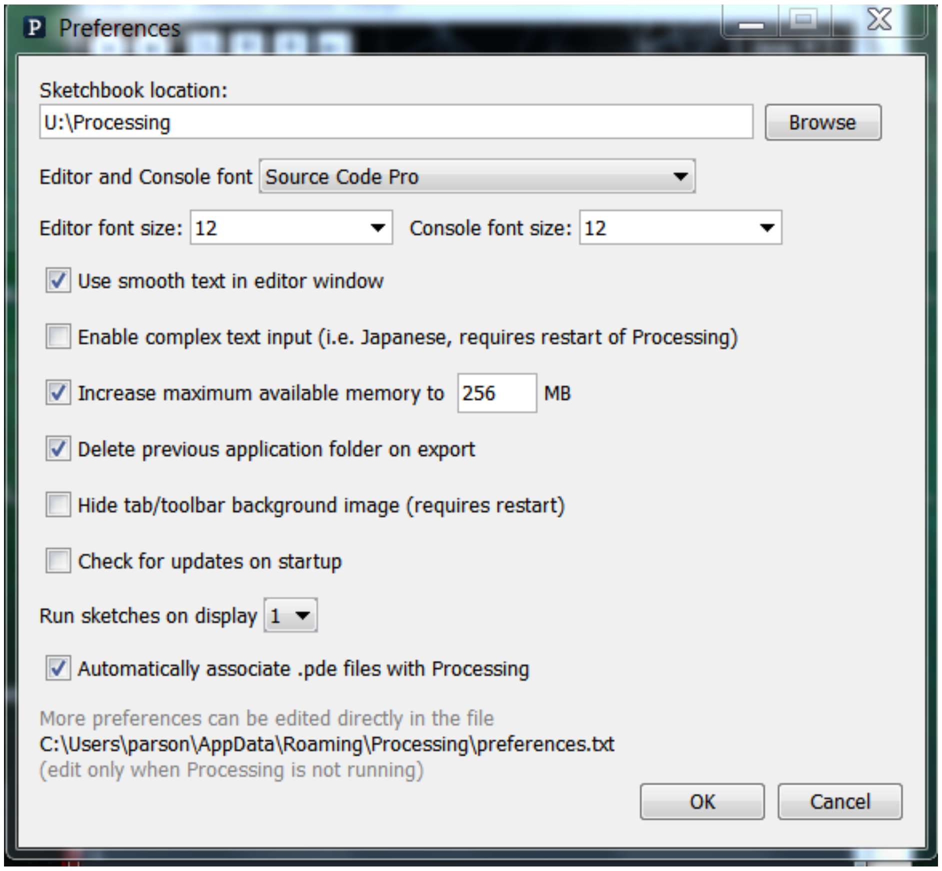
CSC 220 Object-Oriented Multimedia Programming, Spring 2017

Dr. Dale E. Parson, Assignment 5, Recursive Shape OR MIDI Assignment Extension.
This assignment is due via **D2L Dropbox Assignment 5, Final Assignment** by **11:59 PM on Saturday May 6. 10% penalty for each day it is late**.

When using Processing on the Kutztown campus Windows computers, make sure to start out **every time** by setting your Processing Preferences -> Sketchbook Location to U:\Processing. The U:\ drive is a networked drive that will save your work and make it accessible across campus. If you save it to your desktop or the lab PC you are using, you will lose your work when you log out. You must save it to the U:\ drive. If you do not have a folder called Processing under U:\, you must create one using the Windows Explorer. Processing Preferences is under the File menu on Windows.



