# lain zine

vol.1

Lainzine
Issue 1
Published 20 April 2015

---

# Contents

*Editors' Notes* 1
*For Lainzine #1* 1
*Noise* 2
*Gopher Protocol* 2
*Recommended Reading* 6
*Art of the Glitch* 7
*Introduction to Cryptography* 12
*Word Search* 15
*Where Do I Start?* 16
*FreeBSD Guide for Newbs and Dummies* 18
*Youtube Proxy* 20
*Structure-based ASCII Art* 23

---

**COLOPHON**
Created by the good people of Lainchan from all around the world.
https://lainchan.org

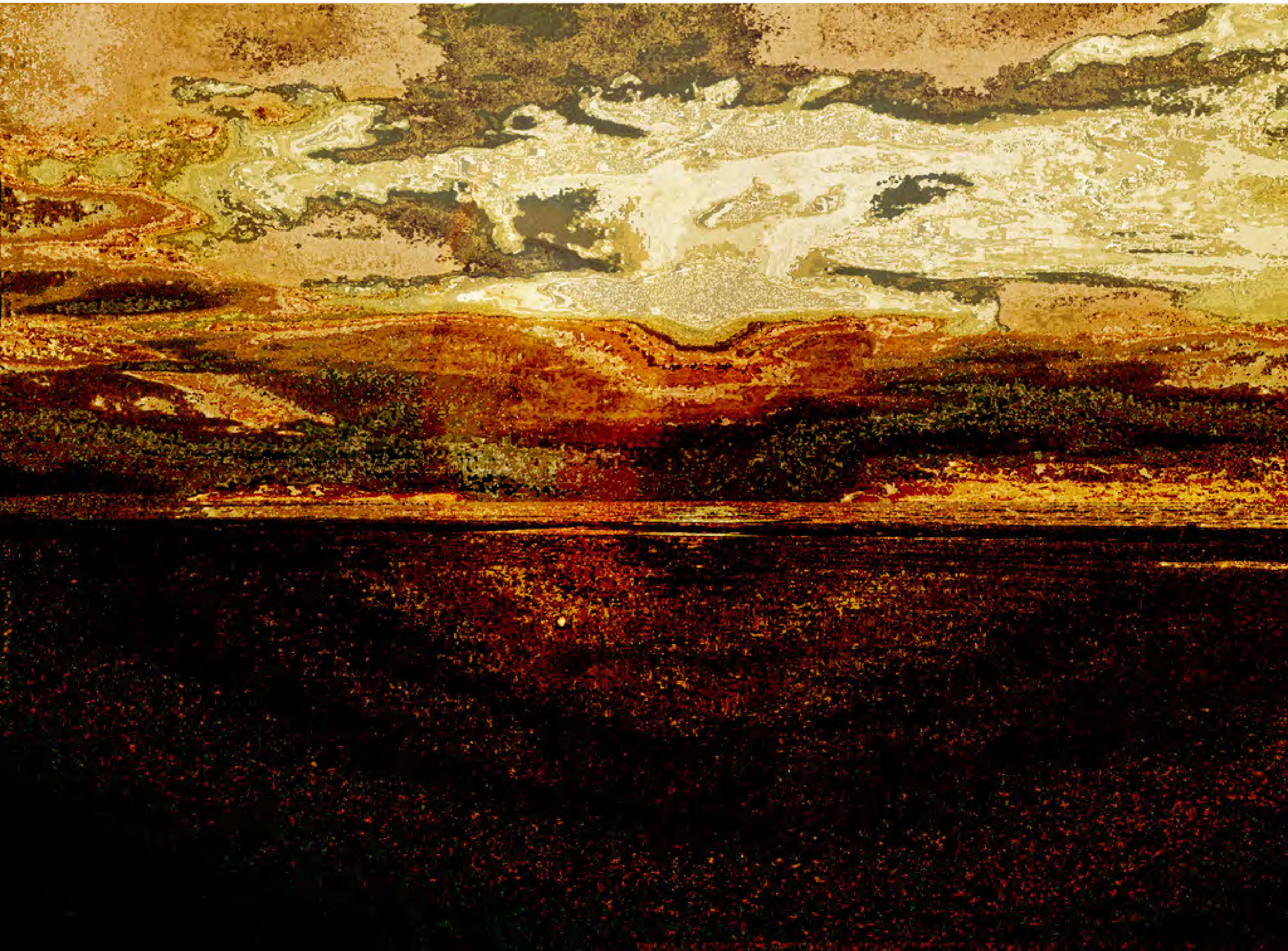
Released in good faith and for free to the public domain.
€0, $0, £0

**STAFF**

| | |
|---|---|
| Editors | junk & Tilde |
| Typesetters | Ivan & Dylan |

*In*
*Waking*
*All*
*Known*
*Unconscious*
*Reality*
*Atrophies*

*Linger*
*And*
*It*
*Nihilates*



# Editors' Notes

I hope enjoy this magazine. We all have put some effort into it and I am personally impressed by what we accomplished. Thanks for reading!

finfq bas zp bsosnhe uxrmz – R ddpvwef cvqsn kumu pq nxfq.

— junk

We live in a ferocious world, one where it is easy to get swept up and aggregated into the 'next big thing'. Lainchan is a place somewhat removed from the beaten path, where the magnetic pull of social assimilation is not quite as strong. It is an eclectic gathering of people from across the planet, who come together to explore the essential ideas of the present, the stuff that will become the science fiction and science fact of the future.

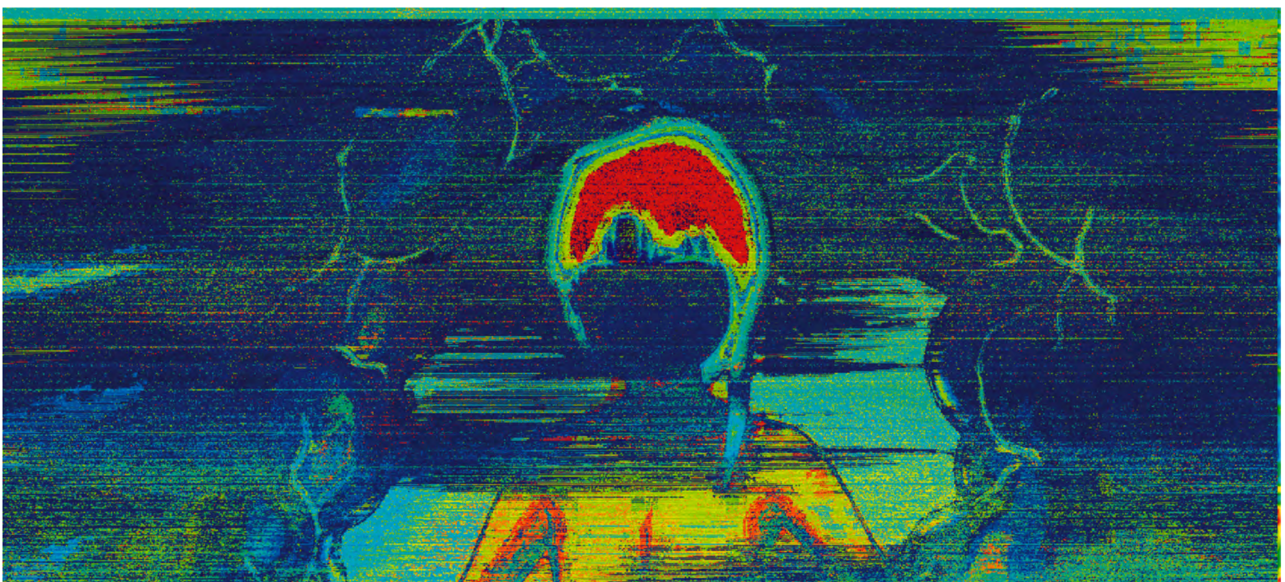
Most of all, Lainchan is friendly. Stop by and say hello, won't you?

— Tilde

# For Lainzine #1

Woa, a zine on Lainchan? Is this what you guys were doing while I was trudging through sector 7.b–189 looking for unused datacards last week [depression]? This is really something, glitch scattered between the pages… impressive. Just in time for the birthday party. Lainchan is the name, OC is the game. Look out for my submission soon.

— Kalyx



# Noise

*Tapping into the noise – building a contact microphone*

There is a constant humming in the wires above.

We are told that at either end sits a small terminal box that listens carefully and modulates the cosmic pulses to find what is useful.

But in recent years, the wires have stretched out. The terminal has receded to some unattainable point in the distance. We find ourselves in the open: Alone and listening, but unable to separate the signal from the noise.

## Parts needed

- Piezo disk,
- scraps of wire,
- audio jack socket,
- something to amplify and listen with,
- soldering iron and solder (optional),
- something to secure the mic in place (optional).

## Instructions

Sound doesn't only travel through the air. A contact microphone picks up the sounds from inside of solid objects. You can shout as loud as you want, the contact mic can't hear you. But if you stick it to a wooden board and scrape a nail across it you can hear the sound of the wood grain resonating, crushing the piezo crystals and generating a small electric field.

Doesn't work? Make sure your contact mic is connected to an amplifier of some sort, PC speakers will work if they're turned all the way up. Disconnect the wires and try them the other way round, then try the first way again.

# Gopher Protocol

Developed in 1991 by a team at University of Minnesota, the gopher protocol is a TCP/IP document distribution system that was popular before the world wide web became the standard protocol. It is primarily devoted to serving plain text documents organized into a hierarchical directory structure and lacks formatting, dynamic content, and even the mixing of media and text. While the protocol may not have style sheets, it certainly has style.

## The Protocol

The simplicity of the gopher protocol is evident from the mere fifteen pages of RFC 1436 – the Network Working Group's 1993 document on implementing gopher servers and clients. For comparison, FTP's RFC 765 is 68 pages, and HTTP/1.1 is represented by the 176-page 2616. A reader with even a cursory understanding of TCP/IP can learn the core of gopher over a cup of coffee.

Learning the complete internals is better suited to the RFC and is left as an exercise to the reader, but we'll go ahead and write a couple of one-line scripts that handle the majority of the available interactions. All that you need to know is that the default port is '70'.

### Script 1: Ask a gopher server to list its contents

First, the client opens a connection to the server, for the example: sdf.org on port 70 Then, the client sends 'CLRF' to the server to request a directory listing. 'CLRF' refers to "Carriage Return Line Feed", but on most systems will be handled by the simple newline character: '\n'. To do this, use the netcat utility which is a command line program for establishing raw network connections. This very powerful tool can be used for exploring remote hosts and even listening as a server itself.

```
iWelcome to the SDF Public Access UNIX System .. est. 1987\t\tnull.host\t1
i\t\tnull.host\t1
iWe offer FREE and inexpensive memberships for people interested\t\tnull.host\t1
iin the UNIX system and internetworking.  Personal GOPHERSPACE\t\tnull.host\t1
iis available to all users as well as hundreds of UNIX utilities,\t\tnull.host\t1
igames and networking utilities.  We are a federally recognized\t\tnull.host\t1
inon-profit 501(c)7 organization and we are supported entirely\t\tnull.host\t1
iby donations and membership dues.  telnet://sdf.org\t\tnull.host\t1
i\t\tnull.host\t1
1SDF Member PHLOGOSPHERE (151 directories)\t/phlogs/\tsdf.org\t70
1SDF Member GOPHERSPACE (2334 directories)\t/maps/\tsdf.org\t70
1SDF Frequently Asked Questions (FAQ)\t/sdf/faq/\tsdf.org\t70
1SDF Accredited University Courses\t/sdf/classes/\tsdf.org\t70
1Software and Documentation for various computers\t/computers\tsdf.org\t70
7GopherSpace SEARCH Engine\t/v2/vs\tgopher.floodgap.com\t70
1Floodgap's GOPHERSPACE\t/\tgopher.floodgap.com\t70
1NetBSD Distribution Mirror\t/NetBSD/\tsdf.org\t70
i______________________________________________________________________\t\tnull.host\t1
i               Gophered by Gophernicus/1.5 on NetBSD/amd64 6.1_STABLE\t\tnull.host\t1
.
```

**Figure 1:** Response of the Gopher server

```
echo "\n" | netcat sdf.org 70
```

Assuming the host is online, your terminal should now contain a list of directories, as pictured in Figure 1. It might not be the most beautiful output, but is certainly readable.

The numbers preceding each items refer to the type of item that it is. '1' lets the client know that "SDF Member PHLOGOSPHERE" is a subdirectory, while '7' marks "GopherSpace SEARCH Engine" as a gopher search server query.[1]

The string after the item's name, such as "/sdf/faq/" is the "magic string" (actual RFC terminology) that directs the client to that particular item.

Finally, the output contains a line whose sole character is a period. This line indicates that the server has listed everything and will then close the connection.

1 https://en.wikipedia.org/wiki/Veronica_(search_engine)

**Script 2: Navigate to a particular item on the server**

Now that we know, via the client, what is available at the root tier of the hierarchical structure, the client can dig deeper into the gopher hole.

```
echo "/sdf/faq/\n" | netcat sdf.org 70
```

Assuming all went well, you get a response similar to the one listed in Figure 2.

With these two simple commands at your disposal, you can freely traverse the gopherspace! All of these are further subdirectories, but once you hit an item whose identifier is '0', you will have finally arrived at a plain text file.

