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

Mission 6
Gosperize

Start date: 01 September 2012
Due: 05 September 2012, 23:59

Readings:

- Textbook Sections 1.3.2

Background:

After days of training, the disciples are now able to freely conjure up curves. However, simple curves are not effective in warding of Force attacks.

Grandmaster Martin explains that there is an interesting way to make the curves much more formidable. Through Gosperization, images of these curves can be multiplied to greatly increase the strength of defence. However, he cautions that the process of Gosperization should be done quickly, so as not to waste the efficiency of the mind.

"Good. The technique of Force comes in many forms, not just from a single source. Now multiply your image in your head, with the images coming in from different angles. Make sure the images come quickly into your mind..."

This mission has **three** tasks.

Task 1:

To show off the power of our drawing language, let's use it to explore fractal curves. Fractals have striking mathematical properties[1]. Fractals have received a lot of attention over the past few years, partly because they tend to arise in the theory of nonlinear differential equations, but also because they are pretty, and their finite approximations can be easily generated with recursive computer programs.

For example, Bill Gosper[2] discovered that the infinite repetition of a very simple process creates a rather beautiful image, now called the *Gosper C Curve*. At each step of this process there is an

[1] A fractal curve is a "curve" which, if you expand any small piece of it, you get something similar to the original. The Gosper curve, for example, is neither a true 1-dimensional curve, nor a 2-dimensional region of the plane, but rather something in between.

[2] Bill Gosper is a mathematician now living in California. He was one of the original hackers who worked for Marvin Minsky in the MIT Artificial Intelligence Laboratory during the '60s. He is perhaps best known for his work on the Conway Game of Life—a set of rules for evolving cellular automata. Gosper invented the "glider gun", resolving Conway's question as to whether it is possible to produce a finite pattern that evolves into an unlimited number of live cells. He used this result to prove that the Game of Life is Turing universal, in that it can be used to simulate any other computational process!

approximation to the Gosper curve. The next approximation is obtained by adjoining two scaled copies of the current approximation, each rotated by 45 degrees.

Figure 1 shows the first few approximations to the Gosper curve, where we stop after a certain number of levels: a level-0 curve is simply a straight line; a level-1 curve consists of two level-0 curves; a level 2 curve consists of two level-1 curves, and so on. The figure also illustrates a recursive strategy for making the next level of approximation: a level-n curve is made from two level-$(n-1)$ curves, each scaled to be $\sqrt{2}/2$ times the length of the original curve. One of the component curves is rotated by $\pi/4$ (45 degrees) and the other is rotated by $-\pi/4$. After each piece is scaled and rotated, it must be translated so that the ending point of the first piece is continuous with the starting point of the second piece.

We assume that the approximation we are given to improve (named `curve` in the function) is in standard position. By doing some geometry, you can figure out that the second curve, after being scaled and rotated, must be translated right by .5 and up by .5, so its beginning coincides with the endpoint of the rotated, scaled first curve.

This leads to the Curve-Transform `gosperize`:

```
function gosperize(curve){
    var scaled_curve = scale(Math.sqrt(2) / 2)(curve);
    return connect_rigidly(rotate_around_origin(Math.PI / 4)(scaled_curve),
                           translate(0.5, 0.5)
                             (rotate_around_origin(-Math.PI / 4)(scaled_curve)));
}
```

Now we can generate approximations at any level to the Gosper curve by repeatedly gosperizing the unit line,

```
function gosper_curve(level){
    return repeated(gosperize, level)(unit_line);
}
```

To look at the level `level` gosper curve, evaluate `show_connected_gosper(level)`:

```
function show_connected_gosper(level){
    draw_connected(200)
      (squeeze_rectangular_portion(-0.5, 1.5, -0.5, 1.5)
        (gosper_curve(level)));
}
```

Your Task:

Define a function `show_points_gosper` such that evaluation of

```
show_points_gosper(level, number_of_points, initial_curve);
```

will plot `number_of_points` unconnected points of the level `level` gosper curve, but starting the gosper curve approximation with an arbitrary `initial_curve` rather than the unit line. For instance,

```
show_points_gosper(level, 200, unit_line);
```

should display the same points as `show_connected_gosper(level)`, but without connecting them. But you should also be able to use your function with arbitrary curves. (You can find the description of the function `squeeze_rectangular_portion` in the file `hi_graph_ce.js`; you don't need to understand it in detail to do this task.)

