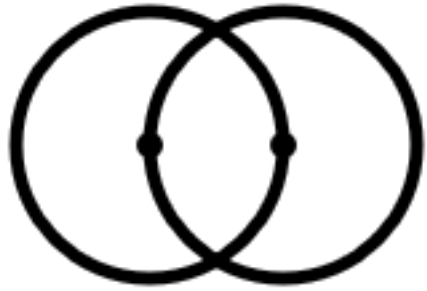
Geometron

Social media for a post-scarcity world

Lafe Spietz



Geometron
Social media for a post-scarcity world

Lafe Spietz

April 18, 2021

Contents

List of Figures

Chapter 1

Civilizations

The nature of our present civilization is one of constant consumption. We mine or drill material out of the ground, re-shape it or burn it for fuel, and ultimately turn it into toxic which waste we dump back into the environment. During the process from mine to landfill, we exchange "money" and this is what we call the "economy". The things which can be exchanged for money are called "property".

We seek to build a new civilization which is based not on a stream from mine to landfill, but on closed loops of material in equilibrium with living ecosystems. We look to the technological civilizations of indigenous people as an example of what mature technology should look like. In mature civilizations, technology is all self-replicating.

An example of technology in a mature civilization is

traditional wooden boats. If people live in equilibrium with a forest, as old trees die or are harvested new trees grow. People carve logs into boats, and then use them to hunt and sustain their communities. A new generation of humans is born from the old, and they are taught all the skills to copy the construction of the boat, the stewardship and harvesting of the trees, and how to pass along the knowledge of boat building(as well as the hunting and fishing that sustains the whole society) to the next generation after them and so on. Unlike a consumer civilization, this type of self-replicating organic technology can continue in theory indefinitely. If it has the resilience built in to improve and change over time in response to changes in the ecosystem it can truly sustain itself without limits. Mature civilizations like this have existed for thousands of years all over the Earth.

Three things to note about a technology such as the one described above: everything replicates, everything can be modified, and everything dies. Every wood boat will rot, and eventually become soil again in the forest which will produce more trees. The technology is soft enough that people can modify it, carve and repair it over time, change it as needed, and evolve it as needed in response to changing conditions in the environment or new innovations in the technology. And of course, every boat can always be copied freely. In this context the idea of property doesn't make sense. Nothing is permanent. Everything is constantly going through a cycle from soil

to tree to boat to soil again, growing, evolving dying, and being reborn.

When we build an economic system based on self-replicating geometric constructions from trash and organic material, money becomes a completely unusable system. To illustrate this, we consider a very simple thought experiment. Suppose I take a pile of trash and transform that into a robot which builds more robots out of trash, and which has media encoded in it which instructs others in how to copy the whole thing. If I and 100 people go from trash pile to trash pile and run the replication, converting all trash piles into trash-fed robot factories, we have created an ever expanding amount of value, with no property or mined materials going in. A vast amount of economic activity is being created, but without a central bank we have no way to denote all this added value. And if we somehow created a mechanism with some innovative new banking system to create enough currency to represent the added value, the instant we went and replicated the system again we would find that again, the numerical currency system breaks down, as it fails to keep up with the replication of value.

This contradiction is precisely the situation we currently find ourselves in. A small fraction of the population, namely the software industry, the media industry and the finance industry, can use replication to create value from nothing. With billions of people having already bought smart phones or other networked devices,

there is zero marginal cost to add some useful new feature. So when an app goes from running on just a single developer's computer to a whole software team to a beta test group to early adopters and ultimately to a billion smart phone users, this is replication of information, not actual production in the sense of a factory. Meanwhile, everyone else, who are compensated for labor or some type of physical material have to directly produce value to get money. As long as some people create value which freely replicates and others do not, with a finite money supply, over time a larger and larger fraction of money will always flow to the replicators rather than the people doing labor.

There is no way out of this trap. Not a new banking system, not a better government policy. The trap is the numerical aspect of money itself. No matter how fast a hypothetical banking system prints money and pushes it out into the economy, a fully networked replication-based society can create value faster, and continue to amplify the inequality until all value flows to the replicators. We have seen this during the 2020-2021 COVID-19 pandemic in the United States. The government poured trillions of dollars into "the economy" and at least half of it appears to have gone directly into the personal fortunes of people in the replication-based segments of the system. While these people also engage in massive exploitation of labor, this is beside the point. Labor can fight back and win some concessions from the capitalists, but as time passes,

the power will always slip away as the replication rate of pure information-based value increases.

Some people have proposed universal basic income as the solution. This is especially popular with the group of people who have benefited the most from the current accelerating inequality: software engineers and their collaborators in the professional classes. In this system, the software people are allowed to accumulate unlimited and ever-accelerating wealth, and simply paying out money directly to everyone else will help the rest of humanity survive. But if a software person is making 1 million dollars a year, do you really want a 2k/month UBI? How about 5k/month? or 10k/month... but now the software people make $1 million/month? Is that really a working economy? If this sounds insane, look at what is happening to rents, look at who all the new housing is built for. This is happening now, but without any UBI to soften the blow, and it will continue to happen as long as we all cling to this failing economic model.

So what is the alternative? The alternative is a switch from an arithmetic to a geometric economy. This means that rather than an economy of exchange, where numbers are used to represent both property and labor, we have an economy of replication of geometric constructions. In this situation, the replication of information *is* the economy. It does not turn into numbers at any point. One can create a thing via a geometric construction from trash which contains information for its own replication. This then

replicates out into the human network in a replication-based future descendant of the current Internet, and we all share the increased value. When we create value we do not "trade" it for something, because the amount of value will *increase* as it replicates, making any trade that truly represents the added value impossible. In a post-scarcity trash-sourced geometric economy, everyone benefits from the creation of creators because the best things naturally replicate to all of humanity, including the back to the creators themselves, but now amplified by the added benefit of billions of people potentially improving it.

The Geometron/Trash Robot system is an attempt to build an information technology system which serves this purpose: to build a purely geometric economy based on replication. In this system, everything replicates, everything evolves, and everything can be deleted at any time. In the beginning we source some parts from trash and some parts from common off the shelf consumer items that are easy to buy without any centralized company. In order to replicate successfully, however, this information networking system has to provide immediate value to the user, both within the existing money-based economy, and also outside that system.

In order for all this to work, we have to give up the three main elements of our existing system: mining, money, and property. As long as money exists, replication will break the economy. As long as mining exists, we will continue on a path to total destruction of the

world and eventually scarcity and extinction. And as long as property exists, replication which could end all scarcity for all of humanity will be hindered by the people who control everything. The extension of property into the domain of pure information is another major reason that people in the replication industries have been able to so brutally exploit the rest of humanity. They claim to "own" most of the freely replicating information we rely on in the new economy, and they're effectively landlords who can create new land as many times as they want for free, but then can charge rent on all of it. Only by creating a new economic model from scratch which simultaneously abandons money, mining and property, can we build a just and sustainable future.

It is not immediately obvious that geometry can completely replace numbers-based thinking. But in most cases, our technology is already completely driven by geometry, we just choose not to look at it that way. What is a microchip? It is a geometric design imprinted in silicon. That design is made by a geometric program interacted with by a user thinking geometrically. Even the code itself, encoding into the physical circuits, is a geometric pattern of either magnetic or electric perturbations of physical objects. And "computers" are not used primarily to compute things, but to display graphical information, both text and pictures and graphics. The repeated actions of automation machinery represent geometric operations(e.e. move over, move up, move down, move over,

etc. . .)

Saying "geometric economy" sounds abstract and to say it self-replicates sounds far fetched, but none of this is a new idea. Many of the oldest and most sustainable civilizations in the world have very very old technologies involving textiles made from natural materials woven into patterns. This is self-replicating geometry. It satisfies all the properties listed here: it dies of natural decay, is taught as a skill from the old to the young and is constantly replaced, and it can be modified and improved over time by the constantly-replicating group of experts in the technology within the community. So the transition to a geometric economy is not to some futuristic new idea but a return to ideas which predate the rise of industrial societies and nation-states.

All that said, we live in times different from any which came before. The Internet is the baseline information system now for all of humanity. Even the few people not connected to the Internet now have their lives completely molded by it, as all power networks exist on it now. And consumer society has injected every single element or type of finished product our civilization uses all over the planet. So while in times before now one group of people might discover the use of bronze and another of steel, now every single person on the planet can get aluminum sheet metal, titanium reinforced steel, rare earth metals, ultra high purity silicon, etc. All these incredible materials, already shaped and processed into the most

useful form, are simply sitting in piles of trash in every corner of every nation on the planet now, all directly adjacent to nodes on an information network which connects to every other part of the planet.

This combination of universal networking and universal access to identical, standardized, trash elements creates a formula for building a new information system from scratch from which a whole stream of new civilizations can rise. It is our task to build a seed from which such civilizations can be built. Geometron is an attempt to build this seed.

This chapter encapsulates the Trash Magic Manifesto and references it. It sets up the problem at the highest level before delving into the specifics of the solution. This chapter describes the problem that all subsequent chapters provide the solution for. Essentially the Trash Magic Manifesto laid out a goal for technology and for civilization. I realized that information technology has to form the basis of all the other technology. If information technology is taken to be general enough, it can be the basis of the entire complete set defined in the Trash Magic Manifesto. Define that set here. Generalized media. Point to the next chapter.

