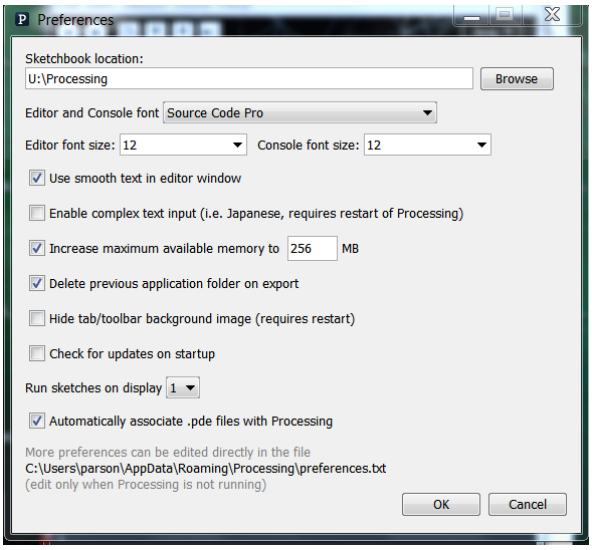
CSC 220 Object-Oriented Multimedia Programming, Spring 2020

Dr. Dale E. Parson, Assignment 5, recursive partitioning of 2D & 3D space.
This assignment is due via **D2L Dropbox** **Assignment 2** due by **11:59 PM on Monday March 9**.
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.



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 **CSC480SP2020Recur2D** to a flash drive on one machine, and then copy it from the flash drive to another Processing sketch folder.

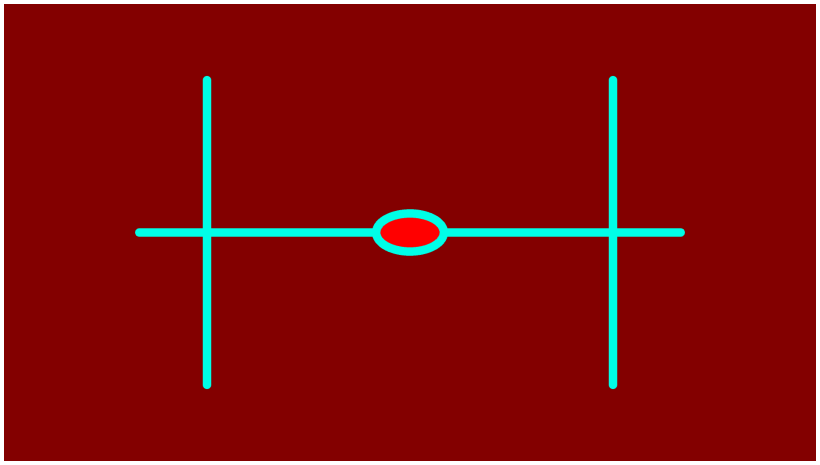
PART A: Recursive 2D Shape

The starting point code for RecursiveShape is in the page entitled **CSC480SP2020Recur2D.txt**[1] linked to the course page. Copy & paste it into **Processing** and **Save As CSC480SP2020Recur2D**. Proceed according to the following instructions.

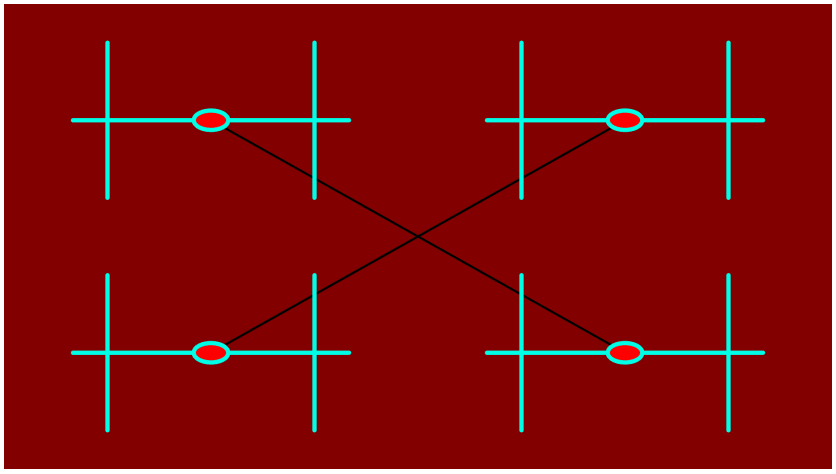
Recursion is a programming technique in which a function calls itself, either directly, or indirectly through another function. Graphical programming often makes use of a recursive function like a "cookie cutter" with parameters that customize the location and scale of the cookie cutter. In PART A you will use recursion within sketch **CSC480SP2020Recur2D** 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