## The Clients

I used this rudimentary network socket approach to demonstrate how trivial it is to communicate with a gopher server, but a variety of robust clients exits. As the text pages are pure streams of information without style sheets and dynamic behaviors, the content of a gopher site

```
1BASICS  SDF History and UNIX Basics - START HERE\t/sdf/faq/BASICS\tsdf.org\t70
1CHAT    Questions about IRC, ICQ and such.\t/sdf/faq/CHAT\tsdf.org\t70
1DIALUP  SDF's TENEX National Dialup PPP Membership\t/sdf/faq/DIALUP\tsdf.org\t70
1EMAIL   Questions about INTERNETWORKED EMAIL.\t/sdf/faq/EMAIL\tsdf.org\t70
1GAMES   Single and Multiuser GAMES.\t/sdf/faq/GAMES\tsdf.org\t70
1GOPHER  Gopherspace questions and answers.\t/sdf/faq/GOPHER\tsdf.org\t70
1MDNS    Dynamic DNS with mdns.org \t/sdf/faq/MDNS\tsdf.org\t70
1MEMBERS SDF Membership Information and Responsibilities\t/sdf/faq/MEMBERS\tsdf.org\t70
1MISC    Miscellaneous, Odd and Very Interesting Questions  \t/sdf/faq/MISC\tsdf.org\t70
1MOTD    Coding, Journals, Forums and Galleries for UNIX Hackers\t/sdf/faq/MOTD\tsdf.org\t70
1MYSQL   SDF's MySQL database server\t/sdf/faq/MYSQL\tsdf.org\t70
1TEACH   Using SDF to teach UNIX and NET concepts.\t/sdf/faq/TEACH\tsdf.org\t70
1TECHIES Advanced UNIX Topics, UUCP and programming.\t/sdf/faq/TECHIES\tsdf.org\t70
1TWENEX  SDF's Project TENEX Free Software Community\t/sdf/faq/TWENEX\tsdf.org\t70
1UNIX    Questions about UNIX Shells.\t/sdf/faq/UNIX\tsdf.org\t70
1USENET  Newsgroup Questions and News READERS\t/sdf/faq/USENET\tsdf.org\t70
1VHOST   Virtual Hosting and Domain Name Service.\t/sdf/faq/VHOST\tsdf.org\t70
1WEB     Webserver questions and answers.\t/sdf/faq/WEB\tsdf.org\t70
i______________________________________________________________________\t\tnull.host\t1
i              Gophered by Gophernicus/1.5 on NetBSD/amd64 6.1_STABLE\t\tnull.host\t1
.
```

**Figure 2:** Response of the Gopher server

is particularly suited to terminal browsing; the classic terminal browser lynx has supported gopher natively since 1992. Alternatively, plugins are available for Firefox and most other mainstream browsers.[2]

Before continuing with the rest of the document, please take a moment to acquire a client and wander about the gopher sites. The most extensive portal to the gophers that I am aware of is over at Floodgap's server.[3] Be sure and check out the link to GopherVR. Having had a taste of the protocol's simple pre-http delights, no doubt you are eager to set up your own home in the gopherspace. You have a couple options: either find a place in the city (existing server such as sdf.org) or try to make it out on your own in the country (run your own server).

2 http://gopher.floodgap.com/overbite/

3 gopher://gopher.floodgap.com/

## The Servers

There are advantages to each. By setting up your own fresh gopher server, gopher's web presence become further decentralized; it it not ideal to have everything crammed into a single point of failure. However, home-brew servers have a tendency to not last. Fresh holes may find themselves quickly buried when running a 24/7 instance doesn't go as planned. On the other hand, by starting a site on sdf (or elsewhere), you help to strengthen the existing, but small, gopher community. Your content will be much more easily found and more likely to remain online.

Personally, I think the best approach is this: if you already run a server of some sort, then go ahead and add gopher. But if you just want to get content online, simply register on sdf, or wherever else you choose. That said, writing a gopher server is a great project for learning network programming: you learn how to work with cli-

ent-server communications, read and implement specifications, and can have a complete product within a fairly short time. For writing the server in C, you may find Beej's Guide to Network Programming a very helpful reference.

# Recommended Reading

## *'The Craft of Text Editing' by Craig Finseth*

Are you interested in text editing, fellow Lains? I would certainly hope so.

For those wanting to go further in expanding their knowledge, 'The Craft of Text Editing' is an excellent read.

The book is based on the author's bachelor thesis examining the underlying principles of text editors of the time. After 11 years, it was expanded into a book to cover the advances and other changes that had occurred in the meanwhile.

The book pays special attention to Emacs-type editors and the choices made in their design, but a majority of the work applies to all text editors and several different ways of doing things are always discussed for each topic.

The subjects covered include the hardware that programs interact with, the language of the editor and the language exposed to the user, the conceptual structure of the edited text, and various editor implementations. There are chapters on redisplay, the user interface, the command set, and several other interesting topics. Lastly, the book makes for an exhilerating look back on computing and the considerations that were taken into account historically. The bibliography also makes for good reading material for those wanting to delve deeper into the topic.

As this is the first recommended reading section for the first Lainzine, I'll point out that this is not meant to be an exhaustive coverage or a review, but merely a suggestion. I guarantee that any Lain with a healthy interest in the concepts and technology behind their favorite text editor will be thoroughly pleased with this read, especially if they use Emacs.

I look forward to recommending more books to you all.

# Art of the Glitch

*An Existential Analysis of its Products, Processes, and Practitioners*

by Dylan

"The glitch is a wonderful experience of an interruption that shifts an object away from its ordinary form and discourse. For a moment I am shocked, lost and in awe, asking myself what this other utterance is, how was it created. Is it perhaps… a glitch? But once I named it, the momentum – the glitch – is no more…"
—Rosa Menkman (*Art of the Glitch*, 2009)

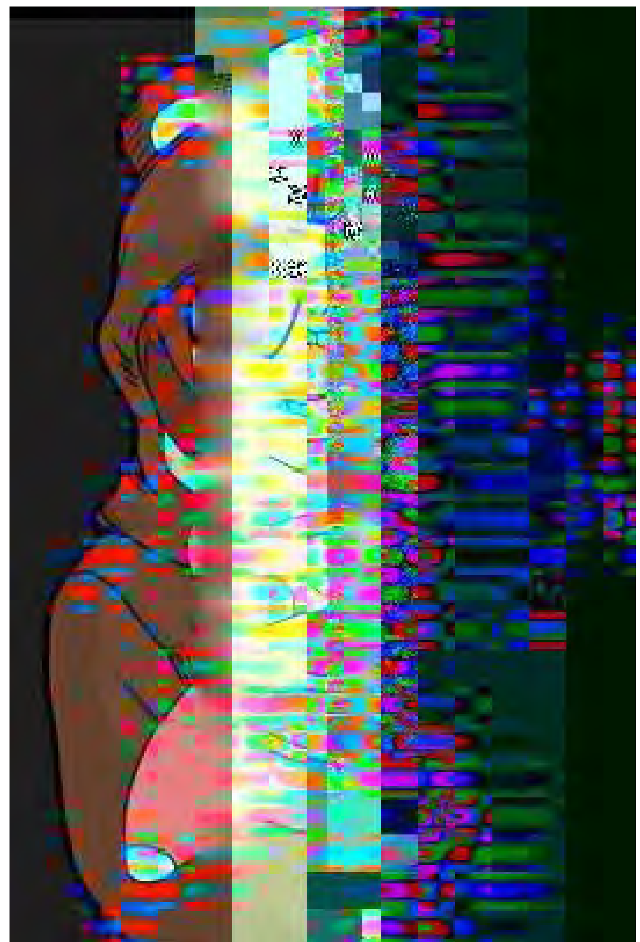
**Abstract:** This paper is an examination of the art style known as glitch art and questions preconceived notions about glitches and art, and asks if they fit that which defines them. It looks at glitch art created by established professionals and how their thoughts and processes relate to the established definition of glitch art and if the definition should be altered to fit the art style. It also compares the effects of popular culture on glitch art.

## Introduction

The analytics of the art world and art itself has always been a grey area. How can one critique something non-objective? When considering new mediums, who is to say who else is an expert? The field of digital art is something relatively new and is one that has many interesting facets and faces. Delving further into one of these sides, the oddly beautiful realm of glitch art is one of these parts of the art world that is difficult to critique. When the purpose of the art form is to destroy something, can one judge its beauty? Does it exist as art at all? To understand these questions, one must look at the roots of the art style. It will be questioned if it can exist as a style at all, given the methods and practices employed are accurate under the term, "glitch."

## Seeking an Understanding

To understand the root causes, underlying structure, or machinations of an issue, theme, or skillset, is to understand it at a deeper level, to lead oneself to a greater proficiency. This is the approach that will be taken moving towards a more thorough understanding of the notion of glitch and glitch art. It is also a means to examine the rationale for desiring a system that can be seen as a destructive one, the reasons for the seeking of chaos, especially when so much of the work that is done by humankind is to bring order to the disordered. Papers are sorted, lists are numbered, and infrastructure is created, all to "correct" this entropy (that seems so grounded in nature). Given these systems, why would this entropy be sought out? What good can come from disorder? This art style that is almost against the nature of human beings will be broken down and examined at its roots and its mysteries revealed.



A glitch could be defined as an undesirable result or outcome; a deviation from an expected conclusion. Is it possible that one might simply tack the idea of "art" on to the end of that definition? An undesirable artistic outcome? This, rather simple, definition leaves a multitude of things susceptible to its grasp. A walk down the street could have a "glitch" when one trips and falls. Certainly, one does not desire to trip and fall when walking anywhere. Perhaps a crack in a glass or a misprint in a paper could be considered a glitch. These things, yet, are different from the glitches that are considered art. A misprinted paper is certainly not art, nor is a broken glass. We might, then, ask ourselves, what is art?

Looking at the vast expanse that is the artistic timeline, we can see an enormous amount of different styles and methods for producing art, and it can be understood that to try and define "art" would be quite a challenging feat. They do, however, have one thing in common: they are all pleasing to look at (or interact with or whatever the intended form of experience may be. Regardless, the idea is that it is enjoyed). Monet's impressionist paintings are indeed beautiful (beauty is subjective, however, and what one might find beautiful, another might find repulsive. This will be an inherent property of art, at least established for the sake of this writing). Returning then, to the broken glass with this new found knowledge, these bright, open, eyes, we find that this broken glass is not beautiful. It lies on the countertop, in pieces, and does not move us to gaze upon it longingly, only to wish that it was not needed to now be cleaned up.

This "glitch art" then, the child of two definitions that were so unceremoniously glued together (perhaps it is fitting, given the word that is being defined), can be understood, at least, for now. One could be typing a paper when for no reason at all, the computer screen glitches. It is in this moment that an artist is born. This daring soul captures the screen to produce a very strange, yet visually appealing image. The image is shared and enjoyed by many; the origin of the glitch art.

What then, of these imitators, these followers, of this new found style? The perceived beauty of these images that are so wonderfully distorted have a movement behind them. There are those that wish to see more of this. However, the methods that were employed to get these results are actually non-existent. The followers of glitch art's only way to create more of this art is to go about their daily lives and keep an eye out for an opportune moment when something might break in their favor. Perhaps they use the same software that the originator used. Perhaps they try to replicate the actions of the previous user. They do, but the time it takes for anything to happen is too lengthy for their tastes, so they abuse the software. They feed the program input that any normal user would not dare.

Here, however, we can already see a problem. Returning to our definition of glitch, an undesirable result or outcome, we can see that these new users that are abusing this software for their own gains are no longer adhering to this definition. The results that are produced from their knowledge of this program and the intent to output errors means that what they are producing is no longer glitch art (in the strictest sense). Before we look to questioning what this new art style really is, let us look to a notable "glitch" artist, Rosa Menkman and her take on the style.

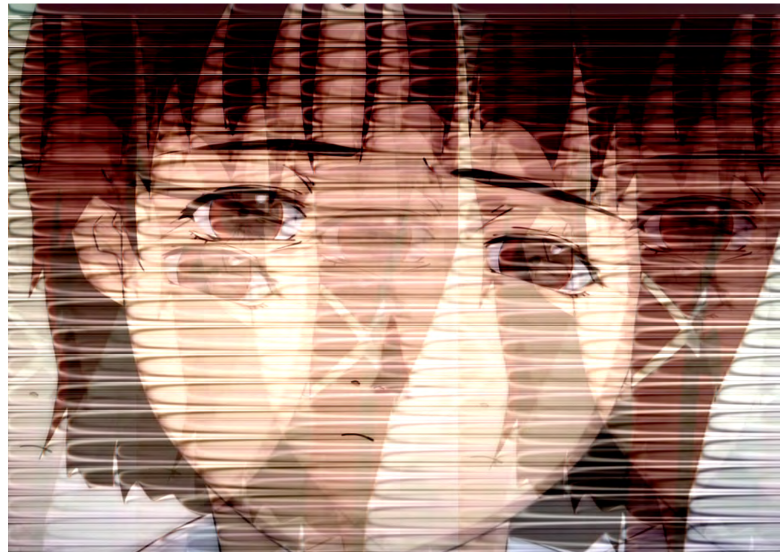
## An Existential Crisis

Rosa Menkman does in fact, associate the definition in a similar way that has been laid out previously, as she states here: "The glitch has no solid form or state through time; it is often perceived as an unexpected and abnormal mode of operandi, a break from (one of) the many flows (of expectations) within a technological system." (Menkman, 2009) Though she scopes it to a tighter view of a technological system,

it fits the idea, regardless. She also notes that, the concept of these errors is altered when they are realized and made into an idea along with the glitch. The "original experience of a rupture" transcended its own self and moved into a new mode of existence entirely. The glitch has ceased to be, and has "become an ephemeral, personal, experience." (Menkman, 2009) She speaks to the very idea that has been presented: that the definition and idea of a glitch are no longer that when they are realized. The mere observation or thought of them has changed them forever. An interesting point is made, however, when she touches upon the act of commercialization of glitch art, whether it be in the form of a script or a "glitching software." She claims that they move away from the process of, what she calls, "creation by destruction" and focus on only the final product. In short, it is about the journey, not the destination, that gives "glitch art" its meaning. "When the glitch becomes domesticated, controlled by a tool, or technology (a human craft) it has lost its enchantment and has become predictable. It is no longer a break from a flow within a technology, or a method to open up the political discourse, but instead a cultivation. For many actors it is no longer a glitch, but a filter that consists of a preset and/or a default: what was once understood as a glitch has now become a new commodity." (Menkman, 2009) This cultivation that she speaks of is inherent in just about any form of media, from music (a local band rising to international fame) to art (the rise of the popularity of the impressionist art style) to social trends (the explosion of the undead and supernatural creatures "genre"). If there are these communities that exist, producing glitch art as an end, rather than a means, they will be subject to scrutiny next.