Chapter 2

Organic Media

2.1 Organic Media

In order to build a world of abundance from waste streams of the old world we need to create a new type of media which self-replicates. We need the media to replicate, the means to produce the media to replicate, and for the media to function as the replication medium for other things of all kinds. In order to build this new type of media we will in this chapter first explore exactly what self-replicating media is and how it works. We refer to self-replicating media as being "organic" media.

Some laws our media will follow in its construction are:

- everything replicates

- everything evolves
- everything dies
- no money
- no mining
- no property

Organic media exist to replicate that which we desire to replicate. Its purpose is to facility replication of things. It contains both information on how to replicate things and information required to inspire people to *desire* to carry out replication. We must give people a reason to replicate as well as a means to replicate. Thus one form of Organic Media is instructions on how to build something you can sell. To maximize replication we are less concerned with how much money someone can make via replication than we are with how little they have to spend to initiate replication. We want to minimize the cost of each item a person might buy to sell, the minimum number required to be purchased, and also minimize the amount of labor and skill required to make the thing. The ArtBox is an example: each box contains the content to make another box. Each box is awesome and can be sold. The ArtBox itself is Organic media. a bunch of SRS gets moved here.

path of replication: how to install and how to teach others to install, how to replicate the code, replicate specific files, how it's all structured, editing the code itself

the structure of sign pointing to domain which points to terminal which is near a place which is labelled by domain(this needs to get explained early since it will be the referenced in everything, as it's basic to the structure of the System)

Editing: how files work, how they don't work. No databases. no private data. No file that cannot be edited.

Deletion: cheap sd cards can be wiped any time. All files can be deleted or written over by anyone at any time. Domains are chosen to be of no real value, are disposed of as soon as burned, never sold. Evaluate entropy of domain names to prove we can always have zero value of names even as our network grows exponentially. "Permanent" deletion less critical in a world without private or personal data.

Cybersecurity: there is none. Security is for private property, and there simply is none on this network. This network must always be disconnected from the property-based Internet or private systems. One way paths of information for protection.

draw from the SRS paper and the existing organic media paper(but the formal SRS idea goes in ontology chapter)

File types independent of technology: symbols, feeds, maps, scrolls. These can be described without any spe-

cific implementation of *either* software *or* hardware. Specific implementations will be dealt with in their own sections.

How files are structured, overview of file types, what they do. All of this is as a generalized specification which users can use to rewrite from scratch quickly.

Example of physical organic media: The Art Box, replicator card, replicator shape, object itself, scroll

We will return to the Art Box again and again as we go through the system, illustrating how it can replicate in the Geometron system and how it fits into all the parts: symbols in "symbols" for the shapes and net, the set in ontology, maps, feeds, scrolls, scroll replicator, role in magic as one of the Trash Robot avatar objects

The user page, basic operation of Geometron as it exists today, just viewing now editing, but with pointers to editors, jump on and look at a file RIGHT NOW, on one of my pages.

In organic media chapter we introduce geometron and say what we're going to be doing in this book. What we are doing is providing a *specification* and an *example* of an instance of that spec. We will use the Pi, and remote hosted pages. But expect this spec to be used by many coders on many platforms in many ways if successful.

This is the chapter which defines Geometron, states the goal here. We want an information network which facilitates the free replication of technology built from trash. To do that, we will need text documents, hyper-

text graphical documents(like slides), feeds of information elements, and generalized symbols and symbolic languages which can be used to create vector graphics and control machines for automation. Creating this language and a hardware platform to transmit it, which replicates itself will be a seed of a new economic system. In order for this to self-replicate we need the media to create value for its users. The next section describes the physical network which our system runs on.

In this work, we describe a system called Trash Robot and a language and IT platform called Geometron which together can serve as a platform for a post-scarcity economy without money. We will describe how the symbolic language can be used to create automation technology from scavenged trash components, and demonstrate with a practical machine which itself is used to fabricate materials which can then be sold.

Trash Robot

Trash Robot is a form of organic media which will be transmitted via the Street Network documented in the next section and the Geometron software platform described below and in the entirity of this book. Trash Robot is a combination of fabrication technology built from trash(the robot itself), self-replicating media consisting of clay coin-like tokens printed with the robot which can be used to stamp out more of themselves by

repeatedly molding and baking batches of clay, and some other elements which are also designed to easily replicate. Trash Robot replication is stimulated by the fact that it can make money. People will buy robots, and the clay tokens can have any symbol on them, and can be made into jewelry and coins which can also be sold for money. As long as there is money to be made from replication we should expect replication to take place freely. A robot can be built for under 50 dollars in parts and easily sold for well over 100 dollars, probably quite a bit more than that. And the products of the robot can be sold on the street for dollars but cost pennies to make.

As with all media in our system, the Trash Robot belongs to no one when it is made. We can sell it and then it belongs to someone but they can also pass it along for free, share it, copy it, and sell it. The fact that it is not property when it's initially created will tend to make it move freely around and replicate.

Geometron

This work loops back in on itself over and over, as does anything that self-replicates. Just as every living cell in our bodies carries a whole vast self-contained and self-replicating system with all our genetic information, so this book must return again and again to describing itself it it is going to work to replicate the system. To that end this final section in the discussion of organic media

describes just what Geometron is, what it is for, and why it is organic media.

Geometron is an information technology system on which we will build a new civilization beyond the artificial scarcity of the mining and consumption based civilization in which we all presently live. This system will be built on technology which is readily available all over the globe for very little money combined with material from waste streams like cardboard and plastic trash. As we grow, however, we will replace every component of the system which currently comes from a factory and mine with one which comes from a waste stream until there are no longer any inputs from the consumption-based civilization. The last chapter of this work will discuss the technological roadmap to get to that point using a hybrid technology of found electronics components from waste streams with components fabricated directly using fabrication technology which is itself built from waste components and which uses either organic materials or waste streams as its feedstock. Trash Robot serves as the seed of this, as it is a system built partly from waste(mechanical parts from old DVD or CD drives) which can be used for a wide variety of fabrication tasks. The Machine Control chapter will discuss how the Geometron software can be used for building practical fabrication and automation technology at all scales, from the scale of 100 meters or even 1000s of meters for agriculture down to the sub-micron scale for nano or micro electronic circuit fabrication, along with

all scales in between.

In order to build all this, we have to start with a social media platform built entirely around the tasks we wish to carry out. These are first and foremost building community of like-minded people who have the desire to build this better system. Then it is about as a group building things which are of direct utility to both the people in the network and other people we interact with. If we can provide tangible value, both in terms of goods and services, our network will naturally replicate, as people can both get money for growing the network and get direct non-monetary benefits(with new and interesting products they can use and share). We also need the network to not just blindly grow but to have an active research and development effort in which we as a community build out the technologies along the roadmap to a closed cycle organic civilization. This means our platform needs to have all the basic tools we need to do that research: automation, measurement, control, graphical communication, technical communication of all kinds, code development, fabrication, numerical processing and free open replacements for all the basic tools used for modern technology development.

Geometron is a seed for such a network. I have created this system to run in web browsers, which run on billions and billions of devices, in dozens of web browsers using the standards which are univeral today on the World Wide Web. Documents are created by users on apps

which allow them to create, edit, copy and destroy everything with no ownership of any files. Servers are small portable computers carried in a cloth shoulder bag which can be carried on the person of the Operators of the network. These are fully functional web servers, but are intended for use on local private networks in a physical place with a shared wifi network.

The Book of Geometron

This book itself is organic media. It is intended to teach its contents to a reader(or rather a small subset of readers) to the level where they can then teach another. This should enable them to re-write future improved versions. In this section I describe how the book was put together, where the files are stored, how to edit them and use the LaTeXdocument preparation system to make the files required to produce a finished book. This means you also need to know what is required to get the physical book printed at an on demand printer, get all the metadata required for publication and distribution, and sell your version in retailers both large and small and online and off. Thus even the book is fully decentralized in principle: if it costs you nothing to set it up to sell, you can sell only a half dozen copies and it will be a net positive, and then if the next person does this it will also be positive and so on.

If the book is decentralized in this way of distribution it has many advantages. If the book turns out to be disruptive enough that people try to use lawsuits to shut it down or harass an author, but there are 10's of thousands of new authors popping up all the time, it will be impossible to shut down. As some versions turn out to be dangerous or illegal, other versions can immediately be published with omit the offending content. Also, many editions will mean some will get much better than this initial manuscript. Decentralization means that as the manuscript finds its way into communities that speak different languages, the translation can happen without any centralized effort. So for instance if someone translates from the original English into say French, and then it spreads around in areas bilingual with French and some other language like Swahili it can go directly from the French to the Swahili without any involvement of the initial English speaking writers. By avoiding copyright, these improved and translated versions can then get translated back into English and sold yet again under yet anotehr edition. Having editions be unique can create a market for unusual editions, further pumping money into the system and stimulating further development of the book. I would rather see 10,000 people make 100 dollars each selling their own editions of this book to just their friends than see me as the initial author make 1 million dollars on 500,000 copies of one edition.

