



UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

Moji

The advent of large identifiers and how
to conquer them as human.

by

Michael Luggen

Thesis for the Master of Science in Computer Science
Supervised by Prof. Dr. Philippe Cudré-Mauroux

eXascale Infolab
Department of Informatics - Faculty of Science - University of Fribourg

September 14, 2012

UNIVERSITY OF FRIBOURG

Faculty of Science

Department of Informatics

eXascale Infolab

Thesis for the Master of Science in Computer Science

Supervised by Prof. Dr. Philippe Cudré-Mauroux

by Michael Luggen

Abstract

This master thesis sets the spotlight on inhumanly large identifiers, shows the problems occurring in their use, enlightens available techniques not used enough and wants to reignite interest in this field by proposing an own solution to visualize large identifiers.

A new implementation 'Moji-Identicon' was developed which takes a large identifier and generates a more human friendly representation in visual form. Neither to memorize nor to transfer, but an easier comparing or matching of such identifiers is targeted by this visualizations.

After a thorough search for existing automated visualizations and the studies of them, a focus on large identifiers that don't hold any semantic information was set. Build on a catalog of requirements we developed multiple visual representations and also the tools to evaluate our approaches in experiments with the end users.

In comparison to existing solutions our simpler and smaller representations of the identifiers did show at least the same performance. We see a lot of potential of improvements and possible features in this still young field.

Finally we hope this kind of tools get more attention in the future and are incorporated at more places of the daily life to help people interact with all kind of systems in this information dominated times.

Acknowledgements

I thank first of all my supervisor Phillipe Cudré-Mauroux which received me with an open mind. Throughout the whole work he provided me with ideas, pointers and guidance. Also was he open to my sudden mind changes but did assure that I focus to get the job done.

For all the fun times at the university and at other opportunities this semester I want to thank the team of the eXascale Infolab and their partners, my office companions Carine and Ilya and all people contributing to the fantastic atmosphere at the university.

Thanks to Don Park and Andreas Gohr which allowed me to get some first hand information about the beginnings of identicons.

Also at this place I thank Urs and Roger and all collaborators at my work place in Biel. Without their support and understanding I never would made it this far.

Thanks to my family and to my love, Maude.

Contents

Abstract	2
Acknowledgements	3
List of Figures	7
List of Tables	8
1 Introduction	9
1.1 Thesis Content	9
1.2 Motivation	9
1.2.1 Larger identifiers	10
1.2.2 New Data Processing Technologies	10
1.2.3 Communication Protocols	10
1.3 Use Cases	11
1.3.1 Memorizing	11
1.3.2 Transferring	11
1.3.3 Comparing	11
2 Visualization of Identifiers	13
2.1 History of Hash Visualization	13
2.2 VisualID	14
2.3 Overview: Identicons	15
2.3.1 Definition	15
2.3.2 Available Implementations	15
2.3.3 Identicon & MonsterID	16
2.3.4 VisualHostKey	18
2.3.5 Other descendants	18
2.4 Focus & Innovation	18
2.4.1 Specialization on the Use Case Comparing	19
2.4.2 Acceptance	19
2.4.3 Dimensions	19
2.5 Requirements of Moji-Identicons	20
2.5.1 Basics	20
2.5.2 Aesthetics	21

2.5.3	Recognizable	21
2.5.4	Colorless media / Colorblind people	21
2.5.5	Bijectivity	22
2.6	Disregarded functionality	22
3	Approach	23
3.1	Approach	23
3.2	Prototyping Environment	24
3.2.1	First tests	24
3.2.2	Transition to JavaScript	25
3.3	Final Environment	26
3.3.1	Technologies	26
3.3.2	2D Geometry Library	26
3.4	Structure	27
3.4.1	Stage: Point Matrix	28
3.4.2	Stage: Visualization	29
3.4.3	Ensuring Bijectivity	30
3.5	Followed Design Guidelines	31
3.5.1	Color	31
3.5.2	Connected Shapes	31
4	Evaluation	32
4.1	Catalog of Identicons	33
4.1.1	Own Development	33
4.1.2	Available Implementations	34
4.2	Experimental Plan	34
4.3	Evaluation Application	34
4.3.1	JavaScript Implementation	35
4.3.2	Frameworks, Components and Services	35
4.3.3	Workflow	36
4.4	Experiments & Results	36
4.4.1	Pre-Test	36
4.4.2	Base Level	36
4.4.3	Text vs. Truncated Text	37
4.4.4	Color Variations	37
4.4.5	Shape Variations	38
4.4.6	Final Validation	38
5	Conclusion	40
5.1	Conclusion	40
5.2	Improvements	40
5.2.1	Additional Functionality	41
5.2.2	Enhancing Barcodes or QR-Codes	41
5.3	Outlook	41
5.3.1	Next Steps	42
5.3.2	Towards a Standard	42

<i>Contents</i>	6
-----------------	---

A Glossary	43
-------------------	-----------

B Proposal	44
-------------------	-----------

Bibliography	48
---------------------	-----------

List of Figures

2.1	Two examples of <i>Random Art</i> which visualize a hash.	13
2.2	A set of <i>VisualIDs</i> which visualize different files.	14
2.3	Nine Block – Digital Artwork by Jared Tarbell [2003].	17
2.4	A <i>VisualHostKey</i> - ASCII visualization of a SSH fingerprint.	18
3.1	The minimal complete visual representation of a SHA1 hash.	23
3.2	Very first prototype tests with <i>Processing</i> of the Bottom-Up approach. .	24
3.3	The first results of the prototyping phase in <i>Processing</i> . Top: First crude examples of the visualization. Bottom: A helper visualization which shows weights.	25
3.4	The end results of the prototyping phase. The top rows show two different variants of connected and centered visuals.	25
3.5	Some examples of final Moji-Identicons. (Codename <i>FigureBlob2</i>)	26
3.6	Uniform distribution of the points generated by our simple algorithm. Top: Saturation value of 1, Middle: Saturation value of 0.9, Bottom: Saturation value of 0.5.	28
3.7	How far to jump according a single hexadecimal input value.	28
3.8	Code which is generating points on the matrix, u and v represent the x and y axis of the final image.	29
3.9	Ordering the points based on the distance to the center.	29
3.10	Ordering the points around the center in a clockwise fashion.	29
3.11	The draw function of the final version, created with the rather poetic name <code>drawIdentFigureBlob2</code> . (For the visualization see Figure 3.5.)	30

List of Tables

1.1	A list of established or emerging harder, better, faster, stronger identifiers.	10
2.1	A survey of existing and used visualizations of (mostly user) identifiers, ordered by time of appearance.	16
3.1	A short overview of the <code>geom.js</code> library.	27
4.1	Overview of the results in color variations.	37
4.2	Overview of the all non significant differences of the shape variations. . .	38
4.3	Overview of the final validation.	38

Chapter 1

Introduction

“Rose is a rose is a rose is a rose.”
Gertrude Stein, 1922, Sacred Emily, p. 187

1.1 Thesis Content

After the motivation in this same chapter we provide an overview of existing approaches to visualize identifiers ([Chapter 2](#)). Also we specify in this chapter a list of requirements for a novel kind of identifier visualization. Provided with this requirements we describe the approach ([Chapter 3](#)) we followed to develop a new kind of visualization which we call Moji-Identicons. To provide scientific evidence that our approach is performing better than identifiers presented in latin alphabet and arabic numerals we present our experimental setup and the results of the evaluation ([Chapter 4](#)). With pointers how to enhance and complete our approach and a conclusion we sum up this work ([Chapter 5](#)).

1.2 Motivation

A lot of work today is motivated by the ever increasing growth of information in general. A side effect of this is the ever increasing size of identifiers which is needed to point to this huge amount of data. In some areas just the addressable *data blobs* grow, nevertheless we collect and manage more unique identifiable data objects. Every year new systems are going online which create a ton of identifiers for all sort of use cases. Transaction ids for receipts of shops, ids for concert tickets, registration numbers in government processes, backup tape identifiers and so on.

1.2.1 Larger identifiers

All this leads to larger identifiers which cover a bigger numerical space. To just give some chosen examples have a look at the table 1.1.

