2D Rendering Assignment v1.0

Written by Bryce Summers (BryceSummers.com)

Last Updated: December 3, 2015

Contents

1 Overview

This assignment runs the gauntlet of common Rendering Techniques. The student will learn about explicit function based forms, implicit paramaterization based forms, coordinate space transformations, Function Scaling, color interpolation, Sampling, antialiasing, bounding boxes, 2D renderer implementation, Optimized Super Sampler implementation.

2 Basic Renderer Setup

2.1 Color Calculation functions

All images can be thought of as a color intensity function on a restricted rectangular domain that takes 2 arguments and returns a color. Many people view images as collections of pixels, but it is usuful to not be restricted by this world view, because often the mathematical specification for an image transcends a given rendering of the image.

2.2 Rendering

Rendering is the proccess of converting a mathematical description of a scene into an image.

The standard rendering algorithm is as follows for scenes defined by Color Calculation functions:

```
image = Color[h][w];
CCF = [Scene Description Function]

for(int y = 0; y < h; y++)
for(int x = 0; x < w;x++)
{
   image[y][x] = CCF.getColor(x, y);
}
```

2.3 Task 1: Implement A Basic Renderer

2.4 Optimized Super Sampling

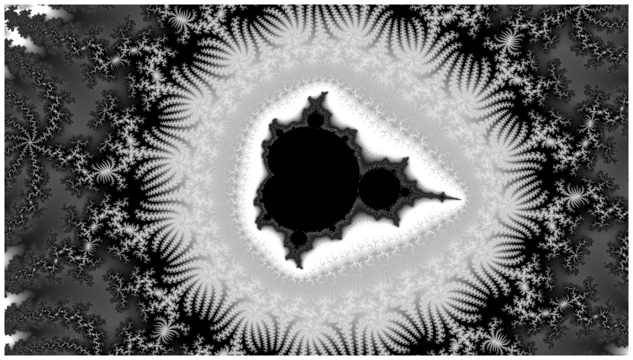
Super Sampling is the proccess of sampling the color calculation function more than once for a pixel and averaging the results in order to eliminate aliasing in the final image do to defficiencies in the closeness of the sampled pixels.

```
image;// Already Rendered using standard rendering.

for(int y = 0; y < h; y++)
for(int x = 0; x < w;x++)
{
   if(pixel differs from a neighboring pixel)
   {
      Color c = average color for all of the sub samples.
      image[y][x] = c;
   }
}
```

2.5 Task 2: Implement an Optimized Super Sampler.

3 Implicit Rendering: Mandelbrot Set



3.1 Color Calculation Function

Create the Color calculation function for the Mandelbrot set with a view from (x1, x2) to (y1, y2).

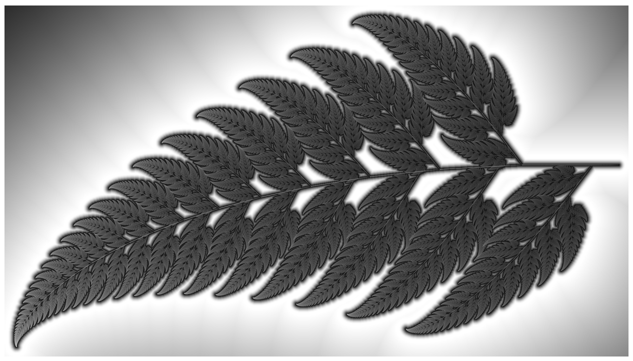
// FIXME Elaborate on the Mandelbrot Generative Procedure.

1. Input x, y;
2. Transform (x,y) from screen space to fractal space.
3. Apply Iterative Mandelbrot Formula to the point until it excapes the unitary complex circle or it hits the iteration limit.
4. Compute a distance estimation value and apply an interpolation function to produce a nice pallet of colors.
5. Return the final color.

3.2 Task 3: Mandelbrot Rendering

Render the appropiate test scenes using super sampling. Make sure you use the optimization so that you do not spend too much of the computation time on the large bodies of set interior points.

4 Explicit Rendering: Barnsley Fern



Create the Color calculation function for the Barnsley Fern.