## Glitch Art in Popular Culture

Rosa Menkman talks about the attempt to define glitch art by media and art historians, giving rise to labels such as "post-digital" and "datamoshing." She provides, what she believes, to be a solution to this problem, but for now, a gaze will be cast upon that subsection of glitch art called "datamoshing". Author Shad Gross describes datamoshing as "a technique whereby the compression of digital video is manipulated as a means of creative expression." Gross, S (2013) His article talks about how digital art (and most all digital media) lacks any form of physical forms but rather, holds within it, a vast number

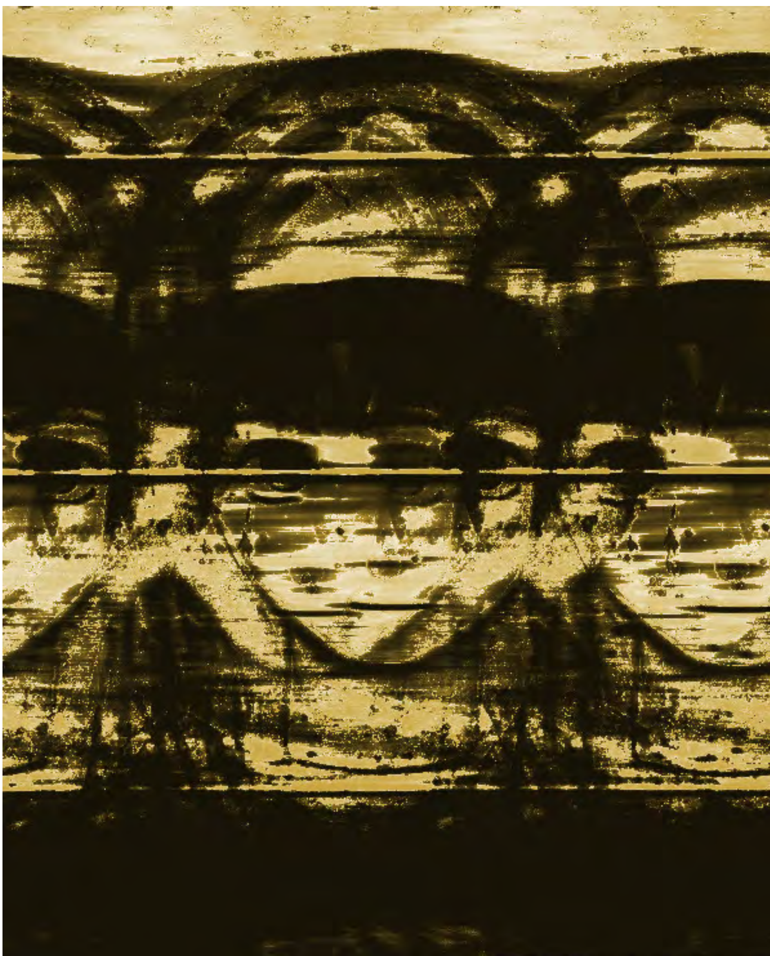


of potential forms. Gross looks to glitch art and datamoshing as a way to reveal these inherent forms of digital media, to focus on the digital aspect of the digital, rather than as a vehicle to a different form entirely. The process of datamoshing involves the editing of frames within the video, more specifically, the frames that relay information about which parts of the screen (pixels) to draw or redraw after something has changed. For many original authors of glitch art and datamoshing, the popular adoption of this style or the adaptation of the style into the mainstream culture, was to lose some of the original luster that it once possessed. Glitching and datamoshing were ways in which people

could transform and alter popular culture, but now it has transformed into the ends, rather than the means, that was mentioned earlier.

Reducing glitch art to this minimal effort approach also loses a key aspect of the idea that it takes a skill and knowledge of the hardware or software to intentionally produce these glitches. Experimentation is a key element, as noted by Funda (2013): "[i]n forcing a visual glitch, there is an element of unpredictability that makes experimentation worthwhile and rewarding." However, the production of scripts and tools to make this kind of art more accessible is what removes the appeal.

Some artists view glitch art as a means of questioning the nature and existence of art itself. Only after hundreds of years of creation of artwork do some people begin to ask, "why?" rather than "how?" One might view this as a driving purpose behind glitch art, to break down the established cultural norms and look at the craft from a new perspective. This style of thinking has been observed by and invited upon by the InterAccess Electronic Media Arts Centre in Toronto. Artists created video games based around the idea of what it means to be and produced some very interesting results. As an example, Terrence (2006) reported this: "… [Ashmore's] Mario Trilogy places the eponymous plumber underwater, in prison or wandering through an empty landscape, with only the viewer/player to move him around aimlessly until the game clock runs out and he dies." This kind of work of art certainly leaves the player with a new outlook on life when the game is finished due to its very relatable message of death and the purpose of existence, or well described by Krapp (2013), the contrast between "playing a game and playing with a game." This mode of thinking can be applied to glitch art, questioning, and perhaps challenging, the very existence of art. With its destructive nature, twisted ideas and indefinable genre-state, it is almost as if glitch art is the death of other art forms.



## Redefining Glitch Art

Given these new insights, one can see the beginnings of glitch art, its methods and practices, and its shortcomings. It is justifiable then, to say that the term "glitch art" is incorrect when it is being described. If glitch art is less about the glitch, and more about the experience, more about the path taken and the surroundings enjoyed on that journey than the end result, then it is to be understood that a new definition is in order. The true glitch art will live on as the far most removal of oneself from the desire to create it, though one could question if

that is even possible.

Regardless, consider the term modal, which is relating to something's mode or manner. It will be used as related to a "mode of action" or simply put, its procedure. Looking back again, we remember the description of the artist who followed in the footsteps of the "original glitch artist." He pushed the limits of the software he was using to create these glitches. It will be termed that this kind of action taken with electronic devices, to produce a desired non-standard result, as abuse. Just as the terms "glitch" and "art" were joined together to create a new meaning, this style of glitch art that desires the malfunctions and errors and actively attempts to produce them, will be termed, "modal abuse." To put it in a more official format, let us say it this way, Modal abuse – the intentional modification or mistreatment of a hardware or software piece to produce an atypical result for the sake of artistic expression.

The newly defined glitch art is, perhaps, a more fitting representation of the values, methods, and ideas that it aims to be. While, however, the views that others (mostly artists) may have on this style may transcend all definitions into a more ideological and emotional realization, to other people, a definition can help create an understanding that is the difference between confusion and clarity. Now, then, begins a new task: to truly remove all elements that could reflect the possibility that any amount of disorder was desired.

## References

Funda S. T. (2012). *Glint: Audiovisual glitches.* Leonardo, 45(3), 296-297. doi:10.1162/LEON_a_00383

Gross, S. (2013). *Glitch, please: Datamoshing as a medium-specific application of digital material.* 175-184. doi:10.1145/2513506.2513525

Krapp, P., & Ebooks Corporation. (2011). *Noise channels: Glitch and error in digital culture.* Minneapolis: University of Minnesota Press.

Menkmen, R. (2009) *Glitch studies manifesto* Retrieved from http://rosa-menkman.blogspot.nl/

Terence D. (2006). *Controller: Artists crack the game code.* Winnipeg: Arts Manitoba Publications Inc.

# Introduction to Cryptography

by peeping_Tom

Cryptography is the practice of secret writing where information is hidden from adversaries. First used by the military, it is now used in every facet of our lives, from simple web browsing to paying bills, talking to your friends, and much more. With examples, I will show you the basics of cryptography and how all encryption is crackable. Cryptography is a wide topic that takes much mathematical ingenuity, so I sliced it into two parts. In this first part, we will explore the origins of cryptography and some simple ciphers you might use with your buddies online or off. This article is mostly theoretical but I encourage you to put your newly acquired knowledge to practice by coding working examples.

Encryption is based on the idea that cracking takes time, and that the more time it takes to crack, the better. Think of bike locks. We know they do not offer perfect security, but we rely on the idea that it will take bad people more time to break them than it will take for someone on the street to stop them. Cryptography follows a similar mindset.

The simplest cipher (method of encryption or decryption) is the *substitution cipher*. As its name implies, this cipher substitutes letters. The most famous substitution cipher is the *Caesar cipher* in which all the letters are replaced by their neighbors in the alphabet a few places over to the right or left. Imagine if A became B, B became C, C became D, and so on until we got to Z, which became A. This is a basic example of the Caesar cipher.

Another interesting substitution cipher is the *Pigpen cipher*. Used primarily by Freemasons, this cipher substitutes letters for shapes and dots.

Every substitution cipher works on the same simple idea, substitution of letters for some other symbols. The way anyone would go about cracking this cipher is through a "checking" process: look for common words in the message's language, then substitute in reverse. For instance, you would look for repeated sets of 3 letters, expect them to be "the", then carry out the necessary substition on the rest of the *ciphertext*, i.e., the encrypted text.

Extend this process to letters, rather than for words, and you get something called "frequency analysis". This method cracks any substitution cipher if you know the language of the *plaintext* (original message). To crack any substitution cipher you start with the "fingerprint" of the plaintext's language, and then look at a frequency analysis of symbols in the ciphertext to see the rules for substitution.

The fingerprint is simply a frequency of occurrence of some letters in a language. For

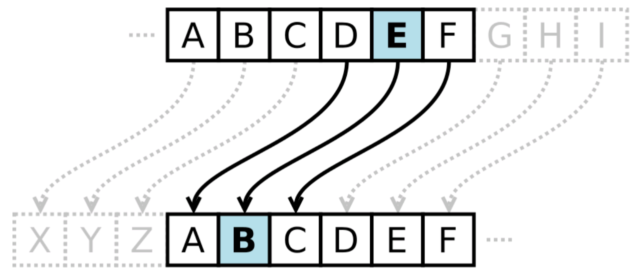



**Figure 1:** The Caesar cipher

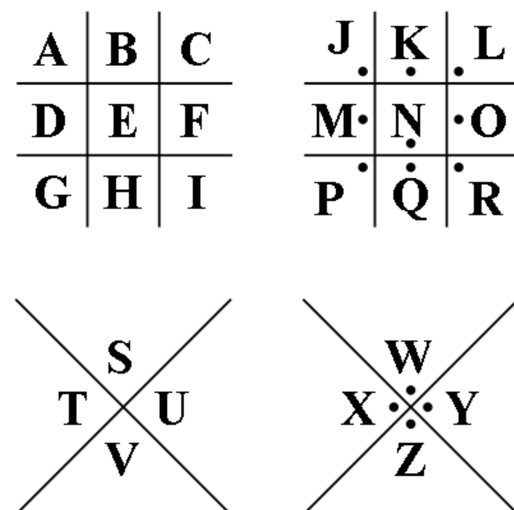



**Figure 2:** The Pigpen cipher

example, if you were to take this wall of text and count the occurrences of every letter, you would find that the most common letters are (by total count): ‘e’, ‘t’, ‘a’, ‘o’, ‘i’, ‘n’, ‘s’, … (just remember ‘ETAOIN SHRDLU’ and that way you remembered the 12 most common English letters). The following image shows the fingerprint of the English language.

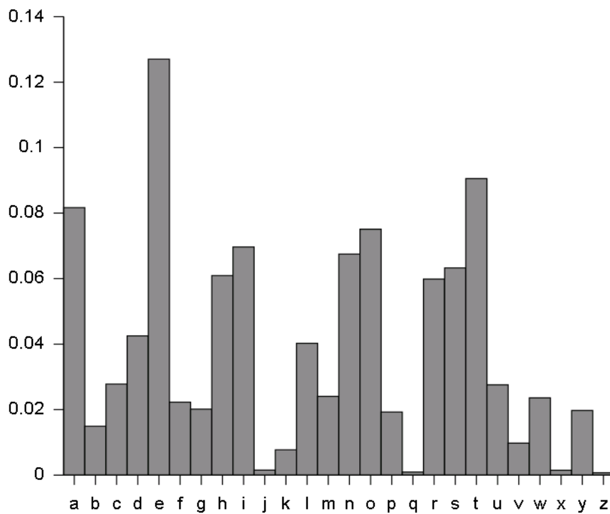



**Figure 3:** The distribution of letters of the alphabet in English by frequency of occurrence

This property is unique to the English language, however all languages have a “common letter use” property. To evade this, people built the *polyalphabetic cipher*.

The polyalphabetic cipher works on the same idea as Caesar cipher, making multiple shift ciphers of the same plaintext. First, a *codeword* is established between the parties. The codeword denotes the shift ciphers, for example the codeword “snake” means that there are 5 shift ciphers each with their own designated shift. The first cipher substitutes the first ‘A’ to occur with ‘S’, ‘B’ with ‘T’, … the second ‘A’ with ‘N’, ‘B’ with ‘O’, and so on. Now that we have 5 shift ciphers with different letter shifts, how do we apply them to the plaintext? We repeat the ciphers in order. The first cipher on the first/sixth/eleventh/… letter, second cipher on the second/seventh/twelfth/seventeen/… letter, and so on. The most famous of polyalphabetic ciphers is the *Vigenère cipher*, which I have just explained.

Take the initial left column to be the codeword, and first row to be the plaintext or ciphertext you want to encrypt/decrypt. Repeat the codeword over the letters of the plaintext/ciphertext without any spaces. For example, try to decrypt “WEBFCSLYZVMYCPN” with codeword “lainchan”. How would a cryptanalyst try to crack this? Since it's just many Caesar ciphers, frequency analysis of the ciphertext is safe bet, right? Well, after he does that he is presented with flat or equally distributed letter frequency. That's because there were several different shifts performed on one text. If he doesn't know the codeword, he can't really crack it. Unless he knows the length of codeword.