It is not my intent in the long run to make money on Geometron. It is my intent to create a network which allows us to live without money by directly bartering what we need to survive(food, a place to sleep and work, medicine, transport) without use of money or any production in the old consumer economy.

All editions are published with a public domain license for everything but the final pdf. The final pdf is published under the minimal copyright required for an author to create the needed publication metadata to get distribution outside of the on demand press used for printing.

Each chapter is a .tex file, using standard LaTeX.

Chapter 3

Street Network

3.1 Street Network

outline:

- what and Why? The power of the physical, local, and free. Organic media, what we want, links to previous chapter. Universal social media for sharing of information in a physically local domain with both content creation and consumption on all Web-enabled devices(laptops, phones, tablets, etc.) Hybrid markets: like Craigslist, but way more local. General description of what the system does(scrolls,maps, feeds, symbols, apps, industrial design and production via Trash Robot). This points to the subsequent chapters about these actual things. Free

boxes and food not bombs. We are a hybrid between free boxes and food not bombs and craigslist.

- The Terminal. This is the heart of this system. What is the Raspberry Pi, why is it powerful? How to build the Terminal. Options involving big screens and projectors, public terminals with large publicly viewable displays. How to adapt it to different situations, how to work with wifi networks, IP addresses, local and global, opening up a local wifi to a public domain. how to avoid ANY property. How the terminal is passed from user to user to operator to operator. Data hygiene: how to keep all personal information of any kind of the machine, to prevent leaks of property. What it means to have a network without property. Replication paths via local laptops(localhost) on wifi, global github repos, replication to global hosted domains. How to constantly back up and replicate to avoid information death. How to kill bad information. How to nuke the whole system if it's too rotten. Grey market and black market commerce.

- The Operators, what we are, what we do, how we do it, how we make money and barter, how we train new Operators . Role of Operator as universal moderator. Forking and avoiding the trap of the network monopolist. How to transition be-

tween money to barter, how to scale up to an all barter system by providing value for

- Psychogeography. Nodes of power, examples at global level and local level, finding the nodal points. Go through the whole philosophy, places, examples such as: parks, intersections, neighborhoods, famous landmarks, bridges, forking below a place, discussion of distance scales and granularity. Targeting nodes of power: Sand Hill Road. Wall Street. SoMa. K Street DC. Jackson Hole. Martha's Vinyard. Navy Memorial DC. Use of power nodes to build extremely powerful local networks, where information is exchanged between players in existing power networks. A network in the right coffee shop could connect people who collectively are processing 10's of billions of dollars of commerce just in that one coffee shop!

- domains. choosing domains, buying them, sharing them, avoiding troubles. Entropy of domain choice, size of name space. use of free web hosting services. Signs, markers, stencils, postcards: physical media which points to domains which point to terminals and link to the IP addresses. The work flow with physical media to software media and back.

- The market

- the coffee shop(or pub) node. How it can build community in the coffee shop, help all coffee shop customers to share and prosper, help neighbors of coffee shop, the developer workflow, use to expand operators. Coffee shop network as service for coffee shop owner for barter for coffee and food at shop. Building collaboration with businesses, both local and global, how to use global chains to scale globally with mutual aid and benefit for all.

- scaling up: the global swarm, going to full stack geometron(see last chapter), building more

What is the Street Network?

We want an information network based around physical replication of technology from trash. To stimulate the replication of the Network, we need it to create value for people who use it and operate it. This value can be of many kinds: it can directly provide physical goods people need, it can facilitate business in the monetary economy, it can provide mutual aid to a community, it can create local social connections, can build network power for users, and any of these values can be traded for materials and space needed to continue to expand the network.

Truly free network. The Humble Pi. The power of a network without property or users. Networked tech-

nology with free sharing can create more value for users than one with private data. Destroy the data brokers by making a network with no personal data.

This book will describe several products and services which Operators in the network can provide to local users. These can be exchanged for money or bartered directly for materials to expand the system(for instance one can ask users who are paying for goods or services to simply buy more computer equipment to build out more network infrastructure.) We aim to have the goods sold be as much as possible based on a stream of trash which are upcycled into sellable products. The prototype product here is a purse, the ArtBox, which can be constructed using the methods of Geometron and which contains the tools to replicate itself. It is, in essence, a self-replicating purse with a unique Open Brand, the Trash Robot Brand, which will be documented in its own chapter.

Services rendered will generally be of the kind which makes money for the user. So for instance a passerby might place an advertisement on a news scroll for a local coffee shop for their service which other coffee shop customers(this is all over the wifi network of the coffee shop so everyone is indeed a customer) might want. This makes the user money, which motivates them to come back and place more ads, so they are making money which they can pay to the Operator. All of this commerce then motivates people to come into the coffee shop and read and share documents. Since wifi has a time limit

and requires purchase, this brings higher traffic to the coffee shop, which then has their revenue go up, so to keep the system running, it is worth it to them to barter snacks and coffee to the Operator of the network.

Structure and purpose of the network: domains, places, operators, mutual aid, markets,

Networks are advertised with physical media which points to domains which point to physical places, specifically to the location of the physical web server, and have a hyperlink which only works on the local wifi network which links to the web server on that network.

The Street Network is social media based around physically local instances of the Web which are not on the public Internet. Wifi networks are used to share documents locally with other users on the same network. The Geometron servers function as public bulletin boards, where documents are shared freely with all other users. Any user can edit or delete any file. All files can be copied by anyone. Documents are all visible by both mobile devices and computers, as long as they are on the same wifi network. The Street Network is an example of Organic Media. It is intended to facilitate replication by users. Users can create and share documents advertising whatever commerce they wish to engage in: they can sell things, share ideas, give things away for free, advertise services, advertise businesses, describe how to make things, or look for others with shared interests. There are no users and no databases. There is just a list of

documents which users can click on to read, edit using the editors or delete using the file deletion tools.

The Geometron documents are stored on a Geometron Terminal, which is based on the Raspberry Pi mini-computer. Raspberry Pi is a non profit project from the UK to create a minimalist Linux based computer, mostly for education, research, art, and maker hobbies. They can be purchased easily online for approximately 50 US Dollars. The two most widely used formats of Geometron documents are the Scroll and the Map. A scroll is just a text file, with some formatting in a markup language called Markdown.

Add paragraph somewhere on role of cryptography in sharing encrypted plain text copy/pastable files. add functionality. PGP? ask someone for help on this section to get it right. How does this all work with various crypto technologies? free sharing of encrypted files which can be replicated but then decrypted privately. This can be a huge vector of replication for the crypto community, including inside the formal and informal intelligence community.

what does the server do, what is an app, what is a document, overview of symbols, maps, scrolls,data,apps,automation.

Edit scrolls.

Edit maps.

feeds. using feeds.

Workflow, developers, apps. reference code structure chapter. Coffee shop developer community.

Users

We are looking to build a network which promotes the development of physically local power structures. To that end, we choose nodes of physical power, such as the most important locations in global cities like key traffic circles in Washington DC. We then study *all* the stakeholders in that physical location. This includes residents, tourists, people walking through, truckers, commuters, workers, panhandlers, local homeless, business owners, the local mail carrier–really everyone, but restricted entirely based on that physical place, rather than any other affiliation. Our network seeks to give the power of networking in that place to everyone in that place. As more people join, more infrastructure is added, until an entire ecosystem of interconnected network nodes grows up in some cases around just a single street corner(say 16th and Mission for example in SF).

The network has to start somewhere, however. We start with Trash Robot, operating the robot to make and sell tokens, and selling all the parts of Trash Robot. We can recruit people to learn the system who can immediately sell Trash Robot elements to make money, which will lead to replication of the network. Since Trash Robot uses the Geometron system to design the symbols, share how to build the robot, program the robot, etc., the social media system will be automatically replicated as Trash Robot replicates. If the network replicates and Trash

Robot Operators learn to operate the Geometron servers, the network will grow, and like all networks its power and value will increase exponentially with size. We also initially need developers to improve the code and build out the more technical elements of the system, essentially setting up Raspberry Pi based web servers and distributing them. So initially we need two groups of people: people to build and sell the elements of Trash Robot and people to develop applications for the Geometron system, build servers and distribute and install them. In both cases, Trash Robot builders and Developers can be recruiting Operators who have less specialized skills but can make money from the system via simple means of just stamping out more and more tokens from already-printed stamps or posting ads for people for money on a bulletin board they operate but did not set up.

Trash Robot Makers: build trash robots. Trash Robot printer Operators: run the robot to print. Trash Robot artists: design new symbols Trash Robot market Operators: operate a bulletin board

Geometron Developers. Geometron Operators.

A Geometron station can have a huge display which all passerby can read without even loggin on, also.

In order for this network to grow it has to create value for people. The more people it provides value for and the more value it provides, the more effectively it will replicate. I will now discuss some of the specific groups of people who can benefit, how they can use the network,

and what the benefits are for them specifically.

