PyZUI
Manual

David Roberts

July 15, 2009

Abstract

PyZUI is an implementation of a Zooming User Interface (ZUI) for Python. Media is laid out upon an infinite virtual desktop, with the user able to pan and zoom through the collection.

PyZUI is compatible with the following media formats:

- All images recognised by ImageMagick[1]
- PDF documents
- Remote webpages
- SVG (vector graphics)

Raster images of arbitrary size can be viewed due to the use of image tiling[2]. The largest image tested so far is the 233 megapixel Blue Marble Next Generation 2km/pixel satellite image[3].

PyZUI also allows viewing of remote media from the following online services:

- OpenStreetMap[4] (world-wide street directory)
- The WMS Global Mosaic[5] (NASA satellite imagery)

Examples of similar projects include Seadragon/DeepZoom[6], OpenZoom[7], Zoomorama[8], and Eagle Mode[9]. More basic ZUIs can also be found in software such as Compiz and Spaces (included in Mac OS X), as well as devices such as the Nintendo Wii and Apple iPhone. A more complete listing of other projects can also be found at Wikipedia[10].

[1]`http://imagemagick.org/script/formats.php`

[2]`http://star.pst.qub.ac.uk/idl/Image_Tiling.html`

[3]`http://earthobservatory.nasa.gov/Features/BlueMarble/images_bmng/2km/world.topo.bathy.200407.3x21600x10800.jpg`

[4]`http://openstreetmap.org/`

[5]`http://onearth.jpl.nasa.gov/`; 1.3TB, max. resolution 15m/pixel

[6]`http://livelabs.com/seadragon/`; Presentation: `http://ted.com/index.php/talks/blaise_aguera_y_arcas_demos_photosynth.html`

[7]`http://openzoom.org/`; Demonstration: `http://tandem.gasi.ch`

[8]`http://zoomorama.com/`

[9]`http://eaglemode.sf.net/`

[10]`http://en.wikipedia.org/wiki/ZUI`

Chapter 1

User Interface

Upon startup of PyZUI, the user is presented with the *home scene*, as shown in Figure 1.1.

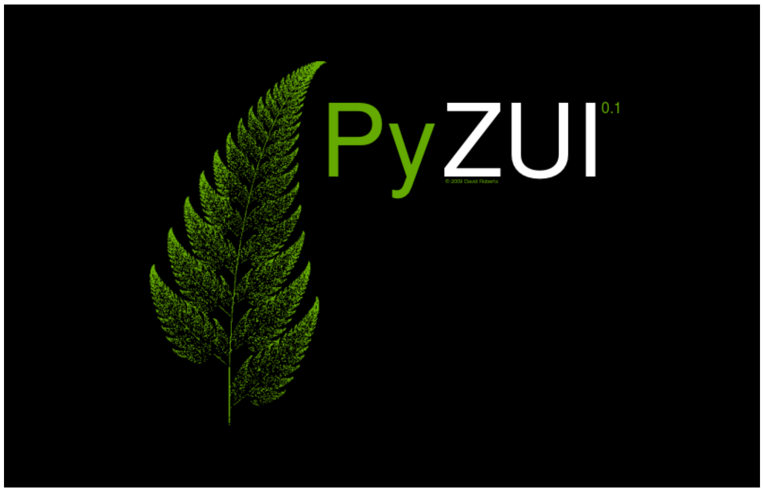

Figure 1.1: Home Scene

The menus provide the following actions:

File

- **New Scene (Ctrl+N)** Create a blank scene
- **Open Scene (Ctrl+O)** Open a saved scene
- **Open Home Scene (Ctrl+Home)** Return to the *home scene*
- **Save Scene (Ctrl+S)** Save the current scene
- **Save Screenshot (Ctrl+H)** Export the current viewport to an image

- **Open Local Media (Ctrl+L)** Open media from a local file
- **Open Media by URI (Ctrl+U)** Open media identified by a URI
  The following URI formats are recognised:
  - any valid URL e.g. `http://google.com/`
  - `dynamic:mandel`: the Mandelbrot set
  - `dynamic:osm`: OpenStreetMap
  - `dynamic:gm`: the WMS Global Mosaic
  - `dynamic:fern`: the PyZUI logo (aka Barnsley's fern)
  - `string:rrggbb:foobar`: a string, where `rrggbb` is a string of three two-digit hexadecimal numbers representing the colour of the text, and `foobar` is the text to be displayed
    e.g. `string:00ff00:hello` will display "hello" in green
- **Open Media Directory (Ctrl+D)** Open the media contained in a directory, and arrange it into a grid (see Figure 1.2)

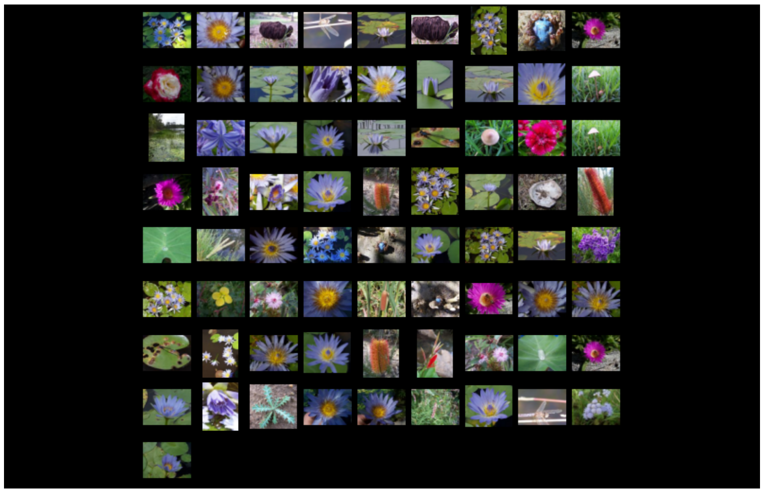

Figure 1.2: Media directory

- **Quit (Ctrl+Q)** Exit the application

View
- **Set Framerate** Set the rendering framerate to the selected frequency
- **Fullscreen (Ctrl+F)** Toggle fullscreen mode

