Micro Tiling Documentation

Aurélien Moisson-franckhauser, Johyn Papin, Quentin Boyer

Mar 21, 2019

Contents:

CHAPTER 1

Brainfuck Interpreter

class bf_interpreter.**Interpret**(*file_name*, *mem_size=65536*, *get_input=<function Interpret.<lambda>>*)

Interpret a brainfuck code, Loads *file_name* in a memory of size *mem_size*, loads the inputs with *get_input*

get_output(*size=0*)

If *size* is 0 gets the current output, else waits for the output size to be *size*

step()

Execute an instruction

CHAPTER 2

Database

database.**open_db**()
: Opens the database

database.**update_state**(*db*, *new_state*, *job_id*)
: Update the state in *db* for the job *job_id* if it is less than *new_state*

CHAPTER 3

Ensicoin Library

Wrapper around ensicoincoin-cli

ensicoin.**generate_keys**()
Generates a pair of (**private_key**, **public_key**)

ensicoin.**send_to**(*value*, *outpoint_hash*, *outpoint_index*, *privkey_from*, *spent_output_value*, *pubkey_to*, *flags*, *uid*)
Sends a transaction

ensicoin.**wait_for_flag**(*flag*)
Blocks until a transaction is issued containing *flag*

ensicoin.**wait_for_pubkey**(*pubKey*)
Blocks until a transaction is issued by *pubKey*

CHAPTER 4

Golfer: a Gopher Server

gopher.golfer.**get_entries**()
: returns *[(DirEntry)]* as defined by the gopher protocol

gopher.golfer.**get_selectors**()
: returns a list of selectors for the known files

gopher.golfer.**main**()
: Starts the gopher server

CHAPTER 5

Micro Tiling Client

5.1 Library usage

Module to generate mosaics using micro-tiling

client_library.client.**generate_mosaic**(*host*, *on_invoice*)
Wraps **manage_state** to return a result, uses *on_invoice* when needing to pay

client_library.client.**get_address**(*host*, *job_id*)
Gets the ensicoin address needing fees for *job*

client_library.client.**launch_job**(*host*)
Create a job for *host* and returns the **job_id**

client_library.client.**manage_state**(*host*, *result*)
Iterator on the completion of a job generation at *host*,

Writes the mosaic in the list *result*

5.2 Command line usage

A command line usage is defined for convienience, however it needs progressbar2.

It is used by invoking client.py with arguments the REST api address for micro-tiling and an ensicoin address used to pay

CHAPTER 6

a_pi

This is the module responsible for generating random segments using the digits of pi.

6.1 Segment generation

Generation of a bounded segment

a_pi.segment_generator.**random_segment**(*x_max*, *y_max*)
Creates a segment with each endpoint x, y such that:

$$x \in [0; x\_max], y \in [0; y\_max]$$

6.2 Flask Application

Flask a_pi executing an action on each digit of pi sent

a_pi.app.**action**(*db*, *job_id*, *job_current*)
Advances the state of the job *job_id* using the state *job_current* as inital state and executes an action, here segment generation

a_pi.app.**close_db**(*e=None*)
Closes the DB

a_pi.app.**create_app**(*test_config=None*)
Generates the Flask application

a_pi.app.**get_db**()
Get the db associated with the application

a_pi.app.**terminate**(*db*, *job_id*, *mill_stub*)
Finishes the job *job_id* and forwards it to the *mill_stub*

CHAPTER 7

Millllllll

This service is rotating segments on a gRPC call

7.1 Module for Rotating points

`Millllllll.rotate.`**`decode`**(*s*)
Takes six bytes *s* and returns a **point**

`Millllllll.rotate.`**`encode`**(*vect*)
Takes a **point** *vect* and converts it to six bytes

`Millllllll.rotate.`**`go_through_brainfuck`**(*file_name*, *point*, *use_python=False*)
passes *points* throught the brainfuck file *file_name*

`Millllllll.rotate.`**`mirror_and_turn`**(*points*, *use_python=False*)
points need to be tuples or lists of floats

`Millllllll.rotate.`**`mirror_and_turn_segments`**(*segments*, *use_python=False*)
segments need to be tuples or lists of **points** (will be returned as tuples), **points** need to be tuples or lists of floats