Test your code by gosperizing the arc of the unit circle running from 0 to π. Find some examples that produce interesting designs (You may also want to change the scale in the plotting window and the density of points plotted)[3]. Some sample tests are given in Figure 2.

Task Files

- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/hi_graph_ce.js
- mission_6_1.html
- **mission_6_1.js**

Task 2:

The Gosper fractals we have been playing with have had the angle of rotation fixed at 45 degrees. This angle need not be fixed. It need not even be the same for every step of the process. Many interesting shapes can be created by changing the angle from step to step.

We can define a function `param_gosper` that generates Gosper curves with changing angles. This function takes a level number (the number of levels to repeat the process) and a second argument called `angle_at`. The function `angle_at` should take one argument, the level number, and return an angle (measured in radians) as its answer:

$$\texttt{angle-at} : \text{JS-Nonneg-Int} \rightarrow \text{JS-Num}.$$

The function `param_gosper` can use this to calculate the angle to be used at each step of the recursion.

```
function param_gosper(level, angle_at){
    if(level === 0){
        return unit_line;
    }else{
        return param_gosperize(angle_at(level))(param_gosper(level - 1, angle_at));
    }
}
```

The function `param_gosperize` is almost like `gosperize`, except that it takes an another argument, the angle of rotation:

[3] One of the things you should notice is that, for larger values of n, all of these curves look pretty much the same. As with many fractal curves, the shape of the Gosper curve is determined by the Gosper process itself, rather than the particular shape we use as a starting point. In a sense that can be made mathematically precise, the "infinite level" Gosper curve is a fixed point of the Gosper process, and repeated applications of the process will converge to this fixed point.

```
function param_gosperize(theta){
    return function(curve){
                var scale_factor = 1 / Math.cos(theta) / 2;
                var scaled_curve = scale(scale_factor)(curve);
                return connect_rigidly(rotate_around_origin(theta)(scaled_curve),
                                       translate(0.5, Math.sin(theta) * scale_factor)
                                         (rotate_around_origin(-theta)(scaled_curve)));
            }
}
```

For example, the ordinary Gosper curve at level `level` is returned by

```
param_gosper(level, function(level){ return Math.PI/4; })
```

Designing `param_gosperize` required using some elementary trigonometry to figure out how to shift the pieces around so that they fit together after scaling and rotating. It's easier to program if we let the computer figure out how to do the shifting.

One convenient Curve-Transform is `rotate_around_origin`. Basically,

```
function rotate_around_origin(theta)(arg_curve)
```

will return a curve obtained by rotating `arg_curve` around the origin for an angle of `theta`.

Another convenient Curve-Transform is `put_in_standard_position`. We'll say a curve is in *standard position* if its start and end points are the same as the unit line, namely it starts at the origin $(0,0)$, and ends at the point $(1,0)$. We can put any curve whose start and endpoints are not the same into standard position by rigidly translating it so its starting point is at the origin, then rotating it about the origin to put its endpoint on the x axis, then scaling it to put the endpoint at $(1,0)$:

```
function put_in_standard_position(curve){
    var start_point = curve(0);
    var curve_started_at_origin = translate(-x_of(start_point),
                                            -y_of(start_point))(curve);
    var new_end_point = curve_started_at_origin(1);
    var theta = Math.atan(y_of(new_end_point), x_of(new_end_point));
    var curve_ended_at_x_axis = rotate_around_origin(-theta)
                                  (curve_started_at_origin);
    var end_point_on_x_axis = x_of(curve_ended_at_x_axis(1));
    return scale(1 / end_point_on_x_axis)(curve_ended_at_x_axis);
}
```

Your Task:

Show how to redefine `param_gosperize` using the functions `put_in_standard_position` and `connect_ends` to handle the trigonometry [4]. In order to avoid naming conflicts, you will be using the function name `your_param_gosperize` instead of the original `param_gosperize`.