Traveling kids, hobos, panhandlers, people asking for money or selling things on the street corner. A physically local free bulletin board shared by passerby in a high traffic area can allow people asking for money who are currently ignored by passerby as just another anonymous face and cardboard sign a chance to really tell their stories and to share all that they have to share. When people share their stories they can become part of the emergent physical community of passerby in a location where the network node is located. When people view others as part of their community they not only are more willing to help, they can have open communication about the best way to help, expanding from just spare change to more comprehensive mutual aid. Because we clone content from the local terminal to web pages on globally visible domains linked to a physical place, which are advertised everywhere in that place, marginalized people whose only ability to get online is the public library can use the computers there to get the information they need to better survive, and ultimately to thrive and build new communities where they already are. The way a local network can help people is twofold. First, it is direct, by asking for money and other mutual aid. But by being physically on location all the time, already with physical media(cardboard signs), people in a given place can aid the network, creating value for the other people in the community who are more resourced, who then no longer

view monetary support as "donation", but rather as an expense which supports their other business activities.

In order to see the power of this second means of network support of marginalized people on the street, we have to look more closely at the network nodes we are building. One of the major types of node is in a business district of a city where there are both homeless people asking for money, on the street all day with physical media, and power brokers who make their living entirely from connections. These people include venture capitalists, entrepreneurs, lobbyists, consultants, and the rest of what might be called the "deal-making class". An example of this confluence is some of the parks along K-Street in Washington DC. K Street and adjacent streets is home to a huge homeless population as well as power brokers whose livelihood depends entirely on connections. If a physical network were built which facilitated direct communication between people along K Street, the people who spend the most time physically on the street can be brokers of information on a network which can be worth a lot to the people who trade in information. Physically local information networks can leverage the power of physical places with very powerful people walking past all the time who normally never communicate. Connecting these people up can be dangerous. But if we provide them with value, it can be worth a both a lot of money to them and also potentially something they can barter for giving us space to live and work nearby. If you facilitate

a 10 million dollar deal and the customer knows you can do it again, the least they can do is give you a 100 dollar gift card to the nicest restaurant in the block. There is no real upper limit on what an enterprising Network Operator could in theory make if they learned to really channel information efficiently in the nodes of global power. And of course we must remember that when dealing with power brokers their currency is not money. When the people who currently have the most power in society find themselves dependent on free open networks, those networks themselves will gain power which penetrates that of the existing power structures, potentially creating an existential threat to them. We must take note of this.

The elements of traveler culture which overlap with "van life" are also key to increasing the network effects of the Street Network. This also links to trucker networks. People who live their lives on the road can use this network infrastructure to set up complex networks and markets in highway rest stops, Walmart parking lots etc. using either wifi networks in these places. These networks can be of utility to passerby of all kinds, from tourists to truckers to the workers who keep the places running. Just as existing global social media networks provide value they can charge money for, a physically local network can provide value which people will pay for. An example use case here is a Street Network Operator agreeing to maintain a backup of and keep posting an advertisement for something a local entrepreneur is try-

ing to sell to truckers. In exchange for that, they can get directly compensated in gas, right there in the rest stop, without money changing hands.

Food not bombs, street outreach, harm reduction people, mutual aid workers. See above. The people who are working to help the most marginalized members of any given community can better reach that community if there is a physically local media platform where people can share information about resources. Documents can be posted which explain how to get access to resources, when and where resources will be available, etc. Because the whole system self-replicates, as with Food Not Bombs, anything which is successful in any given place can be immediately cloned to other nodes on the network. Food Not Bombs already has a global network of free and open nodes with no property but a very recognizable brand identity and set of behaviors and actions. FNB nodes are generally already linked by networks both online and via people who travel from one punk house or FNB house to the next. The whole anarchist network of community houses, FNB's, anarchist infoshops and bookstores, really really free markets, free boxes, etc. can form a basis for a truly free information network carried from house to house and city to city, running on house wifi networks.

Coffee shop owners. Building a network in a coffee shop on the wifi network which requires purchase to use and which has a time limit can create a huge amount

of added business for any local business owner. It also builds community. So coffee shop owners who find themselves with a full shop of laptop drones with headphones on who work for hours, or get kicked out and do the same thing somewhere else can instead find themselves the brokers in a very powerful information network. Much of the commerce of the world is now code written in coffee shops on laptops. Creating physically local networks around these already existing groups can create huge power for the users which then benefits the people who set up the infrastructure(again, just like existing centralized social media platforms.)

Developers. We need developers to be constantly writing more and better software in order to make Geometron a success. Developers who work all day in coffee shops or any other shared space like a co-working space or pub can have a social network based on both co-developing applications useful to all and sharing other resources. Developers will use the resource of the Street Network terminal/server on the local network in the same basic way as others: they can share their resumes, links to pages of personal projects. Developers are key to the whole system. We must recruit developers with this book who will rewrite all the code and also the book, replicating the whole system. The faster our network can get developers into the swarm, the faster the code itself will improve. Developers are key!! Developers create servers to share into the network.

Power brokers. Venture capitalists, financiers, entrepreneurs, deal-makers of all kinds, lobbyists, politicians. Your network is your power. Geography matters. Build a network in the lobby. Post things on street nodes, build your network, build your power, build your literal street cred. Dealflow.

Crafters, makers, jewelers, artists. An alternative to Etsy, street vending, or being in a shop. Post your stuff to the local networks. This is much more free and long form than existing platforms, you can post images, descriptions, contact info, times and places when you'll be in a place. This can be way easier than other sales channels for arts and crafts. You can say when and where you'll be at a place, post a link for contact, and then show up in the network node like a coffee shop to make the physical exchange. In many cases, because the network is physical and local, there will be barter opportunities as well as direct sales. A barter economy can develop where people donate materials you use for your crafts as part of how they pay for the finished product. Removing shipping or transport costs by dealing directly in a physical location removes friction from the market, amplifying dramatically the power of the market, especially for crafts which involve physically bulky objects. For instance, people can bring in motors and properly prepared plastic sheets and cardboard, as well as rolls and rolls of duct tape, and we can exchange finished products built from these materials and tools, as well as free food, drinks, and sup-

plies, creating a market economy without money as well as without formal business structures(making it easier for marginalized people to participate).

Any labor pool of gig economy workers focused on a specific geographic location. The most obvious of these is the drivers who presently drive for the major rideshare apps who all congregate at the airport to pick passengers up in the same exact place, and yet all of it is currently coordinated via the apps(unless you do the cab line). The rideshares apps have proven that cities will ignore illegal cabs if they're done at scale. It would be straightforward for a small team of Network Operators to run a server which replicates to a page which is advertised around, something like a domain of yourairportnamerides.xyz, which tells users how to log onto the wifi network created by an Operator's hotspot near the pickup zone and with a link on the page to the local network address of the server. All all this IT is doing is directing customers to a dispatcher who manages the drivers over a simple app shared by the collective. The whole network is run by a team of about 2-4 people. One person might be a developer, who creates the app to manage all the drivers and post messages from dispatch. Another person is all marketing, putting up the relevant information in the right places to get seen by travelers but not stopped by the rideshare apps, airport authorities, or the cab companies. Riders will never have their destination information on the public network, nor will drivers put personal in-

formation, but they can work on an open trust model where they are known by dispatch, who has code names for them, and operates a queue app which simply adds drivers as the arrive near the Airport and pushes the most senior driver to the top of the stack, which is passed along to a rider. Another Operator might be the one who runs the trust network for the drivers, verifying everyone and organizing meetings for the whole cooperative. This can be used to unionize existing workforces quickly as well, building ad hoc networks which are very hard to suppress visible to everyone on their mobile devices on a local wifi network.

The same model holds for places where workers congregate looking for short term construction work. Those locations can have a server where an Operator runs a labor marketplace where a much larger and deeper labor pool can now advertise, but without all having to be in the physical location. This means a crowd of a dozen workers looking for work can be replaced by an Operator with a sign pointing to the domain where the copy of the market is hosted. Workers who come by can leave an ad on the local Raspberry Pi Geometron server, and anyone coming by looking for construction labor can just scroll through a now much deeper collection of ads and call whoever they need to hire. A market place like this can suddenly go from a dozen general laborers to a construction labor market which includes specialists like plumbers and electricians as well as much larger general contrac-

tors just looking to save on marketing costs. A person holding a cardboard sign on a street corner by a giant box home improvement store can now potentially be the broker of an information network on which millions of dollars of commerce flow.

Trash Robot. Trash robot will be described later in this work. It is a system of technology which can be used to build products from a combination of waste streams and consumer off the shelf(COTS) products, which has a clearly recognizable brand identity owned by no one and provides value to an end consumer. Trash Robot is a meta-business: a system for people to build businesses which can use both barter and sales to make money on the Geometron Street Network, further facilitating the replication of that network. If the right people start doing Trash Robot on the Network, we can create a system which has a significant consumer demand. Trash Robot is structured to be very easy to replicate for an individual but very hard to replicate for a for profit centralized technology company driven by building up value in their equity. This means if we can build up a significant consumer demand, it will provide a very powerful stimulus to replicating the nodes to grow our network, always as a free decentralized system. Building a thing that replicates faster than property that is not property is how we start building a society without property. This is described in detail it its own chapter later in this book.