Type	Example	Use case
Record L.	RMT33W	Airline Booking Records
UPU S10	EE 876 543 216 CA	International Parcel Tracking
TID	8UN34654K0142381H	Paypal Money Transactions
UUID	550e8400-e29b-41d4-a716-446655440000	<i>No-SQL</i> databases
IPv6	2001:0db8:85a3:0042:0000:8a2e:0370:7334	Host Addresses
SHA1	2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12	Version Control Systems

TABLE 1.1: A list of established or emerging harder, better, faster, stronger identifiers.

1.2.2 New Data Processing Technologies

At the beginnig of database technologies identifiers for data tuples were mostly integers beginning at one. This made it easy to work with the identifiers as they (at least in the development stage) seldomly pushed over four to five digit numbers. Such numbers can be kept easily in short term memory by most people which helps to compare them in different contexts. This is a task often recurring while developing new applications. The new kind of often called *No-SQL* databases take advantage of large identifiers in there core functionality. Through using enough big identifiers it is not anymore necessary to have a centralized control instance to assure that they are unique. It is just assumed that the probability a randomly generated ID collides with another is to small to give attention to this case at all. Also at least since the new decentralized version control system (DVCS like e.g. *GIT*) appeared, the idea to identify information blobs by the hash token from the information itself becomes popular. As showed in Table 1.1 you see SHA1 identifiers used by *GIT* are not anymore easily comparable and can only by a small gifted part of the population held in short time memory – if at all.

1.2.3 Communication Protocols

With the switch from the *IP protocol* in version 4 (IPv4: 192.168.0.1) to the new version 6 (IPv6: 2001:0db8:85a3:0042:0000:8a2e:0370:7334) you often loose the ability to easily compare two IP addresses. This is an extreme example but also the worldwide telephone system increased the size of phone numbers to allow more members and to facilitate the independence of the numbers from their geographic location. In Switzerland for example

a local phone number was 20 years ago only 6 digits (23 76 33), today you need to type at least 9 digits (032 823 76 33) to build up a connection inside Switzerland. And mostly the international number is communicated (0041 32 823 76 33).

1.3 Use Cases

1.3.1 Memorizing

One of the basic use cases for identifiers is to memorize them. To know an identifier by heart enables people to directly access information or connect to someone. Like with phone numbers you are able to know a number to call some one without any tools. But as you probably remarked already, today nobody knows anymore a lot of phone numbers by heart. The size and amount of numbers you need to know has become a major challenge to memorize. Therefore are electronic phone books in the mobile phones an important tool for everybody. By this example we want to show that memorizing of identifiers has become some what of obsolete. We have tools which help us with this task.

1.3.2 Transferring

The next task involving identifiers is to transfer them to other persons. It can be by copying them on paper, spelling them in a phone conversation and so on. Identifiers which are used in such scenarios are often optimized like the *Record Locator* which are used in airline reservation systems. Because some letters can easily be confused like L, I and 1 or 0 and O they are exclude of the set of possible tokens which build such an identifier. But to keep these identifiers small they are often not unique for a lifetime but rather are reused after some time. Normally this is no problem as long you deliver some context to the identifier as a date or last name which in combination makes the identifier again unique. (Details about such kind of identifiers would fill another couple of pages and can not be looked into here. Though it is an interesting topic in the field of user experience.)

1.3.3 Comparing

The use case which is becoming more important today by using identifiers is in our view to compare identifiers to check if they are matching. An example is when you have a paper copy in front of you and need to check if it has the same identifier as on your screen in any kind of system. Also sometimes you have to compare identifiers which are

in different applications on your computer like in an accounting system, email application or website. In this use cases you often get presented one of the above mentioned really big identifiers. And exactly in this context lies the main goal of our work. How to visualize these identifiers to facilitate the tasks of the user in comparing multiple identifiers. This has not to be only by checking identifiers, in the same category falls finding a presented identifier in a set of other identifiers.

Chapter 2

Visualization of Identifiers

“Das sinnliche Wahrnehmen z.B. geschieht gewiss mit grösserer oder geringerer Genauigkeit je nach dem Grade des Interesses, es wird fortwährend in andere Bahnen gelenkt durch den Wechsel der äusseren Eindrücke und durch innere Einfälle.”

Hermann Ebbinghaus, 1885, p. 16

Visualization of identifiers is not a new concept. But in our opinion a not enough investigated one. Especially as the importance of large identifiers is arguably is further increasing in the near future ([Section 1.2](#)).

2.1 History of Hash Visualization

The first mention we could research didn't try to visualize identifiers but hash values. It was an attempt to improve the security in public-key applications. As no user ever seriously looked at the cryptic root-keys which were presented to verify the host, [Adrian Perrig and Dawn Song \[1999\]](#) did develop an algorithm to produce a more human friendly visualization of the key hashes. This produced rather artsy so-called *Random Art* ([Figure 2.1](#)) images.

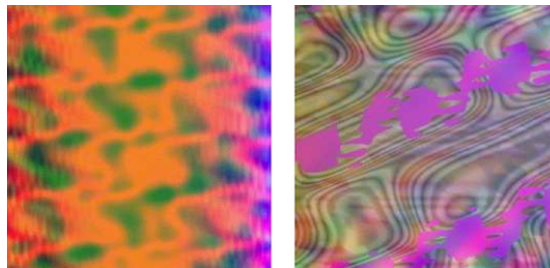


FIGURE 2.1: Two examples of *Random Art* which visualize a hash.

A year later the same algorithm was used to create hash visualizations which were used to test a new visual authentication system [Rachna Dhamija, 2000]. The user was asked to memorize some *Random Art* with which he was later challenged to authenticate. Apart from this we didn't find *Random Art* used at grant scale in a system today. But it was the inspiration of different projects the years to come. A prominent one is the *VisualHostKey* which we refer to later (Subsection 2.3.4). The hash visualization technologies were not developed to visualize identifiers. As you can produce a hash out of any identifier easily, the algorithm can be used immediately as identifier visualization.

2.2 VisualID

The next attempt to visualize random information was made by developing *VisualID*. In the paper of Lewis et al. [2004] they proposed a better representation for icons used in the desktop metaphor. Some file formats don't have any intuitive visual representation. If this was the case generated icons based on a hash of the filename were used. The quite complex algorithm did also incorporate a clustering stage with the goal to create similar visualizations for similar named files.

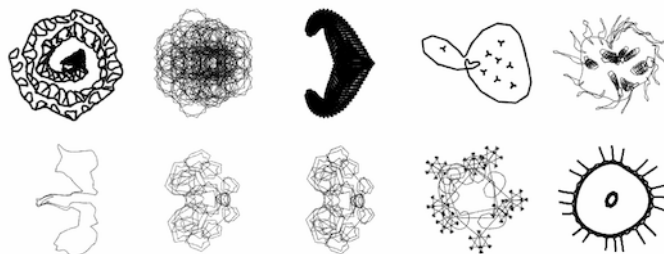


FIGURE 2.2: A set of *VisualIDs* which visualize different files.

2.3 Overview: Identicons

The term *identicon* was coined by Don Park in early 2007. At the time writing this document (September 14, 2012) we were not able to find the term in any official dictionary. The only listing available was found in [Wikipedia \[2007\]](#):

An Identicon is a visual representation of a hash value, usually of the IP address, serving to identify a user of a computer system; compare avatars. The original Identicon is a 9-block graphic, which has been extended to other graphic forms by third parties, some of whom have used MD5 instead of the IP address as the identifier. In summary, an Identicon is a privacy protecting derivative of each user's IP address built into a 9-block image and displayed next to the user's name. A visual representation is thought to be easier to compare than one which uses only numbers, and more importantly, it maintains the person's privacy.

This is not the whole truth about the idea of *identicons* as we will look into in the next section.

2.3.1 Definition

Our current understanding of an *identicon* we phrase the following:

An identicon is a **visual representation** of a piece of information (including identifiers). The representation is unambiguously **generated** by a set of **rules**. The goal of the representation should help the user to **simplify the interaction** with the information itself.

This also reflects more the original idea of Don Park which he mentioned in a comment of his first blog post [[Don Park, 2007](#)]:

I originally came up with this idea to be used as an easy means of visually distinguishing multiple units of information, anything that can be reduced to bits.

2.3.2 Available Implementations

The [Table 2.1](#) contains a compilation of actively used implementations which we consider as identicons. Examples of the different implementations can be found in [Section 4.1](#).