Help
- **About** Show PyZUI copyright information
- **About Qt** Show Qt about dialog

The following mouse/keyboard actions are also available:

Left-click Select the foremost media under the cursor

- if there is no media under the cursor then the currently selected media will be deselected
- if the Shift key is currently being held then no change will be made to the current selection

Click'n'drag Select and move the foremost media under the cursor

- if there is no media under the cursor then the currently selected media will be deselected and the entire scene will be moved
- if the Shift key is currently being held then no change will be made to the current selection and the entire scene will be moved

Esc Deselect the currently selected media

PgUp/PgDn or Scrollwheel Zoom the currently selected media

- if there is no currently selected media, or if the Shift key is currently being held, then the entire scene will be zoomed
- if the Alt key is currently being held, then the zoom amount will reduced allowing for finer control
- Note: the point under the cursor will maintain its position on the screen

Arrow keys Move the currently selected media in the specified direction

- if there is no currently selected media, or if the Shift key is currently being held, then the entire scene will be moved
- if the Alt key is currently being held, then the move amount will reduced allowing for finer control

Space bar Move the point under the cursor to the centre of the screen

- holding the Space bar allows panning by moving the cursor around the centre of the viewport

Del Delete the currently selected media

Chapter 2

Class Design

Contents

2.1 Module pyzui.converter

A threaded media converter (abstract base class).

2.1.1 Class Converter

Converter objects are used for converting media.

Constructor: Converter(string, string)

Derived from: `threading.Thread`, `threading._Verbose`

Converter(infile, outfile) Create a new Converter for converting media at the location given by `infile` to the location given by `outfile`.

Where appropriate, the output format will be determined from the file extension of `outfile`.

run() Run the conversion. If any errors are encountered then `self.error` will be set to a string describing the error.

run() → None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead.

Overrides threading.Thread.run

progress Conversion progress ranging from 0.0 to 1.0. A value of 1.0 indicates that the converter has completely finished.

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.2 Module pyzui.magickconverter

Image converter based upon ImageMagick.

2.2.1 Class MagickConverter

MagickConverter objects are used for converting media with ImageMagick. For a list of supported image formats see `http://imagemagick.org/script/formats.php`

Constructor: MagickConverter(string, string)

Derived from: `pyzui.converter.Converter`, `threading.Thread`, `threading._Verbose`

MagickConverter(infile, outfile) Create a new Converter for converting media at the location given by `infile` to the location given by `outfile`.

Where appropriate, the output format will be determined from the file extension of `outfile`. *(Docstring inherited from Converter)*

run() Run the conversion. If any errors are encountered then `self.error` will be set to a string describing the error.

run() → None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead. *(Docstring inherited from Converter)*

Overrides pyzui.converter.Converter.run

Inherited from pyzui.converter.Converter: progress

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.3 Module pyzui.pdfconverter

PDF rasterizer based upon either Xpdf or Poppler.

2.3.1 Class PDFConverter

PDFConverter objects are used for rasterizing PDFs.

The output format will always be PPM irrespective of the file extension of the output file. If another output format is required then PDFConverter should be used in conjunction with MagickConverter.

Constructor: PDFConverter(string, string)

Derived from: `pyzui.converter.Converter`, `threading.Thread`, `threading._Verbose`

PDFConverter(infile, outfile) Create a new Converter for converting media at the location given by `infile` to the location given by `outfile`.

Where appropriate, the output format will be determined from the file extension of `outfile`. *(Docstring inherited from Converter)*

run() Run the conversion. If any errors are encountered then `self.error` will be set to a string describing the error.

run() → None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead. *(Docstring inherited from Converter)*

Overrides pyzui.converter.Converter.run

Inherited from pyzui.converter.Converter: progress

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.4 Module pyzui.webkitconverter

Webpage renderer based upon QtWebKit.

2.4.1 Class WebKitConverter

WebKitConverter objects are used for rendering webpages.

`infile` may be either a URI or the location of a local file.

Supported output formats are: BMP, JPG/JPEG, PNG, PPM, TIFF, XBM, XPM (see `http://doc.trolltech.com/qimage.html#reading-and-writing-image-files`)

Constructor: WebKitConverter(string, string)

Derived from: `pyzui.converter.Converter`, `threading.Thread`, `threading._Verbose`

WebKitConverter(infile, outfile) Create a new Converter for converting media at the location given by `infile` to the location given by `outfile`.

Where appropriate, the output format will be determined from the file extension of `outfile`. *(Docstring inherited from Converter)*

run() Run the conversion. If any errors are encountered then `self.error` will be set to a string describing the error.

run() → None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead. *(Docstring inherited from Converter)*

Overrides pyzui.converter.Converter.run

start() If a global QApplication has already been instantiated, then this method will call `run()` directly, as Qt requires us to call `run()` from the same thread as the global QApplication. This will not block as Qt will then handle the threading.

Otherwise, if no QApplication exists yet, then the default `Converter.start()` method will be called allowing Python to natively handle the threading.

start() → None

Overrides threading.Thread.start

Inherited from pyzui.converter.Converter: progress

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name)

2.5 Module pyzui.tilestore

Module for managing the disk-based tile storage facility.

2.5.1 Functions

get_media_path(media_id) Return the path to the directory containing the tiles for the media identified by `media_id`.

get_media_path(string) → string

get_metadata(media_id, key) Return the value associated with the given metadata key, None if there is no such value.

get_metadata(string, string) → object or None

get_tile_path(tile_id, mkdirp=False, prefix=None, filext=None) Return the path to the tile identified by `tile_id`.

If `mkdirp` is True, then any non-existent parent directories of the tile will be created.

If `prefix` is omitted, it will be set to the value returned by `get_media_path`.

