rune source files. Please name the files in the format yourname_3d_contest_x.js where x is from 0 to 2. After you have uploaded all your files, click "Finalize Submission". Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.

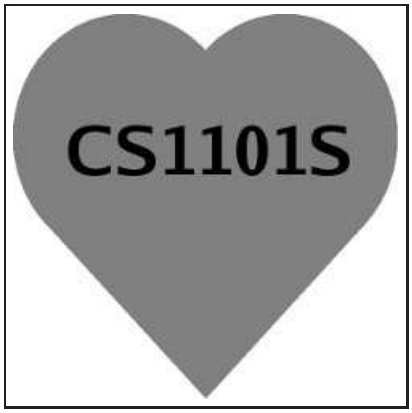
Appendix: `image_to_painter` **and** `function_to_painter`

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_to_painter` and `function_to_painter`.

The `image_to_painter` converts a given image into a depth map painter. For example, the command:

```
var cs1101s = image_to_painter("cs1101s.jpg");
show(overlay(cs1101s, heart_bb));
```

will produce the following depth map (notice that "CS1101S" comes from the file `cs1101s.jpg`).



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)

Note: `image_to_painter` expects the input image to be 600×600 pixels in size. Bigger img/contest/3.3 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_to_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 integer interval 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 $x^2 + y^2 = a^2$).

[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.

Figure 1: *Stereogram created from overlaying "cs1101s" (generated by* `image_to_painter`*) and* `heart_bb`*.*

```
// Note that radius1 should be less than radius2. Can you see why?
// How would you modify this such that the requirement is no longer necessary?
function create_conc_circle_zf(radius1, depth1, radius2, depth2){
    var a1sq = Math.pow(radius1, 2);
    var a2sq = Math.pow(radius2, 2);
    return function(x, y){
                function helper(x, y){
                    var sqsum = Math.pow(x, 2) + Math.pow(y, 2);
                    if(sqsum < a1sq){
                        return depth1;
                    }else if(sqsum < a2sq){
                        return depth2;
                    }else{
                        return 1;
                    }
                }
                return helper(x - 300, y - 300);
            }
}

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

This will result in the following depth map.

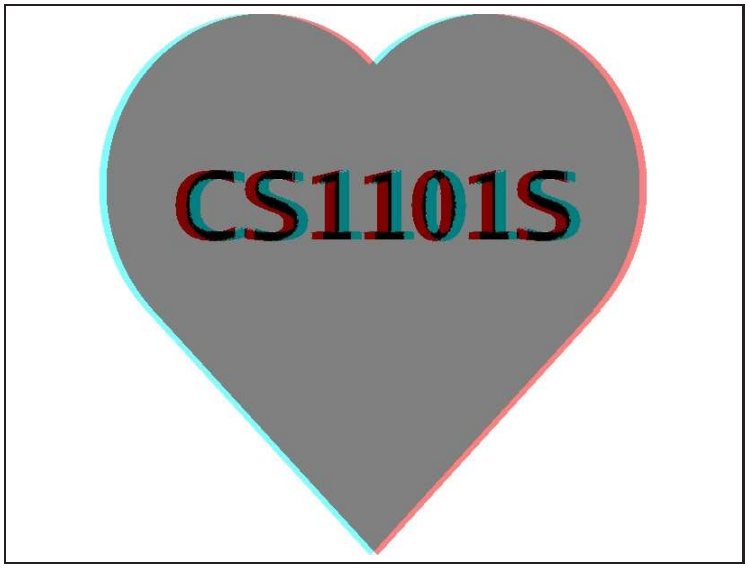


Figure 2: *Anaglyph created from overlaying "cs1101s" (generated by* `image_to_painter`*) and* `heart_bb`*.*