Trash Robot is the most obvious way for those of us

who are building this network to exist. Other people might be Operators, developers, or participants in various markets, but the Trash Robot is the heart of the hardware development which this whole self-replicating social media platform is designed to replicate. Trash robot is the reason we are building the Street Network and Geometron information system. Trash Robot is a system of technology which provides value to people of a variety of kinds. But because it allows Operators of the Trash Robot to be able to print nice clay tokens printed with arbitrary icons designed on the system, we have the ability to issue our own symbolic currency, which can create a new geometric economy not based on money but with some similarities. We will build clothes with an open brand people can use to replicate their own copies. We will make interesting or useful accessories like purses, machine carrying bags, and jewelry. We will make and sell robots, and teach classes on how to make them and use them. And we will make the icon tokens to mean anything and exchange for anything.

The tokens printed on the Trash Robot printer are totally unlike money. They have no numerical value of any kind. What they have is simply two symbols, both created in a universal geometric language which can be copy/pasted freely across the Network. If someone sends you a text message with a string of numbers separated by commas and you give that to the Operator of a Trash Robot Icon Token Printer, they can just paste that text

into their browser, use the code in there to program the Robot Printer and print the pattern into clay from the code. Also, the coins are all themselves self-replicating without the printer. Because they have depressions along where they are printed, clay can be molded around the depressions to make another clay token which has the inverse of the pattern in it but with raised clay instead of depressed. When this piece of clay is baked, it can then be used to print another token, making an exact copy of the original. This can be repeated many times. Thus printing one coin with a Trash Robot Printer can propagate out to in theory a very large number of copies. Copies might degrade as more and more are made but if each copy is used to make many copies and so on, in a giant tree, one can easily imagine one print making thousands of tokens with no more use of the printer! Again, we have made self-replicating media, both self-replicating online where we can share the code which does the print and self-replicating in the physical manifestation where the clay objects are used to make more clay objects.

Building the Geometron Terminal/Server

These should be called servers because they're servers.

Buy the stuff:

- Raspberry Pi 4 board from Sunfounder

- SD card
- SD card reader
- Mini USB keyboard(without number pad)
- mouse
- Sunfounder HDMI display with 12 volt power supply and USB power out to drive PI, wall plug and HDMI cable, or similar display which can run off of a 12 volt barrel connector with a USB power output to drive the Pi.
- 12 V LiPo battery pack with wall plug charger from TalentCell(sold via amazon)
- wifi hotspot

Put it together. Just assemble the Sunfounder terminal as per the instructions.

If you have a display that does not have a mount for the Pi, build an integrated terminal with cardboard, duct tape, and plastic HDPE sheet from milk bottles.

Burn the card with NOOBS, put it in the Pi.

Use paint pens to put symbols on keyboard.

plug in mouse and keyboard.

Make a bag to carry the terminal around in, or find an appropriate backpack and sew symbol onto it. Symbol of Raspberry Pi using Penrose Tiles.

Boot up the pi, set it up with no password

Install Apache and php

copy the Geometron code replicator script replicator.php into the web directory. THis can be found from any geometron server at [serverurl]/php/replicator.txt.

Learn to use with subsequent chapters of this book, customize and deploy, replicate to other people

run in headless mode, or on big screen, discuss display options, how to deploy in different places

The Operators

Building a relationship between Operators and Developers. Operators deal with people and information. Developers deal with code and build the apps to allow operators to operate. Operators connect people: passerby, shop owners, truckers, workers, drivers, developers, community members, other operators. The Operator is like a switch board operator in the old Bell phone system before automated switching. Switch board operators in small towns before automation knew not just phone numbers but the structure of the town, who was likely to be where, who gets called a lot, and in general how information flows through the networks of the town. This role is very old, however and has been held by pub keepers, religious figures, coffee house owners, and numerous others throughout history and throughout the world. The role of the information exchange manager is one all societies

need. The Geometron system creates a specific way for someone to carry out this role in a geographic location, which is easy to replicate in other locations, and to share information from place to place.

The relationship between Operator and Developer is key to structuring the network in a way that scales. Both roles have to be self-replicating in that Operators recruit and train Operators as well as find new people to get trained as Operators

Psychogeography

Introduction, what is Psychogeography, the historical references to the situationists.

Coffee shops and the laptop classes.

Global power nodes. examples.

Domains

Street Market

We help people sell stuff directly on the Street, out in the open, with a sign advertising the Market. People can sell for barter. We also sell directly the items from Trash Robot and other Geometron Things described below. These include the ArtBox as a purse, shirts, pants, flags, bags, clay icon tokens, robots, terminals, laser cut acrylic shapes and rulers and protractors, Pyramids,.

Coffee Shops and Pubs

Scaling Up

Street Network:

- operators
- terminals
- domains
- streets
- places
- developers
- signs
- postcards
- markets
- feeds
- scrolls
- maps
- pages
-

The Terminal is a Raspberry Pi with a keyboard, mouse, display and power supply, which run a web server only visible over the local wifi network. It is carried by the Operator, who uses it to help users create, edit, copy, and share files over the local network.

The files on the Network can be: - Scrolls. These are a type of text file which uses the Markdown markup language for formatting.

- Feeds. Feeds are either a directory with a sequence of files a user can scroll through and select or an array of any kind of information, be it images, symbols, words, links, etc. - Maps. A map is a sort of generalized meme, like a PowerPoint or Keynote slide. It is an array of elements each of which has a position, angle, width, possibly an image url, some text, and possibly a link destination, which might be either a HTML hyperlink or an internal link to a file on the system

All users on the same wifi network as the Terminal can view, edit, delete, and copy all the files on the system. There are no user names, no logins, no passwords, no private data, and no databases.

Users can all see all files, edit them, delete them, copy/paste them, create new ones

Operators carry the Terminal around, share its link with people, talk to people about the system, teach users to use the system, help to share with new Operators.

Roles of the Operator:

- the keeper of the physical Terminal
- maintain relationships with users of the Terminal and Domain
- update the Geometron server at the hosted Domain with links to the IP address on the local wifi network of the Terminal, as well as the wifi network name and password or link to where to get it(e.g. coffee shop register)

- post ads for money or barter on the Geometron server at the hosted Domain for people who ask
- post on the global Geometron server when and where the Operator will appear, or where the terminal is set up on what network if it's installed permanently.
- Teach anyone who wants to learn how to be an Operator, recruit new Operators
- Tell new users about the Network, teach them to use it, how to post, edit, delete
- Promote any kind of business or other venture or project anyone physically local to the wifi network area has in exchange for barter with that user for useful things on location(including just a place to operate)
- Spread Network into new places by finding a location with a wifi network, buying a domain and setting up hosting or getting someone to do that and pay for it,
- Domain names spread in physical space using physical media with depiction of domain which points back to terminal ip address, wifi address and password, photo of terminal and operator other physical media(post cards, book marks, spray paint stencils), spreading the physical media with the domain name

Skills of Operator

Domains

Terminals

Laptops

get ubuntu working under windows, install apache and php

localhost

code goes from terminal to laptop to github to

Chapter 4

Code

home

4.1 Code

Geometron Developers are a critical member of the swarm that makes up the Network, and this chapter is intended to describe how to learn to take on this important role in the swarm. We assume that you are willing to dig in and learn code but not that you already know a lot about it.

If you are a bit technical but not specifically a web develoer you will first need to learn the basics of Web Development. I strongly encourage you to learn how I did(unless you are already a web developer). You need to learn the basics of HTML, CSS, JavaScript, and PHP.

I learned these from https://www.w3schools.com/. I had to learn using Microsoft Visual Studio Code, because I did not have an editor yet(I needed to learn to make web code to make web code to make web code.) But you will be able to entirely learn all material on all four of those languages using the code editor built into each Geometron instance and testing instantly on the same instance.

Philosophy

All our code of all kinds is organic media as described in previous chapters.

All code can be copied by all users. All code is human readable, and can be copied by copy/paste. All code can be edited on all servers by all users. All code can be deleted by all users on all servers. No native code. All code is run from a browser, and can be run from any browser on the local network. All code is part of a server which can be replicated as a block with a single operating, copying all files from any computer on a local network to any other computer on that network. Each instance of the code is totally self-contained, and can be edited and replicated to create a new node with no relations of any kind with any other node. A single Raspberry Pi could be used to replicate a server to a laptop on a coffee shop which pushes the code to a public git repository which is then replicated to a globally visible url, which replicates

to more computers, git repos, terminals, and servers, and so on, replicating a single instance to the whole world of billions of devices in a matter of minutes if the relations between network nodes are set up properly, all with no centralized structure of any kind.

This system does not have "users" in the sense that they exist in current information technology systems. The systems just *are*. Think of them like any other appliance outside of information technology. We will illustrate the idea of a commonly shared thing with an example.