If `filext` is omitted, it will be set to the value returned by `get_metadata`.

get_tile_path(tuple<string,int,int,int>[, bool[, string[, string]]]) → string

Precondition: if `filext` is None, then the metadata file for the media that this tile belongs to exists and contains an entry for filext

load_metadata(media_id) Load metadata from disk for the given `media_id`, and return a bool indicating whether the load was successful.

load_metadata(string) → bool

tiled(media_id) Return True iff the media identified by `media_id` has been tiled i.e. iff both a metadata file and the (0,0,0) tile exist.

tiled(string) → bool

write_metadata(media_id, **kwargs) Write the metadata given in `kwargs` for the given `media_id`.

write_metadata(string, metadata_key=metadata_val, ...) → None

2.5.2 Data

disk_lock <_RLock(None, 0)>

tile_dir ~/.pyzui/tilestore

2.6 Module pyzui.tiler

Threaded image tiler (abstract base class).

2.6.1 Class Tiler

Tiler objects are used for tiling images.

Constructor: Tiler(string[, string[, string[, int]]])

Derived from: `threading.Thread`, `threading._Verbose`

Tiler(infile, media_id=None, filext='jpg', tilesize=256) Create a new Tiler for tiling the media given by `media_id` with the image given by `infile`.

If `media_id` is omitted, it will be set to `infile`.

Tiles will be saved in the format indicated by `filext` and with the dimensions given by `tilesize`.

run() Tile the image. If any errors are encountered then `self.error` will be set to a string describing the error.

run() $\rightarrow$ None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead.

Overrides threading.Thread.run

progress Tiling progress ranging from 0.0 to 1.0. A value of 1.0 indicates that the tiling has completely finished.

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.7 Module pyzui.ppm

Module for reading PPM images.

2.7.1 Class PPMTiler

PPMTiler objects are used for tiling PPM images.

Constructor: PPMTiler(string[, string[, string[, int]]])

Derived from: `pyzui.tiler.Tiler`, `threading.Thread`, `threading._Verbose`

PPMTiler(infile, media_id=None, filext='jpg', tilesize=256) Create a new Tiler for tiling the media given by `media_id` with the image given by `infile`.

If `media_id` is omitted, it will be set to `infile`.

Tiles will be saved in the format indicated by `filext` and with the dimensions given by `tilesize`. *(Docstring inherited from Tiler)*

Inherited from pyzui.tiler.Tiler: progress, run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.7.2 Functions

read_ppm_header(f) Read the PPM header in the given file object `f` and return a tuple representing the dimensions of the image.

Raises `IOError` if the header is invalid and/or unsupported (must be 'P6' binary PPM format with maxval=255).

read_ppm_header(file) → (long, long)

2.8 Module pyzui.tile

Class for representing image tiles.

2.8.1 Class Tile

Tile objects allow storage and manipulation of image tiles.
Constructor: Tile(Image or QImage)

Tile(image) Create a new tile with the given image.

crop(bbox) Return the region of the tile contained in the bounding box `bbox` (x1,y1,x2,y2).

crop(tuple<int,int,int,int>) → Tile

draw(painter, x, y) Draw the tile on the given `painter` at the given position.

paint(QPainter, int, int) → None

resize(width, height) Return a resized copy of the tile.

resize(int, int) → Tile

save(filename) Save the tile to the location given by `filename`.

save(string) → None

size The dimensions of the tile.

2.8.2 Functions

fromstring(string, width, height) Create a new tile from a `string` of raw pixels, with the given dimensions.

fromstring(string, int, int) → Tile

merged(t1, t2, t3, t4) Merge the given tiles into a single tile.

`t1` must be a Tile, but any or all of `t2`,`t3`,`t4` may be None, in which case they will be ignored.

merged(Tile, Tile or None, Tile or None, Tile or None) → Tile

new(width, height) Create a new tile with the given dimensions.

new(int, int) → Tile

2.9 Module pyzui.tilecache

Thread-safe Least Recently Used (LRU) cache for storing tiles.

2.9.1 Class TileCache

TileCache objects are used for caching tiles in memory.

Tiles can be accessed in much that same way as `dict` objects: `tilecache[tile_id]` holds the tile identified by the given `tile_id`.

Constructor: TileCache(int, int)

TileCache(maxsize=256, maxage=60) Create a new TileCache object.

> The maximum number of tiles to store is set by `maxsize`. There will be no limit if `maxsize` ≤ 0.
>
> The maximum age of the tiles (in seconds) allowed before they are discarded is set by `maxsize`. There will be no limit if `maxage` ≤ 0.
>
> None tiles and (0,0,0) tiles do not count towards the number of stored tiles and will therefore not be automatically discarded.

insert(tile_id, tile, maxaccesses=0) Insert the `tile` with the given `tile_id` into the cache.

> If `maxaccesses` ≤ 0, then the behaviour is the same as `tilecache[tile_id]=tile`. Otherwise the tile is set to expire after it has been accessed `maxaccesses` times.
>
> insert(tuple<string,int,int,int>, object, int) → None

purge() Purge all tiles from the cache.

> purge() → None

2.10 Module pyzui.tileprovider

Threaded class for loading tiles into memory (abstract base class).

2.10.1 Class TileProvider

TileProvider objects are used for loading tiles into TileCache objects.

Constructor: TileProvider(TileCache)

Derived from: `threading.Thread`, `threading._Verbose`

TileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`.

purge(media_id=None) Purge all tasks for the given `media_id`. All tasks will be purged if `media_id` is omitted.

purge([string]) → None

request(tile_id) Request the tile identified by `tile_id` be loaded into the tilecache.

Requests are processed in a LIFO order.

If the tile is unavailable, then None will be inserted into the tilecache to indicate this.

request(tuple<string,int,int,int>) → None

run() Run a loop to load requested tiles.

run() → None

Note: for threading to be enabled this method should not be called directly, call the `start()` method instead.

Overrides threading.Thread.run

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.11 Module pyzui.statictileprovider

Class for loading tiles from the local tilestore.

2.11.1 Class StaticTileProvider

StaticTileProvider objects are used for loading tiles from the disk-cache into a TileCache.

