(module

(memory (import "imports" "memory") 1)

(func (export "mandelbrot")

(param $width i32)

(param $height i32)

(param $minX f64)

(param $maxX f64)

(param $minY f64)

(param $maxY f64)

(param $maxN i32)

;; ...

)

)

(func $mandelbrot (export "mandelbrot")

(param $width i32)

(param $height i32)

(param $minX f64)

(param $maxX f64)

(param $minY f64)

(param $maxY f64)

(param $maxN i32)

(local $i i32)

(local $j i32)

(local $dx f64)

(local $dy f64)

(local $x f64)

(local $y f64)

(local $a f64)

(local $b f64)

(local $c f64)

(local $n i32)

(local $index i32)

(set\_local $dx (f64.div (f64.sub (get\_local $maxX) (get\_local $minX)) (f64.convert\_u/i32 (get\_local $width))))

(set\_local $dy (f64.div (f64.sub (get\_local $maxY) (get\_local $minY)) (f64.convert\_u/i32 (get\_local $height))))

(set\_local $j (get\_local $height))

(set\_local $y (get\_local $minY))

(set\_local $index (i32.const 0))

(loop

(set\_local $i (get\_local $width))

(set\_local $x (get\_local $minX))

(loop

(set\_local $a (f64.const 0.0))

(set\_local $b (f64.const 0.0))

(set\_local $n (get\_local $maxN))

(block

(loop

;; $c = $a (just push it)

get\_local $a

;; Set $a = $a \* $a - $b \* $b + $x

(f64.mul (get\_local $a) (get\_local $a))

(f64.sub (f64.mul (get\_local $b) (get\_local $b)))

(set\_local $a (f64.add (get\_local $x)))

;; Set $b = 2 \* $c \* $b + y

(f64.mul (f64.const 2.0)) ;; This pops $a

(f64.mul (get\_local $b))

(set\_local $b (f64.add (get\_local $y)))

;; Set $n -= 1

(set\_local $n (i32.sub (get\_local $n) (i32.const 1)))

;; Break if $a \* $a + $b \* $b >= 4.0

(f64.add (f64.mul (get\_local $a) (get\_local $a)) (f64.mul (get\_local $b) (get\_local $b)))

(br\_if 1 (f64.ge (f64.const 4.0)))

;; Break if $n == 0, else loop

(br\_if 1 (i32.eqz (get\_local $n)))

br 0

)

)

;; Store (0xFF000000 | (($n \* 255 / ($maxN - 1)) & 0xFF)) at index $index

get\_local $index

(i32.trunc\_u/f64 (f64.div (f64.convert\_u/i32 (i32.mul (get\_local $n) (i32.const 255))) (f64.convert\_u/i32 (i32.sub (get\_local $maxN) (i32.const 1)))))

(i32.or (i32.const 0xFF000000))

i32.store

;; Same thing with S-expressions :

;;(i32.store

;; (get\_local $index)

;; (i32.or

;; (i32.const 0xFF000000)

;; (i32.trunc\_u/f64 (f64.div (f64.convert\_u/i32 (i32.mul (get\_local $n) (i32.const 255))) (f64.convert\_u/i32 (i32.sub (get\_local $maxN) (i32.const 1)))))

;; )

;;)

;; Set $index += 4

(set\_local $index (i32.add (get\_local $index) (i32.const 4)))

;; Set $x += $dx

(set\_local $x (f64.add (get\_local $x) (get\_local $dx)))

;; Loop if $i -- != 0, else exit

(set\_local $i (i32.sub (get\_local $i) (i32.const 1)))

(br\_if 0 (i32.ne (get\_local $i) (i32.const 0)))

)

;; Set $y += $dy

(set\_local $y (f64.add (get\_local $y) (get\_local $dy)))

;; Loop if $j -- != 0, else exit

(set\_local $j (i32.sub (get\_local $j) (i32.const 1)))

(br\_if 0 (i32.ne (get\_local $j) (i32.const 0)))

)

)

get\_local $index

(i32.trunc\_u/f64 (f64.div (f64.convert\_u/i32 (i32.mul (get\_local $n) (i32.const 255))) (f64.convert\_u/i32 (get\_local $maxN))))

(i32.or (i32.const 0xFF000000))

i32.store

N’est-il pas possible de simplifier en utilisant plus de S-expressions ? De fait, il est parfaitement possible d’écrire :

(i32.store

(get\_local $index)

(i32.or

(i32.const 0xFF000000)

(i32.trunc\_u/f64 (f64.div (f64.convert\_u/i32 (i32.mul (get\_local $n) (i32.const 255))) (f64.convert\_u/i32 (i32.sub (get\_local $maxN) (i32.const 1)))))

)

)

var width = 11, height = 11;

var importObject = {

imports: {

memory: new WebAssembly.Memory ({ initial: Math.ceil (width \* height \* 4 / 65536) })

}

};

const wasmInstance = new WebAssembly.Instance (wasmModule, importObject);

wasmInstance.exports.mandelbrot (width, height, -2.1, 1.1, -1.4, 1.4, 20);

var i, j, pixels;

pixels = new Uint32Array (importObject.imports.memory.buffer);

for (j = 0; j != height; j ++) {

for (i = 0; i != width ; i++)

console.log (`(${i}, ${j}) = ${pixels[i + j \* width].toString (16)}`);

}

var width = 801, height = 601, canvas;

canvas = document.createElement ("canvas");

canvas.setAttribute ("width", width);

canvas.setAttribute ("height", height);

document.body.appendChild (canvas);

inPixels = new Uint32Array (importObject.imports.memory.buffer);

context2d = canvas.getContext ("2d");

imageData = context2d.getImageData (0, 0, width, height);

outPixels = new Uint32Array (imageData.data.buffer);

outPixels.set (inPixels.slice (0, width \* height - 1));

context2d.putImageData (imageData, 0, 0);

var word = new Uint8Array (2);

word[0] = 0x01;

word[1] = 0x23;

word = new Uint16Array (word.buffer);

if (word[0] == 0x0123)

// Machine big endian

else

// Machine little endian