Your definition should be of the form

```
function your_param_gosperize(theta){
    return function(curve){
                return put_in_standard_position(connect_ends(...,
                                                             ...));
            }
}
```

[4] your_param_gosperize must give the same output as the definition of param_gosperize above.

To test your code, generate some parameterized Gosper curves where the angle changes with the level n. The function `your_param_gosper` has been defined for you and it uses `your_param_gosperize` for the purpose of testing. Sample answers are shown in Figure 3.

Task Files

- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/hi_graph_ce.js
- mission_6_2.html
- **mission_6_2.js**

Task 3:

We now have three functions to compute gosper curves:

- `gosper_curve`
- `param_gosper` with argument `function(level){ return Math.PI / 4; }` using the "hand-crafted" definition of `param_gosperize` above
- `your_param_gosper` in Task 2 that uses `your_param_gosperize` based on `put_in_standard_position`.

The JavaScript `Date` object's `getTime` function can be used to report the elapsed time in milliseconds required to evaluate a series of statements. This function returns the POSIX time[5] at the point when it is called. A `Date` object can constructed and queried for its POSIX time. Another `Date` object can be constructed and queried again later, and the difference between the two recorded times tells the elapsed time between the two statements.

For example, evaluating

```
var start = (new Date()).getTime();
gosper_curve(10000)(0.1);
var end = (new Date()).getTime();
console.log("Elapsed Time: " + (end - start));
```

will print out the time to compute the point at .1 on the level 10000 gosper_curve. Be warned that the resolution of `getTime` is not very high. The accuracy of any elapsed time values below 20 milliseconds is suspect.

Your Task:

Compare the time measurements of these functions for computing selected points on the curve at a **significant level**. [6] Do this for at least 5 times to take the average time measurements for each of the functions for more accurate results. Present your findings in a neat and organized manner.

Use your results to conclude if there is a speed advantage for the more customized functions. [7]

[5] POSIX time is a system based on the amount of milliseconds that have elapsed since January 1, 1970.

[6] Significant level refers to one that does not take a ridiculous amount of time to perform the function, but yet enough to show the difference in time measurements between the different functions.

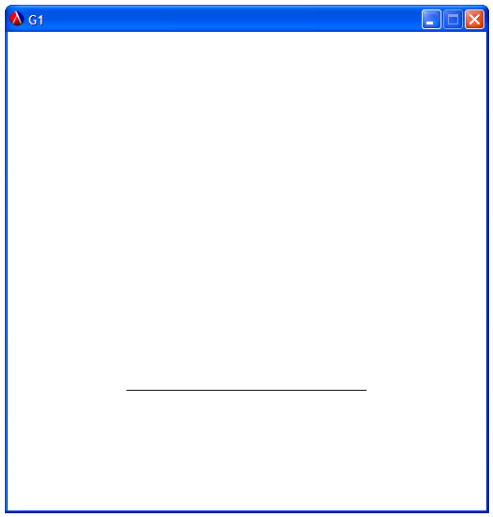
[7] A more customized function here refers to a function that delves in more details and have a lower level of abstraction.

