

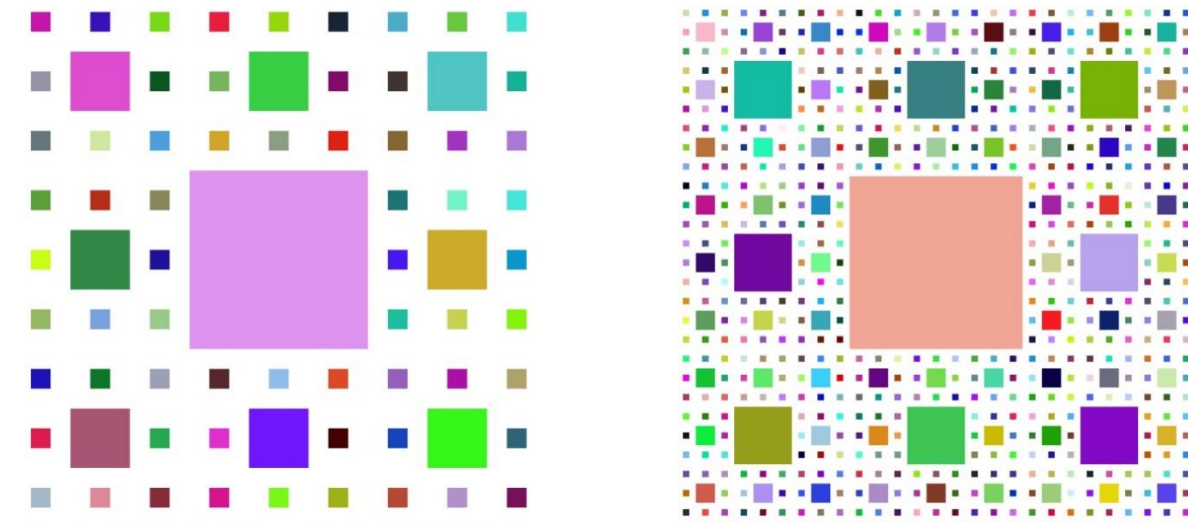
Jeff Registre

CSC-385

Project-1

WebGL Sierpinski Carpet

This code project is called the Sierpinski Carpet. Its purpose is modeling the Sierpinski Carpet fractal and render it on the GPU through the use of WebGL. The structure of this project is as follows; An HTML page acts as the GUI on which to display the application, and a JavaScript application calculates and calls for GPU rendering using shaders written in ESSL. Not much is actually done on the GPU for this project. The JavaScript application takes in input from the HTML page for the number of subdivisions for the fractal: it calculates everything and simply passes all the data to the GPU for rendering. Below is an image of the carpet with 3 and sub subdivisions from left to right.



As you can see from the images, the amount of squares grows extremely quick. So much so that when divisions exceeds 10, the browser likely crashes the program.

The number of squares on screen for any given subdivisions according to my implementation can be calculated with the following piecewise defined equation:

$$A(n) = \begin{cases} 1, & x \leq 1 \\ (A_{n-1} * 8) + 1, & x > 1 \end{cases}$$

The method I've chosen for I've selected for testing is the createCarpet method. This method returns a list of squares making a Sierpinski carpet. The parameters it takes are; subdivisions which drives how many squares will be in the carpet, and InitialSquare which is the

square it'll use to begin the operations. The method begins by analyzing the value of the subdivisions parameter: if it is 0, then it simply returns an array with the initial square as the sole value. If the subdivisions value is 1, then it gets the sub square of the initial square and returns that in an array. So for values of 0 and 1, there is still just one square being rendered. With values greater than one, it'll do the same step as the value being 1 but then it'll get all the "open squares" and continue to act upon them making more and more squares until it finally returns. That is pretty much it: simple but a little bit complex. A screenshot of this code is provided below.

```
function makeCarpet(initialSquare, subdivisions){
  var closedSquares = []
  if(subdivisions > 0){
    var openSquares = []
    closedSquares.push(getFilledSub(initialSquare))
    Util.pushList(getEmptySubs(initialSquare), openSquares)
    subdivisions--
    for(var i = 0; i < subdivisions; i++){
      if(openSquares != null && openSquares.length > 0){
        var temp = []
        $.each(openSquares, function(index, value){
          if(value != null){
            closedSquares.push(getFilledSub(value))
            Util.pushList(getEmptySubs(value), temp)
          }else{console.log("Value null, why??")}
        })
        openSquares = temp
      }else{
        console.log("Open squares is null or has nothing")
      }
    }
    return closedSquares
  }else{
    return [initialSquare]
  }
}
```