Name of Visualization	Appearance	Innovation, Focus
Identicon [Don Park, 2007]	18.01.2007	First implementation of an identicon, based on hashed IP.
MonsterID [Andreas Gohr, 2007]	20.01.2007	Character identicon based on hashed username.
Wavatar [Shamus Young, 2007]	15.12.2007	MonsterID like avatar visualization.
VisualHostKey [OpenSSH, 2008]	22.07.2008	ASCII visualization of SSH Fingerprints.
Retro	May, 2009	Another version of MonsterID style.
Vash	early 2010	High-Fidelity visualization of hash, with a strong focus of memorizing.
Robohash	July, 2011	Visual more appealing version of MonsterID.

TABLE 2.1: A survey of existing and used visualizations of (mostly user) identifiers, ordered by time of appearance.

2.3.3 Identicon & MonsterID

The Identicon and the MonsterID both appeared first early 2007. It was the identicon developed by Don Park [2007] which ignited the concept. Shortly after Andreas Gohr [2007] took another approach. I had the great pleasure to speak in the context of this work with both developers by doing a short phone interview.

2.3.3.1 Identicon

It was the work of Jared Tarbell [2002, 2003] (Figure 2.3) which inspired Don Park to the first incarnation of the identicons. On the website where this digital art-work is presented he describes its own origin as the following:

The nine block is a common design pattern among quilters. It's construction methods and primitive building shapes are simple, yet produce millions of interesting variations.

This nine block figures can be generated by choosing in a set of some basic forms and later rotating by 90° steps. This algorithm is able to produce 10'816 unique different shapes.

The first incarnation of identicons was done by some one which had a lot of experience in network security topics. Therefor the blog title which described the first identicons was: "Visual Security: 9-block IP Identification". A rather technical approach can be inferred here.

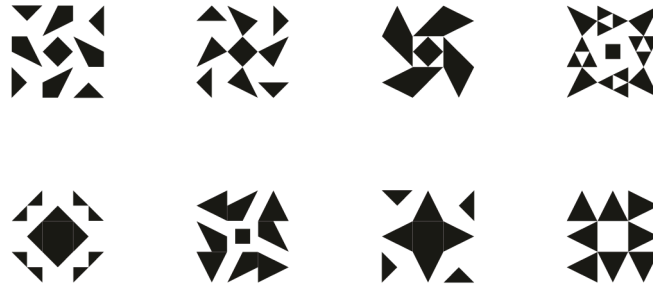


FIGURE 2.3: Nine Block – Digital Artwork by [Jared Tarbell](#) [2003].

As a first application Don Park did implement a visualization for his personal blog. Instead of the non-existing user picture in the comment section, based on the users actual IP address, a visual is created. This visual incorporated the algorithm of the *Nine Blocks* enhanced with a figure color to be able to generate more different possible identicons. People liked it. The user pictures accompany the comments are mostly quite big (> 200 pixels a side), even as the *Nine Block* still work down to a size as small as 16 by 16 pixel.

We like to discuss here some problems we identified, which we estimate is holding the use in more general use cases of identicons back. First of all are the used Nine Blocks quite exciting figures and with the added color really take a lot of the attention of the user. This tends to distract from the task at hand. Further if we consider the quite small space of IPv4 addresses collisions are rather unlikely using the *Nine Blocks*. But on a huge numerical space of a SHA1 hash used as identifier this technique delivers just not enough possibilities and can potentially occur in collisions. Therefore the same visualization would be used for different identifiers.

2.3.3.2 MonsterID

Only some days later, inspired from Don Parks identicons, Andreas Gohr took the concept and did improve it for the use case of generating user pictures for blog software. Thus he gave the visualizations a meaning, every one became a monster. The first improvement was that he based the generation of the visuals not anymore on IP addresses as these change over time, but on the username.

Perfect as substitution of user pictures but not useful for more general use cases. Mostly for the same reasons we see the identicon as problematic, additionally strong semantics, which narrows field of application even further.

2.3.4 VisualHostKey

VisualHostKey is a feature of the recent [OpenSSH \[2008\]](#) packages. We consider this application, which visualizes the fingerprint of the to connecting host in a ASCII art graphic ([Figure 2.4](#)), also as a kind of identicon. Disregarding the fact that it not visualizes identifiers.

This kind of visualization has as origin again the *Random Art* described in the paper of [Adrian Perrig and Dawn Song \[1999\]](#).

```

+--[ RSA 2048]-----+
|          . . . o          |
|      . + o+o .          |
|      . = +.+.+.          |
|      + . oo=             |
|      . . . o E          |
|      . . . .            |
|      . . .              |
|      . . .              |
|      . . .              |
+-----+

```

FIGURE 2.4: A *VisualHostKey* - ASCII visualization of a SSH fingerprint.

2.3.5 Other descendants

The other identicon types we found, listed in [Table 2.1](#), are more or less direct descendants of the other techniques. There are the three more or less direct derivatives of the MonsterID. Wavatar incorporates different graphics, and Robohash takes the graphics to a professional level. Retro on the other hand goes his own way and creates 8-bit computer game figures. And the Vash (visual hash) implementation focus mainly on memorable quite big visual representation of information. The use cases listed by Vash are: Public Key Fingerprints; Reliable password entry; Verification of Sensitive Information and Default Forum or Blog Avatars.

2.4 Focus & Innovation

So there exists already a respectable choice of identicon implementations. Why bother to create even another type of identicons? With this question in mind and the comments provided above we like to develop in this section the focus of our implementation.

2.4.1 Specialization on the Use Case Comparing

Our implementation should clearly favor the use case of comparing two identifiers or to find a given identifier in a set of such. As described in the [Chapter 1](#) visualizations can also be provided to memorize identifiers. Also they can be specialized to be easily explained to someone else in a phone conversation to reproduce it later. These technique are mostly possible for identifiers which have a certain size. But we did not find any implementation of a large identifier as a SHA1 which was potential easier transferable than in alphabetic representation. Some of the concrete use cases we had in mind specifying our implementation was to identify identical data tuples in *No-SQL* databases; the simple task of verifying, if the last version on the own workstation corresponds the version on the production server in a DCVS based system as *GIT*. Or also an easy way to verify if a large bank account number was copied correctly from paper to the online banking system.

2.4.2 Acceptance

The best invention is useless if nobody uses it. Therefore we want to focus also on acceptance and compatibility with existing systems and designs. No developer, designer or user experience specialist will introduce a nervous looking, color full and distracting element in their systems even if it is there for helping the users. We estimate that in our use cases it is also important to keep the physical size on the different media fairly small. In most cases the identicon is just an additional element to the already present information. Further we think it is also important for a broad acceptance and a compatibility with a lot of use cases the identicons should not have any semantics. (It would be rather distracting if your next flight number is represented by a green devil.)

2.4.3 Dimensions

We identified three competing dimension. The art of an implementation is to balance these and find the right mix for an optimal performance.

2.4.3.1 Physical Size

Identicons are mostly an additional element which need to find a place on the same space already existing elements stay. It is important to keep the physical size of the identicons small. In opposite to the implementations which are used to substitute avatars and use

some space on the screen, our solution should be as small as possible. This means that the identicon can stay in the background on a website or on a letter.

2.4.3.2 Representable Possibilities

The physical size works diametral to the amount of different possible identifiers which can be visualized. The extreme will be a one pixel sized image (basically a color) where we can visualize as much different identifiers a person can distinguish. The theoretical space of possibilities is increasing exponential as you increase the size of the visualization. A 20 pixel size image with 8-bit color depth is already enough to fully visualize the quite big SHA1 hash.

2.4.3.3 Graphical feature richness

As further dimension we want to introduce the dimension of graphical complexity. This dimension itself has two directions. First, the more colors and different graphic elements we introduce for an identicon the more different identifiers we can represent. Also are this identicons potentially easier to recognize by humans and show thus a better performance. Second this is likely to conflict with the acceptance which will be higher if the identicon itself is more uniform and don't attract to much attention.

2.5 Requirements of Moji-Identicons

Build on the section [Motivation](#) and also the section [Focus & Innovation](#) we provide in this section the more exact requirements we had in mind while implementing the prototype presented later. The following requirements don't have the same level of importance and are sometimes contradictory.

2.5.1 Basics

As argued on in the motivation for an improvement of identicons the basic requirements of the implementation are to visualize in a human-friendly manner large identifiers. It is possible that both or multiple visualizations are on the same screen. But there are also use-cases where the presentation can be on different devices like a computer and a phone or also on paper.