Constructor: StaticTileProvider(TileCache)

Derived from: `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

StaticTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.12 Module pyzui.dynamictileprovider

Class for loading tiles into memory from somewhere other than the local filesystem (abstract base class).

2.12.1 Class DynamicTileProvider

DynamicTileProvider objects are used for either generating tiles or loading them from a remote host, and then loading them into a TileCache.

Constructor: DynamicTileProvider(TileCache)

Derived from: `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

DynamicTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

aspect_ratio `1.0`

filext `'png'`

tilesize `256`

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.13 Module pyzui.osmtileprovider

Dynamic tile provider for OpenStreetMap.

2.13.1 Class OSMTileProvider

OSMTileProvider objects are used for downloading tiles from OpenStreetMap (`http://openstreetmap.org/`).

Constructor: OSMTileProvider(TileCache)

Derived from: `pyzui.dynamictileprovider.DynamicTileProvider`, `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

OSMTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

Inherited from pyzui.dynamictileprovider.DynamicTileProvider: aspect_ratio, filext, tilesize

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.14 Module pyzui.globalmosaictileprovider

Dynamic tile provider for the WMS Global Mosaic.

2.14.1 Class GlobalMosaicTileProvider

GlobalMosaicTileProvider objects are used for downloading tiles from the WMS Global Mosaic server `http://onearth.jpl.nasa.gov/`.

Constructor: GlobalMosaicTileProvider(TileCache)

Derived from: `pyzui.dynamictileprovider.DynamicTileProvider`, `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

GlobalMosaicTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

aspect_ratio `2.0`

base_params `{'layers': 'global_mosaic', 'srs': 'EPSG:4326', 'version': '1.1.1', 'width': 512, 'styles': 'visual', 'format': 'image/jpeg', 'request': 'GetMap', 'height': 512}`

base_url `'http://wms.jpl.nasa.gov/wms.cgi?'`

filext `'jpg'`

Inherited from pyzui.dynamictileprovider.DynamicTileProvider: tilesize

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.15 Module pyzui.mandeltileprovider

Dynamic tile provider for the Mandelbrot set.

2.15.1 Class MandelTileProvider

MandelTileProvider objects are used for generating tiles of the Mandelbrot set using jrMandel (`http://freshmeat.net/projects/jrmandel/`).

Constructor: MandelTileProvider(TileCache)

Derived from: `pyzui.dynamictileprovider.DynamicTileProvider`, `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

MandelTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

Inherited from pyzui.dynamictileprovider.DynamicTileProvider: aspect_ratio, filext, tilesize

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.16 Module pyzui.ferntileprovider

Dynamic tile provider for Barnsley's fern.

2.16.1 Class FernTileProvider

FernTileProvider objects are used for generating tiles of Barnsley's fern iterated function system.

Constructor: FernTileProvider(TileCache)

Derived from: `pyzui.dynamictileprovider.DynamicTileProvider`, `pyzui.tileprovider.TileProvider`, `threading.Thread`, `threading._Verbose`

FernTileProvider(tilecache) Create a new TileProvider for loading tiles into the given `tilecache`. *(Docstring inherited from TileProvider)*

color `(100, 170, 0)`

max_iterations `50000`

max_points `10000`

transformations `[(0.01, (0.0, 0.0, 0.0, 0.0, 0.16, 0.0)), (0.070000000000000007, (0.20000000000000001, -0.26000000000000001, 0.0, 0.23000000000000001, 0.22, 1.6000000000000001)), (0.070000000000000007, (-0.14999999999999999, 0.28000000000000003, 0.0, 0.26000000000000001, 0.23999999999999999, 0.44)), (0.84999999999999998, (0.84999999999999998, 0.040000000000000001, 0.0, -0.040000000000000001, 0.84999999999999998, 1.6000000000000001))]`

Inherited from pyzui.dynamictileprovider.DynamicTileProvider: aspect_ratio, filext, tilesize

Inherited from pyzui.tileprovider.TileProvider: purge(media_id=None), request(tile_id), run()

Inherited from threading.Thread: daemon, getName(), ident, isAlive(), isAlive(), isDaemon(), join(timeout=None), name, setDaemon(daemonic), setName(name), start()

2.17 Module pyzui.tilemanager

The TileManager is responsible for requesting tiles from TileProviders, caching them in memory, and providing them to MediaObjects when requested to do so.

It is also responsible for creating new tiles from available ones when no tiles of the requested resolution are available.

2.17.1 Class MediaNotTiled

Exception for when tiles are requested from a media that has not been tiled yet.

This exception will never be thrown when requesting a tile from a dynamic media.

Derived from: `exceptions.Exception`, `exceptions.BaseException`
Inherited from exceptions.Exception: __init__
Inherited from exceptions.BaseException: args, message

2.17.2 Class TileNotAvailable

Exception for when an attempt to load the requested tile has previously failed.

Derived from: `exceptions.Exception`, `exceptions.BaseException`
Inherited from exceptions.Exception: __init__
Inherited from exceptions.BaseException: args, message

2.17.3 Class TileNotLoaded

Exception for when tiles are requested before they have been loaded into the tile cache.

Derived from: `exceptions.Exception`, `exceptions.BaseException`
Inherited from exceptions.Exception: __init__
Inherited from exceptions.BaseException: args, message

2.17.4 Functions

cut_tile(tile_id, tempcache=0) Create a tile from resizing and cropping those loaded into the tile cache. Returns a tuple containing the tile, and a bool `final` which is False iff the tile is not the greatest resolution possible and should therefore not be cached indefinitely.

If `tempcache` > 0, then tiles with `final`=False will cached in the TileCache, but will expire after they have been accessed `tempcache` times.

This function should only be called if a `TileNotLoaded` or `TileNotAvailable` error has been encountered.

cut_tile(tuple<string,int,int,int>, int) → tuple<Tile,bool>

Precondition: the (0,0,0) tile exists for the given media

Precondition: the requested tile doesn't fall outside the bounds of the image

get_metadata(media_id, key) Return the value associated with the given metadata `key` for the given `media_id`, None if there is no such value.

get_metadata(string, string) → object or None

get_tile(tile_id) Return the requested tile identified by `tile_id`.