When you go into a coffee shop, the counter where orders come out and are picked up by customers does not have divisions by property. Customers are not assigned a special area on the counter to the exclusion of others. Likewise the baristas do not have zones marked off that are theirs, with some complex system for assigning a space over from barista to customer(like the driver and rider in a ride sharing app). Instead, the counter is a shared resource: its value to customer and barista alike is maximized by it being a common space where cups of coffee appear and are taken based on whatever is most convenient at the moment. Sometimes someone breaks the model here, and throws a huge dirty backpack on the counter or spills something sticky everywhere, putting the system out of use. But again we all accept that the risk of this and the added work to correct it is outweighed by the benefit of the counter being an open space.

Our information systems in Geometron are like the

counter top in a coffee shop. Indeed, one of the primary use cases is to have the software run on a server literally sitting on the counter of a coffee shop. As with the counter, the benefit to customers, workers, and owners of the coffee shop is such that it simply has more value shared with no property than it would if there were *any* barriers to the flow of information. Anyone with any experience designing and building information technology systems will need to constantly remind themselves how different a model this is to any existing system. Even anonymous message boards are not structured this way. Users are still users, even if they're just an anonymized IP address. In our model there *simply are no users!!* No users means no passwords. It also means there are no "transactions" involving money or property, as any application which actually does commerce can't function without a user on each side of the transaction. Our media can *enable* all kinds of commercial transactions! People can post things which say how to get ahold of them, or link to pages with more specific content some of which might be private. But the point is that something like an advertisement for some good or service on a local network is worth more the more people see it. If members of a community are all posting to each other and all getting value from seeing those posts, it is in everyone's best interest to have things be as free as possible, and this means no users.

Without users, there are also no passwords, no databases,

and no "security" of any kind. Security is physical in a physical space, and our networks are physically local and not on the open Internet. In order for this to work, people need to not put any private information on the network, ever. The only way users on their private devices interact with Geometron is through their web browser.

All that said, encryption can be used in interesting ways in a free open physically local network node. Files can be encrypted in a human-readable format, dropped on a server, and copied out across the web from server to server and laptop to laptop. One can think of numerous applications for this. In the initial version of Geometron that is being released with this book, there are no built in cryptography technologies of any kind. But with the application development workflow described in this work users can easily build that tech in whatever direction they choose. Again, this is the power of a fully open system: you can sit in a coffee shop anonymously editing code, write a totally new kind of cryptography application, share with other users in the coffee shop, who then copy to github and share with the world, and in no time your application has spread to the whole world, untraceable to you, and with others forking the code, riffing on it and improving it. Your project shared with one person on a coffee shop can come back to you in the form of a vast new ecosystem of derived code, shared freely across the global network and finally back to your coffee shop.

We also need to specifically state here that part of

the philosophy of Geometron is to have as little server side code as possible, and in particular to have no command line installations needed at all. I *strongly* advocate against any use whatsoever of systems like node.js or other server side packages which might do things the user cannot immediately see, copy, edit, or destroy. This is why I am such a strong proponent of using only PHP for almost all server side code. The only exceptions are for direct physical control of machines. For building robots using the Raspberry Pi we will use Python, along with shell scripts to set them up, and for Arduino-based robots we'll use that language(C++ with some nice libraries).

It is inevitable that developers who like these kinds of things, be it Ruby, Node, or the latest fad language like Rust or Go will want to rewrite all this in their favorite server-side language. I again cannot overstate how strong my opposition is to to this. PHP is unique in how it can run on pretty much any server with no added installations. It is installed by default on all new macs. It is installed with one command and actually always works on the Raspberry Pi and other standard *nix command prompts. And most importantly it just runs in the browser, where a file of the form filename.php in the browser just runs the script. This is what enables the self-replication and self-editing that makes this a form of organic media and not just another software platform. Please keep an open mind about this approach before trying to advocate for a supposedly "more advanced" lan-

guage than PHP.

The same statement applies to my choice to use JSON instead of XML for most of the data. This was not a choice made at random. I've played around a lot with both formats and feel very strongly that JSON is more appropriate for how we handle information in this system than XML is, for numerous reasons.

Code Structure

All Geometron apps run in a web browser and so are composed of HTML, CSS and JavaScript. PHP scripts are used to communicate between the web browser and the file system on the server(even if it is just a localhost server on a laptop).

Each instance of Geometron has a standard file structure where various types of files are always stored. Whatever directory on the web server a Geometron instance is in can have subdirectories which also have yet another Geometron instance. We aim to have the whole system be well under 10 megabytes, so a 1 gigabyte storage capacity server can host a hundred instances easily, each in its own subdirectory. All Geometron applications are html files which sit in the root directory of that instance(although this might be a subdirectory on a server). JavaScript libraries are stored in the directory jscode/ as .js files. Symbol files or other image files used as icons and graphics in a Geometron server can be stored in

the directory iconsymbols/. Scrolls are markdown files, which are stored in the directory scrolls/. JSON data for most applications is stored in the directory data/, and are always stored with the extension .txt so that they can be read as raw text in a web browser. The files in the specific JSON format called the "Geometron Map", documented in a later chapter, are all stored in the directory maps/. Two copies of all the php scripts exist on each server. One copy is in the directory php/ and has the file extension .txt. The other identical copies of the files are in the root directory of the Geometron instance and have the file extension .php. In other words each php script has one copy which anyone can read in a web browser and copy/paste without running the code and another which is exectuable but not human readable.

All these files of the above type are listed in a file called dna.txt which is stored in the data/ directory along with other JSON files. A script called replicator.php has in it a url of a dna.txt file. Replicator.php will extract the root directory of the Geometron instance being copied from the global url of that dna.txt file, and use that along with the JSON data in dna.txt to copy each individual file from the source server to the destination server. Any server can copy from any other server with this. Any given replicator script can have the one line of code changed which points to a different dna.txt file to switch the source server from which Geometron is replicated. So in a complex network, any node can copy from

any other node, replicating the whole structure. Note that replication is a destructive process, and in general the server onto which the code is replicated then has all the old files deleted automatically. All information is thus volatile. Anything can instantly be destroyed or instantly be copied an infinite number of times.

If you want to get the replicator on any given server, point your browser to [the main url of the server]/php/replicator.txt. Copy and paste that into a new file called replicator.php, and put that in the web directory of a new server. Then if you just point a browser to [address of server]/replicator.php, and wait until there is a link to click on and you'll be looking at a new instance of Geometron replicated from wherever the origin was of the instance you copied the replicator from.

You can try this by looking at the code at for instance http://www.trashrobot.org/php/relicator.txt. Or go find raw code at the main Geometron Thing repository under the username lafelabs, repo name "thing"(`https://www.github.com/lafelabs/thing/`).

To become a Geometron developer you will want to have a laptop which can run a local php web server, as well as a Geometron Terminal as described in the section on the Street Network. The server can also be a Street Network Terminal, but it doesn't have to be, it can be a Raspberry Pi which you install the system on. In other words, any Terminal is a full Geometron server, but a server does not have to be a full terminal. You can just

use a headless pi dangling from a USB wall plug, as the author does at home, setting it up once with a tv, then unplugging everything and leaving it on permanently.

When your laptop is on the same wifi network as your server, point your browser to the IP address of the server and you will be looking at that Geometron instance. Now on your laptop you will want to create a new github repository. This can be as generic as a repository called "thing". Make a working copy of the repository in some directory on your computer. Now get some type of linux command line working with php installed. In the root directory of your new project directory, make a new file replicator.php and paste into it the replicator of some other Geometron instance. At the command line, run replicator.php. This should copy the whole existing code system. Once the replication is completed, still at the command line, type

```
php -S localhost:80
```

And then navigate your browser to http://localhost. From there you can edit documents and code on that instance of Geometron just as you would on a Raspberry Pi based or globally hosted server. When you save that whole system to your github you are then publishing that entire copy of the code there. Now you want to edit the version of replicator.txt on your local server to point to the global url on github which points to your dna.txt.

This means for instance with lafelabs/thing the line in replicator.txt which we want to edit is

```
$dnaurl = "https://raw.githubusercontent.com/LafeLabs/t
```

In this line of the code, you will want to replace "LafeLabs" with your Github account name and "thing" with whatever the name of your github repository is(a logical choice might be "geometron"). Once this change is made, that global replicator.txt file which points to that global dna.txt file can be used to replicate your local instance to any other place on the Open Web. This means any other private Raspberry Pi Geometron Terminal on the Street Network can instantly be running copies of your code, without you paying for bandwidth or building a site of any kind. You can private message people the url of your replicator, and they can all be copying from github to private terminals in coffee shops, truck stops, and public libraries on every continent!

So this is how we replicate the system from a pi server to a local laptop to a global code repo to the whole Web. But how do we actually edit the code?

We use a program called editor.php which is based on the open source JavaScript library Ace.js to edit all code on the system. Editor.php reads the files in the various directories and lists them, color coded by code type. You can just edit live, and as you type the code changes. So watch out!! You can just pound keys and

wipe out index.html in a few keystrokes. You can randomly delete one character somewhere or add a random letter in a JavaScript library and catastrophically break the whole system. But no fear!! This is what replication is for!! A corrupted system can be totally wiped out by just running the replicator again, to wipe out the corrupted files. If there are billions of copies of a good app, however, the loss can be be on average negligible.