2.5.2 Aesthetics

To ensure that graphical representations of any kind get adopted and used in a broad manner calls to be perceived as aesthetically pleasing. Which is, if the task in mind to provide 2^{160} different visualizations, not trivial at all. Further the Moji-Identicons will be later presented in a context. In a dialog box, a paper form or on a website you will have other graphical elements which interact with the inserted Moji-Identicons. As more attention seeking the identicon will be as more difficult it will be to integrate in a context without disturbing the other elements.

There are some basic rules which are easy to follow but are restricting us a lot in providing unique visualizations for each identifier possible. First of all, at least in western cultures, it is not at all perceived as aesthetic to use too much different and fluent colors. By controlling the use of colors the disturbance of the Moji-Identicons can be controlled a lot.

2.5.3 Recognizable

Another feature should be the recognizability as a Moji-Identicon. As stated in [Focus & Innovation](#) we try to generate a small representation which is more easy to integrated in documents and screens. Therefore it would be counter instinctive if you need to label the identical visualization as such. Like a *QR-Code* which now a days from most people is as such recognizable it should be with the Moji-Identicons too.

2.5.4 Colorless media / Colorblind people

In the most cases we have colorful and also quite color accurate devices or prints at our disposition. Nevertheless we also have to take into account that this is in some cases not possible as some devices like eInk-Displays or most of photocopiers still produce black and white or grey-tone outputs. The Moji-Identicons don't have to function with the same performance in this cases, but they should work at least at all.

Also there is the constraint of color blind people which we need to take into account. By doing a large part of the encoding by generating a distinctive figure we can assure that the Moji-Identicon rest usable.

2.5.5 Bijectivity

An identifier is encoded into an identicon to be easier readable for humans. Not necessary does a visualization encode all information of the identifier visually. But anyway if you have a Moji-Identicon, at least in digital form, it should be possible to extract the identifier encoded in it. This property can be assured in multiple ways. The easiest would be to integrate the identifier as meta data in the image format. On the other hand this is reducing your choice of image formats and will be lost most probably if the visualization will be converted. A more secure solution to this problem can be achieved by using a kind of image watermark. Just by adding small variations of the color values in the image, not visible to the bare eye, you can encode the information of the identifier.

2.6 Disregarded functionality

Again we want emphasize that the implementation of Moji-Identicons is specialized for the use case of comparing. It will not perform well if you ask people to reproduce this kind of identicons. This means the *memorability* is not guaranteed. Further we don't want to incorporate the possibility to explain a Moji-Identicon to someone else. Neither for recognition nor for reproduction. This means Moji-Identicons are not easy to explain someone through a phone (as long it is not a video call).

Chapter 3

Approach

“The minimum we should hope for with any display technology is that it should do no harm.”

Edward Tufte

3.1 Approach

To start working on a new implementation we had to decide on an approach. We did see two general roads we could take. One was to start from the identifier and work out a visualization, in a sort a bottom-up approach. Also there is the top-down approach, where we imagine what a good visual would be and try to find a way to get from the identifier to the visualization.

Bottom-up

We did do some test following the bottom-up strategy. Some examples of this approach can be found in the [Figure 3.1](#). This visualization just uses 16 different colors per digit of the hexadecimal representation of a SHA1. The digits are arranged in a rectangle.

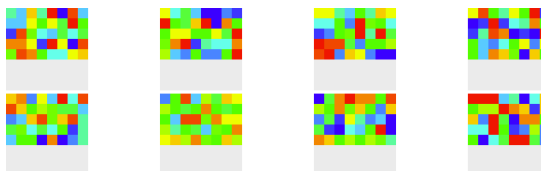


FIGURE 3.1: The minimal complete visual representation of a SHA1 hash.

It is quickly evident that it is difficult to distinguish the identicon even if only eight are presented. No potential success was perceived to produce identicons this way which our requirements would match. Therefore we did not investigate any further this strategy.

Top-down

To illustrate the concept in the proposal ([Appendix B](#)) we did come up with a mock-up. This mock-up was created without any algorithm in mind. Therefore the first steps ([Subsection 3.2.1](#)) of the practical development were done by creating in a top-down approach prototypes of Moji-Identicons.

3.2 Prototyping Environment

As starting point we did use *Processing* [[Ben Fry and Casey Reas, 2001](#)]. This Java based integrated development environment was ideal to produce in record time the first visuals ([Figure 3.2](#)). With only a minimal initial investment we were able to try some ideas and play around.

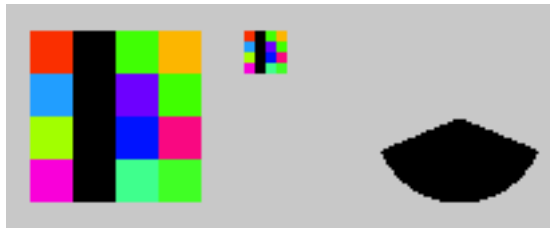


FIGURE 3.2: Very first prototype tests with *Processing* of the Bottom-Up approach.

3.2.1 First tests

The first tests did include a simple color based visualization as shown in [Figure 3.2](#). The most time was invested on the visualization shown in [Figure 3.3](#).

As base for both visualizations it was necessary to first create randomly some hashes. Therefore we used the Java built-in `java.security.MessageDigest` functionality. To have a base to construct visuals we did use the 160 bit input of the hash to map to a 8 by 8 grid with weights. To create 4 levels of weights we did combine 2 bits for each of 64 points in the grid. This leaves us with 32 not assigned bits.

In a second step all points with the highest weight get connected by a line. This produces the stroke images seen in [Figure 3.3](#). To create a connected figure the closest line segments are connect. In the example this lines are visualized with red color.

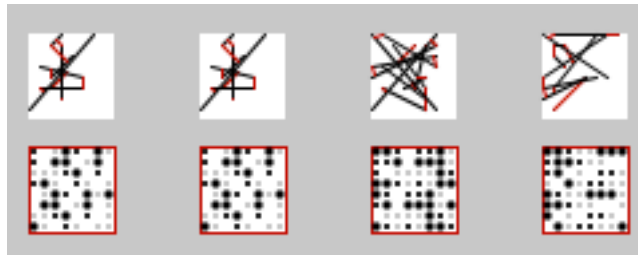


FIGURE 3.3: The first results of the prototyping phase in *Processing*. Top: First crude examples of the visualization. Bottom: A helper visualization which shows weights.

After this we tweaked the stroke drawing algorithm. But the main change was drawing a figure from the center towards the edges. Also did we try with different stroke weights to introduce more distinct visualizations. At the end a color element was introduced. The color was calculated by taking a checksum over the provided SHA1 hash. This ensures that if two provided SHA1 only differ in one bit still create visual clearly distinguishable results. (Figure 3.4)

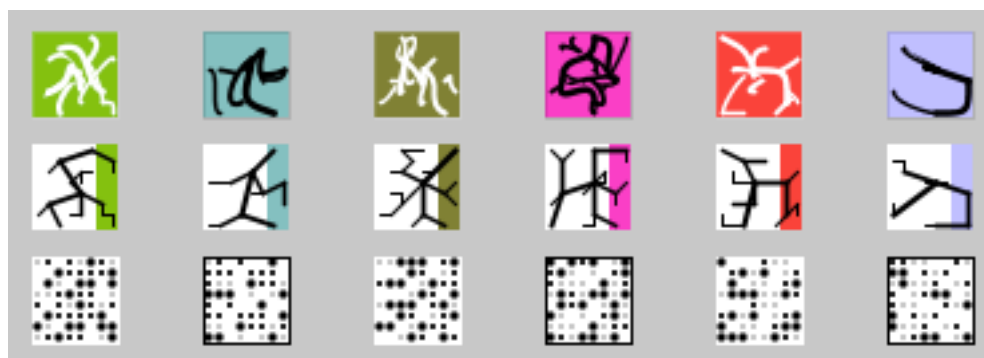


FIGURE 3.4: The end results of the prototyping phase. The top rows show two different variants of connected and centered visuals.

3.2.2 Transition to JavaScript

After we had a basic concept and a first working implementation of the Moji-Identicons we wanted to present our work to future users and let them test if they are effective. We did some investigation if it is possible to distribute the code as Java-Applets. But after some tests we did see that it was rather difficult to distribute the experiments this way.