[1] http://faculty.kutztown.edu/parson/spring2020/CSC480SP2020Recur2D.txt

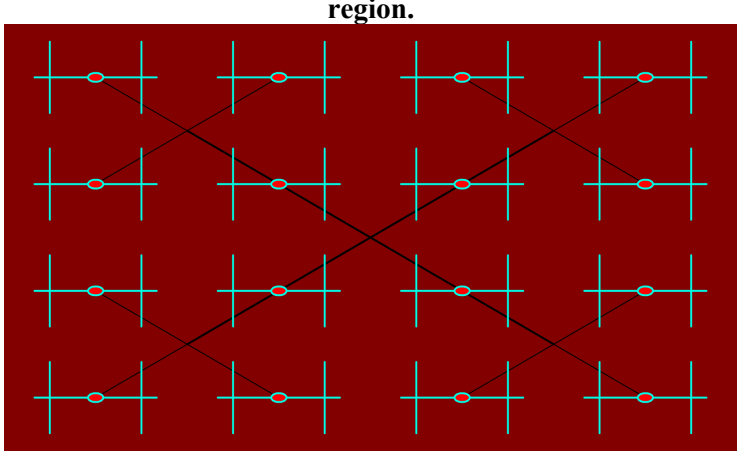
requirements appearing later in this document. We will go over my handout code and the concepts of recursion in class.



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.

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%) 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 and 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 2 (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.

Save this sketch as **CSC480SP2020Recur2D**. You will turn in both **CSC480SP2020Recur2D** and **CSC480SP2020Recur3D** of the next set of instructions. The latter sketch derives from the former.

PART B: Recursive 3D Shape

After you have **CSC480SP2020Recur2D** working to your satisfaction, save a copy as **CSC480SP2020Recur3D**. Part B relates to **CSC480SP2020Recur3D**.

REQUIREMENT 3 (5%) Change P2D to P3D in the setup() function. Uncomment my supplied moveCameraRotateWorldKeys() function down in the bottom of the sketch, and add the **float** global variables required by that function (e.g., **xeye** and **worldxrotate**). Global **float** variable **degree** is the radian equivalent of 1 degree, and **around** is TWO_PI (radians(360) degrees).

REQUIREMENT 4 (25%) Change your recursive portion of function **drawRecursiveShape**() to partition space in 3 dimensions. I simply added an outer for-loop in zdelta space, around the now-nested

```
for (int xdelta = ...) {
        for (int ydelta = ...) {
```

loops. The call to translate() now uses all three deltas. Compare my 2D, 1-deep recursive screenshots at the top of the next page, to the 3D, 1-deep examples at the center & bottom in the illustration on the next page.

The second is rotated around the Y axis using the 'y' keyboard command to show depth.

In the base case, replace a 2D shape or just add a fitting 3D shape such as a box(), sphere(), or cylinder from the 3D demo sketch CSC220F19DemoPShapeCylinder, function makeCylinder() at http://faculty.kutztown.edu/parson/fall2019/CSC220F19DemoPShapeCylinder.txt. I replaced an ellipse() with a sphere().

