Mandulia

A zooming visualization of the Mandelbrot Set as many Julia Sets.

Fractal definitions

The Equation

- does iterating $z \mapsto z^2 + c$ remain bounded?

Julia Sets $J(c)$

- fix one c for the whole plane with z_0 at each point

The Mandelbrot Set M

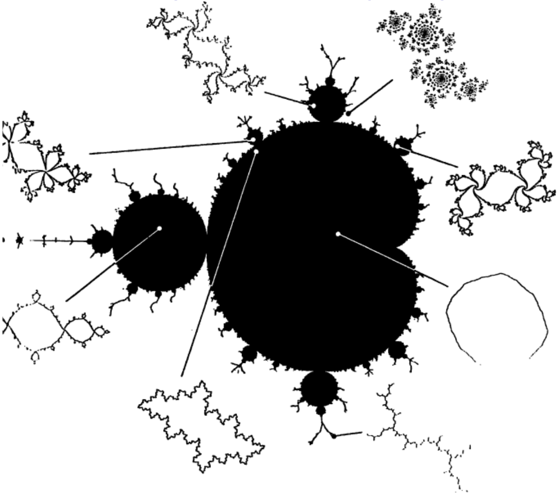
- vary c over the plane with $z_0 = 0$ for each point

The Connection

- $w \in M \Leftrightarrow J(w)$ is connected

Example

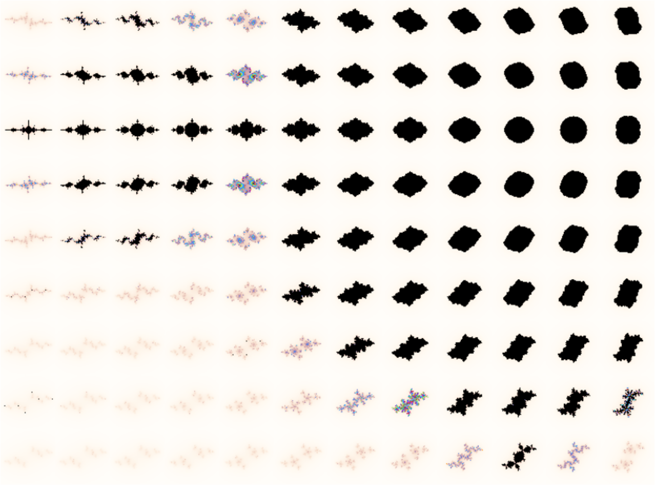
The Mandelbrot Set with some Julia Sets



(source: Falconer "Fractal Geometry: Mathematical Foundations and Applications")