The final goal was anyway to have an implementation which works easily in modern web browsers we did start reimplementing the code in JavaScript. Also in expectation to have a simpler possibility to switch to a more test driven development with this change.

3.3 Final Environment

The final version of the Moji-Identicons (Figure 3.5) was implemented in JavaScript running in browser environments. The generation of the visuals itself is completely done client-side which has different advantages. There is no need to install software or allocate additional resources on a server for generating the visuals. As images tend to use more size to be transmitted as a simple string we can reduce the bandwidth used. On performant clients we can argue that the page loading is shorter as there is no need to open multiple connections to get the images of the Moji-Identicon. There are disadvantages too. It is not possible to control the correct generating and display of the visualizations. Further on slow devices (like older smartphones) it may need some time to create the Moji-Identicons. Also there needs to be a client which supports some newer technologies.



FIGURE 3.5: Some examples of final Moji-Identicons. (Codename *FigureBlob2*)

3.3.1 Technologies

In general we did write our implementation with HTML5 in mind. This is not a hard requirement and the code should work with some tweaking also in older HTML standards. Where we can't make compromises is the `canvas` element which is an integral part of the library we wrote. The `canvas` element delivers a surface which we can manipulate with basic performant graphic operators down to each single pixel. The solution does reportedly work in the state of the art browser of all well known manufacturers.

The operator which `canvas` element provides were developed as basic as possible to be easily implemented by all clients. This challenged us to find a basic 2D geometry library.

3.3.2 2D Geometry Library

There are tons of really nice and extremely powerful frameworks available, most of under an open source license. The most provide complete library enabling two dimensional geometry calculations on points and lines. But also a lot of other functionality. The prospect to add a huge frame work just for providing some basic functionality was not ideal.

Also it is good to ensure implementation to be as independent as possible. Therefore we decided to implement the needed functionality again, thus `geom.js` (Table 3.1) was born.

<code>Geom.Point2D(x,y)</code>	takes x,y number	Constructor of a point.
<code>*.toString</code>	returns a string	Outputs a string representation of a point.
<code>*.distance(oP)</code>	takes a <code>Point2D</code> , returns a number	Calculate distance to another point.
<code>*.checkType(oP)</code>	takes a <code>Point2D</code> , returns true	Check if is a valid point.
<code>Geom.Line2D(a,b)</code>	takes a,b <code>Point2D</code>	Constructor of a line.
<code>*.toString</code>	returns a string	Outputs a string representation of a line.
<code>*.closestPointToPoint(oP)</code>	takes a <code>Point2D</code> , returns a <code>Point2D</code>	Returns the closest point on the line in reference to the provided point.
<code>*.closestLineToPoint(oP)</code>	takes a <code>Point2D</code> , returns a <code>Line2D</code>	Returns the closest line connecting the given line to the provided point.
<code>*.closestLineToLine(oL)</code>	take a <code>Line2D</code> , returns a <code>Line2D</code>	Returns the closest line connecting the given line to the provided line.
<code>*.checkType(oL)</code>	takes a <code>Line2D</code> , returns true	Check if is a valid line.

TABLE 3.1: A short overview of the `geom.js` library.

With all tools ready to go we could start implementing our approach of Moji-Identicon again in JavaScript.

3.4 Structure

With the experience gained by writing the prototype code we did structure our code from beginning in two main parts. The first stage processes the input and generates points on a matrix. On the base of this points in a second step the visualization itself is build. This splitup made it possible to easily test different combinations of both stages.

3.4.1 Stage: Point Matrix

In `Identicon.PointMatrix` a quadratic matrix, in our tests with a size of 32x32, was specified. According to the hash input provided the different algorithms generated *points* on this matrix.

3.4.1.1 Rastering

Developed for the prototype we did re-implement the method, which we call *rastering*. The input in hexadecimal representation is mapped to pixels. If the value of the input was reaching a certain level, a point was generated. This method has the crucial disadvantage that all values below the pre set level don't provide any information to the point matrix. Further was the input repeated until the whole matrix was filled. To minimize these effects we did look for a better algorithm.

3.4.1.2 Distribution

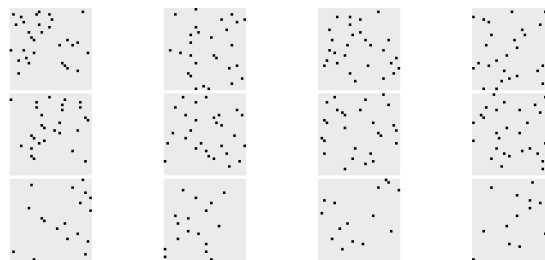


FIGURE 3.6: Uniform distribution of the points generated by our simple algorithm. Top: Saturation value of 1, Middle: Saturation value of 0.9, Bottom: Saturation value of 0.5.

In this attempt we uniformly distribute the points (Figure 3.6) based on the input over the defined matrix. This is done by jumps defined through the input of a single hexadecimal value (called cypher in Figure 3.7). To match the defined matrix the size of the jumps are adjusted according the maximal possible points in the matrix.

```

1 getJump = function (cypher) {
2     return cypher * (Math.pow(size,2) / ((hash.length+1) * 16 / 2));
3 };

```

FIGURE 3.7: How far to jump according a single hexadecimal input value.

Still we need a way to adjust the density of the generated points. Therefore we introduce again a value which regulates the smallest possible size of a jump. By decreasing this value called *saturation* we can produce a more sparse matrix. (Figure 3.8, Line 5)

```

1 for (var u=0; u<size; u++) {
2     for (var v=0; v<size; v++) {
3         if (jump <= 0 && cypher < hash.length) {
4             var data = Identicon.Help.getIntFromHexAt(cypher++,hash);
5             if ((data/15 > 1-saturation)) {
6                 m.push(new Geom.Point2D(u,v));
7             }
8             jump = getJump(data);
9         } else { jump--;}
10    }
11 }

```

FIGURE 3.8: Code which is generating points on the matrix, u and v represent the x and y axis of the final image.

3.4.1.3 Ordering

The later visualization is dependent on the order we provide the points to the draw algorithm. Therefore we implemented in a step in-between two different ordering algorithms.

The first approach orders the points according the distance a point has to the center of the matrix. (Figure 3.9)

```

1 m.sort(function(a,b) {
2     return center.distance(b) - center.distance(a);
3 });

```

FIGURE 3.9: Ordering the points based on the distance to the center.

Later we used mainly the second algorithm. Here the points are ordered circular around the center. Like this we could evade the problem that the direct connections of two following points did cross itself. (Figure 3.10)

```

1 m.sort(function(a,b) {
2     return Math.atan2(a.x-center.x,a.y-center.y) -
3           Math.atan2(b.x-center.x,b.y-center.y);
4 });

```

FIGURE 3.10: Ordering the points around the center in a clockwise fashion.

3.4.2 Stage: Visualization

Having a nicely prepared matrix with points distributed dependent on the hash we can start to play around with creating visuals. We did create more than ten different visualizations (Subsection 4.1.1). The algorithms have all a similar composition, we briefly explain based on the example (Figure 3.11) of the final Moji-Identicon algorithm how a draw method is structured.

```

1 drawIdentFigureBlob2: function() {
2     // initialized variables
3     var m = new Identicon.PointMatrix(this.PIXEL, this.HASH.toString(), 0.5
4                                     ).getByDistribution(2);
5
6     var ctx = this.ctx;
7     var size = this.PIXEL;
8     ctx.lineWidth = 1;
9     var center = new Geom.Point2D(size/2, size/2);
10    // draw background
11    ctx.fillStyle = 'hsl(' +
12                  (Identicon.Help.getChecksum(this.HASH.toString())*16) +
13                  ', 80%, 70%)';
14    ctx.beginPath();
15    ctx.moveTo(0,0); ctx.lineTo(size,0); ctx.lineTo(size,size);
16    ctx.lineTo(0,size); ctx.lineTo(0,0); ctx.fill();
17    // draw a shape using quadratic curve
18    ctx.beginPath();
19    back = m[0];
20    for (var i = 1; i < m.length; i=i+2) {
21        var a = m[i-1]
22        var b = m[i]
23        ctx.quadraticCurveTo(a.x, a.y, b.x, b.y);
24    }
25    ctx.lineTo(back.x, back.y);
26    // fill the shape
27    ctx.fillStyle = "#000";
28    ctx.fill();
29    ctx.closePath();
30 }

```

FIGURE 3.11: The draw function of the final version, created with the rather poetic name `drawIdentFigureBlob2`. (For the visualization see Figure 3.5.)