Sketchbook location: U:\Processing
Editor and Console font Source Code Pro
Editor font size: 12
Console font size: 12
✓ Use smooth text in editor window
☐ Enable complex text input (i.e. Japanese, requires restart of Processing)
✓ Increase maximum available memory to 256 MB
✓ Delete previous application folder on export
☐ Hide tab/toolbar background image (requires restart)
☐ Check for updates on startup
Run sketches on display 1
✓ Automatically associate .pde files with Processing
More preferences can be edited directly in the file
C:\Users\parson\AppData\Roaming\Processing\preferences.txt
(edit only when Processing is not running)

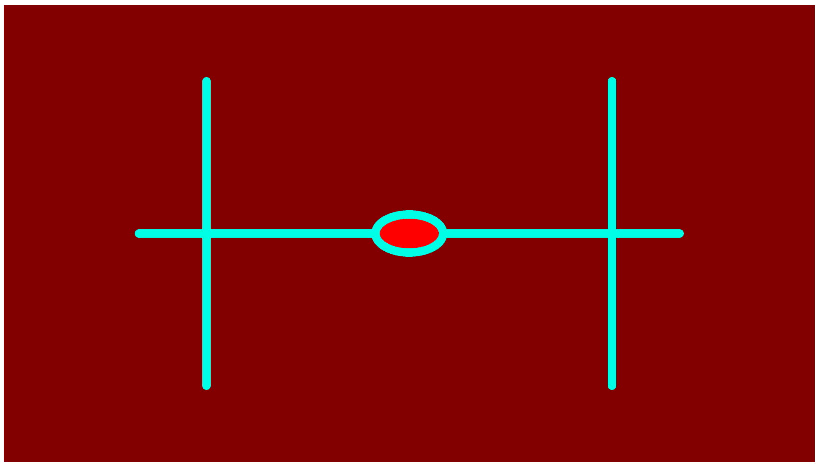
If you will be downloading Processing 3.X and running it using an off-campus computer (do not use version 2.X for assignments), you can copy your project sketch named **ShapePaint3DIntro** to a flash drive on one machine, and then copy it from the flash drive to another Processing sketch folder.

You may choose either to do either **Option A Recursive Shape** for this assignment, or **Option B Extended MIDI Keyboard**, which is an extension of our Assignment 3. You can use either your or my solution to Assignment 3 as your starting point for Assignment 5, if you decide to do Option B.

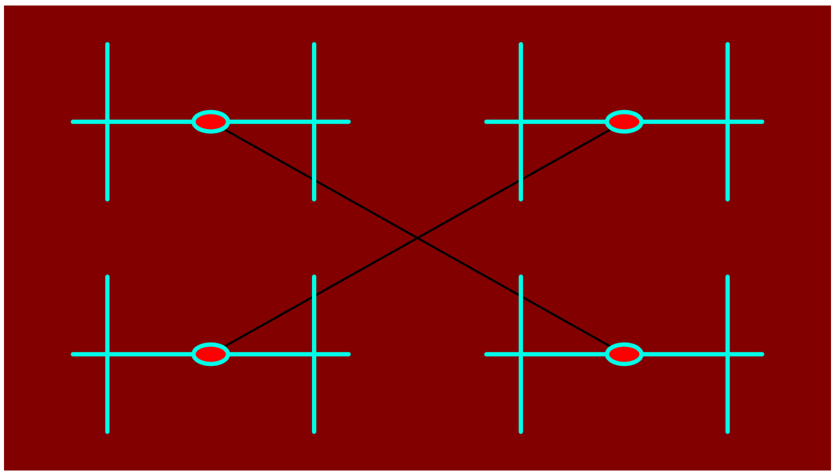
OPTION A: Recursive Shape

The starting point code for RecursiveShape is in the page entitled **RecursiveShape.txt** linked to the course page. Copy & paste it into **Processing** and **Save As RecursiveShape**. Proceed according to the following instructions.

Recursion is a programming technique in which a function calls itself, either directly, or indirectly through another function. Graphical programming often makes use of a recursive function like a "cookie cutter" with parameters that customize the location and scale of the cookie cutter. In this assignment you will use recursion to divide the space of one Processing graphical window that is **width X height** in size into a number of smaller, adjacent "virtual graphical windows" that typically may maintain the same width X height aspect ratio, but where the actual width and height have been scaled, and the 0,0 center location has been translated, to fit each of the smaller virtual windows. The next few illustrations show my handout solution to the problem. You will create your own shape according to requirements appearing later in this document. We will go over my handout code and the concepts of recursion in class of April 25 and 26.