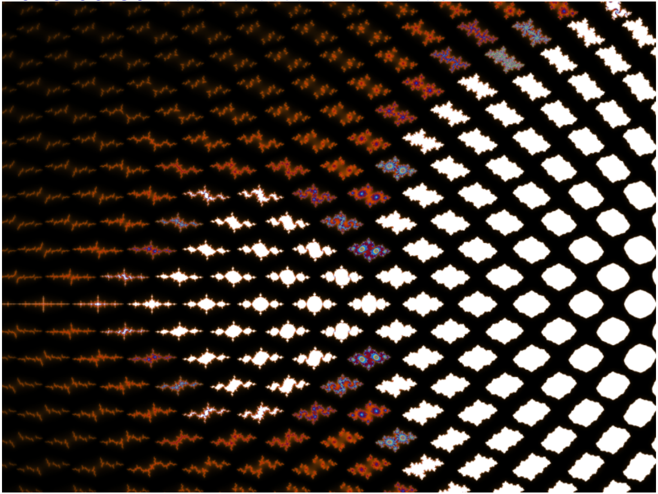
Changing Aesthetics

2010-06-29 first test: squares



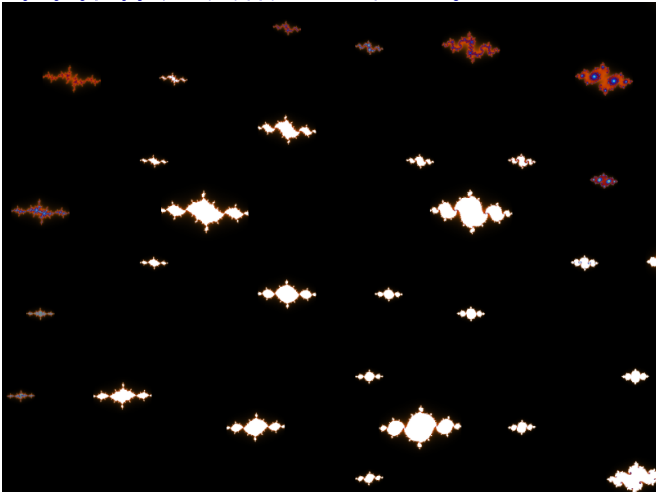
Changing Aesthetics

2010-06-30 second test: diamonds



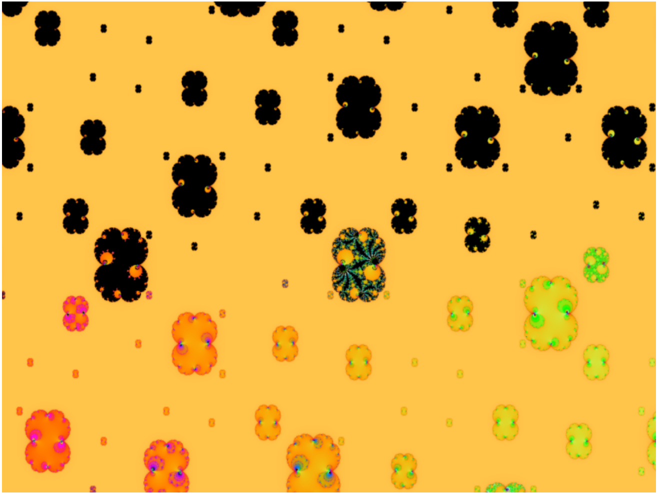
Changing Aesthetics

2010-07-06 third test: Ammann A3



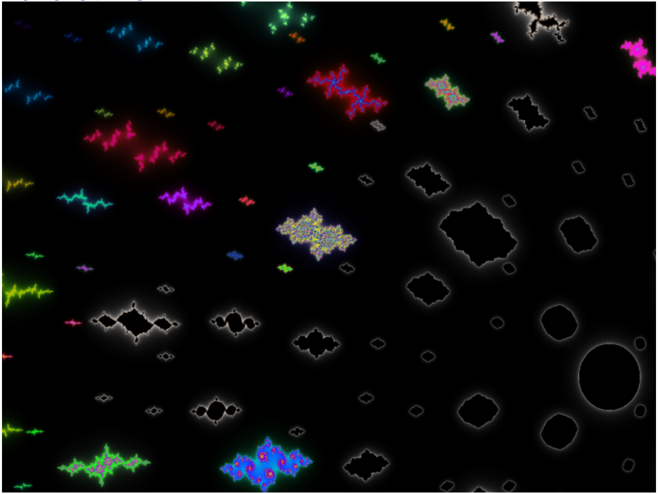
Changing Aesthetics

2010-07-10: golden sun



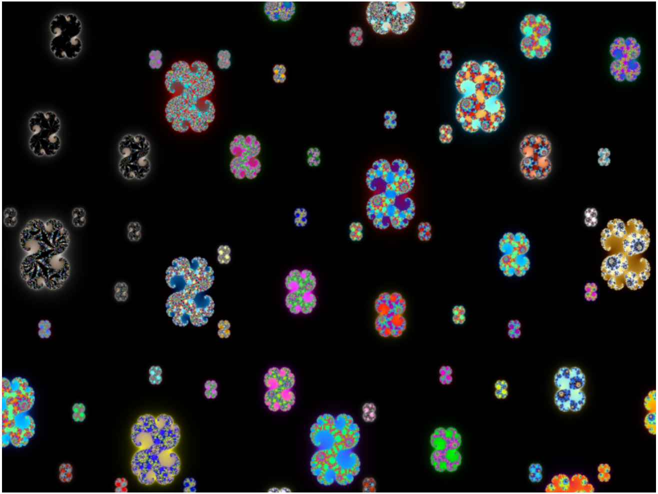
Changing Aesthetics

2010-07-23: neon



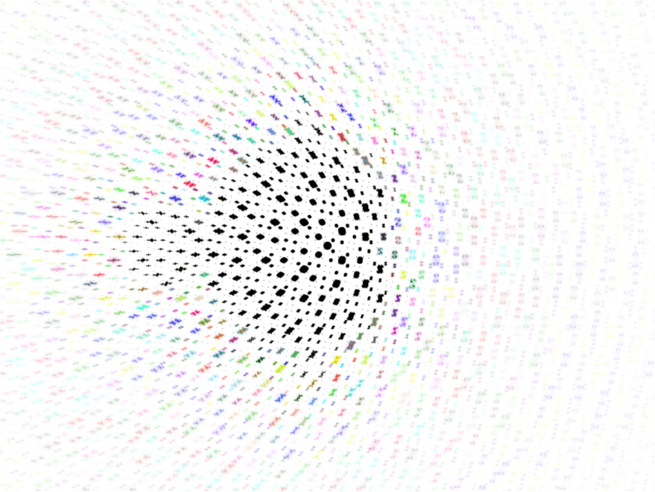
Changing Aesthetics

2010-07-23: neon (still)