In the first part the initialization of the necessary variables and the generation of the `PointMatrix` is done. Later a box, eventually colored, of the size of the visual is drawn. And the last part is handling the drawing of the figure. In the example listed two points at a time are used as input to draw a quadratic curve segment. All these segments are connected and form a shape which at the end is filled with black.

3.4.3 Ensuring Bijectivity

In a last step the identifier itself is encoded in the image. This is done using a simple algorithm which adds three bits per pixel of the identifier in the final image. We did not explore more advanced techniques in this field.

The used approach defines even color values as an encoded zero and odd values as an encoded one. Beginning in the top-left the pixels are adjusted in sequence. On each color channel the input is adjusted based on the input of the identifier by one bit.

3.5 Followed Design Guidelines

3.5.1 Color

There have been conducted many studies [Ware, 2004] to find the number of colors a human can distinguish in a set of symbols. All studies recommend a maximum of between 6 and 12 different colors. Therefore where we used colors it was always a goal to minimized the amount of hues.

3.5.2 Connected Shapes

If we belief David Marr [1982] to be able to recognize objects, the human creates in a first stage a so called *primal sketch*. In this stage the human tries to recognize and separate with basic processes a figure from his background. We believe if the identicons consist out of a foreground object and uniform background, the users are faster differing such objects. Based on this beliefs we did concentrate mainly on fully connected simple shapes. Further we did only produce one shape per identicon.

Chapter 4

Evaluation

“Mais pour le moment, il est certain que l’on éprouve quelque embarras à exprimer par un chiffre brutal toutes les oscillations d’une pensée ; le chiffre ne peut avoir qu’une précision trompeuse; comment en effet pourrait-il résumer ce qui aurait besoin de plusieurs pages de description !”















Alfred Binet, 1900, p. 199

Which is the best kind of identicon? This is not a trivial question. Because each human perceives different and the identicons should work in different media the visualizations needs to be highly versatile. Even if we just consider the use of identicons on screen, different configurations, the resolution and the brightness influence the result and make a complete evaluation nearly impossible.




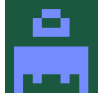


But nevertheless it is important to test especially in this kind of field as the human perception and processing part has the same importance as the technological facet of generating the identicons. Because of time constraints it was not possible to test the different implementations on all different media. Therefore we decided to investigate the media we think will be the most used for identicons. And this is arguably the web.

4.1 Catalog of Identicons

4.1.1 Own Development

Example	Name	Description
	Minimal	Hash as color field.
	Figure	The first implementation. Distance sorting.
	FigureBW	A colorless variant.
	FigureC1	Base color variant.
	FigureC2	More fluent colors.
	FigureC3	Only edge element colored.
	FigureGrey	Grey background.
	FigureBlob	First circle sorting implementation.
	FigureBlob2	With quadratic curves.
	FigureMoji	FigureBlob with added eyes.
	FigureMoji2	FigureBlob2 with added eyes.
	FigureMoji3	One eye, no color.
	FigureTwo	Two independent shapes.
	FigureTwoTail	Two connected shapes.

4.1.2 Available Implementations

Example	Name	Description
	Classic	JavaScript implementation of Nine Block http://www.moreslowly.jp/products/identicon/
	GravIdenticon	Identicon provided by http://gravatar.org
	GravMonsterid	MonsterID provided by http://gravatar.org
	GravRetro	Retro provided by http://gravatar.org
	GravWavatar	Wavatar provided by http://gravatar.org
	Robohash	Robohash provided by http://robohash.org

4.2 Experimental Plan

It is not the goal to measure if people are able to memorize identicons we don't let the subjects reproduce the visualizations. Therefore the time to find identical identifiers is measured. A possible setup for this case is to present a set of identifiers and ask to find a duplicate in this set.

This simple form of experimental setup helps to control a lot of side effects which we don't want to measure like; the speed with which the subject can interface with the website, the mental capabilities to remember visuals, the size of the working or mid-term memory of the candidates. Simply by measuring the time consumed until the decision is made and counting the correct decision gives a solid performance index. The sets including a duplicate and the lack of are showed with the same probability. The presented visuals as also the order in which the different kinds of identicon appear was fully randomized.

4.3 Evaluation Application

To be able to collect easily enough data we decided to let the subjects do the experiment at home on their own computers. This simplifies the organization of the experiments enormously. We lose some control of the testing environment which the subjects choose

them self. But we can argue that this simulates real life situations in which the developed identicon should work at the end.

4.3.1 JavaScript Implementation

As explained (Chapter 3) we want to create a easily distributable application to conduct our experiments. Therefore we did decide to write a browser based application based on JavaScript which incorporates our identicon library. (This implementation figures at the same time as a proof-of-concept for our Moji-Identicon library itself.)

4.3.2 Frameworks, Components and Services

The use of JavaScript brings the advantage of a huge choice of professional and mature libraries, frameworks and tools.

4.3.2.1 Application unspecific

The HTML5 Boilerplate¹ creates a common ground on different browsers. It also includes a feature detection library called Modernizr². As a base for the layout and design we did incorporate Bootstrap³. Another useful framework which was used for template based generation of the content is Handlebars⁴. On the base of these common web development tools we could concentrate on our use case, the evaluation application.

4.3.2.2 Application specific

To generate identicons we need a way to create easily hashes. We loaded just the SHA1 generation part of crypto-js⁵ to handle this task. To structure the program flow of our application we decided to incorporate the *Javascript Finite State Machine*⁶. Finally we needed a way to save the results of our experiments in a persistent way on a server. This task was achieved by the document oriented, NoSQL database service Mongolab⁷ based on MongoDB⁸.

¹<http://html5boilerplate.com/>

²<http://modernizr.com/>

³<http://twitter.github.com/bootstrap/>

⁴<http://handlebarsjs.com/>

⁵<http://code.google.com/p/crypto-js/>

⁶<http://github.com/jakesgordon/javascript-state-machine/>

⁷<http://www.mongolab.com/>

⁸<http://www.mongodb.org/>

4.3.3 Workflow

The application is based on multiple screens which first briefs the user and guides him through the experiment. If the user conducts the the experiment the first time he will be introduced with a test run. Once the real experiment begins the browser switches to fullscreen to block as good as possible from distractions. At the end the result is loaded anonymously and automatically to the server. (If this is not possible the result is show to the user with the request to send it by email.) Finally the user has the possibility to provide a general feedback about the experiment.

4.4 Experiments & Results

All together we did conduct four experiments. Each of the experiments was announced by email. The experimental group were mostly working in the field of software development or computer science. The participation was thanked for with a small sweet treat. The duration of the experiments in its whole were in average well under five minutes.

The results are statistically verified by a dependent t -test for paired samples as the experiment is done by repeated measurements of the different conditions (types of identicon). If referred to as statistically significant have a p value of at least 0.001.

Only important results are discussed in the following sections.

4.4.1 Pre-Test

Before investing time in a probably dry land we did a short pre-test. Just comparing the first *Figure* identicon with the simple text representation of the identifiers. (8 users with 8 trials per condition, $N=64$, $p=0.001$)

In the mean the text representation consumed 7899 ms (SD 8255.37ms) until decision where the *Figure* only consumed 2895 ms (SD 2733.39ms). This result is statistically highly significant. Also the percentage of correct answers is in favor of the *Figure* which was answered correct 88% (SD 0.3) in comparison to 59% (SD 0.5) of the text.

This was the green light to further investigate our approach.

4.4.2 Base Level

The first experiment was designed to establish a base level of the measured variables. The *Figure* implementation was tested against the two variants of showing the whole

identifier and truncated version with only ten digits. Further did we test also the *Minimal* implementation to see if it still performs better than the text representation. (16 users with 6 trials per condition, N=96)

4.4.2.1 Text vs. Minimal

Neither the time to decision nor the amount of correct answers is significantly different between the use of the *Minimal* and the text representation.

4.4.3 Text vs. Truncated Text

Also the comparison between the full presentation and the truncated text representation did show any significant differences. (Text: Mean 5146.80ms , SD 3884.67ms; Truncated Text: Mean 4864.38ms, SD 5119.46).

