Homework 2: 2D Graphics

CS 440 Computer Graphics
Habib University
Fall 2019

Due: 18h on Mon, 23 Sep

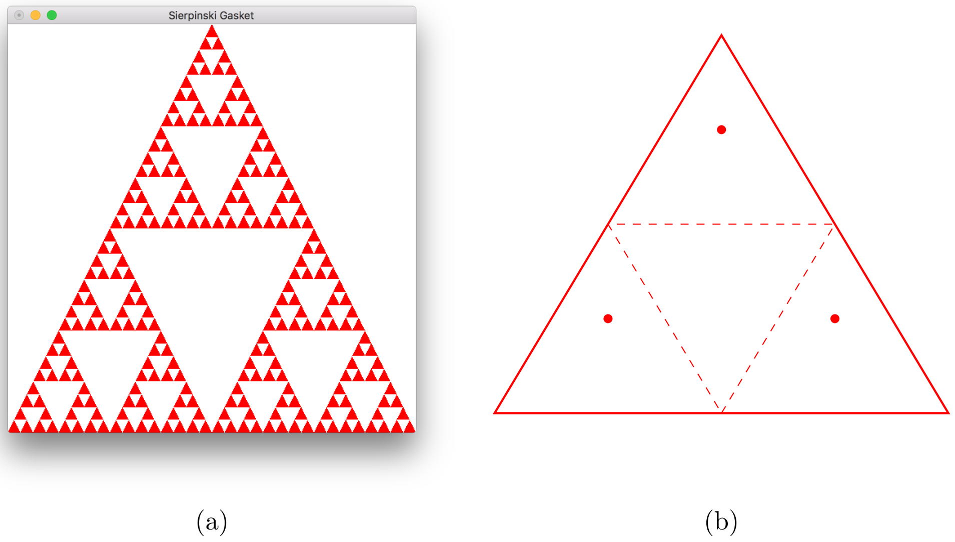


Figure 1: (a) A Sierpinski triangle at recursion level 5. (b) The recursive step in the generation of the Sierpinski triangle.

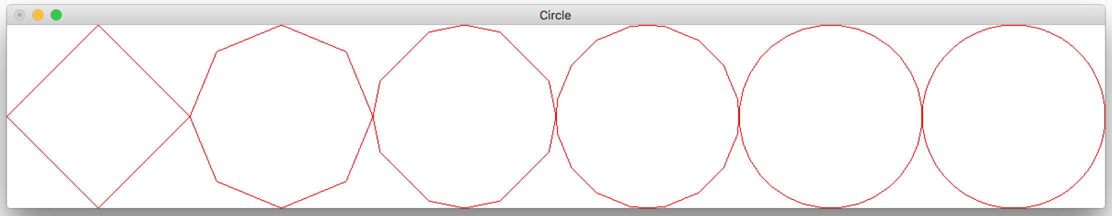
1. Triangle in a Triangle in a Triangle ...

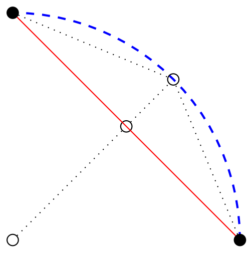
The Sierpinski triangle is a self-similar geometric shape with many interesting properties, not the least of which is the variety of ways in which it can be generated. A sample triangle is shown in Figure 1a).

For this problem, the Sierpinski triangle is defined recursively as illustrated in Figure 1b). Starting with a triangle, connect all its edge midpoints. This yields 4 new triangles. Repeat the process for all the new triangles shown with a dot in them. Do not subdivide the triangle in the middle (without a dot). Note that the dots in the figure are for demonstration purposes only–they are not part of the Sierpinski triangle.