Changing Aesthetics

Postcard: optimized for print



Implementation

The main program

- GLUT/OpenGL, initialize threads, interface with Lua

Julia Set renderer

- number crunching C with FFI, concurrency, resource pools

2D point layout

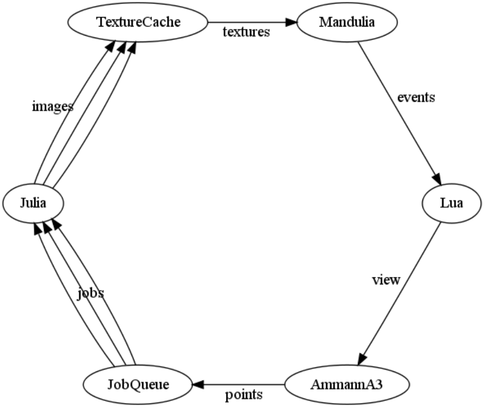
- Ammann A3 substitution tiling, irregular but uniform

Lua scripting

- configure and control animations and interactivity

Implementation

Data flow



The main program

Interaction

- passing user input to Lua scripts
- passing Lua state to view configuration

Calculation

- passing view configuration to Ammann A3 point layout
- adding points to the Julia Set renderer job queue

Visualization

- uploading rendered Julia Sets to GPU textures
- displaying the Julia Sets laid out in space

The Julia Set renderer

Number crunching

- tight loops in C to iterate the equation
- "foreign import" the function from c to image buffer

Concurrency

- multiple worker threads (1 per CPU core)
- each runs the "best right now" job

Recycle resources

- image pool: buffers in CPU memory, re-used after GPU upload
- texture cache: keep only the most relevant points on the GPU

Concurrency and OpenGL

Bound threads

- OpenGL can only be accessed by the “main” bound thread
- so, the Julia renderers cannot upload to GPU directly
- similarly the Lua virtual machine might not be thread-safe

Smooth appearance

- unpredictability of time taken to render each Julia Set
- so, number of images to upload each frame varies
- reduce jitter: swapBuffers at start of display callback

Job queue

High priority

- points that are visible but have no Julia Set yet
- points that are nearby and might be visible soon
- priorities can change every frame

Difficulties

- hard (impossible?) to abort “foreign” jobs
- limited GPU resources: cache the most relevant
- slowly completed jobs might now be irrelevant

Ammann A3

Substitution tiling

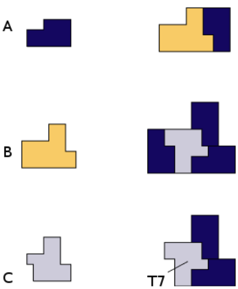
- similar to a quad tree, but instead of squares...
- ...three differently shaped tiles
- inflation factor $\phi = (\sqrt{5} + 1)/2$

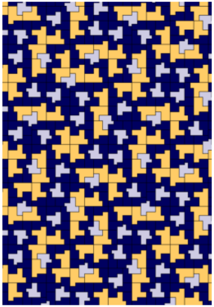
Attractive properties

- fixed point in rules gives stable zoom
- aperiodic (maybe?), irregular, ...
- ...but still has uniform point density

Ammann A3 rules

Diagram





(source: `http://tilings.math.uni-bielefeld.de`)

Ammann A3 implementation 1/4

Tiles, rules, IDs

```
data Tile = A | B | C -- tile shape
data TrID = T1 | ... | T9 -- (sub)rules

-- affine transformations from rules
transforms :: Tile -> [(Tile, (Matrix, TrID))]
bounds0 :: Tile -> Bounds -- initial bounds

normalizeID :: [TrID] -> [TrID]
normalizeID = dropWhile (T7 ==) -- fixed point
idToInteger :: [TrID] -> Integer
idToInteger = foldr ...
idToLevel :: [TrID] -> Int
idToLevel = length
```