4.1 Precomputations

Fractal Space:

$$-2.1820 < x < 2.6558 \textbf{ and } 0 \leq y < 9.9983$$

We will store irradiance values for every pixel in the image. $\mathcal{O}(ITERATIONS)$ We will do a precomputation where we will generate irradiance values for a point traveling according to the Barnsley Fern iterative generation function.

The irradiance points will always hit 4 pixels at a time. The relative additive contributions should be determined by the fractional componant of the floating point location. The location the values are stored at will need to be determined by transforming the location from fractal space to screen space.

We will then do a precomputation where we compute the distance every point is from the fractal. This requires a breadth first search like specialized algorithm form propogating the chaotic values. It also requires a Queue. $\mathcal{O}(PIXELS)$

Compute the Maximum irradiance in order to scale the values appropriately.

// FIXME Elaborate on the math for the generative procedure.

4.2 Color Calculation Function

Using the procomputed data, we will specify the Color Calculation Function as follows:

1. Input x, y;
2. Transform (x,y) from screen space to fractal space.
3. Shade according to the precomputed distance estimation function if the point is not in the fractal. Shade according to the irradiance value relative to the maximum irradiance value otherwise.
4. Return the final color.

Please note that their is no need for super sampling, because the fast changing parts of the signal have already been interpolated in the precomputation step.

4.3 Task 4: Render The Barnsley Fern

5 Piesewise Rendering: Geometric Shapes

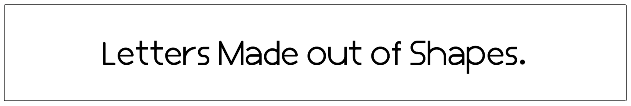
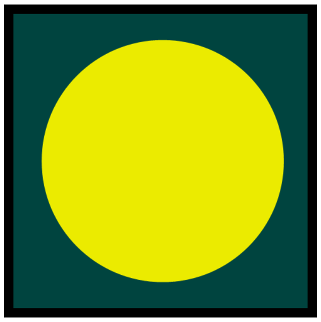

Letters Made out of Shapes.

Figure 1: Example Text written in Bryce Font 2, which is made out of circles, lines, and locus curves.

// FIXME : Expand on the transformation math required to properly define these shapes.

5.1 Circles



The region withing a bounded distance from a central point.

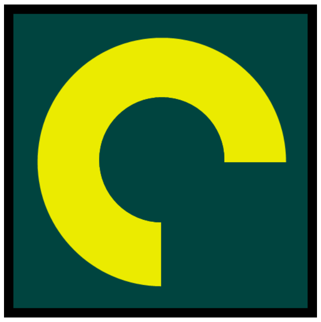
5.2 Lines

The region defined by two points that is of a given stroke size perpendicular to the direction specified by the difference between the two points.

5.3 Subdivision Curves

Defined by the subdivision of a polyline.

5.4 Locus Curves



The portion of a circle that goes through 3 points.

//FIXME : Write up my nifty line side inclusion / exclusion algorithm for explicitly checking for inclusion in theis shape.

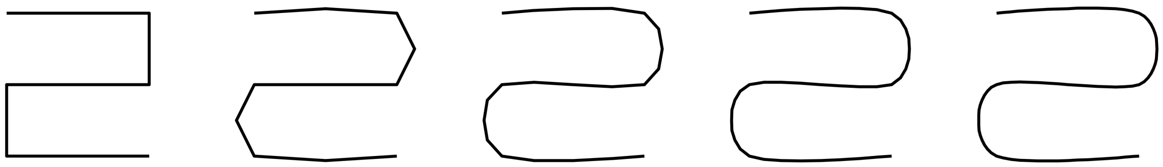


Figure 2: A 2D polyLine being subdivided into an interpolated curve.

5.5 Task

Render the test scenes that specify geometry of the forms discussed earlier in this section. Invent the specification for forms that represent one of the letters of the alphabet.

// Notes: we may want to have the shapes define interpolated in boundedness functions. We could also have people get some hands on experience with alpha blending. We might also want to include some data structures such as bounding boxes to make this task go better.

5.6 Submission

There is no submission procedure as of yet.

5.7 Starter Code

There is no starter code as of yet.