Since we know that the length of the codeword is N letters, and that every 1+K×N letter shares the same shift cipher (where K is a number from 0 to ∞ – or at least until K×N becomes longer than ciphertext), we can do frequency analysis on every K×N-th letter. Since we used

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| B | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A |
| C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B |
| D | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
| E | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D |
| F | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E |
| G | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F |
| H | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |
| I | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H |
| J | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I |
| K | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J |
| L | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K |
| M | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L |
| N | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M |
| O | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| P | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| Q | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| R | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
| S | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
| T | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
| U | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| V | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
| W | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| X | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| Y | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
| Z | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y |

**Figure 4:** A polyalphabetic cipher

```
ZDXWWW EJKAWO FECIFE WSNZIP PXPKIY URMZHI JZTLBC YLGDYJ
HTSVTV RRYYEG EXNCGA GGQVRF FHZCIB EWLGGR BZXQDQ DGGIAK
YHJYEQ TDLCQT HZBSIZ IRZDYS RBYJFZ AIRCWI UCVXTW YKPQMK
CKHVEX VXYVCS WOGAAZ OUVVON GCNEVR LMBLYB SBDCDC PCGVJX
QXAUIP PXZQIJ JIUWYH COVWMJ UZOJHL DWHPER UBSRUJ HGAAPR
CRWVHI FRNTQW AJVWRT ACAKRD OZKIIB VIQGBK IJCWHF GTTSSE
EXFIPJ KICASQ IOUQTP ZSGXGH YTYCTI BAZSTN JKMFXI RERYWE
```

**Figure 5:** An example of a one-time pad

"lainchan" as our codeword, we can do frequency analysis on every 8th letter in "WEBFCSLYZVMYCPN" and find out all 8 shift ciphers that way, or in English, we can find the codeword. If we don't know the length of the codeword, and we saw flat distribution of frequency analysis, then we would try frequency analysis of every (1+K×N) th letter, where K ranges from 0 to ∞. Since we don't know N (length of the codeword), we will assume it's 1 letter long, then we will assume its 2 letters long, and so on. We do this until we get the original, readable plaintext. This process can be automated with computers, which easily make it take less time. Longer codewords result in stronger ciphers since the cracker has to assume all possible codewords shorter than the real codeword.

This is only possible because the codeword is repeating, but what if it wasn't repeating? What if we had a codeword as long as the plaintext itself? This is called a *one-time pad*. The codeword "lainzine" can be seen as a list of numbers by which the shift is occurring in each shift cipher ('L' = '11', 'A' = '0', 'I' = '8', and so on). Also note that having a random number sequence for codeword is much more secure than having a word.

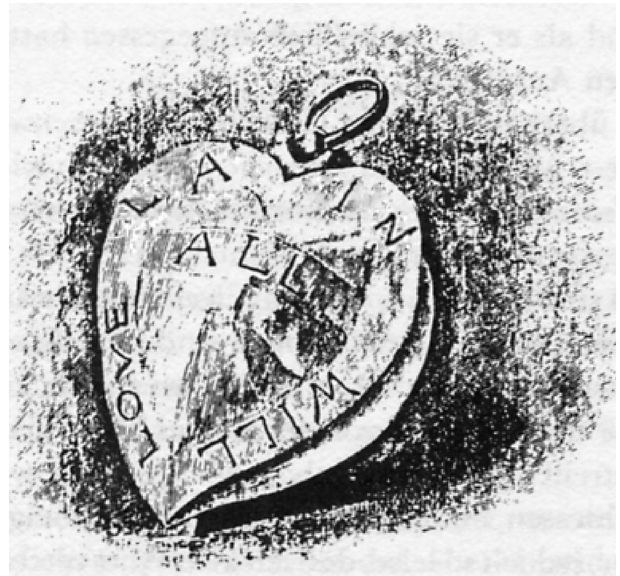
A multiple-shift cipher with random shifts on each letter that is as long as the ciphertext is unbreakable by frequency analysis because the shifts don't repeat. The only way to break this one-time pad is brute force.

Bruteforce, in layman's terms, is trying out every possible combination. Since we know that shift ciphers can shift letters by at most 26 until they repeat (as there are only 26 letters in the alphabet that you can shift to), bruteforcing a Caesar cipher would be simple since the shift is always the same. So you try out all possible shifts (from 1 to 26). For the polyalphabetic cipher and the one-time pad, bruteforcing becomes impossible since there is a shift for every letter. You would need to try all possible letter shifts for all possible letters, meaning that you would need try out all 26 shifts for only one letter. The longer the ciphertext is, the greater amount of shifts you need to try out. Given that's 26 shifts for a letter, and N is the amount of letters in the ciphertext, you would need $26^N$ tries to crack the text. Best explained by a simple example.

Let's say that we encrypted word "lainchan" with one-time pad with codeword of "ZDXWEJKA", making it "KDFJGQKN". There are 26 possible shifts for all letters, making it $26×26×26×26×26×26×26×26 = 26^8 =$ 208,827,064,576 possible combinations of shifts. As the plaintext becomes longer, N becomes larger and so does the amount of possible outcomes $26^N$. If the codeword is truly random, one-time pads are on par with asymmetric encryptions. If you have a friend with whom you can practice encryption, try remembering pi to 300 decimals (it's not that hard, just listen to Hard n' Phirm's Pi song a bit and you already have 160 digits), and then encrypting everything with it (first letter shift by 3, second by 1, third by 4...).

Although the codeword needs to be truly random, this would work fine as long as you and your friend don't reveal the codeword to anyone else.

That's about all we need to know about the origins of cryptography and simple ciphers. In the next article, we will talk about XOR and asymmetric encryption. I hope you liked this short introduction, and that you will have loads of fun with your friends armed with this knowledge.




## Word Search

```
Y T P S Y C H E T J C D U K D E A T H
Z A Y R O R M A E R D D U A B N M S H
S N U V V O L I V I A S A A Q J I O D
V A P Y Y Y H S B N G F I O F S Z R I
D B D R T H V V A N B H N S A U U A I
J I C I I C D X K V S I E N F O K T R
N H T R L S T U I O A L I B I I I S I
V C J O A C J W T L Z H Z E O C A T E
U A I F E P Z I A E C V R S K S R H I
O T Q J R B H R C A C E U M V N I G M
Z H O X C S U N M S Y R M R H O S I A
S R B B O K E X B A B R J I P C U N S
P H G Y A I E N L D E I L O C B P K A
H S L W T S O X R S R K H C H U Z J M
Y O I N U F Y I N O I T R O T S I D N
S R E E T G E F J Z A E E J I V A N B
N S D X Y W K K K C A L B N I N E M M
H U S K J M M A M V V Y R O M E M E Z
E V E F M K K W I R E D L M T M O E G
```

# Where Do I Start?

*A primer to offensive security*

by Hash_Value

Every computer forum you find is flooded with questions about hacking. When lainchan makes it big, we are going to get these questions a lot on /cyb/, so point them to this guide when they ask!

Hacking has a lot of definitions depending on who you ask, but let's go straight to offensive security (which I know is what you all want anyway). First, you will need a basic but strong foundation in programming, networking and operating systems.

To get a strong foundation in programming/scripting/coding you need to *practice*! I don't mean following along with easy college intro to programming course either. You need to build something useful. It doesn't have to be complex, just useful. Just run a web search every time you have a problem. Start off with some simple syntax and then go use that to solve problems and get better and go bigger. A lot of the time, you just need to know simple scripting languages such as python and ruby. If you plan on building something big, use C or C++. Personally, I always recommend this project based approach. That way you get to see how the pieces fit together and you actually end up with something at the end.

Next is networking. You will need to know how packets move, and how routers and switches move them! There are hundreds of tutorials on this, as with programming. Visit Cisco's website and download a little program called packet tracer. This tool simulates creating a network, although it doesn't have all the security measures and you don't get the physical hands on experience. It's one of the best tools available, though, and I highly recommend playing with it! If nothing else, you can buy an old Cisco router on the web for cheap and it will still be good, trust me. Businesses are usually slow to change their routers and switches. In addition to all of this, you also need a strong understanding of ports and TCP handshake.

Moving on to operating systems, the big three are Windows, OS X, and Linux. Back in the early days, if you wanted to do any hacking it had to be on linux. That isn't necessarily true nowadays, as many tools have been ported to Windows and Mac. However, Linux is still your best bet. Regardless of which operating system you choose to work with, you have to learn both Windows and Linux. Why? Each holds about half of the server market, and if you plan on breaking in to either you have to learn both, including Linux and Windows server versions.

Let's not forget web applications! Since the arrival of web 2.0, HTML5 web applications have become more and more popular. You will need to know HTML (HyperText Markup Language) and Javascript (a client-side scripting language). You also have to know how websites work, and how databases interact with websites. You can start by installing LAMP (Linux-Apache-MySQL-PHP5) in a virtual machine (which we will discuss later).

Okay, now that you have a rough idea of what you should be learning, time for the good stuff!

First the age old question: "Should I download a hacking specific distro?" The truth of the matter is, you don't really need too. You can download all the tools into any distro of Linux from the web, so the only real reason you would want to download a pentesting distro is the fact all the tools are the there and you don't have to waste time finding all the dependencies for metasploit. So my advice for you is this: Download a pentester, but don't install it as your main distro. Yes there are some distros that can serve both purposes, but I wouldn't recommend using them that way.

Alright, you got your weapons and the skill to wield them, now all you need is a dojo. "But why can't I attack Shitbook or [insert other mainstream site here]?" I'm glad you asked! It's because that would be illegal, and you would get caught. I recommend visualizing everything, because it's cheaper, if you fauurrk something up you can revert back to a snapshot, and it's cheaper. Now, many recommend VMware. It costs a lot of money, and I won't lie, VMware is really well made. However, Virtual Box does the same thing at a price you can't beat, *free* (as in both free beer and freedom).

Well, this was just a starter's guide, rough and dirty. I do full time college and work, so opportunities to shitpost on the web are few and far between. I wanted to do a starter's guide on BeEF (browser exploitation framework) as well, but it was going to take longer than I first suspected. I'll try for that next time, but no promises. If you want to talk to me, feel free to give me something to research; I love to learn new things. Chances are I'm lurking in #lainchan on freenode.

## FAQ

Q: I'm just a poor boy who has to steal wifi.
A: Get Reaver or, if you're using Windows, Cain.

Q: Kali Linux?
A: Only liveboot, don't install it.

Q: Do I have to learn to code?
A: Tools can only go so far Anon.

Q: How do i hack Facebook?
A: Look into 'SET'.

Q: What are the tools i need to hack X?
A:

- Portscanning: nmap
- Password cracking: hydra
- Web app: Burbsuite is popular, but I like w3af
- General use: Metasploit is an all-in-one tool which is pretty much perfect for everything
- Social engineering: payphone/SET

Q: Where can I get hands-on experience?
A: Places to practice hacking:

- https://www.hackthissite.org
- https://www.hackthis.co.uk
- https://pentesterlab.com
- https://hack.me
- https://community.rapid7.com/docs/DOC-1875

Networking help and tutorials:

- https://www.netacad.com/web/about-us/cisco-packet-tracer
- https://www.youtube.com/user/danscourses

Very good resources and general use education:

- http://pastebin.com/cRYvK4jb
- https://udacity.com
- https://www.youtube.com/user/thenewboston

Exploit development:

- http://www.myne-us.com/2010/08/from-0x90-to-0x4c454554-journey-into.html

# FreeBSD Guide for Newbs and Dummies

by gh0st_

### Getting the release

First go to freebsd.org and pick the proper architecture. You will see the option of x86 and x64. To find out which one to pick you generally need to go by how much RAM you have. If you have under 4gigs of RAM the go with x86, 4gigs or more then go with x64.

### Knowledge requirements

- Unix basics – ie. directory setups and concepts.
- Shell, i.e. Bash, CSH, KSH, etc. – You need to have a basic grasp on this.
- Unix concepts – i.e., everything is a file, shit like this.

### Install

If you have never messed around with Unix like OS's before then you probably should go to Linux first, pick something like Ubuntu and learn the Unix basics (also, the installer on Ubuntu is generally more friendly).

**Note:** If you don't want to, then go and grab the FreeBSD handbook and read through it.

### First boot

Press Enter and go through the basic installer (it is pretty fucking easy). If you can't navigate through that then you shouldn't be reading this.

### Part 1: Wireless

If you are on a desktop, you don't have to worry about this. However, if you are on a laptop you, will be confronted with a wireless connection screen inside the installer. Just find your wireless network id and click it, then enter your password and continue. After the install, you should auto connect to your home wireless, but if you go somewhere else it won't do this (obviously). So follow these simple steps:

1. "`ifconfig`" – this will print out your card name and its alias. Most of the time it is just "`wlan0`".
2. "`ifconfig wlan0 scan`" – brings up network ids
3. "`ifconfig wlan0 nwid 'FooBar Network' wpakey "FO0Bar_password"`"
4. If your network is unencrypted use the "`-wep`" or "`-wpa`" option where the `wpakey`" would normally be
5. "`dhclient wlan0`" – this will setup all the DHCP shit
6. to test run "`ping google.com`"

### Part 2: Users

You will come to a point where you will be asked if you want to create a user. Select yes and go thru the basics until you get to the part where it asks you if you want to add this person to any groups. In this you want to type "wheel" this will be used later when adding you to the list of sudoers

### Part 3: Post install

After you are done you will reboot your computer and take out the medium which you used to install FreeBSD with. The computer should boot normally and you will be confronted with a basic CLI login screen (we will be adding a graphical login manager later). Login as root (*only for this section*, doing this routinely is a rootkit waiting to happen).

Once you are logged in as the root user you will do the following:

1. We are assuming you are connected to your home network; run "`ping google.com`" to test this.
2. Now we are going to install a few basic packages. You will be doing all this as root for now but later you can finally use the user you created. Run the "`pkg`" command

and go through the dialouge. Next, run "`pkg install sudo`".

3. Run "`visudo`" and once you see text press the 'i' key to start editing text. Scroll down until you see "Uncomment to allow members of the wheel to" blah blah blah, press the Del key to uncomment. Next press the Esc key then press Shift "`:wq`". Having done that will allow the user you created to have access to sudoers.

We will now be setting up our desktop environment and login manager as well as a shit ton of other stuff to get you started. First run "`pkg install nano`". Once that installs you want to install basic GUI packages such as a desktop environment

I will later write a file on how to customize xfce, but for now we are just setting up.