7.2 Millllllll Reference

gRPC server to rotate segments

`class` `Millllllll.millllllll.`**`MillServicer`**
Serves a Millllllll gRPC server

`Turn`(*request*, *context*)
Turns segments contained in the RPC message *request*

`Millllllll.millllllll.`**`segment_to_tuple`**(*segment*)
Transforms a *segment* from a dict to a tuple

Millllllll.millllllll.**serve**()
Start the Millllllll server

Millllllll.millllllll.**write**(*job*, *job_id*)
Write a *job* to a gopher served directory while notifing a golfer server of the *job_id*

CHAPTER 8

Unitator

This services clips segments fetched from a gopher server into the unit square, then writes them to firefox

8.1 Unitator Service

Unitator.unitator.**listen**()
Start a Unitator server listening for notifications of a golfer server

Unitator.unitator.**recv_data**(*conn*)
Receive all data from a socket *conn*

Unitator.unitator.**send**(*data*, *host*, *port*)
Sends *data* using a gopher connnection on *host*:*port*

Unitator.unitator.**unit**(*segments*, *job_id*)
Clip *segments* of the job *job_id* into a unit square and forwards them

Unitator.unitator.**write**(*job_id*, *text*)
Writes a *text* by an html form

8.2 Clipping Reference

Module clipping lines in the unit square

Unitator.clipping.**clip_right**(*lines*)
Clip *lines* with a vertical line to the right at x = 1

Unitator.clipping.**clip_unit_square**(*lines*)
Clip *lines* to fit in the unit square

class Unitator.clipping.**p**(*x*, *y*)
Class defining a two dimensional point in space

class Unitator.clipping.**segment**(*a*, *b*)
Class representing a segment by the two endpoints

Unitator.clipping.**turn_right**(*lines*)
Rotates *lines* by 90 degrees

CHAPTER 9

Translator

This translates and replicates segments from a inotify event and sends them by mail

9.1 Translator Reference

`Translator.translator.`**`decode`**(*s*)
Takes six bytes *s* and returns a **point** (tuple of coordinates)

`Translator.translator.`**`decode_nine`**(*s*)
Decode nine **points** from *s*

`Translator.translator.`**`encode`**(*vect*)
Takes a *vect* and converts it to six bytes

`Translator.translator.`**`go_through_brainfuck`**(*file_name*, *point*, *use_python=False*)
Passes *point* through the brainfuck file *file_name* Returns a list of translated points

`Translator.translator.`**`translate_segments`**(*segments*, *use_python=False*)
Translate and copies each segment from the list *segments* in all eight directions

9.2 Translator Service

`Translator.app.`**`create_app`**(*test_config=None*)
Launches the flask web application

`Translator.app.`**`encrypt`**(*string*, *gpg*)
Encrypt the *string* in pgp encoded using the context *gpg*

`Translator.app.`**`send`**(*host*, *segments*, *job_id*, *gpg*)
Sends the encoded *segments* to the next service by smtp throught *host* encrypted using the *gpg* context

`Translator.app.`**`translation`**(*segments*, *job_id*, *gpg*)
Creates replicas of the segments in the eight directions

CHAPTER 10

Cruxingator

This service cuts all segments such that no two segments cross, and publishes the result in an ensicoin transaction

10.1 SMTP Server

class Cruxingator.smtp_serv.**Handler**
Handles SMTP requests

handle_DATA(*server*, *session*, *envelope*)
Handles the content of a mail

handle_RCPT(*server*, *session*, *envelope*, *address*, *rcpt_options*)
Refuse all mails not in @micro-tiling.tk

Cruxingator.smtp_serv.**decrypt**(*string*)
Decrypts the *string* using gpg

10.2 Segment Spliting

class Cruxingator.split.**Point**(*pos*)
Points in two dimensions

get_pos(*precision=0.0003*)
Get a rounded position at *precision* level

merge(*other*)
Tells two points are equivalent

class Cruxingator.split.**Segment**(*a*, *b*)
Segment between two endpoints

intersect(*other*)
Calculate the intersection point beetween *self* and *other*

class Cruxingator.split.**Vect**(*x*, *y*)
Two dimensional Vector

norm2()
Squared norm of the vector

Cruxingator.split.**cut**(*segments*)
Cuts the *segments* such that no two segments cross

Cruxingator.split.**find_intersections**(*segments*)
Find all intersections between *segments*

