CSC 220 Object Oriented Multimedia Programming, Fall 2021

Dr. Dale E. Parson, Assignment 4, 3D Recursive Graphics. SEE STUDENT B REVISION BELOW.

Assignment is due via **D2L Assignments CSC220F21Assn4_Recursion** by **11:59 PM on Tuesday November 23 via D2L**. You need to supply your own PImage file (JPEG, PNG, or similar). Make sure to turn **that** in as well as the code. There will be an in-class overview on November 2 and work session on November 4. The usual 10% per day late charge applies. Please start early, use the work session, and come to Zoom office hours as needed.

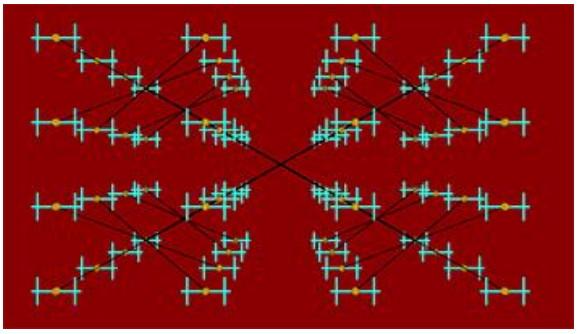
Follow the instructions in our Assignment 1 for setting up your sketchbook location or download the Processing 3.x environment if you need to do this. You probably don't need to do this. However, if your laptop caused unresolved problems with 3D Assignment 2, then use the campus Windows PCs that have access to S: networked drive. Windows computer labs on the KU campus such as the CSC Lab in Old Main 248A at the end of the CSC office hallway and in Rohrbach Library should allow you to run Processing from the networked S:\ComputerScience\processing3.5.4 subdirectory. I believe we have fixed the new Mac problem by calling frameRate() immediately after size() in setup().

Download handout code from https://faculty.kutztown.edu/parson/fall2021/CSC220Fall2021Recur3D.txt and save it as Processing sketch **CSC220Fall2021Recur3D**. Make enhancements detailed in STUDENT comments in the code which also appear below.

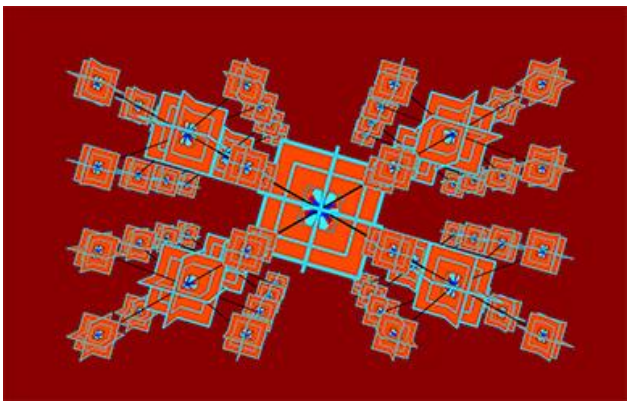
Search and read all **STUDENT** comments in the code before starting.

```
/* Author: STUDENT NAME

PImage STUDENTIMAGE ;
PShape STUDENTSHAPE ; // STUDENT I set this via createShape in setup() but use this in drawShape().
// See STUDENTSHAPE B requirement in drawShape().

  // fullScreen(P3D); // STUDENT adjust size() for your monitor.
  size(1920, 1080, P3D);    // Use size() to avoid Zoom problem on fullScreen.
  frameRate(fRate);         // Call frameRate() after size to avoid the new Mac problem.
  colorMode(HSB, 360, 100, 100, 100);
  STUDENTIMAGE = loadImage("Hex306090B.png"); // STUDENT replace with your own image file for
drawShape()

// STUDENT A 20%: At this bottom section of the recursive case call drawShape(mydepth, rotation);
// just like in the non-recursive case, with its full size, not scaled via scale(),
// but showing the same rotation in effect as the base case (non-recursive) drawShape.
// Achieving this requires some additional lines of code, properly placed.
```

Here is a screen shot from my handout code **without** the STUDENT A feature. Note the recursive case centers.



Here is a screen shot from my solution code **with** the STUDENT A feature. Note the recursive case centers. This pic shows some rotation from the RIGHT arrow key.