Task Files

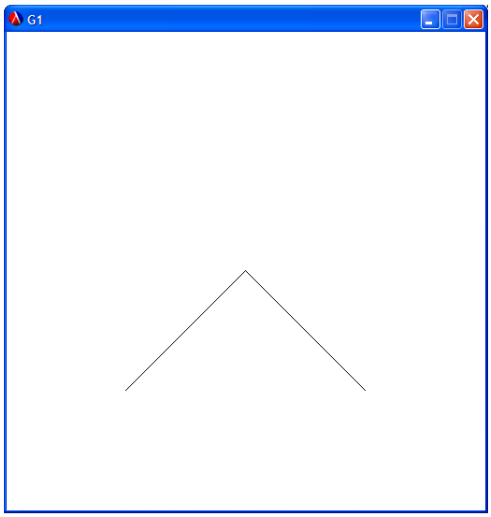
- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/hi_graph_ce.js
- mission_6_3.html
- **mission_6_3.js**

Submission

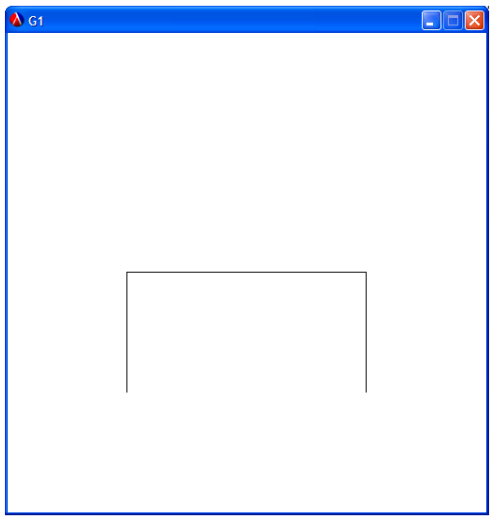
To submit your work to the Academy, copy the contents from the template file(s) into the box that says "Your submission" on the mission page, click "Save Code", then click "Finalize Submission". Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.



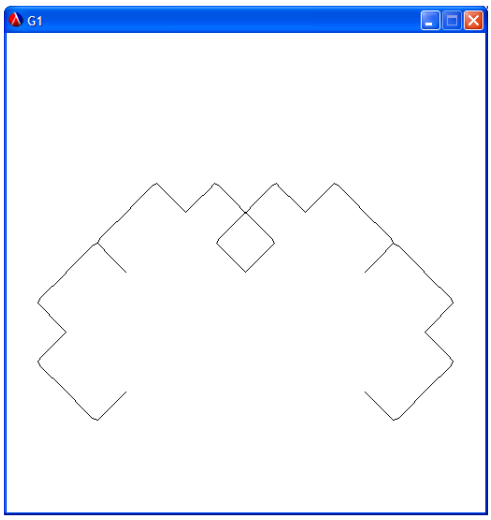
show_connected_gosper(0);



show_connected_gosper(1);

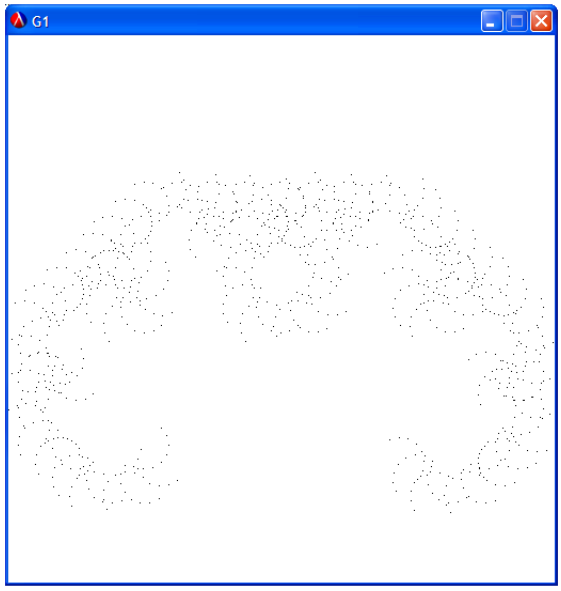


show_connected_gosper(2);