There is a window in editor.php which lists all the html and also some php files as clickable links. This is how you run another critical replication script, called dnagenerator.php. This script is where the JSON file dna.txt comes from. It scans the directories, looks at all the .html files, and writes to data/dna.txt on whatever server it is being run on. Any time you make new files of any kind on a server, always run that again.

The next most important script in understanding the system is text2php.php. This script takes all the files in the php/ directory which end in .txt and copies them into the main directory with the extension changed to .php, making them executable. This is why we need two copies of every PHP script: because we can't be editing PHP scripts which are actively running, and we want editor.php to be able to edit every single file on the server including itself! Editor.php, like so many code editors, was largely written in itself. You can use it to edit the file php/editor.txt, then run text2php.php and you'll find the editor has changed!! If you corrupt the editor, you'll

probably want to just run the replicator again to wipe and replace the whole system. But this is a very useful first exercise if you're learning to become a Geometron developer, and I strongly encourage you to actually do this, right when you get started. Go into php/editor.txt and change something in the CSS to change something like background color, run the script, and click back to editor.php and you should see your customized editor. You can change themes by referring to the documentation on Ace.js.

Editor.php functions by using two other php scripts: filesaver.php and fileloader.php. When you click on a file to load it into the editor, fileloader.php is engaged using the XMLHttpRequest object in JavaScript to fetch arbitrary files and load them into the Ace.js editor. As you hit keys when the cursor is in the editor, the file is updated using filesaver.php with every keystroke. Once again, there is no safety net here! If you mess around with the code, it changes instantly.

To create a new file using editor.php you use the question mark to input information to the script, with the field "newfile". For example, to create a new application entitled triangles.html, you would put "editor.php?newfile=triangle.h and scroll down to the very bottom of the list of files to find your new file. Then just start editing it, and when you want to add it to the geometron instance, click on dnagenerator.php and it will be added to the dna to replicate to the next instance. Also, it will now be

linked from editor.php, so when you reload the editor, you can click that link to get to your new app. This also works for other files besides new applications, you just include the directory name with the file, for instance "editor.php?newfile=jscode/newjslib.js".

To create another instance of the full Trash Robot/Geometron system, we copy a program called "replicator.php" into the main web directory of the server. The raw code can be found at either locally on this server at `php/replicator.txt` or globally on the original lafelabs Github "thing" repository at `https://raw.githubusercontent.com/LafeLabs/thing/master/php/replicator.txt`.

We generally run Trash Robot/Geometron in one of three ways:

1. Run it on a hosted remote server somewhere
2. Run it on a Raspberry Pi and serve it over a local wifi network.
3. Run it locally on a computer we are using for active development

To host it on a remote server, we first buy a domain name representing a local place which is not property: a public street, public park, public body of water name for instance. We always choose obscure domains, do not use .com, and avoid any personal information or names of businesses. Then we pay for hosting service. We find the root directory for web hosting, and create a new file

called replicator.php. We copy the code in the replicator into that and save it. Then we point a browser to [your domain name]/replicator.php and wait for the script to copy all the files.

To run it on a Raspberry Pi, after installing the normal Pi software, install Apache and PHP as follows:

Then install the Geometron software type copy/paste these commands into the terminal:

To run on a local laptop as localserver, if you're on a mac, just open a terminal. You can use the "command" button combined with searching for "terminal" to find it, then pin it to the menu bar. On a Windows machine, install Ubuntu under windows. Then as with mac you can use control-escape to bring up the Start Menu, and type in "ubuntu" and click on it to open a terminal. Once the terminal is open, pin Ubuntu to the task bar for easy use in the future.

In the terminal, you want to type

Or open .bashrc

```
nano ~/.bashrc
```

And copy this line after the last existing line of the file:

```
alias s='php -S localhost:80'
```

And then just hit the letter "s" every time you get to the command line.

When the local PHP server is running you can open a browser on that machine and point it to http://localhost and you will be running the full Trash Robot/Geometron software on that machine. You can use this for purely local interaction where no one in the world can see what you do, and can edit various files which you then paste into other instances of the software, send to other users, or import when other users send you date(scrolls, maps, feeds, symbols).

You can fork the whole software when you run it locally on a laptop by replicating the whole system into a directory which is a Git repository, then pushing the code to a public repository(like on Github) and then replicating the new version of the code to the whole Web by pointing the code in replicator.php which has a url for "dna.txt" to the global url for your dna.txt file. Dna.txt has all the files to copy organized by type. Replicator.txt uses that to figure out what to copy. The DNA is generated using another PHP script called dnagenerator.php. PHP files are all stored as .txt files in the directory php, and a script called text2php.php copies all of those files to the main web directory and changes the extensions from .txt to .php.

All code is edited with the program `editor.php`. This is a code editor which edits all code directly on the server. This is how all code development works in Trash Robot/Geometron. It is all in the Web Browser. Code formatting is carried out using the free open JavaScript library Ace.js, hosted

on Cloudflare CDN at `https://cdnjs.cloudflare.com/ajax/libs/ace/1.2.6/ace.js`. With this we can edit all the HTML, all the JavaScript, all the PHP, the raw Geometron, and various data files. This editor is used to make and edit all kinds of files.

To create a new file we can use "newfile" after editor.php as follows: editor.php?newfile=[filiename]. The file will appear at the very end of the list of files, with the right color coding and syntax highlighting based on the file extension.

A coffee shop-centered community code work flow is now described. A Raspberry Pi sits on the coffee shop wifi network. All users in the shop share in making scrolls, maps, symbols, feeds, pages and apps. Then any user can back all that up to a full new code instance, and push that to their public facing Github page. That copy of replicator.php is the pointed to that copy of dna.txt. The next instance of the software can use the code from this new replicator.php and it will clone the whole code base of that coffee shop, with no reference at all to the original code. Each fork creates a fully independent copy of the code.

To fork a whole full instance of the software down a level, use `fork.html`. This lets you create new branches with whatever name you want, as well as delete whole branches. Deletion is real!! There are no backups. We prevent data loss with massive redundancy of replication. If all users frequently not only replicate but pass along

all information, loss is a normal part of information life cycle and easy deletion is healthy.

There is a standard structure to applications in Geometron. An application generally uses the PHP scripts fileloader.php and filesaver.php via the JavaScript XMLHttpRequest object to edit some data file. That data file is always human readable in the sense that it has some global url which you can navigate a browser to and it will display the text in a way that can be copy/pasted via email, text messages etc. With the specific exceptions of formats specific to Geometron, all data will be in the "data" directory. In general, the format will either be some form of plain text for human consumption or in the JSON format. The actual function of the application is then in completely client-side JavaScript. For our JavaScript libraries We always use either a library which is free and open source and exists on a publicly available CDN or a library which is written by us, in the Geometron network(and is therefore public domain), and which is replicated from the "jscode" directory by replicator.php by way of dna.txt.

Another element we try to have in all Geometron applications is a text input/output, either an "input" HTML element or a "textarea" HTML element, which can be used to import and export the JSON data. There will generally also be a button to import and a button to export. Whatever other buttons and inputs will be added to manipulate the data, and in general as mod-

ifications are made, the JSON file is instantly updated in real time as you go, just like with editor.php. Then, when you press the EXPORT button, the JSON code will appear in the text area or input. You can then copy it to the clipboard of a computer or mobile device, paste into a text message, email, private file, or public pastebin, and share with anyone anywhere, and if they have their own local copy of the same application, they can drop it into the same text window in their browser and hit the IMPORT button to load the same data into their app instance.

Perhaps the simplest Geometron app is the wall.html app. This is just an html page where the whole screen is a text box in which everything you type is saved to the file data/wall.txt. That's all! The HTML code in the body of the document is simply

```
<textarea id = "wall"></textarea>
```

A little bit of CSS code sets the size of the textarea element to take up the whole screen. The function of the app is in a script element and is as follows:

```
wall = "";

var httpc2 = new XMLHttpRequest();
httpc2.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
```

```
        wall = this.responseText;
        document.getElementById("wall").value = wall;
    }
};

httpc2.open("GET", "fileloader.php?filename=data/wall.txt",
httpc2.send();

document.getElementById("wall").onkeyup = function() {
        wall = this.value;
        var httpc = new XMLHttpRequest();
        var url = "filesaver.php";
        httpc.open("POST", url, true);
        httpc.setRequestHeader("Content-Type", "application
        httpc.send("data="+encodeURIComponent(wall)+"&filen

}
```

This is very simple: keystrokes save the file. If you want to copy/paste the whole thing, just select all the text and do it, and then you can drop the whole thing into another wall on another Geometron page instance and it can be edited there and shared and copied etc. etc. from Geometron instance to Geometron instance.

Another replicator script which is useful is copy.php. To see the source code, as with all PHP scripts in Geometron, go to any Geometron server and look at php/copy.txt. This program takes as inputs a from and to field and

copies a file from one url which can be global or local to a local file location on the server. So for instance to copy wall.html from trashrobot.org to your local raspberry pi, you can navigate a browser on the same local network as the Pi to http://[the pi's IP address]/copy.php?from=https://www.t this is a long URL. But it can be put into a link with a shorter text or an image or icon, so that on a page a user can just click it like a button and they will copy the data/wall.txt file from the one server to the other. If links like this are used to copy a file from one server to the next, decentralized replication can propagate very quickly. Again, we can replicate using this tool from a Pi to a local developer's laptop, to a Github repository, to a public web page, to other local Pi servers all over the world, where the whole process is then repeated again and again. And we can be doing this with *any* file in the whole Geometron system: apps, data, JavaScript libraries, PHP scripts, scrolls, maps, feeds, symbols, all the other formats to be discussed here. And again there are now owners of any of these documents. They are not saved in a user directory and emailed to another user or shared via app from user to user. They simply are not owned by anyone. They are shared resources to be replicated or destroyed.