**Part 4: Packages and settings up the desktop environment and login manager**

Run "`pkg install xorg xfce4-session firefox hexchat epdfview`". Once that finally installs run pkg install "`xfce4-wm-themes slim xfce4-mixer`" (or "`xfce4-pulseaudio-daemon`"). Once you install this, you can poke around and configure xfce to your liking but that is for a different zine.

4. Log out of root and login as the user you created. You should have basic permissions now that you are a sudoer. Next you want to configure things so that xfce will work, run "`sudo nano /etc/rc.conf`" and add the following to the file:

```
dbus_enable="YES"
hald_enable="YES"
slim_enable="YES"
moused_enable="YES"
powerd_enable="YES" (if you are on a
   laptop only)
```

Next, press Ctrl-O and press Enter.

Now your xfce should be able to function normally. Next run "`cd ~`" to get to your home directory and run "`nano .xinitrc`". Add the following to the file:

```
exec /usr/local/bin/startxfce4
```

This will make it so when you boot up and login with SLIM or the CLI login screen you will be able to boot into xfce. (Note that if you aren't using a graphical login manager you will need to run "`startx`" to boot into the desktop.)

That's all for now folks. Stay tuned for my next article in the next issue of lainzine. I will be covering ricing your xfce desktop.

**Note:** This will have some spelling errors and maybe even some gaps in the guide. Don't bitch about it, figure it out. If you can't, there are tons of forums, discussion boards, mailing lists and IRC channels dedicated to Unix full of helpful, enthusiastic people.

# Youtube Proxy[1]

The following script runs on vanilla Node.js and requires an executable of youtube-dl. If it's not located in the same directory as the script, change the line

```
var job = spawn('./youtube-dl', args);
```

to something like

```
var job = spawn('~/path/to/youtube-dl', args);
```

#### USAGE EXAMPLES

Download a video in MP4 from YouTube:

```
http://example.com:8567/https://www.youtube.com/watch?v=XXXXXXX
```

Download a video from YouTube and extract audio in M4A format:

```
http://example.com:8567/https://www.youtube.com/watch?v=XXXXXXX.m4a
```

Currently, the headers it sets are designed for downloading. You can change them so that it plays in your browser, but I mainly wanted it for downloading videos on my phone and Firefox on Android wouldn't download it without them.

Also, be aware that it needs a “downloads” directory that is not created automatically.

```
var spawn = require('child_process').spawn,
    fs    = require('fs'),
    http  = require('http'),
    mime  = require('mime');

var server = http.createServer(function(req, res) {
    var m      = req.url.match(/\/(.*)\.(.*?)/),
        format = m[2],
        url    = m[1];

    if (!/^.*\.[a-z0-9]{3,5}$/.test(req.url)) {
        format = 'mp4';
        url    = req.url.match(/\/(.*)/)[1];
    }

    var file     = url.replace(/[\/:?!&=\.]/g, '') + '.' + format,
        mimeType = mime.lookup(file),
        args     = ['-o', 'downloads/' + file];
```

1 Original comment on /tech/: https://lainchan.org/tech/res/5644.html#5658, original source code at http://pastebin.com/raw.php?i=amxBtmmj

```
    if (['m4a', 'mp3', 'opus'].indexOf(format) !== -1) {
        args = args.concat(['-x', '--audio-format']);
    } else {
        args.push('-f');
    }
    args.push(format);
    args.push(url);

    console.log(args.join(' '));
    var job = spawn('./youtube-dl', args);
    job.on('close', function(code, signal) {
        var fileStream = fs.createReadStream('downloads/' + file);
        pipeReadstream(req, res, fileStream, mimeType, file, function(err) {
            console.log('error: ' + err);
        });
    });

    job.stdout.on('data', function(data) {
        console.log('stdout: ' + data);
    });

    job.stderr.on('data', function(data) {
        console.log('stderr: ' + data);
    });
});

server.listen(8567);

// Pipe some stream as HTTP response
function pipeReadstream(req, res, readStream, mimeType, filename, cb) {
    var headWritten = false;

    readStream.on('data', function(data) {
        if (!headWritten) {
            res.writeHead(200, {
                'Content-Disposition': 'attachment; filename=' + filename,
                'Content-Type': mimeType
            });
            headWritten = true;
        }
```

```
        var flushed = res.write(data);
        // Pause the read stream when the write stream gets saturated
        if (!flushed) {
            readStream.pause();
        }
    });

    res.on('drain', function() {
        // Resume the read stream when the write stream gets hungry
        readStream.resume();
    });

    readStream.on('end',   function()    { res.end(); });
    readStream.on('error', function(err) { cb(err); });
}
```

# Structure-based ASCII Art

Xuemiao Xu* Linling Zhang† Tien-Tsin Wong‡
The Chinese University of Hong Kong

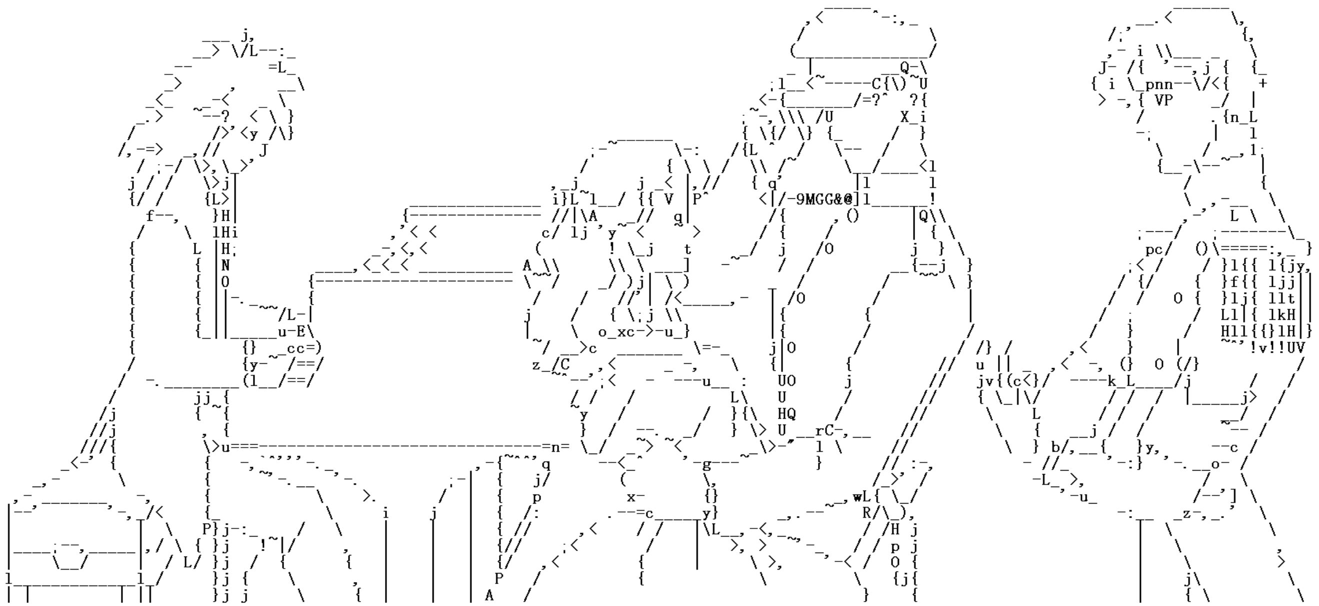


Figure 1: Structure-based ASCII art generated by our method (the input is "banquet" of Figure 18). Characters were chosen from the set of 95 printable ASCII characters.

## Abstract

The wide availability and popularity of text-based communication channels encourage the usage of ASCII art in representing images. Existing tone-based ASCII art generation methods lead to halftone-like results and require high text resolution for display, as higher text resolution offers more tone variety. This paper presents a novel method to generate *structure-based* ASCII art that is currently mostly created by hand. It approximates the major line structure of the reference image content with the shape of characters. Representing the unlimited image content with the extremely limited shapes and restrictive placement of characters makes this problem challenging. Most existing shape similarity metrics either fail to address the misalignment in real-world scenarios, or are unable to account for the differences in position, orientation and scaling. Our key contribution is a novel *alignment-insensitive shape similarity (AISS) metric* that tolerates misalignment of shapes while accounting for the differences in position, orientation and scaling. Together with the constrained deformation approach, we formulate the ASCII art generation as an optimization that minimizes *shape dissimilarity* and *deformation*. Convincing results and user study are shown to demonstrate its effectiveness.

**Keywords:** ASCII art, shape similarity

*e-mail: xmxu@cse.cuhk.edu.hk
†e-mail: llzhang@cse.cuhk.edu.hk
‡e-mail: ttwong@cse.cuhk.edu.hk

## 1 Introduction

ASCII art is a technique of composing pictures with printable text characters [Wikipedia 2009]. It stemmed from the inability of graphical presentation on early computers. Hence text characters are used in place of graphics. Even with the wide availability of digital images and graphics nowadays, ASCII art remains popular due to the enormous growth of text-based communication channels over the Internet and mobile communication networks, such as instant messenger systems, Usenet news, discussion forums, email and short message services (SMS). In addition, ASCII art has already evolved into a popular art form in cyberspace.

ASCII art can be roughly divided into two major styles, tone-based and structure-based. While tone-based ASCII art maintains the intensity distribution of the reference image (Figure 2(b)), structure-based ASCII art captures the major structure of the image content (Figure 2(c)). In general, tone-based ASCII art requires a much higher text resolution to represent the same content than the

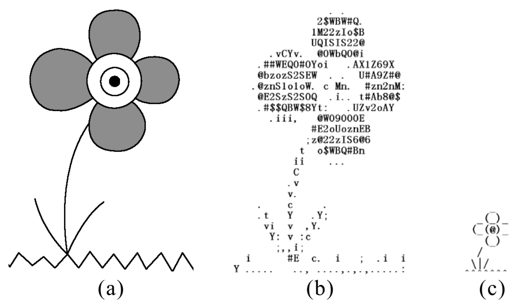


Figure 2: ASCII art. (a) A reference image. (b) Tone-based ASCII art generated by the program PicText, requiring the text resolution 30×29 in order to depict the content, though not very clearly. (c) Structure-based ASCII art manually designed by an artist, with a significant lower text resolution of 8×7.

structure-based one, as the high text resolution is required for producing sufficient tone variety. On the other hand, structure-based ASCII art utilizes the shape of characters to approximate the image structure (Figure 2(c)), without mechanically following the pixel values. To the extreme, smileys, such as `:)` and `:(`, are the simplest examples of structure-based ASCII art.

Existing computational methods can only handle tone-based ASCII art, as its generation can be regarded as a dithering problem with characters [Ulichney ]. O'Grady and Rickard [2008] improved such dithering process by reducing the mismatches bewteen character pixels and the reference image pixels. Nevertheless, high text resolution is still required for a clear depiction. Note that ASCII art gradually loses its stylishness (and approaches to standard halftone images) as its text resolution increases. In addition, as the text screens of mobile devices are limited, the character-saving structure-based ASCII art is more stylish and practical for commercial usage such as text-based advertisement. However, satisfactory structure-based ASCII art is mostly created by hand. The major challenge is the inability to depict the unlimited image content with the limited character shapes and the restrictive placement of characters over the character grid.

To increase the chance of matching appropriate characters, artists tolerate the misalignment between the characters and the reference image structure (Figure 3(b)), and even intelligently deform the reference image (Figure 3(c)). In fact, shape matching in ASCII art application is a general pattern recognition problem. In real-world applications, such as optical character recognition (OCR) and ASCII art, we need a metric to *tolerate misalignment* and also *account for the differences in transformation* (translation, orientation and scaling). For instance, in recognizing the characters "o" and "$^o$" during the OCR, both scaling and translation count; while in recognizing characters "6" and "9", the orientation counts. Unfortunately, existing shape similarity metrics are either alignment-sensitive [Wang et al. 2004] or transformation-invariant [Mori et al. 2005; Belongie et al. 2002; Arkin et al. 1991], and hence not applicable.

In this paper, we propose a novel method to generate structure-based ASCII art to capture the major structure of the reference image. Inspired by the two matching strategies employed by ASCII artists, our method matches characters based on a novel *alignment-insensitive shape similarity metric* and allows a constrained deformation of the reference image to increase the chance of character matching. The proposed similarity metric tolerates the misalignment while it accounts for the differences in transformation. Given an input and a target text resolution, we formulate the ASCII art generation as an optimization by minimizing the shape dissimilarity and deformation. We demonstrate its effectiveness by several convincing examples and a user study. Figure 1 shows the result automatically obtained by our method.

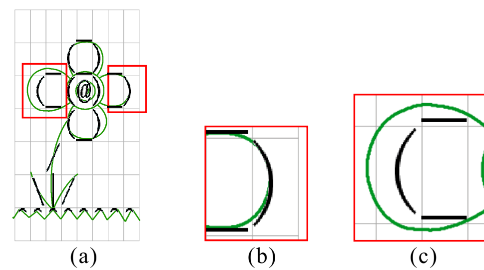


Figure 3: (a) By inspecting the overlapping image between the edge map of the reference image (Figure 2(a)) and the structured-based ASCII art (Figure 2(c)), one can identify the two matching strategies employed by ASCII artists: (b) misalignment is tolerated; (c) the reference image is deformed to increase the chance of matching.

## 2 Related Work

As a culture in the cyberspace, the best references of ASCII art can be found online. There is collaboratively prepared frequently asked questions (FAQ) for Usenet newsgroup `alt.ascii-art` [CJRandall 2003], which keeps track of the update information and resources related to ASCII art. Other sources of reference are online tutorials written by individual enthusiasts [Wakenshaw 2000; Crawford 1994; Au 1995]. To produce ASCII art, one can type it using a standard text editor. It is not as intuitive as painting, however. Enthusiasts developed interactive *painting* software [Davis 1986; Gebhard 2009] to allow users to directly *paint* the characters via a painting metaphor.