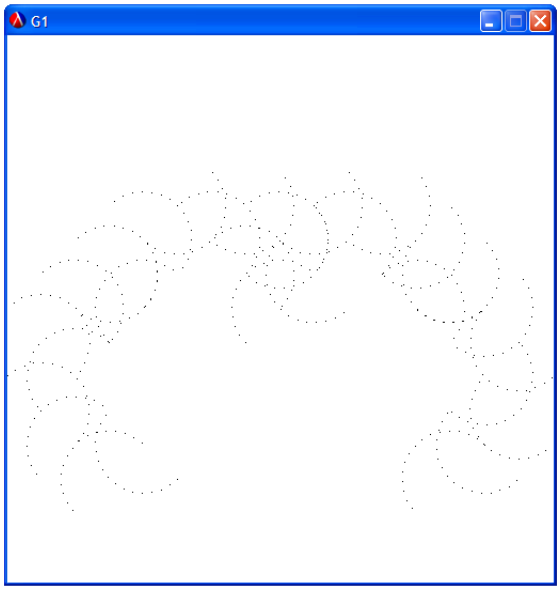


show_connected_gosper(5);

Figure 1: Samples for Gosper Curve.

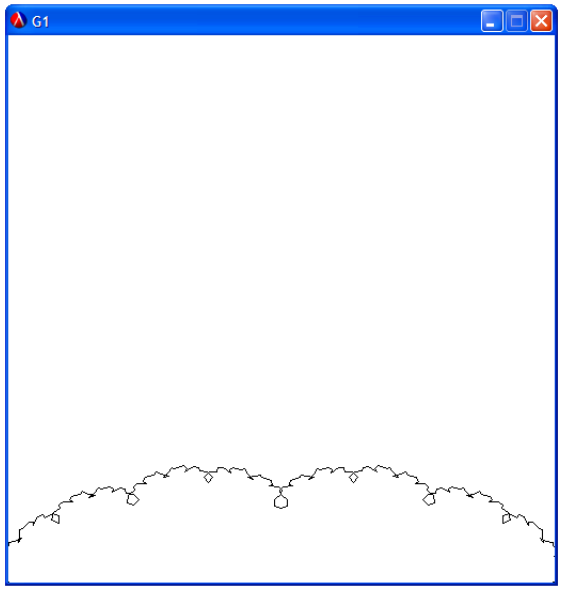


show_points_gosper(7, 1000, arc);

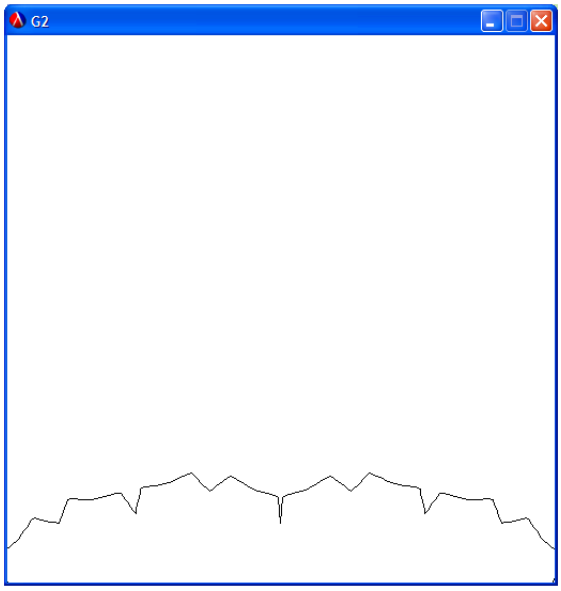


show_points_gosper(5, 500, arc);

Figure 2: Sample Tests for Task 1.



draw_connected(200)(your_param_gosper
(10, function(n) return Math.PI / (n + 2);));



draw_connected(200) (your_param_gosper
(5, function(n) return Math.PI / 4 / Math.pow(1.3, n);));

Figure 3: Sample answers for Task 2.