4.4.4 Color Variations

To get more knowledge about the influence of the color we did test different variations of the *Figure* implementation. Namely *FigureBW*, *FigureGrey*, *FigureC1*, *FigureC2* and *FigureC3*. The first two don't incorporate any color in its visualization. The second of the too distinguishes in less contrast. The *FigureC3* does not exactly differ only in the color but also on the used space of the color cue. (16 users with 7 trials per condition, N=112)

Type	Decision Mean	(ms) SD	Correct Mean	SD
FigureBW	2780.29	3348.73	91.07%	0.29
FigureC1	2343.62	2212.09	87.50%	0.33
FigureC2	2512.48	2227.46	87.50%	0.33
FigureC3	3130.63	3021.93	87.50%	0.33
FigureGrey	2985.42	3395.20	81.25%	0.39

TABLE 4.1: Overview of the results in color variations.

4.4.4.1 Black and White vs. FigureC1

Surprising in our opinion is that the addition of color not results in significant better performance. ($p=0.0724$, Color: Mean 2343.62ms, SD 2212.09ms; Black and White: Mean 2780.29ms, SD 3348.73ms)

4.4.5 Shape Variations

In this test series the focus was set on variants of different shapes. Refer to the catalog (Section 4.1) to see examples of the different approaches. The test incorporated the following implementations: *FigureBlob*, *FigureBlob2*, *Figure*, *FigureMoji*, *FigureMoji2*, *FigureTwo* and *FigureTwoTail*. Because of a technical problem the results for the *FigureTwoTail* were not collected. (19 users with 4 trials per condition, N=76)

Type	Decision Mean	(ms) SD	Correct Mean	SD
Figure	4202.20	2669.20	85.53%	0.35
FigureBlob	4298.07	2828.79	89.33%	0.31
FigureBlob2	4122.63	2863.50	92.11%	0.27
FigureMoji	4845.47	3172.84	88.16%	0.33
FigureMoji2	5058.00	4040.81	84.21%	0.37
FigureTwo	4664.04	3153.31	84.21%	0.37

TABLE 4.2: Overview of the all non significant differences of the shape variations.

None of the shape variations were significant different (Table 4.2). Which allows us to argue that the simpler, aesthetic more pleasing figures show the same performance as the more complex ones.

4.4.6 Final Validation

To finally validate against existing implementations we did test all types out of the catalog (Subsection 4.1.2) against *FigureBlob2* and *FigureMoji3*. We like to mention here that we are conscious about the fact that some representation of the other implementations were presented smaller as they were developed for. But it was necessary to guarantee a common ground for all tested. (11 users a 3 trials per condition, N=33)

Type	Decision Mean	(ms) SD	Correct Mean	SD
Classic	1988.31	1067.64	75.00%	0.44
FigureBlob2	3616.73	2701.59	90.91%	0.29
FigureMoji3	4304.82	2376.76	78.79%	0.42
GravIdenticon	2823.42	1781.32	90.91%	0.29
GravMonsterid	3059.94	1917.41	87.88%	0.33
GravRetro	3364.73	2660.78	87.88%	0.33
GravWavatar	3161.85	2494.72	81.82%	0.39
Robohash	3562.06	2594.36	87.88%	0.33

TABLE 4.3: Overview of the final validation.

4.4.6.1 Nine Block vs. Blob2

The *Classic* implementation which is based on the *Nine Block* identicon shows the best performance if measuring the time to decision. ($p=0.0054$, Classic: Mean 1999.21ms, SD 1052.69ms; Blob2: Mean 3616.73ms, SD 2701.59ms).

But on the other hand were the results more correct using the *FigureBlob2* implementation with 91% correct answer of our implementation against 76% correct answer of the *Classic* implementation. This second results but didn't show statistical significant values ($p=0.096$).

This makes it difficult to interpret this results. Unfortunately we had only 11 participants in the final validation which possibly was not enough to gain meaningful results.

Chapter 5

Conclusion

“I love standards, there are so many to choose from!”

Unattributed

5.1 Conclusion

Thorough investigation of different concepts and approaches allowed us to find a set of basic features. The described dimensions and thereof resulting requirements build a starting point for a new development of identicons. These requirements target a general not well defined use case.

By using a test-driven development we could decide at each iteration which direction was worth exploring. This allowed us to refine our implementation to the end product. The experiments provide at the same time first scientific evidence about the performance in a laboratory environment. How the new implementation performs in a real world situation needs still to be discovered.

The results show that any kind of structured visualization provide a clear advantage over working with the only text representation of identifiers. Which makes us wonder why such techniques are not more often seen in daily used applications.

5.2 Improvements

The presented work is only the beginning of a field of possible powerful visualizations of identifiers. We tried to create a general visualization of identifiers which can be applied

in different domains. But we see immense improvement of our approach as also domain specific features which could be added to enhance the use of the identicons.

5.2.1 Additional Functionality

Our approach was based on hash identifiers which are completely free of any meaning. It is but easily imaginable to incorporate features of the data in the visualization: to visually group data from the same user, to visualize the age of a document, etc. An easy approach to do this would be to use color for this information. But more subtle and interesting solutions can be imagined by enhancing the shape, adding structures or extending the visual.

5.2.2 Enhancing Barcodes or QR-Codes

This proposition is not directly related to the main topic of this thesis. We would like to document the idea which occurred us in the course of working on the Moji-Identicons. Our approach was solemnly focused at the development of a new visualization of identifiers which is independent of other technologies.

The approach we would like to see is enhancing existing machine readable barcodes or 2D-codes to compare them more easily by humans. This is a rather delicate task as it should not interfere with the reading capability of machines. We imagine by adding color features to the mainly black and white codes would give users the ability to easily exclude differing identifiers. Aside of changing the color it is also feasible to play with the length of barcode bars. Or by providing additional graphical elements to the user.

5.3 Outlook

In a scope of a semester at the university we were able to investigate the topic of identifier visualization thoroughly. Also we could create some prototypes which show that the specified requirements are implementable. The development process was iterative and test driven, meaning the different stages were tested immediately with a small population of users.

5.3.1 Next Steps

After finalizing the own implementation we want to open-source our work and provide easy to integrate code. The main problem is to get to the interested people which can comment with constructive criticism, ideas for extensions and spread the word.

This document itself will be publicly available but will not officially published. If we find time in the near future an attempt will be made to publish a short paper. This would at least introduce the concept of an *identicon* to the scientific community. At best trigger some future work in this field.

5.3.2 Towards a Standard

Because in our perspective the main use case for identicons is comparing identifiers throughout different systems and media we strongly recommend to develop a standard. It is necessary that different systems generate the same visualizations to make the life of the users easier. Without the standardization of this process the whole point of our concept is rendered useless.

It is surpassing the scope of this work to define a standard as such. We estimated that it needs an expert board to evaluate the different features such a visualization needs to incorporate. Further intensive testing would be required. The definitive solution should perform with people out of different geographic regions and cultures. The goal is to find the optimal parameters which work and are additionally to this performant in as big as possible amount of use cases.

fin

Appendix A

Glossary

canvas	Element of HTML5 providing a graphic canvas
DVCS	Distributed/Decentralized Version Control System http://en.wikipedia.org/wiki/Distributed_revision_control
HTML5	Hypertext Markup Language Version 5 http://www.w3.org/TR/html5/
SHA1	Secure Hash Algorithm Version 1 http://en.wikipedia.org/wiki/SHA-1
TID	Paypal Transaction Identifier https://www.paypalobjects.com/IntegrationCenter/ic_transaction-search-and-details.html
UUID	Universally unique identifier http://tools.ietf.org/html/rfc4122
UPU S10	Universal Postal Union S10 UPU [2011]

Appendix B

Proposal

Msc Proposal, *eidos*

Abstract

This document describes a proposal for a MSc Thesis in Computer Science. In the field of information management (databases, version control systems, digital archiving, data exchange) current and emerging technologies take advantage of long unique identifiers. These are either based on the information at hand, like different forms of hash functions or are chosen completely by random. In both cases the users (mostly developers) have trouble working with these identifiers because of their relatively big size. In the scope of this MSc Thesis we propose a way to help the users to work in a more human friendly way with long unique identifiers.