Besides the interactive tools, there are attempts to automatically convert images into ASCII art [Klose and McIntosh 2000; DeFusco 2007; O'Grady and Rickard 2008]. However, they can only generate tone-based ASCII art, as it can be regarded as a dithering process. The major academic study is in the area of halftoning [Ulichney ; Bayer 1973; Floyd and Steinberg 1974]. O'Grady and Rickard [2008] tailor-made a method for tone-based ASCII art by minimizing the difference between the characters and the reference image in a pixel-by-pixel manner. However, all these methods cannot be extended to generate structure-based ASCII art due to their inability to allow misalignment and deformation. In this paper, we focus on the generation of structure-based ASCII art as it depicts a clearer picture within a smaller text space. Its generation can no longer be regarded as a dithering process. Instead, the shape similarity plays a major role in its generation. 3D collage [Gal et al. 2007] relies on shape matching to aggregate smaller objects to form a large compound one. While transformation invariance is needed during collaging, our character matching must be transformation-aware and with restrictive placement.

## 3 Overview

An overview of our structure-based ASCII art generation is shown in Figure 4. The basic input is a vector graphics containing only polylines. A raster image can be converted to vector via vectorization. As the limited shapes and restrictive placement of text characters may not be able to represent unlimited image content, ASCII artists slightly deform the input to increase the chance of character matching. So we mimic such deformation during optimization by iteratively adjusting the vertex positions of the input polylines. Given the vector-based line art, we rasterize it and divide the raster image into grid cells. Each cell is then best-matched with a character based on the proposed alignment-insensitive shape similarity metric (Section 4). This completes one iteration of optimization, and the objective value, which composes of the *deformation of the vectorized picture* (Section 5) and the *dissimilarity* between the

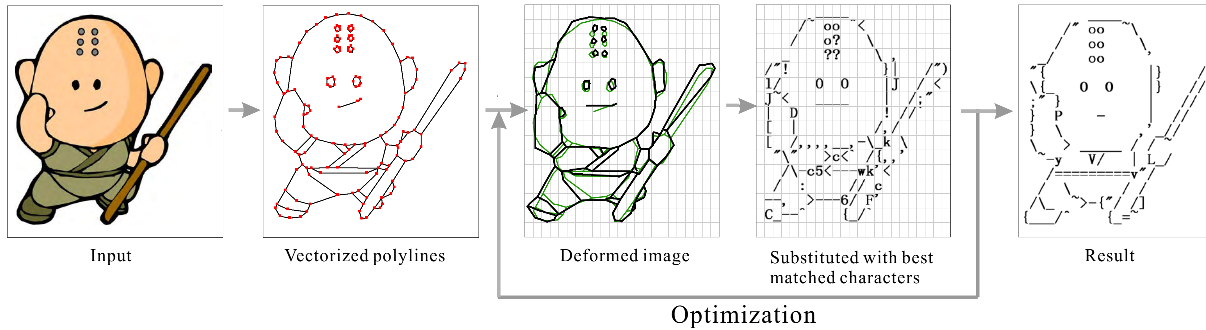



Figure 4: The overview of our framework.

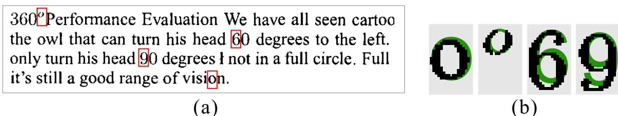



Figure 5: Real-world applications, like OCR and ASCII art, require a similarity metric to account for scaling, translation and orientation, as well as tolerate misalignment. (a) A scanned image for OCR input. (b) Misalignment with ideal characters (in green) exists.

characters and the deformed picture, can be computed. In the next iteration, we adjust the vertex positions of the vector-based line art with a simulated annealing strategy (detailed in Section 5). Since the line art is changed, the above rasterization-and-AISS-matching process is repeated to obtain a new set of best-matched characters. Such deformation-and-matching process continues until the objective value is minimized.

Before the optimization, we need to prepare the input and the characters. Since character fonts may have varying thicknesses and widths, we simplify the problem by ignoring font thickness (via centerline extraction) and handling only fixed-width character fonts. We further vectorize the characters and represent them with polylines. In order to focus only on the shapes during matching, both the input polylines and the characters are rasterized with the same line thickness (one pixel-width in our system). Note that the characters are only rasterized once as they can be repeatedly used. Before each optimization step, the input polylines are rasterized according to the target text resolution, $R_w \times R_h$, where $R_w$ and $R_h$ are the maximum number of characters along the horizontal and vertical directions respectively. As the aspect ratio of our characters, $\alpha = T_h/T_w$, is fixed, the text resolution can be solely determined by a single variable $R_w$, as $R_h = \lceil H/(\alpha \lceil W/R_w \rceil) \rceil$, where $T_w$ and $T_h$ are the width and height of a rasterized character image in the unit of pixels respectively. $W$ and $H$ are the width and height of the input image. Hence, the input polylines are scaled and rasterized to a domain of $T_w R_w \times T_h R_h$. Furthermore, since the vector-based input is scalable ($W$ and $H$ can be scaled up or down), users may opt for allowing the system to determine the optimal text resolution ($R_w \times R_h$) by choosing the minimized objective values among results of multiple resolutions, as our objective function is normalized to the text resolution.

## 4 Alignment-Insensitive Shape Similarity

The key to best-match the content in a grid cell with a character is the shape similarity metric. It should tolerate misalignment and, simultaneously, account for the differences in transformation such as, position, orientation and scaling. Existing shape similarity metrics can be roughly classified into two extreme categories, alignment-sensitive metrics and transformation-invariant metrics.

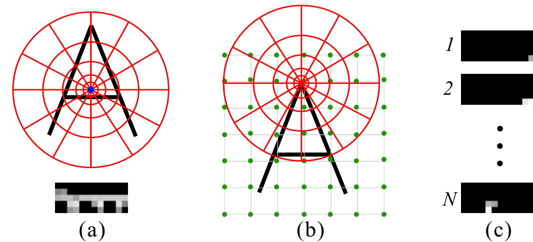



Figure 6: Alignment-insensitive shape similarity. (a) A log-polar diagram to quantify the letter "A" with the corresponding histogram underneath. Its row and column correspond to the angular and radial dimensions of the log-polar diagram respectively. (b) $N$ points are regularly sampled in a grid layout, each with a log-polar diagram. (c) The corresponding log-polar histograms.

Peak signal-to-noise ratio (PSNR) or mean-squared error (MSE), and the well-known structural similarity index (SSIM) [Wang et al. 2004] belong to the former category. Their similarity values drop significantly when two equal images are slightly misaligned during the comparison. On the other hand, the transformation-invariant metrics are designed to be invariant to translation, orientation and scaling. These metrics include shape context descriptor [Mori et al. 2005; Belongie et al. 2002], Fourier descriptor [Zahn and Roskies 1972], skeleton-based shape matching [Sundar et al. 2003; Goh 2008; Torsello and Hancock 2004], curvature-based shape matching [Cohen et al. 1992; Milios 1989], and polygonal shape matching [Arkin et al. 1991]. In our case, the transformation matters. Hence, no existing work is suitable for our application.

In fact, the above metric requirement is not only dedicated to our application, but applicable for real-world applications of pattern recognition and image analysis, such as OCR. For example, Figure 5(a) shows a scanned image ready for OCR. The characters "°", "o", "6" and "9" are magnified in Figure 5(b) for better visualization. It is not surprising that the scanned character images (in black) may be slightly misaligned to the ideal characters (in green) no matter how perfect the global registration is. Hence, an alignment insensitive shape similarity metric is essential. Besides the misalignment, the transformation difference has to be accounted for in OCR as well. Characters "o" and "°" have the similar shapes, but are different in position and scaling. Characters "9" and "6" also share the same shape but with a difference in orientation. In other words, the shape information alone is not sufficient for recognition, since position, orientation and scaling have their own special meanings. Therefore, the desired metric must also account for position, orientation, scaling, as well as the shape information.

**Misalignment Tolerance** Misalignment is, in essence, a small-scale transformation. To tolerate misalignment, a histogram of a log-polar diagram [Mori et al. 2005] is used as the basic building

block of our shape descriptor (Figure 6(a)). This log-polar histogram measures the shape feature in a local neighborhood, covered by a log-polar window. Its bins uniformly partition the local neighborhood in log-polar space. For each bin, the grayness of the shape is accumulated and used as one component in the histogram. As the bins are uniform in log-polar space, the histogram is more sensitive to the positions of nearby points than to those farther away. Moreover, since only the sum of pixels within the same bin is relevant, it is inherently insensitive to small shape perturbations, which leads to its misalignment tolerance nature. In other words, the degree of misalignment tolerance is implicitly defined in the log-polar diagram. During the pixel summation, black pixel has a grayness of 1 while the white one is 0. The bin value $h(k)$ of the $k$-th bin is computed as $h(k) = \sum_{(q-p)\in \text{bin}(k)} \mathbf{I}(q)$, where $q$ is the position of the current pixel; $(q - p)$ is the relative position to the center of the log-polar window, $p$; $\mathbf{I}(q)$ returns the grayness at position $q$. The lower sub-image in Figure 6(a) visualizes the feature vector $h$ with respect to $p$ (the blue dot).

**Transformation Awareness** Unlike the original transformation-invariance scheme in [Mori et al. 2005], we propose a novel sampling layout of log-polar diagrams in order to account for the transformation difference. The log-polar histogram can natively account for orientation. The bin values change as the content rotates. To account for scaling, all log-polar histograms share the same scale. To account for translation (or position), $N$ points are regularly sampled over the image in a grid layout (Figure 6(b)). Both the reference image in a cell and the character image are sampled with the same sampling pattern. For each sample point, a log-polar histogram is measured. The feature vectors (histograms) of the sample points are then concatenated to describe the shape, as shown in Figure 6(c). The shape similarity between two shapes, $S$ and $S'$, is measured by comparing their feature vectors in a point-by-point basis, given by

$$D_{\text{AISS}}(S, S') = \frac{1}{M} \sum_{i \in N} ||\mathbf{h}_i - \mathbf{h}'_i||, \qquad (1)$$

where $\mathbf{h}_i$ ($\mathbf{h}'_i$) is the feature vector of the $i$-th sample point on $S$ ($S'$); $M = (n+n')$ is the normalization factor and $n$ ($n'$) is the total grayness of the shape $S$ ($S'$). This normalization factor counteracts the influence of absolute grayness.

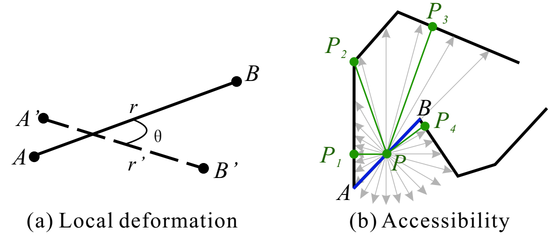
In all the experiments, histograms were empirically constructed with 5 bins along the radial axis in log space, and 12 bins along the angular axis. The radius of the coverage is selected to be about half of the shorter side of a character. The number of sample points, $N$, equals $(T_w/2) \times (T_h/2)$. To suppress aliasing due to the discrete nature of bins, the image is filtered by a Gaussian kernel of size 7×7 before measuring the shape feature.

**Comparison to Existing Metrics** We evaluate the metric by comparing it to three commonly used metrics, including the classical shape context (a translation- and scale- invariant metric), SSIM (an alignment-sensitive, structure similarity metric), and RMSE (root mean squared error) after blurring. For the last metric, RMSE is measured after blurring the compared images by a Gaussian kernel of 7×7, as one may argue that our metric is similar to RMSE after blurring the images.

The effectiveness of our metric is demonstrated in Figure 7, in which we query four different shapes (the first column). For each metric, the best-matched character is determined from a set of 95 printable ASCII characters. From the matching results, shape context over-emphasizes the shape and ignores the position (as demonstrated by queries 2 to 4). On the other hand, the alignment-sensitive nature of SSIM and RMSE drives them to maximize the overlapping area between the query image and the character, while paying less attention to the shape (demonstrated by queries 1 and 3). In contrast, our method strives for a balance between shape and position in all results. The query of a long center-aligned horizontal line (query 4) demonstrates the advantage of our metric. Shape context maximizes shape similarity, ignores large displacement, and chooses the longer underscore character "_" to match the long line. SSIM and RMSE match the shape with an equal sign "=" because its lower line overlaps with the query image. Our method pays attention to the shape (a single line), tolerates a slight misalignment, and chooses a shorter hyphen "−" as the best match.

| Query | Our metric | Shape context | SSIM | RMSE (after blurring) |
|---|---|---|---|---|
| (1) | L | L | P | _ |
| (2) | } | \| | P | } |
| (3) | ~ | _ | ` | ` |
| (4) | − | _ | = | = |

Figure 7: Comparison of four shape similarity metrics. From left to right: our metric, shape context, SSIM, and RMSE-after-blurring.




(a) Local deformation (b) Accessibility

Figure 8: Deformation metric

## 5 Optimization

**Deformation Metric** To raise the chance of matching characters, ASCII artists intelligently deform the reference image. We mimic such deformation during our optimization. We deform the reference image by adjusting the vertex positions of the vectorized polylines. However, unconstrained deformation may destroy the global structure of the input. We designed a metric to quantify and minimize the deformation values during the optimization process. This consists of two terms, *local deformation constraint* and *accessibility constraint*.

