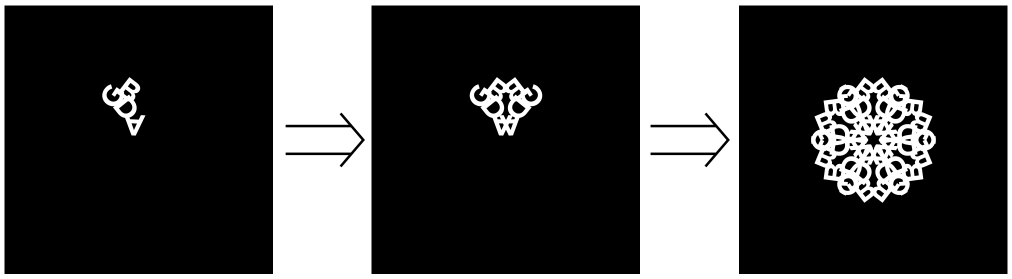
CS 116x Winter 2015
Craig S. Kaplan

Assignment 05: Geometric context

Due date: Wednesday, 25 February, 12:00pm

Question 1: Typographical snowflakes



A snowflake is composed of six evenly-rotated arms, all identical. Each arm in turn is made up of left and right halves that are mirror images of each other, as in the middle image above. Each half of that arm is some arbitrary shape, such as the jumbled letters "GBDA" on the left. We can neatly encode this hierarchical structure using layers of geometric contexts. In this exercise you will design a snowflake by overlapping letterforms to create an abstract geometric design.

Download the starter sketch `A05_1.zip` from the course web page. The sketch sets up a simple framework with an empty `drawFlake()` function. Your job is to fill in this function, and add helper functions, in order to draw a single snowflake centred in the sketch window.

Your snowflake must be made up of overlapping letterforms, in white on a black background. In other words, you must jumble some letters together to make half an arm, and then use geometric context to reflect and rotate it. You can choose any letters, layout, font and size, with the following minimal requirements:

- You need to have at least two distinct letters.
- The snowflake as a whole must be a single connected shape.
- Your sketch must contain exactly one use of the built-in `text()` function for each letter.

That last requirement forces you to use hierarchical modelling to build up the repetitive structure of a snowflake. To accomplish this, you could follow a process similar to the way we developed the house drawing in lecture, as in the `HierarchicalStreet` sample sketch:

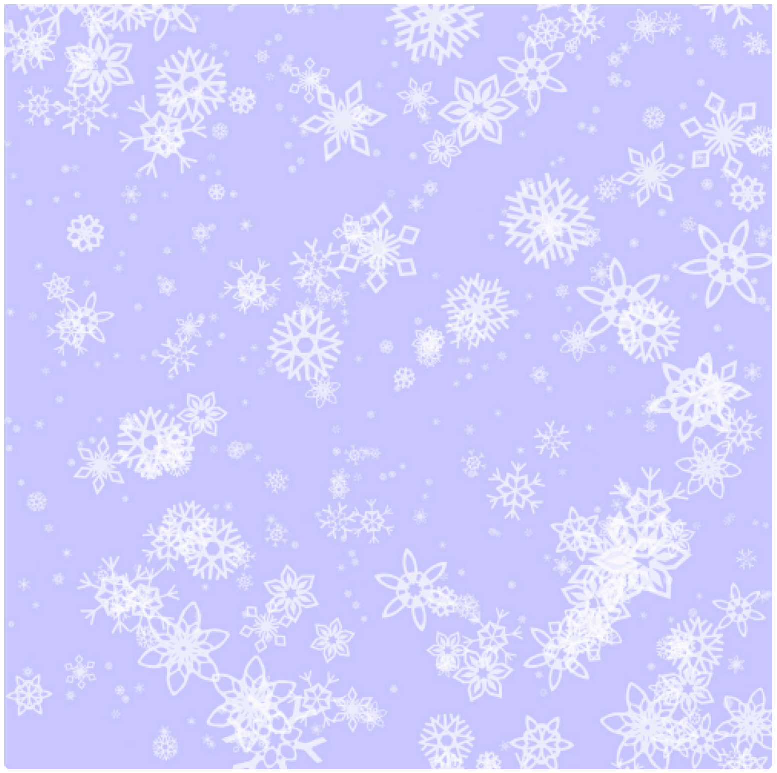
- First, write a helper function that draws a jumble of letters (i.e., half an arm) near the point $(0, 0)$. Test that function by calling it directly from `setup()` (in place of the current call to `drawFlake()`).