// **STUDENT B 50%:** Each student creates their own shape according to handout spec.
// You must use 2D shapes only, with copies rotated in X, Y, and Z directions, and
// nested to the degree given by global recursionDepth+1. Also, plot one 2D PShape STUDENTSHAPE
// per https://processing.org/reference/createShape_.html, texture it with STUDENTIMAGE
// via STUDENTSHAPE.setTexture, and plot it as the innermost nested shape per recursionDepth.
// I initialized STUDENTSHAPE via createShape() in setup() and textured it using
// STUDENTSHAPE.setTexture() there to avoid doing it repeatedly inside drawShape().
// I will demo on 11/2. The handout code does none of this.
// NOTE: If you plot 2 2D shapes of different colors in the same Z plane, their pixels will
// munge together. The later one plotted will NOT cover the earlier ones.
// Therefore, as you nest deeper to plot smaller shapes, you could plot 2 of each offset at
// increased z and -z distances from the outermost one. I just stuck with the same color instead
// of plotting multiple copies. However, I plotted 2 copies of the textured STUDENTSHAPE,
// one at a positive and one at a negative Z translate, to correct the problem.

STUDENT B REVISION:

You do **NOT** need to texture a STUDENTSHAPE with the STUDENTIMAGE. That works only for a subset of PShapes, notably RECT PShapes. Instead, plot via **shape(STUDENTSHAPE, 0, 0, yourwidth, yourheight)** one library-supplied PShape built with **STUDENTSHAPE = createShape(…)** (could be a 2D vector file via **loadShape**() if you know how to build those), AND display one **STUDENTIMAGE** loaded with **loadImage**() and plotted with **image(STUDENTIMAGE, 0, 0, yourwidth, yourheight)**. Call **createShape**() and **loadImage**() up in **setup**() to avoid the overhead of creating them in each call to **drawShape**().

ALSO (11/17): You do **NOT** need to use vertex() in creating your PShape. You can use a canned library shape as documented in the **createShape**() documentation.

```
// STUDENT C 10% (within function keyPressed()):
// on key of '2' set spanSteps to 2,
// on key of '3' set spanSteps to 3.
// on key of '4' set spanSteps to 4.
// on key of '5' set spanSteps to 4.
// on key of '6' set spanSteps to 6.
// on key of '7' set spanSteps to 7.
// on key of '8' set spanSteps to 8.
// Note you can treat character variables like key and constants like '0'
// as integers for the purpose of arithmetic such as subtraction and
// numeric comparisions of key to character constants. You don't necessarily
// need to but it can shorten your code. Or you can just use individual
// "else if" statements for each key constant or a switch statement.
// Also, setting spanSteps greater than 4 runs slowly with more than 1 level
// of recursion. When testing, the starting point is spanSteps == 2.
// Hitting '2' after recursing 1 level deep should show no change.

// controlKey, altKey, or both are currently engaged:
// STUDENT D 20% (within function moveCameraRotateWorldKeys()):
// IMPLEMENT THE FOLLOWING
//  VVV CONTINUOUS CONTROL and ALT key closures, CONTROL or ALT key held down VVV
// CONTROL-x CONTROL-y CONTROL-z continuously increment these by spanIncr.
// CONTROL-X CONTROL-Y CONTROL-Z (upper case) continuously decrement these by spanIncr.
// float xSpanLimit = 0.5, ySpanLimit = 0.5, zSpanLimit = .5 ;
// The above SpanLimits may range from -[xyz]SpanRange to [xyz]SpanRange.
// spanIncr is the increment, -spanIncr decrement for the above SpanLimits.
// final float xSpanRange = 2.0, ySpanRange = 2.0, zSpanRange = 2.0, spanIncr = .002;
if (controlKey) {

}
// ALT-x ALT-y ALT-z continuously increment these by spanIncr.
// ALT-X ALT-Y ALT-Z (upper case) continuously decrement these by spanIncr.
// float xScaleAmount = 0.5, yScaleAmount = 0.5, zScaleAmount = 0.5 ;
// The ScaleAmounts may range from -[xyz]SpanRange to [xyz]SpanRange.
// ^^^ END CONTINUOUS CONTROL and ALT key closures ^^^
if (altKey) {
  // If both CONROL & ALT are down, do both.

}
```

I will demo all STUDENT requirements on 11/2. Watch the video if you need a refresher.

Make sure to test thoroughly when you believe you are done and turn both the Processing code file and your PImage file into D2L by the due date. The usual 10% per day late penalty applies. There is plenty of lead time to work on this.