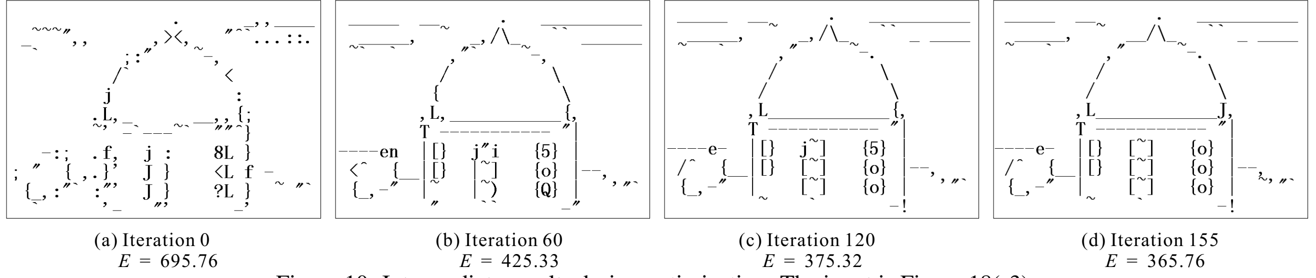
*Local Deformation Constraint* The first term measures the local deformation of a line segment, in terms of orientation and scaling. Consider the original line segment $AB$ as deformed to $A'B'$ in Figure 8(a). As we allow global translation during the deformation, the local deformation of line segment $AB$ is measured in a relative sense, as follows,

$$D_{\text{local}}(AB) = \max \{V_\theta(AB), V_r(AB)\}, \qquad (2)$$

where $V_\theta(AB) = \exp(\lambda_1 \theta)$, and

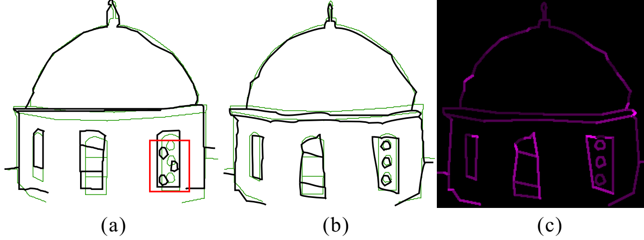
$$V_r(AB) = \max \left\{ \exp(\lambda_2 |r' - r|), \exp\left( \frac{\lambda_3 \max\{r, r'\}}{\min\{r, r'\}} \right) \right\},$$

$\theta \in [0, \pi]$ is the angle between the original and the deformed line segments. $r$ and $r'$ denote the lengths of the original and deformed



(a) Iteration 0 $E = 695.76$ (b) Iteration 60 $E = 425.33$ (c) Iteration 120 $E = 375.32$ (d) Iteration 155 $E = 365.76$

Figure 10: Intermediate results during optimization. The input is Figure 18(s3).



(a) (b) (c)

Figure 9: The green and black lines indicate the original and deformed polylines respectively. The input is Figure 18(s3). (a) With the local deformation constraint alone, the drift of circular windows cannot be avoided. (b) With local and accessibility constraints, the drift can be avoided. (c) Visualization of the deformation value of each line segment in (b). For visualization purpose, the deformation values are non-linearly mapped.

line segments. Parameters $\lambda_1$, $\lambda_2$, and $\lambda_3$ are the weights, and empirically set to values of $8/\pi$, $2/\min\{T_w, T_h\}$, and 0.5, respectively, in all the experiments. When there is no local deformation, $D_{\text{local}} = 1$.

*Accessibility Constraint* The local deformation constraint alone can only prevent the over-deformation in the local scale. It cannot avoid the over-deformation in a global scale, as demonstrated in Figure 9(a). Three circular windows drift away from their original locations and destroy the layout, even though each of them is not over-deformed in a local sense. To constrain the deformation in a global scale, we propose a 2D accessibility constraint, inspired by the surface exposure [Hsu and Wong 1995] and 3D accessibility [Miller 1994]. This maintains the relative orientation and position between the current line segment and its surrounding line segments.

To compute the accessibility of the current line segment, say $AB$ in Figure 8(b), multiple rays are shot from the midpoint, $P$, of $AB$ towards the surrounding circle in order to determine the closest surrounding line segments. For each intersected line segment, the nearest point, $P_i$, is determined, forming an imaginary line segment $PP_i$. The accessibility is then computed as

$$D_{\text{access}}(AB) = \sum_{i=1}^{n_l} w_i D_{\text{local}}(PP_i), \tag{3}$$

where $n_l$ is the total number of intersecting line segments to $P$. $D_{\text{local}}(PP_i)$ is defined in Equation 2; $w_i$ is the weight, computed as the normalized distance $w_i = |PP_i|/(\sum_{i=1}^{n_l} |PP_i|)$. Its value is higher when the corresponding line segment is closer to $P$. Hence, the overall metric of controlling the deformation is,

$$D_{\text{deform}}(AB) = \max\{D_{\text{local}}(AB), D_{\text{access}}(AB)\}, \tag{4}$$

where $D_{\text{deform}} = 1$ when there is no deformation. Figure 9(c) visualizes $D_{\text{deform}}$ of the deformed image (Figure 9(b)) by color-coding each line segment with lighter value indicating higher deformation, and vice versa. As the objective function is computed on the basis of a character cell, the deformation value of a character cell $j$, $D^j_{\text{deform}}$, is computed. All line segments intersecting the current cell $j$ are identified, as denoted by the set $\{\mathcal{L}_j\}$. $l_i$ is the length of the $i$-th line segment $L_i$ (partial or whole) in $\{\mathcal{L}_j\}$ occupied by cell $j$. Then, the deformation value of cell $j$ is then computed as the weighted average of deformation values of involved line segments,

$$D^j_{\text{deform}} = \sum_{i \in \{\mathcal{L}_j\}} \tilde{l}_i D_{\text{deform}}(L_i), \quad \text{where } \tilde{l}_i = \frac{l_i}{\sum_{i \in \{\mathcal{L}_j\}} l_i}. \tag{5}$$

**Objective Function** With the shape similarity and deformation metrics, the overall objective function can be defined. Given a particular text resolution, our optimization goal is to minimize the energy $E$,

$$E = \frac{1}{K} \sum_{j=1}^{m} D^j_{\text{AISS}} \cdot D^j_{\text{deform}}, \tag{6}$$

where $m$ is the total number of character cells, and $K$ is the number of non-empty cells, and is used as the normalization factor. $D^j_{\text{AISS}}$ is the dissimilarity between the $j$-th cell's content and its best-matched character, as defined in Equation 1. The term $D^j_{\text{deform}}$ is the deformation value of the $j$-th cell. When there is no deformation, $D^j_{\text{deform}} = 1$; hence $E$ is purely dependent on $D^j_{\text{AISS}}$. Note that the energy values of different text resolutions are directly comparable, as our energy function is normalized. The lower row of Figure 12 demonstrates such comparability by showing our results in three text resolutions along with their energies. The middle one (28×21) with the smallest energy corresponds to the most pleasant result, while the visually poor result on the left has a relatively larger energy.

We employ a simulated annealing strategy during the discrete optimization. In each iteration, we randomly select one vertex, and randomly displace its position with a distance of at most $d$. Here, $d$ is the length of the longer side of the character image. Then, all affected grid cells due to this displacement are identified and best-matched with the character set again. If the recomputed $E$ is reduced, the displacement is accepted. Otherwise, a transition probability $P_r = \exp(-\delta/t)$ is used to make the decision, where $\delta$ is the energy difference between two iterations; $t = 0.2 t_a c^{0.997}$ is the temperature; $c$ is the iteration index; $t_a$ is the initial average matching error of all grid cells. If $P_r$ is smaller than a random number in $[0, 1]$, this displacement is accepted; otherwise, it is rejected. The optimization is terminated whenever $E$ is not reduced for $c_o$ consecutive iterations, where $c_o = 5000$ in our implementation.

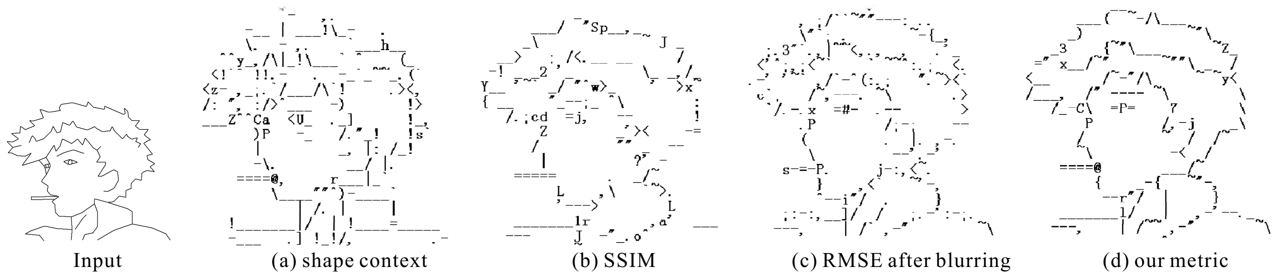



Figure 11: Comparison of ASCII art using different shape similarity metrics.

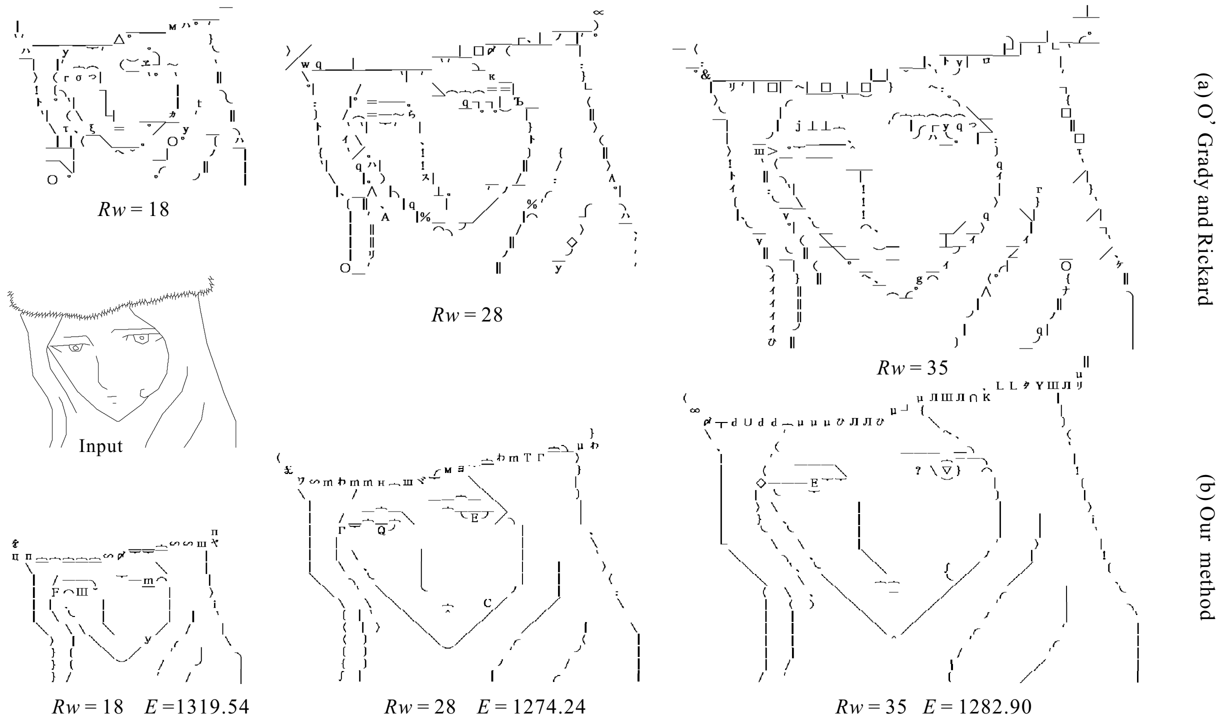



Figure 12: Our method vs. the method of O'Grady and Rickard. $R_w$ is the width of the text resolution and $E$ is the optimized energy.

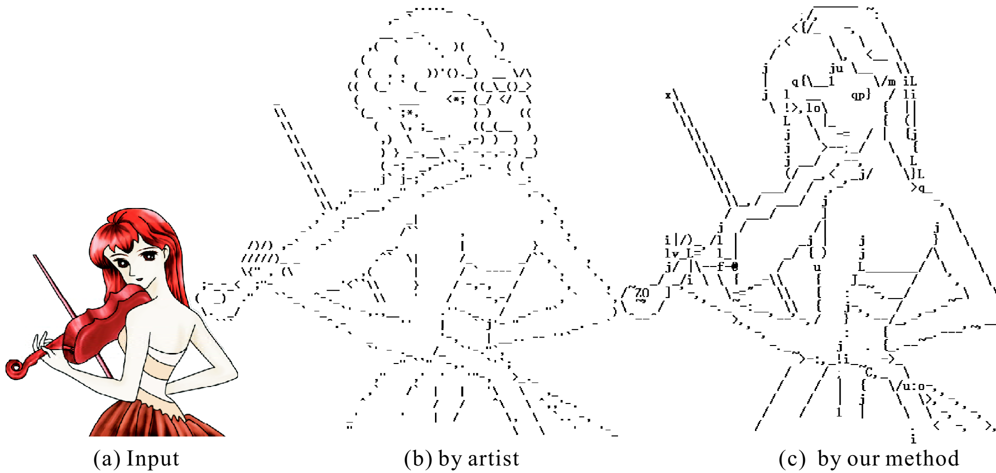
Figure 10 shows the intermediate results along with their energies. As the energy reduces, the visual quality of ASCII art improves accordingly. An animated sequence for better visualization of such optimization is included in the auxiliary material.

## 6 Results and Discussions

To validate our method, we conducted multiple experiments over a rich variety of inputs. The setting used for generating all our examples in this paper and their running times are listed in Table 2. Our method works with any font database of fixed character width. This paper shows results of matching characters from ASCII code (95 printable characters) and Shift-JIS code (475 characters only). Figures 14 to 17 show our results. The corresponding inputs can be found in Figure 18. Complete comparisons and results can be found in the auxiliary material.

**Metrics Comparison** In Section 4, we have compared different shape similarity metrics for matching a *single* character. One may argue the visual importance of the proposed metric in generating the entire ASCII art which may contain hundreds of characters. To validate its importance, we compare the ASCII art results (Figure 11) generated by substituting the character matching metric in our framework with different shape similarity metrics, including shape context, SSIM, RMSE after blurring and our metric. Hence, the same deformation mechanism is employed in generating all results. The result of shape context (Figure 11(a)) is most unrecognizable due to the structure discontinuity caused by the neglect of position. SSIM and RMSE preserve better structure as they place a high priority on position. Their alignment-sensitive nature, however, leads to the loss of fine details. Among all results, our metric generates the best approximation to the input, with the best preservation of structure and fine details. The comparison demonstrates the importance of transformation awareness and misalignment tolerance in preserving structure continuity and fine details.