If the tile is not available in the tilecache, one of three errors will be

raised: `MediaNotTiled`, `TileNotLoaded`, or `TileNotAvailable`

get_tile(tuple<string,int,int,int>) → Tile

get_tile_robust(tile_id) Will try returning the result of `get_tile`, and if that fails will return the result of `cut_tile`.

This function will not raise `TileNotLoaded` or `TileNotAvailable`, but may raise `MediaNotTiled`.

get_tile_robust(tuple<string,int,int,int>) → Tile

init(total_cache_size=192) Initialise the TileManager. This **must** be called before any other functions are called.

init() → None

load_tile(tile_id) Request that the tile identified by `tile_id` be loaded into the tilecache.

load_tile(tuple<string,int,int,int>) → None

purge(media_id=None) Purge the specified `media_id` from the `TileProviders`. If `media_id` is omitted then all media will be purged.

purge([string]) → None

Precondition: the media to be purged should not be active (i.e. no `MediaObjects` for the media should exist).

tiled(media_id) Returns True iff the media identified by `media_id` has been tiled.

Will always return True for dynamic media.

tiled(string) → bool

2.18 Module pyzui.physicalobject

An object that obeys the laws of physics.

2.18.1 Class PhysicalObject

PhysicalObject objects are used to represent anything that has a 3-dimensional position and velocity, where the z-dimension represents a zoomlevel.

Constructor: PhysicalObject()

PhysicalObject() Create a new PhysicalObject at the origin with zero velocity.

aim(v, s, t=None) Calculate the initial velocity such that at time `t` the relative displacement of the object will be `s`, and increase the velocity represented by `v` by this amount.

If `t` is omitted then it will be taken that `s` is the limit of the displacement as `t` approaches infinity i.e. the initial velocity will be calculated such that the total displacement will be `s` once the object has stopped moving.

velocity(string, float[, float]) → None

Precondition: `v` is either 'x', 'y', or 'z'

move(dx, dy) Move the object by (`dx`,`dy`).

move(float, float) → None

step(t) Step forward `t` seconds in time.

step(float) → None

zoom(amount) Zoom by the given `amount` with the centre maintaining its position on the screen.

zoom(float) → None

centre The on-screen coordinates of the centre of the object (the point that will maintain its position on the screen as the object is being zoomed).

moving Boolean value indicating whether the object has a non-zero velocity.

zoomlevel The zoomlevel of the object.

damping_factor `256`

2.19 Module pyzui.mediaobject

Media to be displayed in the ZUI (abstract base class).

2.19.1 Class LoadError

Exception for if there is an error loading the media.

Derived from: `exceptions.Exception`, `exceptions.BaseException`

Inherited from exceptions.Exception: __init__

Inherited from exceptions.BaseException: args, message

2.19.2 Class MediaObject

MediaObject objects are used to represent media that can be rendered in the ZUI.

Constructor: MediaObject(string, Scene)

Derived from: `pyzui.physicalobject.PhysicalObject`

MediaObject(media_id, scene) Create a new MediaObject from the media identified by `media_id`, and the parent Scene referenced by `scene`.

fit(bbox) Move and resize the image such that it the greatest size possible whilst fitting inside and centred in the onscreen bounding box `bbox` (x1,y1,x2,y2).

fit(tuple<float,float,float,float>) → None

hides(other) Returns True iff `other` is completely hidden behind `self` on the screen.

hides(MediaObject) → bool

move(dx, dy) Move the image relative to the scene, where (`dx`,`dy`) is given as an on-screen distance.

move(float, float) → None

Overrides pyzui.physicalobject.PhysicalObject.move

render(painter, mode) Render the media using the given `painter` and rendering `mode`.

render(QPainter, int) → None

Precondition: `mode` is equal to one of the constants defined in `RenderMode`

zoom(amount) Zoom by the given `amount` with the centre maintaining its position on the screen.

zoom(float) → None

Overrides pyzui.physicalobject.PhysicalObject.zoom

bottomright The on-screen positon of the bottom-right corner of the image.

centre The on-screen coordinates of the centre of the object (the point that will maintain its position on the screen as the object is being zoomed). *(Docstring inherited from PhysicalObject)*

Overrides pyzui.physicalobject.PhysicalObject.centre

media_id The object's media_id.

onscreen_area The number of pixels the image occupies on the screen.

onscreen_size The on-screen size of the image.

pos The position of the object.

scale The factor by which each dimension of the image should be scaled when rendering it to the screen.

topleft The on-screen positon of the top-left corner of the image.

Inherited from pyzui.physicalobject.PhysicalObject: aim(v, s, t=None), damping_factor, moving, step(t), zoomlevel

2.19.3 Class RenderMode

Namespace for constants used to indicate the render mode.

Draft `1`

HighQuality `2`

Invisible `0`

2.20 Module pyzui.tiledmediaobject

Tiled media to be displayed in the ZUI.

2.20.1 Class TiledMediaObject

TiledMediaObject objects are used to represent tiled media that can be rendered in the ZUI.

If `autofit` is True, then once the media has loaded it will be fitted to the area occupied by the placeholder.

Constructor: TiledMediaObject(string, Scene[, bool])

Derived from: `pyzui.mediaobject.MediaObject`, `pyzui.physicalobject.PhysicalObject`

TiledMediaObject(media_id, scene, autofit=False) Create a new MediaObject from the media identified by `media_id`, and the parent Scene referenced by `scene`. *(Docstring inherited from MediaObject)*

render(painter, mode) Render the media using the given `painter` and rendering `mode`.

render(QPainter, int) → None

Precondition: `mode` is equal to one of the constants defined in `RenderMode` *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.render

onscreen_size The on-screen size of the image. *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.onscreen_size

default_size `(256, 256)`

tempcache `5`

transparent `False`

Inherited from pyzui.mediaobject.MediaObject: bottomright, centre, fit(bbox), hides(other), media_id, move(dx, dy), onscreen_area, pos, scale, topleft, zoom(amount)

Inherited from pyzui.physicalobject.PhysicalObject: aim(v, s, t=None), damping_factor, moving, step(t), zoomlevel

2.21 Module pyzui.stringmediaobject

Strings to be displayed in the ZUI.

