

Take a look at my live demo at:

*http://cs.iupui.edu/~ibsardar/ibr\_mandelbrot.html*

MATH 35100 Mandelbrot Challenge 2016

**SOURCE CODE**

***explanations within comments***

<!--

​

        AUTH:  Ibrahim Sardar

        FILE:  ibr\_mandelbrot.html

        DESC:  Print the Mandelbrot Set using colours with a precision input.

                The entire picture is present at every Fractal Geometry.

        LOG:

            08/31/2016 @ 12:00am  -  initial project completed

            08/31/2016 @ 1:00am  -  fixed zoom to be centered

                                  -  notify users of ctrl+u shortcut

​

-->

​

<!DOCTYPE HTML>

<html lang="en-US">

<head>

    <meta charset="utf-8"/>

    <title> Mandelbrot Visualization </title>

    <!-- logic -->

    <script style="text/javascript">

​

        //globals

        var canvas = null;

        var brush = null;

        var pic = null;

        var pixels = null;

        var x\_min=0, x\_max=0;

        var y\_min=0, y\_max=0;

        var iter\_max = 0;

        var x\_offset = 0, y\_offset = 0;

        var color\_mult = 0;

        //ready the canvas

        canvas = document.createElement('canvas');

        brush = canvas.getContext('2d');

​

        //init canvas

        function init( w,h,x\_bounds,y\_bounds,iters ) {

            //set canvas properties

            canvas.width = w;

            canvas.height = h;

            //set mandelbrot bounds

            [x\_min, x\_max] = x\_bounds;

            [y\_min, y\_max] = y\_bounds;

            //set # of iterations per pixel

            iter\_max = iters;

            //multiply this with n iterations to scale to 255 rgb

            color\_mult = 255/iter\_max;

            //multiply these to canvas x,y to get mandelbrot x,y

            // note: will be centered at 0,0 of canvas

            x\_mult = Math.abs( x\_min-x\_max ) / w;

            y\_mult = Math.abs( y\_min-y\_max ) / h;

            //normalize x\_min,y\_min without altering distance between min & max

            // note: Subtract these to mandelbrot coords to make the canvas

            //      center the actual center instead of topleft corner, vice

            //      versa to do the opposite.

            //

            // ex:  -2,4  =>  0,6  =>  offset=2

            // ex:  3,6  =>  0,3  =>  offset=-3

            x\_offset = -x\_min;

            y\_offset = -y\_min;

        }

        //get scaled coordinates

        function scale( x, y ) {

            //scale x,y coordinates to mandelbrot coordinates

            var x\_mandelbrot = x\*x\_mult;

            var y\_mandelbrot = y\*y\_mult;

            //push transformed coordinates back to original

            x\_mandelbrot -= x\_offset;

            y\_mandelbrot -= y\_offset;

            //return scaled coordinates

            return [x\_mandelbrot,y\_mandelbrot];

        }

        //sets color based on how many iterations happened

        function colorize( px,py,iters,sx,sy ) {

            // rgb depends on value of iters

            var R = 0, G = 0, B = 0;

            /\*\*\*

            // Basic colorize (if in m-set, black, else white)

            if( iters != iter\_max ){

                R = 255; G = 255; B = 255;

            }

            \*\*\*/

            // typical colorize algorithm

            R = 10; G = 20; B = 30;

            if( iters != iter\_max ){

                R = 0;

                G = 0;

                B = Math.floor(color\_mult \* iters);

            }

            /\*\*\*

            // More complex colorize algorithm (pretty colors!)

            R = Math.floor(color\_mult \* iters / 1.5);

            G = Math.floor(color\_mult \* iters / 1);

            B = Math.floor(color\_mult \* iters / 2);

            if (iters/iter\_max < (1/4)) {

                R = 0;

                B = Math.floor(color\_mult \* iters) \* 2;

                G = Math.floor(color\_mult \* iters) \* 4;

            }

            if (iters/iter\_max < (1/16)) {

                R = 0;

                B = Math.floor(color\_mult \* iters) \* 8;

                G = Math.floor(color\_mult \* iters) \* 14;

            }

            if (iters/iter\_max < (1/32)) {

                R = 0;

                B = Math.floor(color\_mult \* iters) \* 16;

                G = Math.floor(color\_mult \* iters) \* 28;

            }

            \*\*\*/

            // apply rgb

            brush.fillStyle = "rgb("+R+","+G+","+B+")";

            // blit colored pixel

            brush.fillRect(px, py, 1, 1);

        }

​

        //store mandelbrot pixel data

        function m() {

            //for each pixel [i,j] on canvas

            for (var i=0;i<canvas.width-1;i++) {

                for (var j=0;j<canvas.height-1;j++) {

                    //vars

                    var xm=0.0, ym=0.0;

                    var iter = 0;

                    //scale pixel coordinates to mandelbrot coordinates

                    var [x\_scaled,y\_scaled] = scale( i,j );