- Second, write a helper function that draws a complete arm by combining half an arm with its mirror image (hint: look at the side-by-side mirrored doors in the house example). Again, test this arm in isolation by putting it temporarily in the `setup()` function.
- Finally, you're ready to fill out the `drawFlake()` function by drawing six rotated copies of the helper function that draws an arm.

All you need to do is fill in the `drawFlake()` function and add these two helpers. You don't need a `draw()` function or anything else to complete this exercise.

What to submit: On LEARN, you should submit a sketch entitled `A05_1`. Submit the entire sketch folder.

Question 2: Blizzard



As it turns out we didn't get our snow day on Groundhog Day, but we can try to recapture that moment by creating a blizzard simulator.

Download the starter sketch `A05_2.zip` from the course web page. You will find a sketch that loads a dozen snowflake vector illustrations and stores them in an array of `PShapes`. Your task is to create a simple animated demo in which snowflakes gradually fall from the top to the bottom of the screen. The snowflakes fall in random locations, have random sizes, and spin at random rates.

To accomplish this, you'll need to create a lot of global array variables. At the very least, you'll need:

- An array that determines which illustration to use for each snowflake. You can do this with either an array of `ints` or an array of `PShapes`, depending on how you structure your code.
- An array of `floats` containing the x position of every snowflake.
- An array of `floats` containing the y position of every snowflake.
- An array of `floats` containing the scaling factor of every snowflake.
- An array of `floats` containing the current rotation of every snowflake.
- An array of `floats` containing the rotation speed of every snowflake.

All of these arrays should have the same length, corresponding to the number of snowflakes you want to support in your sketch. That number should also be a global constant. It should be at least 100, but the exact number is up to you.

During the setup phase, you'll need to initialize all of these arrays. You should create a random blizzard. That means you should choose random initial values for every element in each of these arrays. Try to be as random as possible, but within reason:

- For each snowflake, use one of the 12 illustrations at random.
- Snowflake x positions should be chosen so that some part of the flake will lie between the left and right edges of the window as it falls. Similarly, snowflake y positions should also be on-screen. Or, you are also permitted to set your initial y values to be within a box above the sketch, so that you can watch the blizzard start from an empty screen.
- Scales should be chosen so that every snowflake is large enough to be seen, but small enough to avoid too much clutter.
- Every snowflake should spin at a random rate around its centre. Don't forget to allow snowflakes to spin in either direction (or not at all, if you happen to choose a random rotation speed of zero).

During the draw phase, you'll need to draw all the snowflakes in their current positions, with their current scales and rotations. The core of that code will be a loop with a bunch of geometric context functions surrounding a single call to `shape()`. You should use the three-argument version of `shape()`, and the second and third arguments must both be zero; that is, you are required to use geometric context functions to do all the positioning and scaling of snowflakes.

After drawing all the snowflakes, you must update their parameters, not unlike a physical simulation:

- Move each snowflake slightly downward. The speed at which a snowflake falls should be proportional to its scale (small snowflakes will fall slowly, giving the impression that they're far away). Snowflakes should not accelerate from gravity—each one falls at a constant speed.

- Rotate each snowflake according to its rotation speed. A snowflake must rotate around its centre. (Hint: as with `PImage`, you can ask a `PShape` for its `width` and `height`.)
- If a snowflake travels so far downward that it's no longer visible in the window, "recycle" it: move it back to a random position at the top of the window. You can also randomly vary its other parameters, though that's not necessary to create a convincing demo. By continually recycling snowflakes, the demo will run forever.

Finally, you must implement at least **one additional feature** of your choosing. There are several possibilities you can choose from:

- If a snowflake is sufficiently small, replace it by a circle with approximately the same position and size. This will make the sketch run faster.
- Make every snowflake partially transparent. You must do this in code, not by editing the SVG files.
- Find a way to convey accumulation of snow at the bottom of the window. Don't store actual snowflakes there; instead, think about creating a simple white curvy shape that grows upward slowly as the sketch runs.
- Simulate the sky colour slowly darkening as night falls. If you want to be very clever, choose the sky colour based on your current geographic location, date, and time of day. But make sure there's a way for us to see the effect in order to mark it!
- Have the snowflakes drift horizontally while they fall. It's probably easiest to "join" the left and right edges of the screen, so that a flake that leaves one side of the screen enters on the other side.
- Find a way to make the snowflakes swirl from side to side in a visually appealing way, as if caught in eddies of wind.
- I don't mind if you think of some other feature you want to add, as long as it's not too trivial. If in doubt, you can always ask.

What to submit: On LEARN, you should submit a sketch entitled `A05_2`. Submit the entire sketch folder.