2.21.1 Class StringMediaObject

StringMediaObject objects are used to represent strings that can be rendered in the ZUI.

`media_id` should be of the form 'string:rrggbb:foobar', where 'rrggbb' is a string of three two-digit hexadecimal numbers representing the colour of the text, and 'foobar' is the string to be displayed.

Constructor: StringMediaObject(string, Scene)

Derived from: `pyzui.mediaobject.MediaObject`, `pyzui.physicalobject.PhysicalObject`

StringMediaObject(media_id, scene) Create a new MediaObject from the media identified by `media_id`, and the parent Scene referenced by `scene`. *(Docstring inherited from MediaObject)*

render(painter, mode) Render the media using the given `painter` and rendering `mode`.

render(QPainter, int) → None

Precondition: `mode` is equal to one of the constants defined in `RenderMode` *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.render

onscreen_size The on-screen size of the image. *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.onscreen_size

base_pointsize `24.0`

transparent `True`

Inherited from pyzui.mediaobject.MediaObject: bottomright, centre, fit(bbox), hides(other), media_id, move(dx, dy), onscreen_area, pos, scale, topleft, zoom(amount)

Inherited from pyzui.physicalobject.PhysicalObject: aim(v, s, t=None), damping_factor, moving, step(t), zoomlevel

2.22 Module pyzui.svgmediaobject

SVG objects to be displayed in the ZUI.

2.22.1 Class SVGMediaObject

StringMediaObject objects are used to represent SVG images that can be rendered in the ZUI.

Constructor: SVGMediaObject(string, Scene)

Derived from: `pyzui.mediaobject.MediaObject`, `pyzui.physicalobject.PhysicalObject`

SVGMediaObject(media_id, scene) Create a new MediaObject from the media identified by `media_id`, and the parent Scene referenced by `scene`. *(Docstring inherited from MediaObject)*

render(painter, mode) Render the media using the given `painter` and rendering `mode`.

render(QPainter, int) → None

Precondition: `mode` is equal to one of the constants defined in `RenderMode` *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.render

onscreen_size The on-screen size of the image. *(Docstring inherited from MediaObject)*

Overrides pyzui.mediaobject.MediaObject.onscreen_size

transparent `True`

Inherited from pyzui.mediaobject.MediaObject: bottomright, centre, fit(bbox), hides(other), media_id, move(dx, dy), onscreen_area, pos, scale, topleft, zoom(amount)

Inherited from pyzui.physicalobject.PhysicalObject: aim(v, s, t=None), damping_factor, moving, step(t), zoomlevel

2.23 Module pyzui.scene

A collection of media objects.

2.23.1 Class Scene

Scene objects are used to hold a collection of MediaObjects.

Constructor: Scene()

Derived from: `pyzui.physicalobject.PhysicalObject`

Scene() Create a new scene.

add(mediaobject) Add `mediaobject` to this scene.

This has no effect if `mediaobject` is already in the scene.

add(MediaObject) → None

get(pos) Return the foremost visible `MediaObject` which overlaps the on-screen point `pos`.

Return None if there are no `MediaObject`s overlapping the point.

get(tuple<float,float>) → MediaObject or None

remove(mediaobject) Remove `mediaobject` from scene.

This has no effect if `mediaobject` is not in the scene.

remove(MediaObject) → None

render(painter, draft) Render the scene using the given `painter`.

If `draft` is True, draft mode is enabled. Otherwise High-Quality mode is enabled.

If any errors occur rendering any of the `MediaObject`s, then they will be removed from the scene and a list of tuples representing the errors will be returned. Otherwise the empty list will be returned.

render(QPainter, bool) → list<tuple<MediaObject, MediaObject.LoadError> >

save(filename) Save the scene to the location given by `filename`.

It is recommended (but not compulsory) that the file extension of filename be '.pzs'.

save(string) → None

step(t) Step the scene and all contained `MediaObject`s forward `t` seconds in time.

step(float) → None

Overrides pyzui.physicalobject.PhysicalObject.step

moving Boolean value indicating whether the scene or any contained `MediaObjects` have a non-zero velocity.

Overrides pyzui.physicalobject.PhysicalObject.moving

origin Location of the scene's origin.

viewport_size Dimensions of the viewport.

standard_viewport_size `(256, 256)`

Inherited from pyzui.physicalobject.PhysicalObject: aim(v, s, t=None), centre, damping_factor, move(dx, dy), zoom(amount), zoomlevel

2.23.2 Functions

new() Create and return a new `Scene` object.

new() → Scene

open(filename) Load the scene stored in the file given by `filename`.

open(string) → Scene

Precondition: `filename` refers to a file in the same format as produced by `Scene.save`

2.24 Module pyzui.qzui

QWidget for displaying the ZUI.

2.24.1 Class QZUI

QZUI widgets are used for rendering the ZUI.

Constructor: QZUI([QWidget])

Derived from: `PyQt4.QtGui.QWidget`, `PyQt4.QtCore.QObject`, `PyQt4.QtGui.QPaintDevice`, `sip.wrapper`

QZUI(parent=None, framerate=10) Create a new QZUI QWidget with the given `parent` widget.

keyPressEvent(event) *Overrides PyQt4.QtGui.QWidget.keyPressEvent*

keyReleaseEvent(event) *Overrides PyQt4.QtGui.QWidget.keyReleaseEvent*

mouseMoveEvent(event) *Overrides PyQt4.QtGui.QWidget.mouseMoveEvent*

mousePressEvent(event) *Overrides PyQt4.QtGui.QWidget.mousePressEvent*

mouseReleaseEvent(event) *Overrides PyQt4.QtGui.QWidget.mouseReleaseEvent*

paintEvent(event) *Overrides PyQt4.QtGui.QWidget.paintEvent*

resizeEvent(event) *Overrides PyQt4.QtGui.QWidget.resizeEvent*

timerEvent(event) *Overrides PyQt4.QtGui.QWidget.timerEvent*

wheelEvent(event) *Overrides PyQt4.QtGui.QWidget.wheelEvent*

framerate Rendering framerate.

scene Scene currently being viewed.