Write a program to generate the Sierpinski triangle at any desired recursion level.
Files: sierpinski.html, sierpinski.js

2. Approximating a Smooth Surface: $\lim_{n\to\infty} \diamond = \circ$

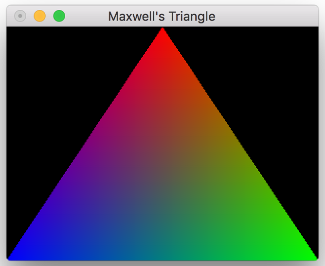




This problem explores the approximation of a smooth shape by a many-sided polygon. Starting from a diamond on the left in the above figure, each successive shape is recursively generated by inserting edge midpoints into the shape and projecting the new points to the circumference of the circle being approximated. This is illustrated in the figure on the left. The red line is approximating the blue arc. In the subdivision steo, the midpoint of the red line is inserted and projected to the arc. The red line is then replaced with the dotted black line.
Generate the above figure by repetitively refining a coarse initial approximation.

Files: circle.html, circle.js

3. Maxwell's Triangle

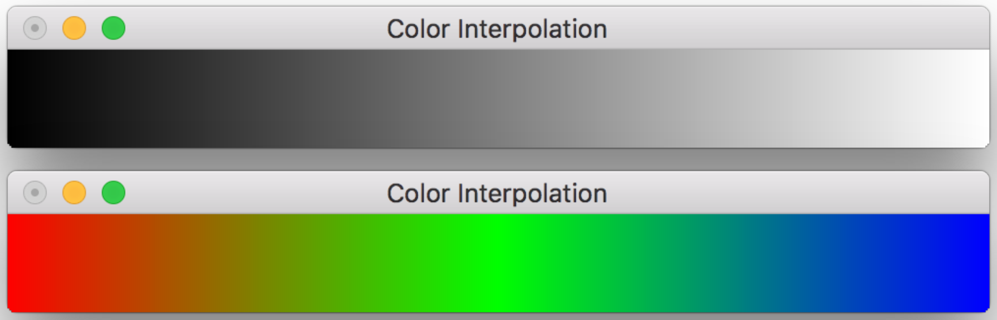


James Clerk Maxwell is best known for his work in electromagnetic radiation resulting in the well known Maxwell's equations.
Not surprisingly, he was also among the first ones to explore the trichromatic theory of color. This has led to what is now known as Maxwell's triangle, simply called the color triangle. It is a triangle with the primary colors at its vertices and its interior shaded by linear combinations of the colors at the vertices. At any interior point, the amount of color contributed by a vertex is proportional to the distance of the vertex from that point.

In WebGL, if the endpoints of a line segment are of different colors, each point on the line segment is rendered with a color value that is interpolated from the endpoints' colors. The same applies to polygons. Use this property to display Maxwell's triangle: a triangle whose vertices are red, green, and blue.
Files: maxwell.html, maxwell.js

4. Color Interpolation

.



Problem 3 relied on the GPU's native color interpolation. For this problem, you will draw color bars for which you will perform the interpolation using the `map_point` function from the previous homework.

Display the colorbars above in your display window with the grayscale bar above the color bar. Each bar is comprised of vertical lines rendered next to each other, i.e. on adjacent pixels, in the display window. Both bars have the same dimensions. The width has to be sufficiently large to include every color in the following RGB ranges.

- (0,0,0) to (255, 255, 255), and
- (255,0,0) to (0, 255, 0) to (0, 0, 255).

Files: colorbar.html, colorbar.js

5. The Mandelbrot Set

The sequence z_n over a complex number, c, is defined for positive integers, n, as

$$z_n(c) = \begin{cases} 0 & n = 0 \\ z_{n-1}^2(c) + c & \text{otherwise} \end{cases}$$