                    //iterate function on parameters xm, ym

                    while(true){

                        //use Pythagoras Th. to determine escape

                        var pythag = (xm\*xm) + (ym\*ym);

                        if (pythag < (2\*2)) {

                            //make sure iterations are capped

                            if (iter < iter\_max) {

                                //apply function f = z^2 + c

                                // where:

                                //

                                //  f = x^2 + i\*2xy - y^2 + x\_scaled + i\*y\_scaled

                                //

                                //  xm is the REAL part of f

                                //    xm = x^2 - y^2 + x\_scaled

                                //

                                //  ym is the IMAGINARY part of f (w/out the i's)

                                //    ym = 2xy + y\_scaled

                                //

                                var tmp = (xm\*xm)-(ym\*ym)+(x\_scaled);

                                ym = (2\*xm\*ym)+(y\_scaled);

                                xm = tmp;

                                //iterate

                                iter += 1;

                            } else {

                                break;

                            }

                        } else {

                            break;

                        }

                    }//end iterate

                    //color and blit pixel

                    colorize( i,j,iter,x\_scaled,y\_scaled );

                }

            }

        }

        //events

        canvas.addEventListener("click", zoom, false);

        //zoom into canvas

        function zoom(e) {

            //get canvas properties (for simplification)

            var w = canvas.width;

            var h = canvas.height;

            //clear canvas

            brush.clearRect(0, 0, w, h);

            //get mouse position

            var view = canvas.getBoundingClientRect();

            var msx = e.clientX - view.left;

            var msy = e.clientY - view.top;

            //convert mouse pos to mandelbrot position

            msx = msx\*x\_mult;

            msy = msy\*y\_mult;

            msx -= x\_offset;

            msy -= y\_offset;

            //get current mandelbrot width, height

            var mw = Math.abs( x\_min-x\_max );

            var mh = Math.abs( y\_min-y\_max );

            //scale bounds by 50% where mouse pos is the center

            var xb = [msx-(mw/4), msx+(mw/4)];

            var yb = [msy-(mh/4), msy+(mh/4)];

            //redraw m-set with zoomed in bounds

            init( w,h,xb,yb,iter\_max );

            m();

        }

    </script>

</head>

<body>

​

    <div style="text-align:center">

        <div style="color: #333333">

            <h1>Mandelbrot Set Demo by Ibrahim S.</h1>

        </div>

        <div style="color: #999999">

            <h3>click anywhere to zoom in</h3>

        </div>

        <div style="color: #aa2299; font-style: bold;">

            <h5>CTRL+U for source code</h5>

        </div>

    </div>

    <!-- print the m set here -->

    <div id="m" style="text-align: center">

    </div>

    <!-- run logic -->

    <script>

        //place canvas onto page

        document.getElementById("m").appendChild(canvas);

        //initialize parameters

        init( 900,600,[-2,1],[-1,1],1000 );

        //run algorithm, print results

        m();

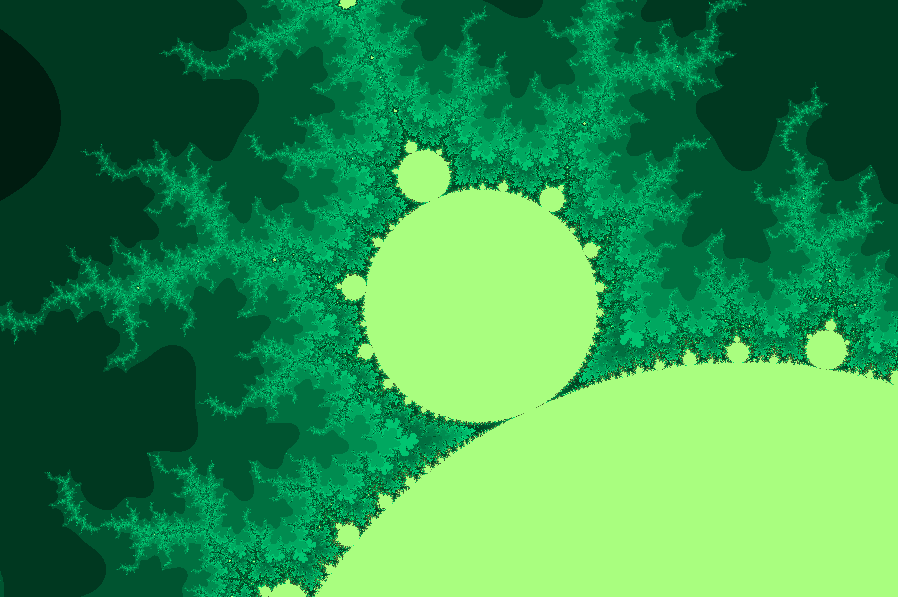
    </script>

​

</body>

</html>

**EXAMPLE OUTPUT**

**

*2500 iterations per pixel. Partial Set. Green.*