Idea

A UUID (Unique Universal Identifier) like `2efd5421-dbf0-4fea-82c0-77d634a5d9a2` or a SHA1 as used by GIT `43c5ed1b003ced7f396b7c21f8b73845cbeb80bb` are identifier strings which are non-trivial to parse for a human being. These identifiers have in general no order in any way. Therefore the most important task done on such kind of data is to search a specific identifier in a list of such. Further this task can basically described as a matching task where two identifiers at a time are compared and tested for a match.

Such tasks are mostly performed by systems in the background, which are extremely performant at it. This makes sense as this kind of identifiers is developed exactly with machines in mind. But in the development phase the developers are over and over faced with the problem for debugging and testing purposes to compare such identifiers with the naked eye.

Humans are pretty good in recognizing and comparing different strings, but only if they make sense to them. But identifiers make no sense and are huge. One of the tricks used in working with such identifiers is to only represent a small part of the identifier. This works quite good because of the randomized nature of these identifiers. (GitHub as example uses the first 10 chars of a string e.g. `43c5ed1b00`).

But the human can do much better if these strings would be represented more adapted to his visual capabilities. The main idea of *eidos* is to introduce a library which allows to generate a visual representation of the identifiers described above. An identifier could have the size of an icon. It should be optimized that the human eye can compare differing such icons as fast as possible. Also here only a part of the identifier can be represented to the human. It will not be possible to generate representations which show the whole identifier and are pleasing to look at too.

Following a first not at all elaborated visualization to show how long identifiers could be represented only for explanation purposes:



Two examples of 16x16 pixel icons representing different identifiers. To compare such icons and find matches should be much easier as comparing identifier strings.

Scope

The choice of the design of the representation of the visualization as also the right image format (PNG, JPG, GIF) to work with.

User tests of the visualizations to find the optimal version for a performant comparison task.

Developing a library which allows to generate visualizations of the most used different identifier types; like the different hash functions SHA-1, SHA-256, MD5, crc16/32, further the random identifiers like GUID and UUID, and also imaginable is a representation of MAC, and IPv4, IPv6 addresses.

The implementation of a reference library in a programming language to be chosen yet. And further an example of at least one other implementation in an other programming language.

Creating a small website to publish the idea and the library. Spreading the word ...

Problem Dimensions

Visual

- The design of the representation itself. The choice and combination of the different elements which can be used; form, structure, symbols, color, gradients.
- Testing how the user performs by doing the different task like finding, and comparing representations.
- Distinction of the different identifier types through the design.

Aesthetics

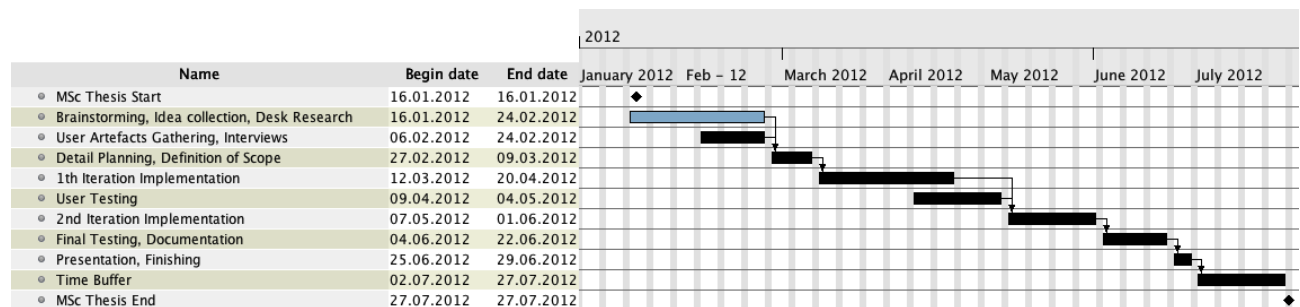
- The aesthetics of the visualization will play an important part in the adaption of the concept by the public.
- A static part of the visualization should make the recognition factor of the visualizations evident. (The visualization should be recognized as such without any additional labeling.)

Technical

- Interchangeable representation as visualization. It should be possible that the full identifier can be extracted out of the visualization of the generated image. (Even if not all information is clearly presented to the viewer.)
- The use of image formats without generating any licenses problems. Also without the use of external libraries as this could break the extraction of the identifier if the external dependencies change there implementation.
- Probabilities of collisions when creating the human readable part of the representations.

Planning

The following Gant-Chart shows a fairly rough planning of the work through the semester. An updated and more detailed version will be provided at the end of the exploring phase.



Sources

[1]

Chroma-Hash Demo: <http://mathtt.github.com/Chroma-Hash/>. Accessed: 2012-01-04.

[2]

Dhamija, R. 2000. Hash Visualization in User Authentication. (2000).

[3]

Globally unique identifier - Wikipedia, the free encyclopedia:

http://en.wikipedia.org/wiki/Globally_unique_identifier. Accessed: 2012-01-08.

[4]

Klanten, R., Bourquin, N. and Ehrmann, S. 2008. *Data Flow: Visualising Information in Graphic Design*. Die Gestalten Verlag.

[5]

List of hash functions - Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/List_of_hash_functions. Accessed: 2012-01-08.

[6]

Menezes, A.J., Menezes, A.J. and Menezes 1996. *Handbook of Applied Cryptography*. Crc Press.

[7]

Miller, G. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*. 63, (1956), 81-97.

[8]

Perrig, A. and Song, D. 1999. Hash Visualization: a New Technique to improve Real-World Security. *IN INTERNATIONAL WORKSHOP ON CRYPTOGRAPHIC TECHNIQUES AND E-COMMERCE*. (1999), 131--138.

[9]

Tufte, E.R. and Tufte, E.R. 2005. *Envisioning information*. Graphics Press.

[10]

Ware, C. 2004. *Information Visualization. Perception for Design.: Perception for Design*. Morgan Kaufmann.

Bibliography

- Adrian Perrig and Dawn Song. Hash visualization: a new technique to improve real-world security. In *In International Workshop on Cryptographic Techniques and E-Commerce*, page 131–138, 1999.
- Alfred Binet. *La suggestibilité*. Paris : Schleicher, 1900. URL <http://archive.org/details/lasuggestibilit00bine>.
- Andreas Gohr. MonsterID, January 2007. URL <http://www.splitbrain.org/projects/monsterid>.
- Ben Fry and Casey Reas. Processing.org, 2001. URL <http://www.processing.org/>.
- David Marr. *Vision: A computational investigation into the human representation and processing of visual information*. WH Freeman, 1982.
- Don Park. Visual security: 9-block IP identification, January 2007. URL <http://www.docuverse.com/blog/donpark/2007/01/18/visual-security-9-block-ip-identification>. <http://web.archive.org/web/20080703155519/http://www.docuverse.com/blog/donpark/2007/01/18/visual-security-9-block-ip-identification>.
- Gertrude Stein. *Geography and plays*. Boston, The Four seas company, 1922. URL <http://archive.org/details/geographyplays00steirich>.
- Hermann Ebbinghaus. *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie*. Leipzig: Duncker & Humblot, 1885. URL <http://archive.org/details/berdasgedchtnis01ebbigoog>.
- Jared Tarbell. Nine block pattern generator, March 2002. URL <http://www.levitated.net/daily/lev9block.html>.
- Jared Tarbell. Nine block, October 2003. URL <http://www.complexification.net/gallery/machines/nineblock/>.

- J. P. Lewis, Ruth Rosenholtz, Nickson Fong, and Ulrich Neumann. VisualIDs: automatic distinctive icons for desktop interfaces. *ACM Trans. Graph.*, 23(3):416–423, August 2004. ISSN 0730-0301. doi: 10.1145/1015706.1015739. URL <http://doi.acm.org/10.1145/1015706.1015739>.
- OpenSSH. OpenSSH 5.1 release notes, July 2008. URL <http://www.openssh.com/txt/release-5.1>.
- Rachna Dhamija. Hash visualization in user authentication. 2000. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.2948>.
- Shamus Young. New website features - twenty sided, December 2007. URL <http://www.shamusyoung.com/twentysidedtale/?p=1460>.
- UPU. Identification of postal items – 13-character identifier, October 2011. URL www.upu.int.
- Colin Ware. *Information Visualization. Perception for Design.: Perception for Design*. Morgan Kaufmann, 2. a. edition, May 2004. ISBN 1558608192.
- Wikipedia. Identicon - wikipedia, the free encyclopedia, February 2007. URL <http://en.wikipedia.org/wiki/Identicon>.