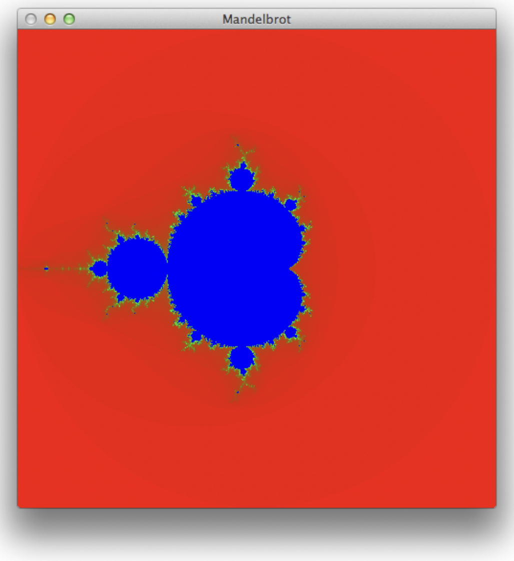
The Mandelbrot set, M, is the set of all complex numbers, c, such that z_n above is bounded, i.e. $|z_n|$ is finite for all n. It can be shown that if for some $n = n_0$, $|z_{n_0} > 2|$, then the sequence z_n *escapes to infinity*, i.e. $|z_n|$ is not bounded for $n > n_0$. This leads to the conclusion that $\forall c \in M \; z_n(c) \leq 2$. Note that since $z_1 = c$, the above condition also implies that $|c| \leq 2$. The above yields the following two handy conditions whose fulfillment by a complex number, c, establishes its membership in M:

1. $|c| \leq 2$
2. $\forall n \geq 0 \; z_n(c) \leq 2$

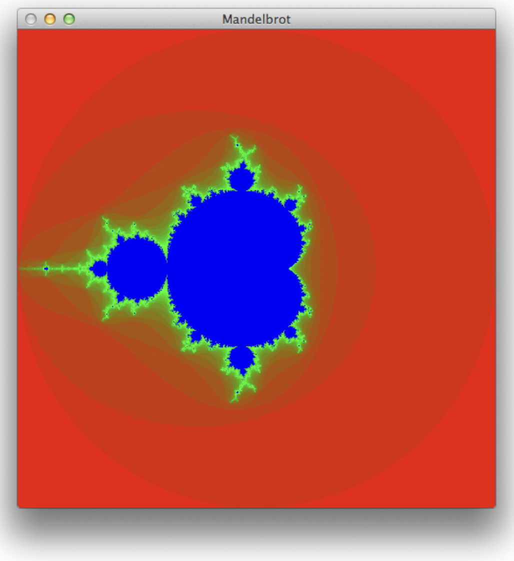
The value of n for which condition 2 becomes false for a given c is dubbed the *escape time* of c.

Write a program to visualize the Mandelbrot set. You will need the following steps.

- Map $Re(c)$ and $Im(c)$ to the x and y axes respectively.
- Your viewing rectangle should span the ranges [-2,2] in the x and y directions. The origin lies at the center of the window.
- Map every coordinate in the viewing window to its corresponding c and determine its escape time. Set an appropriately high threshold n_t, such that, if $z_n(c)$ has not escaped till $n = n_t$, you may assume that $c \in M$.



(a)



(b)

Figure 2: The Mandelbrot set for $n_t = 100$. Points at the corners of the windows are outside the circle of radius 2, and z_n escapes for these points at $n = 0$. They are thus completely red. The points in blue toward the center of the image are the ones for which z_n up to z_{100} has not escaped. (a) Uniform color interpolation from Red to Green to Blue. (b) Adjusted interpolation.

- Color the pixel according to its escape time such that an escape time of 0 corresponds to red, pixels whose corresponding c have not escaped are colored blue, and pixels whose c has escaped between 0 and n_t are colored green. This is a similar color interpolation as in Problem 4. An example for $n_t = 100$ is shown in Figure 5a).
- Figure 5a). contains a lot of red. Think about why that is and what it means in terms of escape times. It is possible to adjust the color interpolation to yield Figure 5b). Adjust your interpolation function to yield an aesthetically appealing coloring of your choice. Some strategies you could implement are
  - move green away from the middle of the range
  - use more colors for interpolation
  - use non-linear interpolation

Files: mandelbrot.html, mandelbrot.js