Inherited from PyQt4.QtGui.QWidget: DrawChildren, DrawWindowBackground, IgnoreMask, RenderFlag, RenderFlags, acceptDrops, accessibleDescription, accessibleName, actionEvent, actions, activateWindow, addAction, addActions, adjustSize, autoFillBackground, backgroundRole, baseSize, changeEvent, childAt, childEvent, ... *251 more*

Inherited from PyQt4.QtCore.QObject: blockSignals, children, connect, deleteLater, disconnect, dumpObjectInfo, dumpObjectTree, dynamicPropertyNames, emit, eventFilter, findChild, findChildren, inherits, installEventFilter, isWidgetType, killTimer, metaObject, moveToThread, objectName, parent, ... *29 more*

Inherited from PyQt4.QtGui.QPaintDevice: PaintDeviceMetric, PdmDepth, PdmDpiX, PdmDpiY, PdmHeight, PdmHeightMM, PdmNumColors, PdmPhysicalDpiX, PdmPhysicalDpiY, PdmWidth, PdmWidthMM, depth, heightMM, logicalDpiX, logicalDpiY, numColors, paintingActive, physicalDpiX, physicalDpiY, widthMM

2.25 Module pyzui.mainwindow

PyZUI QMainWindow.

2.25.1 Class MainWindow

MainWindow windows are used for displaying the PyZUI interface.

Constructor: MainWindow()

Derived from: `PyQt4.QtGui.QMainWindow`, `PyQt4.QtGui.QWidget`, `PyQt4.QtCore.QObject`, `PyQt4.QtGui.QPaintDevice`, `sip.wrapper`

MainWindow(framerate=10) Create a new MainWindow.

minimumSizeHint() *Overrides PyQt4.QtGui.QWidget.minimumSizeHint*

showEvent(event) *Overrides PyQt4.QtGui.QMainWindow.showEvent*

sizeHint() *Overrides PyQt4.QtGui.QWidget.sizeHint*

Inherited from PyQt4.QtGui.QMainWindow: AllowNestedDocks, AllowTabbedDocks, AnimatedDocks, DockOption, DockOptions, ForceTabbedDocks, VerticalTabs, actionEvent, addDockWidget, addToolBar, addToolBarBreak, centralWidget, changeEvent, childEvent, closeEvent, connectNotify, contextMenuEvent, corner, create, createPopupMenu, ... *93 more*

Inherited from PyQt4.QtGui.QWidget: DrawChildren, DrawWindowBackground, IgnoreMask, RenderFlag, RenderFlags, acceptDrops, accessibleDescription, accessibleName, actions, activateWindow, addAction, addActions, adjustSize, autoFillBackground, backgroundRole, baseSize, childAt, childrenRect, childrenRegion, clearFocus, ... *211 more*

Inherited from PyQt4.QtCore.QObject: blockSignals, children, connect, deleteLater, disconnect, dumpObjectInfo, dumpObjectTree, dynamicPropertyNames, emit, eventFilter, findChild, findChildren, inherits, installEventFilter, isWidgetType, killTimer, metaObject, moveToThread, objectName, parent, ... *29 more*

Inherited from PyQt4.QtGui.QPaintDevice: PaintDeviceMetric, PdmDepth, PdmDpiX, PdmDpiY, PdmHeight, PdmHeightMM, PdmNumColors, PdmPhysicalDpiX, PdmPhysicalDpiY, PdmWidth, PdmWidthMM, depth, heightMM, logicalDpiX, logicalDpiY, numColors, paintingActive, physicalDpiX, physicalDpiY, widthMM

Chapter 3

Dependencies

PyZUI depends on the following packages:

- PyQt4
- Python Imaging Library (PIL)
- ImageMagick
- pdftoppm
- jrMandel

For further information see the `INSTALL.txt` file, which is replicated below:

```
PyZUI INSTALLATION INSTRUCTIONS
===============================
PyZUI was developed under Ubuntu 8.10 Intrepid Ibex. It has also been tested on
Ubuntu 9.04 Jaunty Jackalope.

DEPENDENCIES
============
PyZUI depends on the following Python packages:
- PyQt4
- Python Imaging Library (PIL)

These can be installed in Debian-based distributions by:
  apt-get install python-qt4 python-imaging

The following non-Python packages are also required by certain features of the
application:
- ImageMagick (highly recommended, but strictly optional if you do not intend
  using images of any format other than PPM)
- pdftoppm from Poppler or Xpdf (optional if you do not intend viewing PDFs);
  Note that PyZUI has been developed using the pdftoppm binary provided by
  Poppler
- jrMandel (optional if you do not intend viewing the Mandelbrot set dynamic
  media)

ImageMagick can be installed in Debian-based distributions by:
  apt-get install imagemagick

pdftoppm can be installed in Debian-based distributions by either:
  apt-get install poppler-utils
or:
  apt-get install xpdf-reader

jrMandel can be downloaded from <http://freshmeat.net/projects/jrmandel/>,
and installed with the standard:
  ./configure && make && make install

RUNNING PyZUI
=============
PyZUI can be run by executing the script 'main.py'. It is not necessary to run
this from the command-line (unless you want to view the logging), and it can be
run from any directory (the script will set the set the working directory
appropriately by itself).
```

Chapter 4

Efficiency

To judge the efficiency of the application, a small script (`test/benchmark.py`) was written and run on eight images[1] of various sizes on a machine with a 1.73GHz CPU and 512MB RAM. The results of this are summarised in the following table (for full results see `test/benchmark.log`):