Cruxingator.split.**generate_id_tuple**(*points*)
Generates a tuple for each point (id, x, y, [id of linked])

Cruxingator.split.**pairs**(*l*)
Iterate on pairs of segments

CHAPTER 11

Solidator

This service works in two parts: a master program managing subprograms

11.1 Master

signification of messages to solidator : r : the point is ready. d : there is one less point of degree 1 alive. e : a point process has ended.

class Solidator.solidator.**Point**(*pos*)
A two dimensional point whose *pos* is a **Vect**

get_pos(*precision=1e-10*)
Returns a position rounded up to *precision*

merge(*other*)
Merges with another **Point**

class Solidator.solidator.**Segment**(*a*, *b*)
A segment of two vectors

class Solidator.solidator.**Vect**(*x*, *y*)
A two dimensional vector

Solidator.solidator.**create_points**(*point_list*)
Takes in a list of [(point_id, pos_x, pos_y, [list of neighbours' ids]),...] and output a list of **Point**

Solidator.solidator.**open_process**(*point*)
Opens a process representing a *point* tells it how many neighbours it should expect and its *Point id*

Solidator.solidator.**remove_deg_1**(*points*, *job_id*, *multiprocess=True*)
Write an svg with only closed polygons displayed

11.2 Subprograms

processes are sent the IDs of their neighbours at start once they have them, they start telling others if they have a degree of 1 and they listen for neighbours signaling they have a degree of 1

signification of messages to point_processes :

e [message comming from main meaning it is now time to write] the result.

d <neighbour_id> [the neighbouring point with neighbour_id] had only one neighbour left 'alive' and is therefore now 'dead'.

a : death acknowledge means the neighbour now knows the point is 'dead'

A <neighbour_id> <neighbour_x_pos> <neighbour_y_pos> [the] corresponding neighbour is still 'alive' (sent after the neighbour received a message 'e').

D [a neighbour is 'dead' and no line should be displayed between the] two points.

`Solidator.point_process.`**`debug`**`(`*string*, *own_id*`)`
write a *string* in stderr and add the point's *own_id* for clarity

`Solidator.point_process.`**`main`**`()`
Waits for processes to die around you to check if you can survive

`Solidator.point_process.`**`read_neighbours`**`(`*amount*`)`
Reads *amount* neighbours from **stdin**

`Solidator.point_process.`**`read_position`**`()`
Reads a line from **stdin** and interprets it as a position

`Solidator.point_process.`**`signal_death`**`(`*neighbour_messagers*, *own_id*`)`
Tells all neighbours of *own_id* death by *neighbour_messagers*' fifo files

`Solidator.point_process.`**`svg_coord`**`(`*x*`)`
Returns the coordinate in svg space for *x*

`Solidator.point_process.`**`svg_output`**`(`*message*, *own_id*, *position*, *res*`)`
Outputs to *res* the svg line from *position* to the neighbour indicated in *message* if *own_id* is smaller than the neighbour's id (this is to avoid writing twice the same line)

`Solidator.point_process.`**`who_am_i_at_the_end`**`(`*is_dead*, *neighbour_messagers*, *own_id*, *position*`)`
Tells all neighbours by *neighbour_messagers* the vertex *own_id* and *position* if not *is_dead*, else send them D

CHAPTER 12

REST API Usage

A running API is hosted at `https://micro-tiling.tk`

12.1

POST /

Creates a new job

- **Description:**
- **Produces:** [‘application/json’]

Responses

200 - Succesfuly created job

Name	Description	Type
id		string

405 - Invalid input

GET /{jobId}/state

Gets the state of a job

- **Description:**
- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - State of the job

Name	Description	Type
completion		integer
state		string

400 - Invalid ID supplied

404 - Unknown Job

405 - Invalid Method

GET /{jobId}/address

Gets the address needing ensicoins for a job

- **Description:**
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - Ensicoin address

Name	Description	Type
address		string

400 - Invalid ID supplied

404 - Unknown Job or No Address needed

405 - Invalid Method

GET /{jobId}/result

Get the final mosaic in a job

- **Description:**
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
jobId	path	The ID of the requested job	string

Responses

200 - SVG file

Name	Description	Type
result		string

400 - Invalid ID supplied

404 - Unknown Job or Job not done

405 - Invalid Method

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

N

O

P

R

S

T

U

V

W