Ammann A3 implementation 2/4

Trees

```
-- transformed tiles
data Tile'' = Tile'' Tile [TrId] Matrix
-- immediate ``children''
builder :: Tile'' -> [Tile'']
-- build a tree (a variant of unfoldTree)
tree' :: Tile'' -> Tree Tile''
tree' = unfoldTree2 builder

-- annotated tiles with depth <= level
data Tile' = Tile' Tile [TrId] Bounds
                            Vertex Int Int
-- with a specified maximum radius
tree :: Real -> Tree Tile'
```

Ammann A3 implementation 3/4

Zooming

```haskell
data AmmannA3 = AmmannA3
    (Forest Tile') -- tiles within bounds
    (Forest Tile') -- tiles overlapping bounds
    Bounds -- bounding box
    Real -- in-radius
ammannA3 :: Bounds -> AmmannA3
-- fails when new bounds aren't inside
zoomTo :: Bounds -> AmmannA3 -> Maybe AmmannA3
-- add another level of detail
stepIn :: AmmannA3 -> AmmannA3
-- flatten
tiles lod = map rootLabel
          . (\ a3 -> outer a3 ++ inner a3)
          . (!! lod) . iterate stepIn
```

Ammann A3 implementation 4/4

Pruning when zooming

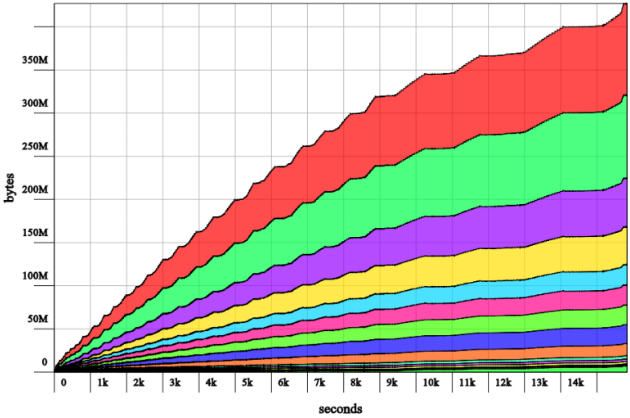
```
-- partition tiles
triPart :: Bounds -> Forest Tile'
        -> ( Forest Tile' -- inside bounds
           , Forest Tile' -- overlapping
           , Forest Tile' -- outside bounds
           )
```

Stepping in

- only check the overlapping tiles
- children of inside tiles will be inside
- outside tiles have been discarded already

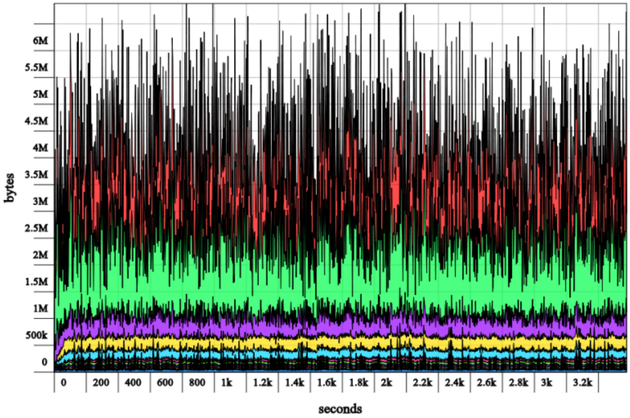
Ammann A3 profiling 1/2

Heap profile with too much sharing: $> 400MB$



Ammann A3 profiling 2/2

Heap profile with recomputation: $< 7MB$



Lua scripting

Interface

- find scripts using Cabal's "Paths_pkg" module
- call Lua functions on events (keyboard, each frame, ...)
- read Lua variables for configuration

Warts

- current implementation lacks proper error handling

Demo

(live or video)

Unanswered issues

Priority metrics

- smooth animation step size depends on current zoom level
- prioritization is unevenly spread between translation and zoom
- ...and also varies with the zoom level

Resource pools

- want: bounded number of resources (avoid death spiral)
- want: allocated only on demand (then kept for re-use)
- easy to have just one: both is hard?

The End

Mandulia

- cabal install mandulia
- cabal install mandulia -ffast
- cabal install mandulia -ffast -fSSE4

Claude Heiland-Allen

- http://claudiusmaximus.goto10.org
- mailto:claudiusmaximus@goto10.org