Image	**1**	**2**	**3**	**4**
Area (megapixels)	3.54	3.98	4.98	15
Width (px)	2304	2304	1932	5400
Height (px)	1536	1728	2580	2700
Conversion time	0.66s	0.70s	0.88s	3.02s
Tiling time	1.81s	1.91s	2.56s	7.95s
Tiling memory use (MB)	4.81	4.49	3.80	8.04
Cold zooming mean FPS	20	21	21	21
Warm zooming mean FPS	26	27	27	28
Image	**5**	**6**	**7**	**8**
Area (megapixels)	35	75	160	233
Width (px)	11756	22194	20634	21600
Height (px)	3009	3367	7751	10800
Conversion time	1m52s	13m23s	2m33s	3m9s
Tiling time	27s	1m8s	1m31s	2m11s
Tiling memory use (MB)	5.79	19	31	33
Cold zooming mean FPS	13	10	19	18
Warm zooming mean FPS	30	41	29	27

[1]The first three images are from an average digital camera. The fourth and eighth are satellite imagery from NASA: `http://earthobservatory.nasa.gov/Features/BlueMarble/images_bmng/8km/world.topo.bathy.200407.3x5400x2700.jpg`; `http://earthobservatory.nasa.gov/Features/BlueMarble/images_bmng/2km/world.topo.bathy.200407.3x21600x10800.jpg`
The remaining images are from *Life In Megapixels*: `http://lifeinmegapixels.com/location.php?location=gorne`; `http://lifeinmegapixels.com/location.php?location=mscnt`; `http://lifeinmegapixels.com/location.php?location=kingsn`

4.1 Conversion

It can be seen that the conversion time for the first three average sized images is negligible, and not overly significant for the 15MP image. For the other four images, the conversion time is much more significant, especially in the case of image 6 which took 13m23s.

This anomaly is likely because of the way in which ImageMagick caches pixel data[2] during the conversion. For smaller images, the pixel data is cached directly in RAM. However, once the image size passes a certain threshold, it is cached to disk instead. On the machine this was tested on, the "automagically" determined default threshold is 740MB.

For the Q16 version of ImageMagick, each pixel in the image consumes 8 bytes in the pixel cache. Therefore, the first 5 images consumed 29MB, 32MB, 40MB, 120MB and 280MB respectively, meaning that they could be stored entirely in RAM. The last 2 images consumed 1.3GB and 1.9GB respectively, meaning that they would be cached to disk instead. However, image 6 consumed 600MB — below the threshold, but larger than the amount of physical memory available. Therefore, ImageMagick would have attempted to cache this image to the machine's virtual memory, but as it was larger than RAM a portion of it would have been required to be swapped to disk, causing the system to thrash, hence accounting for the substantial time taken.

Also, although the 280MB consumed by image 5 could theoretically be stored entirely within in the 512MB of RAM, in practice a substantial amount of the available RAM would be occupied by other applications. Therefore the conversion of this image may have also caused minor thrashing. This is evident in the much longer conversion time when compared to the first 4 images.

If this situation were likely to be an issue, the user could alter the configuration of ImageMagick to lower the threshold to a reasonable value. PyZUI could also be altered to attempt to alter the threshold value itself, but this would not necessarily result in optimal values being set.

It is likely that thrashing caused by the conversion of images 5 and 6 may have impacted on the rest of the benchmarking of those images. Therefore they will be counted as a anomalies and will not included in the calculations in the following sections.

4.2 Tiling

4.2.1 Time

Inspecting the values given in the above table shows that tiling time is almost directly proportional to the area of the image — performing a linear regression results in a coefficient of determination (R^2) of $\sim$ 0.99994. From this it can be tentatively assumed that the tiling process has a time complexity of $\Theta(n)$, where n is the area of the image.

[2] `http://imagemagick.org/script/architecture.php`

Running `benchmark.py` on image 8 with profiling enabled (using the `cProfile` module), shows that 76% of the tiling time was spent loading pixels from the file (`Tiler.__load_row_from_file`), and another 21% of the tiling time was spent saving tiles to disk (`Tiler.__savetile`) — a total of 97% of tiling time being spent on these IO operations. As both of these operations are proportional to the size of the image, this agrees with the time complexity assumed in the previous paragraph.

4.2.2 Space

Similarly, inspecting the values in the table shows that that tiling memory usage is almost directly proportional to the *width* of the image ($R^2 \approx 0.9988$). From this it can be tentatively assumed that the tiling process has a space complexity of $\Theta(w)$, where w is the width of the image.

This agrees with the fact that only a single row is processed at a time, and since every row has a height of 256px, the memory consumed by each row is proportional to the width of the image.

4.3 Rendering

The difference between the efficiency of cold and warm rendering should be noted. In this context, cold rendering means the tile cache began empty, and warm rendering means most of the required tiles were already in the tile cache when rendering began. This difference in rendering framerate shows the increase in efficiency provided by the tile caching mechanism.

Discounting the anomalies discussed previously, it can be seen that all images have approximately the same rendering framerate independent of image size. The larger images have a slightly reduced cold rendering framerate, but this is only because they required tiles to be loaded from disk for higher resolutions, whereas the other images simply resized existing tiles in-memory. This shows how tiling images allows arbitrarily large images to be rendered efficiently.

4.4 Possible Improvements

The `Tiler.__load_row_from_file` method is optimised for memory usage, by only loading a single line of pixels from the file at a time. However, as discussed previously, this method accounts for around 75% of the total tiling time. Therefore, it may be beneficial for a compromise to be made between memory usage and time efficiency, possibly by loading larger chunks of pixel data at a time.

Rendering efficiency could be improved by adding support for rendering with OpenGL. However, this would not replace the current rendering system since OpenGL is not currently supported by all video cards.

SVGMediaObject is very slow at rendering complex SVG files. This could be improved by implementing an SVGTileProvider which renders tiles on demand

and provides them to a TiledMediaObject. This would improve the responsiveness of the application as SVG rendering would be performed in the background, and tile caching would mean that the SVG file need not be rendered for every frame.