# visual-hash Documentation

***Release 0.0.0***

**David Roundy**

December 14, 2016

Contents

A visual hash is an image that is generated from a large string, just as an ordinary cryptographic hash is usually represented as a hexadecimal string. The advantage of a visual hash is that it is easier for humans to remember and compare.

CHAPTER 1

# What makes a hash good enough?

A good visual hash should have the following properties:

1. A high information content, as measured by its Shannon entropy. This results in a hash that has a low chance of collision. This provides pre-image resistance, i.e. it makes it difficult to create a second input that results in a hash identical to a given hash.

2. A high minimum self-information. i.e. the lowest self-information value should be as high as possible. This property implies the former property, but is itself distinct. The lowest self-information output is most prone to collisions. Testing for this property is challenging, more challenging than the mean information content discussed previously.

   A second-best for this property would be to be able to identify hashes that have low self-information. Of course, if we can reliably identify them, we could (in principle) eliminate them entirely by checking for a low self-information during the hashing process, and trying again when this is encountered.

3. Second pre-image resistance, which means that knowing the hash input, we should not be able to produce a similar (or identical) hash. We achieve this by using a cryptographic hash as input to our visual hashes, which ensures the second pre-image resistance is identical to the first pre-image resistance.

   In order to generate testable visual hashes, we actually aim for raw algorithms that *lack* second pre-image resistance (which we then add on via the cryptographic hash). This allows us to more readily explore "similar" images in order to estimate the information content in the visual hash, and thus its first pre-image resistance.

CHAPTER 2

# The VisualHash module

Create a visual hash of a string.

VisualHash is a package that includes several functions to create a visual hash of an arbitrary string. Each function implements a distinct algorithm that given a random number generator produces a visual image. The cryptographic strength of the hash relies on using a cryptographically strong random number generator that is seeded by the data to be hashed.

We provide a strong random number generator (called StrongRandom), which is based on taking the SHA512 hash of the data, followed by the SHA512 hash of the hash, and so on. This puts an upper bound of 512 bits of entropy on any of our hashes (which should not be a problem).

We also provide a "tweaked" random number generator TweakedRandom, which gives a slight variation on a specific strong random number sequence. This will enable us to test the effect of small changes in the generated hashes.

The visual hash styles supported are:

- Fractal
- Flag
- T-Flag
- RandomArt
- Identicon

VisualHash.**Flag**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create a hash using the "flag" algorithm.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a hash generated by the random generator.

VisualHash.**Fractal**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create a hash as a fractal flame.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a hash generated by the random generator.

VisualHash.**Identicon**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create an identicon hash.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a hash generated by the random generator. This hash has only 32 bits in it, so it is not a strong hash.

VisualHash.**OptimizedFractal**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create a hash as a fractal flame.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a hash generated by the random generator.

VisualHash.**RandomArt**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create a hash using the randomart algorithm.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a hash generated by the random generator.

VisualHash.**TFlag**(*random=<VisualHash.random.StrongRandom object>*, *size=128*)
Create a hash using the "flag" algorithm.

Given a random generator (and optionally a size in pixels) return a PIL Image that is a a hash generated by the random generator.

Contents:

## 2.1 Estimating Entropy

We are interested in the amount of entropy (or information) that is transferred to humans in a visual hash. It is worth taking a moment to define terms in a human-centric manner.

### 2.1.1 Self-information

The **self-information** of a given image is given by

$$I(x) = -\log_2 P(x)$$

where $P(x)$ is the probability of observing state $x$.

### 2.1.2 Information

The **information** (or entropy) of a system is actually the weighted average of the self-information all possible states.

$$H = -\sum_x P(x) \log_2 P(x)$$

### 2.1.3 Estimating the entropy take II

Let us consider a system consisting of $N$ parts, each of which has $n$ distinct configurations. If I randomly modify one part, then the chance of changing the system is $q = 1 - \frac{1}{n}$. The entropy of such a system is given by

$$H = N \log_2 n$$

If I change a fraction $f$ of the parts of the system, the probability of changing the system is given by

$$P = (1 + f - fq)^N (1 - A)$$

where $q$ is the probability of a given thing *not* changing the system when it is modified, and math: 'A' is the variable accounting for user error. Let us now consider a fraction $f_\gamma$ that has a probability $\gamma$ of *not* changing the system.

$$\gamma = (f_\gamma q)^N$$

We will assume that we can measure $f_\gamma$, and also that we can (and do) measure $f_{\gamma^2}$, the fraction that leads to the square of probability $\gamma$.

$$\gamma^2 = (f_{\gamma^2} q)^N$$

We can solve for the following:

$$q = \frac{f_{\gamma^2}}{f_\gamma^2} = 1/n$$

$$N = \frac{\ln \gamma}{\ln (f_\gamma q)}$$

$$N = \frac{\ln \gamma}{\ln \left( \frac{f_{\gamma^2}}{f_\gamma} \right)}$$

Taking these together, we can show that the entropy is given by

$$H = \frac{\ln \gamma}{\ln \left( \frac{f_{\gamma^2}}{f_\gamma} \right)} \log_2 \left( \frac{f_\gamma^2}{f_{\gamma^2}} \right)$$

A nice option seems to be

$$\gamma = \frac{\sqrt{5} - 1}{2}$$

$$\gamma^2 = \frac{3 - \sqrt{5}}{2}$$

which balances the two at an average of 50%, thus avoiding a bias in one direction or the other. This is the golden ratio minus one.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

## V

## F



## I



## O



## R



## T



## V