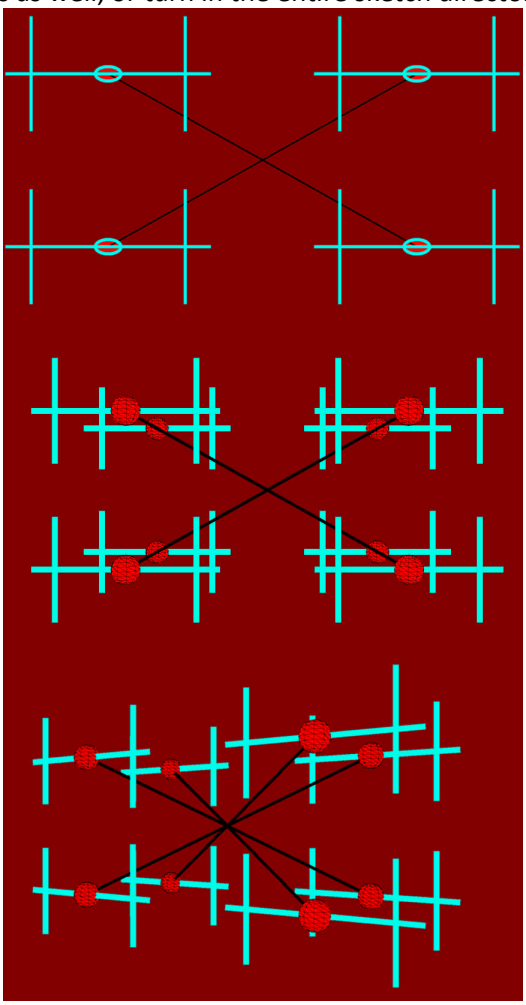
REQUIREMENT 5 (10%) Uncommenting moveCameraRotateWorldKeys() adds some polled keyboard commands. You have a few other keyboard commands to change. Also, change all calls to rotate() in **drawRecursiveShape**() to the identical rotateZ(). Changes all calls to shearX() to rotateX(), and all calls to shearY() to rotateY(). Update keyPressed() to implement the 3D keyboard commands below. I have highlighted the changes. See the handout code for **CSC480SP2020Recur2D** to see its keyboard commands. Do not change the code inside of moveCameraRotateWorldKeys(), but add a call to it in the correct place within the draw() function to get 3D camera navigation and scene rotations to work.

Test to make sure that all of your **CSC480SP2020Recur3D** keyboard commands work. From **CSC480SP2020Recur3D**:

```
/* NEW IN BOLD. I have underlined the section supplied with my moveCameraRotateWorldKeys()
// keyPressed() WORKS 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).
//    'R' also eliminates global rotation.
// 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.
// 'g' toggles rotation of the whole space, default is false
// 'O' enters rotateZ, no rotateX or rotateY (upper case O, replaces 'r' for 2D rotation)
// '-' enters rotateX, no rotateZ or rotateY (dash character, (replaces 'x' for 2D shearX)
// '|' enters rotateY, no rotateX or rotateZ (vertical bar, replaces 'y' for 2D shearY)
// 'F' toggles freezing the display
 *  'p' sets perspective projection; 'o' sets orthographic (NEW TO 3D)
 *  vvv START OF KEY POLLING SUPPLIED IN HANDOUT IN moveCameraRotateWorldKeys() vvv
 *  'u' when held down moves camera up in Z direction slowly
 *  'U' when held down moves camera up in Z direction quickly
 *  'd' when held down moves camera down in Z direction slowly
 *  'D' when held down moves camera down in Z direction quickly
 *  'n' when held down moves camera up in Y direction slowly
 *  'N' when held down moves camera up in Y direction quickly
 *  's' when held down moves camera down in Y direction slowly
 *  'S' when held down moves camera down in Y direction quickly
 *  'e' when held down moves camera right in X direction slowly
 *  'E' when held down moves camera right in X direction quickly
 *  'w' when held down moves camera left in X direction slowly
```

```
 *  'W' when held down moves camera left in X direction quickly
 *  'x' when held down rotates image positive degrees around x
 *  'X' when held down rotates image negative degrees around x
 *  'y' when held down rotates image positive degrees around y
 *  'Y' when held down rotates image negative degrees around y
 *  'z' when held down rotates image positive degrees around z
 *  'Z' when held down rotates image negative degrees around z
*  SPACE BAR held down moves camera x,y to mouseX*2-width, mouseY*2-height
 *  ^^^ END OF KEY POLLING SUPPLIED IN HANDOUT IN moveCameraRotateWorldKeys() ^^^
 *  'R' resets to original camera point of view, STUDENT MUST ADD FOLLOWING ACTIONS:
 *     also resets xeye, yeye, zeye, worldxrotate, worldyrotate, worldzrotate
 *     to their original, default positions
*/
```

Turn in BOTH files **CSC480SP2020Recur2D.pde** and **CSC480SP2020Recur3D.pde** by the due date to avoid late penalties. If you use image files or .svg vector files in your sketch (this is an option open to you), make sure to turn in those files as well, or turn in the entire sketch directories if you use such files.



2D (top) and 3D (middle and bottom) of recursive partitioning of space