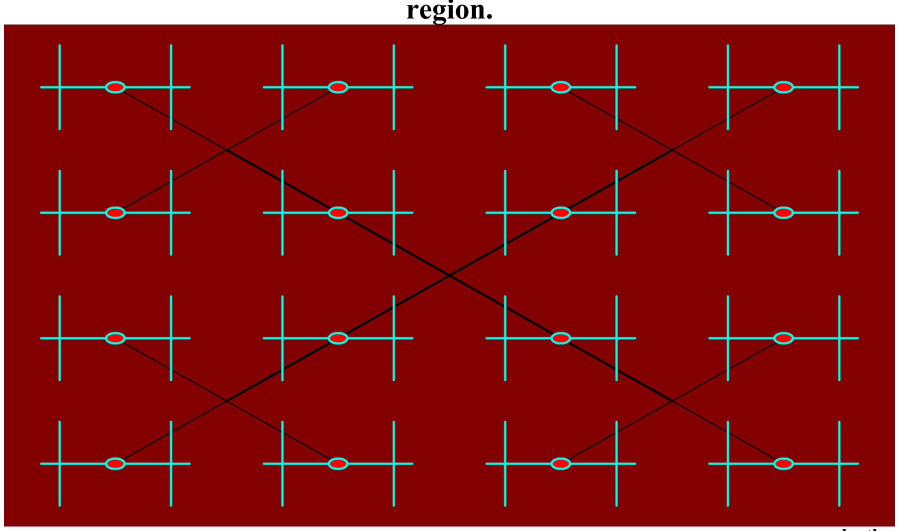


Base case showing my custom shape from function drawShape with a recursion depth of 0.



Recursive case with a recursion depth of 1. Each level of recursion subdivides a rectangular region into 4 adjacent rectangular regions, each with ½ the width and ½ the height of the original "parent"

region.



Recursive case with a recursion depth of 2 gives 16 sub-regions, or generally, 4^{depth} sub-regions.

I will demonstrate deeper levels of recursion in class, along with a different sketch, CCurve, that is linked on the course page. The basic strategy is this:

1. Decide whether it is time to draw the shape, based on a depth parameter to function **drawRecursiveShape** in the handout code.
2. If it is time to draw the shape:
   a. pushMatrix(), then rotate() if the rotate parameter to drawRecursiveShape is non-0.
   b. Call **drawShape**() to draw the shape at the current scale, then popMatrix().
3. Else (not time to draw the shape due to insufficient depth):
   a. For each region to be subdivided from this region (Mine is 4; yours will be something else.)
      I. pushMatrix(), set strokeWeight() and stroke()
      II. Optionally draw a line from the region center to this sub-region center.
      III. translate() to the center of the sub-region.
      IV. scale() to the size of the sub-region, relative to the region being divided.
      V. Call **drawRecursiveShape** recursively to subdivide that sub-region.
      VI. popMatrix() and return.

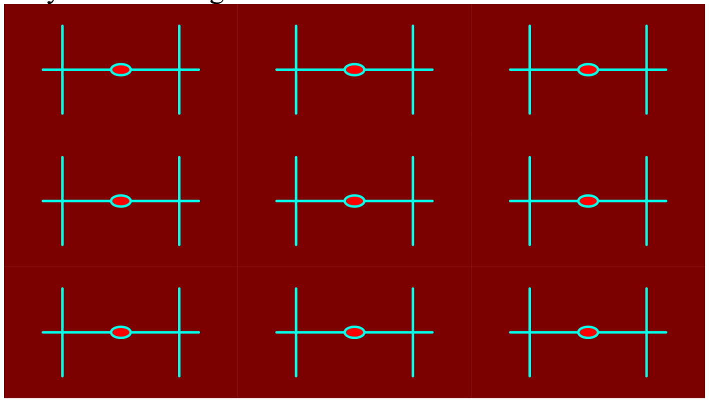
Base case function **drawShape**() simply draws the shape, using **width and height** to delimit its size. We will go over the whole thing in class.