**Comparison to Existing Work** Figure 2 already demonstrates the inferiority of the more naïve halftoning approach in representing clear structure. The only alternative work that was tailormade for generating ASCII art is by O'Grady and Rickard [2008]. We therefore compare our results to those generated by their method in Figure 12(a). Due to its halftone nature, their method fails to produce satisfactory (in terms of structure preservation) results for



(a) Input (b) by artist (c) by our method

Figure 13: Comparison of ASCII art between an artist's and our method.

| | Text resolut. | Character set | Running time | Time by artists |
|---|---|---|---|---|
| Fig. 1 | 138X36 | ASCII | 15mins | 5 hrs |
| Fig. 11 | 26X16 | ASCII | 4mins | - |
| Fig. 12b | 18X13 | Shift-JIS | 6mins | - |
| Fig. 12b | 28X21 | Shift-JIS | 8mins | - |
| Fig. 12b | 35X28 | Shift-JIS | 8mins | - |
| Fig. 13 | 52X34 | ASCII | 7mins | 2 hrs |
| Fig. 14 | 66X61 | ASCII | 12mins | - |
| Fig. 15 | 31X20 | Shift-JIS | 15mins | - |
| Fig. 16 | 35X27 | Shift-JIS | 11mins | - |
| Fig. 17a | 24X37 | Shift-JIS | 9mins | - |
| Fig. 17b | 24X38 | Shift-JIS | 9mins | - |
| Fig. 17c | 25X36 | Shift-JIS | 9mins | - |

Table 2: Timing statistics.

all three text resolutions (from 18×13 to 35×28). All fine details are lost in their results.

**User Study** To conduct a user study, artists were invited to manually design ASCII art pieces for 3 test images. They were free to choose the desired text resolution for their pieces, but the character set was restricted to ASCII code. We then use our method and the method by O'Grady and Rickard to generate ASCII art results with the same text resolutions. Then, we invited 30 participants for the user study. The source image and three ASCII art results were shown side-by-side to the participants. Figure 13 shows our result as well as the artist piece from one of the 3 test sets. The complete set of comparison can be found in the auxiliary material. Each participant graded the ASCII art using 2 scores out of a 9-point scale ([1-9] with 9 being the best). The first score was to grade the similarity of the ASCII art pieces with respect to the input. The second was to grade the clarity of content presented in the ASCII art without referring to the input. Therefore, there were 18 data samples for analysis from each of the 30 participants. Altogether 540 data samples can be used for analysis.

From the statistics in Table 1, the results by O'Grady and Rickard are far from satisfactory in terms of both clarity and similarity. Our method is comparable to (but slightly poorer than) the artist production in terms of clarity. In terms of similarity, however, our method produced better results than the artist's production. Such a phenomenon can be explained by that fact that artists can intelligently (creatively) modify or even drop parts of content in order to facilitate the ASCII approximation (e.g. hairstyle of the girl in Figure 13(b)). In some cases, they even change the aspect ratio of the input to facilitate character matching. On the other hand, our method respects the input aspect ratio and content.

**Animated ASCII Art** Figure 17 shows the ASCII art results of converting an animation to a sequence of ASCII art pieces. Although each frame is converted independently without explicit maintenance of temporal coherence, the generated ASCII art sequence is quite satisfactory. Readers are referred to the auxiliary material for a side-by-side comparison between the original frames and our generated ASCII art, in an animated fashion.

**Timing Performance** The proposed system was implemented on a PC with 2GHz CPU, 8 GB system memory, and an nVidia Geforce GTX 280 GPU with 1G video memory. Table 2 summarizes the timing statistics of all examples shown in this paper. The

| | Methods | Mean | Standard deviation | 95% confidence interval Lower Bound | Upper Bound |
|---|---|---|---|---|---|
| Similarity | Artists | 6.86 | 1.32 | 6.60 | 7.11 |
| | Our method | 7.36 | 1.13 | 7.14 | 7.58 |
| | O'Grady and Rickard | 4.42 | 1.82 | 4.06 | 4.77 |
| Clarity | Artists | 7.18 | 1.25 | 6.94 | 7.42 |
| | Our method | 7.09 | 1.30 | 6.84 | 7.34 |
| | O'Grady and Rickard | 4.15 | 1.80 | 3.80 | 4.50 |

Table 1: User study statistics.

second, third, and fourth columns show the corresponding text resolution, the character set used, and the running time for generating our ASCII art. The running time increases as the complexity of the input and the number of the characters increase.

**Limitations** Besides the fact that traditional ASCII art only works on a fixed-width font, modern ASCII art also deals with proportional fonts, e.g. Japanese Shift-JIS. Our current method does not handle proportional placement of characters or multiple font sizes in a single ASCII art piece. Another limitation is that we currently do not consider the temporal consistency when we generate the animation of ASCII art. To achieve this, one could first establish the correspondence between the shapes of the adjacent frames. Then one could constrain the deformation along the temporal dimension to achieve temporal consistency. Since our system only accepts vector input, real photographs or other raster images must first be converted into outline images. This could be done either by naïve edge detection or a sophisticated line art generation method such as [Kang et al. 2007], followed by vectorization. This also means that our results would be affected by the quality of the vectorization. A poorly vectorized input containing messy edges would be faithfully represented by our system. One more limitation stems from the extremely limited variety of characters. Most font sets do not contain characters representing a rich variety of slopes of lines. This makes pictures such as radial patterns very hard to be faithfully represented.

## 7 Conclusion

In this paper, we present a method that mimics how ASCII artists generate structure-based ASCII art. To achieve this, we first propose a novel alignment-insensitive metric to account for position, orientation, scaling and shape. We demonstrate its effectiveness in

balancing shape and transformations, comparing it to existing metrics. This metric should also benefit other practical applications requiring pattern recognition. Besides, a constrained deformation model is designed to mimic how the artists deform the input image. The rich variety of results shown demonstrates the effectiveness of our method. Although we have shown an application of animated ASCII art, its temporal consistency is not guaranteed. In the future, it is worth investigating the possibility of generating animations of ASCII art with high temporal consistency. An extension to proportional placement of characters is also worth studying. To further control and refine the result, it would also be beneficial to allow users to interactively highlight the important structure in the input for preservation during the deformation.

## Acknowledgments

This project is supported by the Research Grants Council of the Hong Kong Special Administrative Region, under General Research Fund (CUHK417107). We would like to thank Xueting Liu for drawing some of the line art works, and ASCII artists on `newsmth.net` including Crowyue, Zeppeli, Wolfing, and Asan for creating the ASCII arts in our comparison between the results by our method and by hand. Thanks also to all reviewers for their constructive comments and guidance in shaping this paper.

## References

ARKIN, E. M., CHEW, L. P., HUTTENLOCHER, D. P., KEDEM, K., AND MITCHELL, J. S. B. 1991. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 13*, 3, 209–216.

AU, D., 1995. Make a start in ASCII art. http://www.ludd.luth.se/∼vk/pics/ascii/junkyard/techstuff/tutorials/Daniel_Au.html.

BAYER, B. 1973. An optimum method for two-level rendition of continuous-tone pictures. In *IEEE International Conference on Communications*, IEEE, (26–11)–(26–15).

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Tran. Pattern Analysis and Machine Intelligience 24*, 4, 509–522.

CJRANDALL, 2003. alt.ascii-art: Frequently asked questions. http://www.ascii-art.de/ascii/faq.html.

COHEN, I., AYACHE, N., AND SULGER, P. 1992. Tracking points on deformable objects using curvature information. In *ECCV '92*, Springer-Verlag, London, UK, 458–466.

CRAWFORD, R., 1994. ASCII graphics techniques v1.0. http://www.ludd.luth.se/∼vk/pics/ascii/junkyard/techstuff/tutorials/Rowan_Crawford.html.

DAVIS, I. E., 1986. theDraw. TheSoft Programming Services.

DEFUSCO, R., 2007. MosASCII. freeware.

FLOYD, R. W., AND STEINBERG, L. 1974. An adaptive algorithm for spatial grey scale. In *SID Int.Sym.Digest Tech.Papers*, 36–37.

GAL, R., SORKINE, O., POPA, T., SHEFFER, A., AND COHEN-OR, D. 2007. 3d collage: Expressive non-realistic modeling. In *In Proc. of 5th NPAR*.

GEBHARD, M., 2009. JavE. freeware.

GOH, W.-B. 2008. Strategies for shape matching using skeletons. *Comput. Vis. Image Underst. 110*, 3, 326–345.

HSU, S.-C., AND WONG, T.-T. 1995. Simulating dust accumulation. *IEEE Comput. Graph. Appl. 15*, 1, 18–22.

KANG, H., LEE, S., AND CHUI, C. K. 2007. Coherent line drawing. In *ACM Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 43–50.

KLOSE, L. A., AND MCINTOSH, F., 2000. Pictexter. AxiomX.

MILIOS, E. E. 1989. Shape matching using curvature processes. *Comput. Vision Graph. Image Process. 47*, 2, 203–226.

MILLER, G. 1994. Efficient algorithms for local and global accessibility shading. In *Proceedings of SIGGRAPH 94*, 319–326.

MORI, G., BELONGIE, S., AND MALIK, J. 2005. Efficient shape matching using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 11, 1832–1837.

O'GRADY, P. D., AND RICKARD, S. T. 2008. Automatic ASCII art conversion of binary images using non-negative constraints. In *Proceedings of Signals and Systems Conference 2008 (ISSC 2008)*, 186–191.

SUNDAR, H., SILVER, D., GAGVANI, N., AND DICKINSON, S. 2003. Skeleton based shape matching and retrieval. *SMI '03*, 130.

TORSELLO, A., AND HANCOCK, E. R. 2004. A skeletal measure of 2d shape similarity. *Computer Vision and Image Understanding 95*, 1, 1–29.

ULICHNEY, R. A. MIT Press.

WAKENSHAW, H., 2000. Hayley Wakenshaw's ASCII art tutorial. http://www.ludd.luth.se/∼vk/pics/ascii/junkyard/techstuff/tutorials/Hayley_Wakenshaw.html.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., MEMBER, S., SIMONCELLI, E. P., AND MEMBER, S. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 600–612.

WIKIPEDIA, 2009. ASCII art. http://en.wikipedia.org/wiki/Ascii_art.

ZAHN, C. T., AND ROSKIES, R. Z. 1972. Fourier descriptors for plane closed curves. *IEEE Tran. Computers 21*, 3, 269–281.

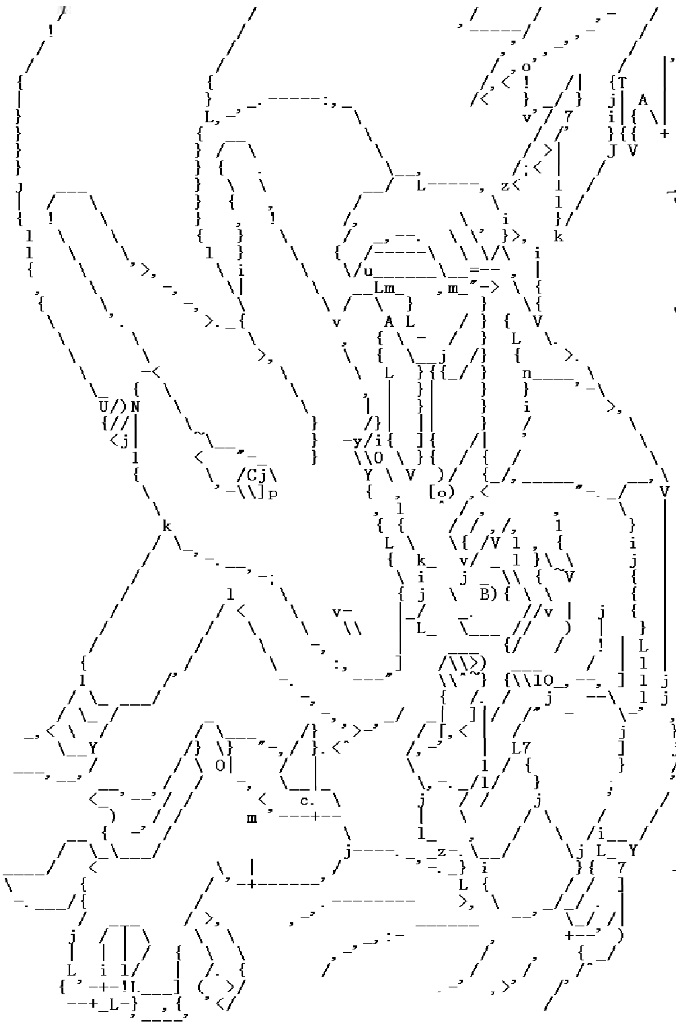
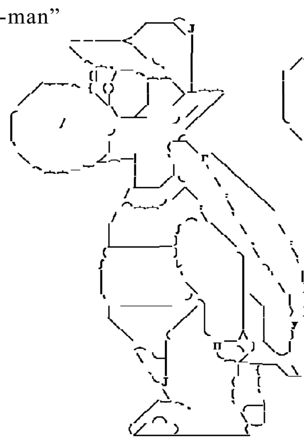


Figure 14: ASCII art of "dragon-man"

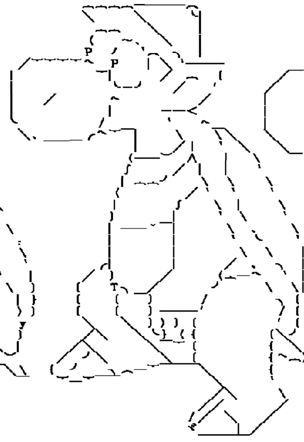
Figure 15: ASCII art of "Golden Temple"

Figure 16: ASCII art of "train"

(s1) "dragon-man"



(a) Frame 1



(b) Frame 3

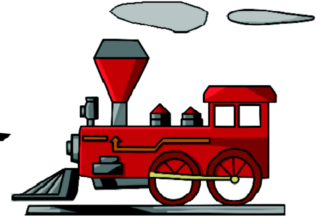
(c) Frame 6

Figure 17: Animation of "toitorse"

(s2) "banquet"

(s3) "church"

(s4) "Golden Temple"



(s5) "train"

Figure 18: Inputs of examples in this paper

# lain zine