REQUIREMENT 1 (25%) is to complete the keyPressed() function according to these instructions in the function's comments. You can test using my drawShape() after completing this step.

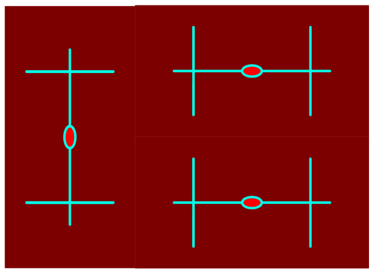
```
// You must write keyRessed() as follows:
// UP increments recursionDepth with no limit.
// DOWN decrements recursionDepth, does not take it < 0.
// RIGHT increments rotationIncrement, wrapping from 359 to 0.
// LEFT decrements rotationIncrement, wrapping from 0 to 359.
// key between '0' and '9' sets eraseRate 0..9, use (key - '0') to get an int value 0..9.
// Upper case 'C' sets eraseRate to 100 ('C' for Clear).
// Upper case 'R' sets eraseRate to 100, rotation and rotationIncrement to 0 ('R' for Reset).
// Lower case 'a' to 'f' sets fRate and calls frameRate to value 10..60, just take
// (key - 'a') and multiply by 10 to get a new frameRate.
```

REQURMENT 2 (25%) Replace my code in function drawShape() with your own distinct shape, using at least two shape-drawing functions different that the line() and ellipse() functions that I used. You can still use calls to line() and ellipse(), but you **must** use at least two other shape-drawing functions. Also, your code should use width a height to help ensure that the shape displays in the currently scaled sub-region. I found it useful to figure out endpoints for my shape by using s sketch on graph paper.

REQUIREMENT 3 (35%) Change else potion of function **drawRecursiveShape** so that it subdivides its **width X height** region into different sub-regions than mine. Mine divides the region into 4 adjacent, identically sized sub-regions. The simplest way to satisfy this requirement would be to subdivide the region into 9 adjacent, identically sized sub-regions, but you could also subdivide it into 3 adjacent, non-identically sized sub-regions as shown here.



Nine sub-regions with aspect ratio equal to the original



Three sub-regions with varying aspect ratios

REQUIREMENT 4 (15%) All of the rotation and user interface capabilities that I demo in class, as supported by your keyPressed() function, must work correctly. Your name must be in the comments at the top of the code.

<u>**OPTION A: Extended MIDI Keyboard (Do either OPTION A or OPTION B, not both.)**</u>
Extend either your solution or my solution to assn3 (posted on the course page) to do the following.
REQUIREMENT 1 (25%): Use a scale other than the 12-note chromatic scale of assignment 3. For example, here are two scales, noting that non-chromatic scales have fewer than 12 notes.

```
int [] major = {0, 2, 4, 5, 7, 9, 11};
int [] minor = {0, 2, 3, 5, 7, 9, 11};
```

Use the mouseX position to determine an octave 0..11 via "(mouseX * 127 / width) / 12", and an offset into a scale such as major via "mouseX % major.length". Add 12 X the octave + offset. This places the "0" note in the key of C; you can add an additional offset that is < major.length to get a different key. I recommend taking the final note "% 128" in order to avoid InvalidMidiDataException.
REQUIREMENT 2 (25%): Use at least 2 distinct instrument voices (PROGRAMS) on at least 2 different MIDI channels. You can hard code the instruments-to-play if you like, or let the user select.
REQUIREMENT 3 (25%): Use 3 distinct, simultaneous notes on a given channel at least some of the time, in order to play a chord.
REQUIREMENT 4 (25%): Use whatever user interface enhancements you like to make it playable, including enhancements to keyPressed(). Keep the same basic mouse UI of increasing pitch as you sweep from left to right, and volume from bottom to top, but you might want to make a note sustain for more than 1 call to draw(). Right now there are very rapid noteon-noteoff sequences that make for a very fast tempo. Feel free to keep a note on across multiple draw() calls, fading it out, or adding CONTROL_CHANGE effects if you like.