The power of copy.php can only become clear as more Geometron servers are linked. These can be directories on a single server, of which, remember, there can be hundreds. Imagine a collection of maps and scrolls which

each contain links which activate copy.php to replicate scrolls and maps from other pages. One can start to try to calculate the amount of information in such a network and get some shocking results at the potential power of a network like this. If developers are actively creating apps on each node, and replicating them freely and destroying and modifying them and replicating them more we get a flow of information with similar complexity of the topology of the network to Twitter interactions with retweets and hashtags etc. But in this case the amount of information is vastly higher, as the actual structure of apps in evolving. If swarms of developers are all working in this way in a coffee shop on a shared Raspberry Pi Geometron server, while they all push code to their own github repositories, which are then replicated to other swarms in other shops, all over the world, and if they are using this swarm to actively develop new applications in applied math, the potential is staggering. Like the Logopolitans in Dr. Who, a global swarm of developers all working on a math problem in parallel with code that evolves in real time can challenge classical computers at applied math problems by creating advantages of information content in a similar way to who quantum computing claims it supposedly can in the future. If such technology were developed for real time swarm computation in a way that can be applied to cryptography, the potential could be an existential threat to the status quo, especially since swarms of anonymous devel-

opers on private networks are obviously a much harder to control technology than quantum computers, and can scale up globally with zero marginal cost using existing infrastructure.

To replicate only the data from a server we can use copydata.php. It only has the one parameter, which is the "from" url, which is the location of the Geometron server to copy, not specifically of the dna file, but of the whole url, to which this script adds "data/dna.txt" to fetch the actual information to get the data. This is like replicator.php only it is just the contents of the maps/, scrolls/ and data/ directories.

We must also address how bad code is dealt with. Not all bad code is from typos. Malicious code or media of a bad nature can be intentionally created. What do we do with bad information? We totally destroy it. Not by simply editing it but by replicating a non-corrupted version, totally annihilating the bad instance. In the most extreme case we step out of the Geometron system and just wipe the whole thing clean and then go back and re-replicate. A Raspberry Pi's whole system lives on a cheap little SD card the size of a fingernail. If you really think it's infected with something so nefarious that you can't deal with it, you can always pull that out and destroy it and put a new one for just a few dollars, without any impact on your overall setup. Our goal in Geometron is not to destroy bad things but to create good things that replicate faster than bad things and destroy them

by overwhelming numbers. If a bad thing copies a million times and a good thing copies a billion times, on average we all still win.

Also, data files in the data directory can be directly edited with editor.php, which always gives us yet another way of copy/pasting the raw data, even in cases where some data is hidden in the app, or where there is no copy/paste input/output window, which might be inconvenient for some apps.

Some more technical details on editor.php should be elaborated on here. The look of the editor, which developers are likely to have strong opinions about, can be easily changed in two ways. You can edit the CSS code in the style element at the end of the file and/or you can change the theme used in the Ace.js library. The Ace themes are somewhat poorly documented but there are lots of them. You can see how they look by using the pull-down menu of themes(as well as various other properties) in the menu at the page https://ace.c9.io/build/kitchen-sink.html. You can just google ace.js themes to find this page, also. The thing to note with setting themes is that when you use the "kitchen sink" to test a theme, there are spaces, but the names used in the code have underscores. A small detail but it can be frustrating if you don't look out for it. The list of the actual names is in the github repo linked from the main Ace website at ace.c9.io. If you are willing to dive deeper into the documentation of Ace, there are lots of other neat features you can turn on

in the JavaScript. For details on fancier things, again go to the main ace website and click around on the howto and other documentation. If you have time, please do this and make a better version! What exists as of the writing of this book lacks numerous basic features which could be added, such as undo.

The code in editor.php which creates the editor itself is

```
editor = ace.edit("maineditor");
editor.setTheme("ace/theme/vibrant_ink");
editor.getSession().setMode("ace/mode/html");
editor.getSession().setUseWrapMode(true);
editor.$blockScrolling = Infinity;
```

This is about three quarters of the way down through the main script element in the file.

To change the theme without delving into Ace.js lore, just replace "vibrant$_ink"$ $with whatever theme you want from the liston$

By default, editor.php starts out editing index.html. As with all web servers, this is the file displayed automatically when a user puts the URL into their browser of your server. We keep a backup of the default user page at user.html, which allows us to copy/paste arbitrary applications into this file and then revert back easily later if we want to have the default user page. This means that by default the editor starts out with syntax highlighting for html. Ace has syntax highlighting for a lot

of languages. By default, editor.php sets themes based on what type of file it is for the files which are most commonly used in the Geometron system. So for example, if a file is in the jscode/ directory and has the file ending .js it will have syntax highlighting for JavaScript. Files in the data directory are also formatted with JS, because they're in JSON which is part of JS. Markdown files are recognized by .md and highlighted accordingly. Python files are also detected, and these are useful for building hardware control on the Raspberry Pi as well as various other science things. Arduino is close enough to C++ that using the c++ formatting is tolerable, but one thing that would be very useful is if someone wrote a highlighter for Ace.js which actually knew all the Arduino keywords. The software also automatically detects .sh files which are *nix shell script files which can be very useful for setting up things on a Raspberry Pi like an off button and robotics controls. There is an older version of the Trash Robot documented later in this work which used the Pi for control which had several of these setup scripts, but that is now unused in the current version.

We must also talk about forks and deleters, both of which are critical to the operation of our system. Any Geometron instance can be forked down into a subdirectory using the app fork.html. This creates new directories within whatever directory the existing instance is in(initially just the main web directory on the server).

fork.html is an app for seeing all the Geometron in-

stances that are in subdirectories of whatever directory you are in, making new ones, and deleting them if you want them destroyed. This is a very powerful program, where just a single click will recursively delete *everything* in a directory: every file, every directory, every subdirectory and files in them etc. There is no undo. Destroyed is destroyed. This is part of what keeps the system safe. It is easy for very bad things to evolve on the system: unsavory files of the worst kind can be uploaded at any time, content which is both terrible to see and which causes havoc with executable code. We therefore choose rather than defending against bad information to wipe it out completely. The ultimate Destroyer program is rdelete.php. Rdelete.php has one input, called "filename", which is the name of the directory to be destroyed. It has a function which calls itself recursively which uses the PHP function unlink to delete everything in its path.

The list of directories is generated using the function dir.php, which lists files or directories in a directory and returns them as a JSON array. Numerous Geometron apps use dir.php. It is used to list the various files in various directories in editor.php, to list the scrolls and maps in the main user page, and to list scrolls in the scroll editor and maps in the map editor. To create a new fork, you enter the name of the new fork in the text input field and fork.html will use mkdir.php to create a new directory and copy the file replicator.php into it, and then

give you a link to click on to that replicator. Clicking on it will run the replicator and then you can click on the link after that runs to go to the main user page of the new instance, which is now down in the new directory you just created. It is a good exercise at this point to create a new fork using fork.html, go into it and change index.html with editor.php, see your changes, then go back to the level where you started, look at fork.html, turn the delete mode on, and recursively delete your new fork you just made. Then reload fork.html and see that it's destroyed forever. You can repeat this again, this time replicating the new fork to another instance elsewhere before you destroy it. Or build several branches and create paths of replication of files between them using copy.php, put them on a shared wifi network with your friends and all open different instances, clicking on copy files to move data around. You have the power to create, evolve and destroy complex active information in a swarm of people with a shared hive mind and no property. It is not clear how powerful such technology can prove to be, but again, we must look to the claims of quantum computers and compare the information theory of our situation to see what might be possible. You do the math yourself: first look at the information theory of these swarms, then get your applied math friends together on a shared network and build a swarm up to attack a problem and see what happens.

In Geometron a slogan we repeat is "no war but the

math war". This has several meanings.

Chaos feed

Example: Duality App

This example should further clarify what we want to do with Geometron and how we can build and use apps for people to freely share information. This app example shows a number of more advanced aspects to operation than the wall example above. It is intended to explore dualities between images. Duality can be interpreted any number of ways. This is a purely artistic app, which is typical of what we want to be doing with this system.

Duality.html uses the hammer.js library, another open source JavaScript library which is used repeatedly in the Geometron system. It also has its own JSON data format which can be freely shared in raw text from user to user. Go read the source code for it using editor.php! Whether or not you read the code, try using it by putting duality.html into the browser after whatever address you're using for your geometron server, be it http://localhost/ or the IP address of your local Raspberry Pi Geometron server. You will see two images, each partially transparent. This is best on a touch screen, drag a finger sideways across the screen, and you will see a fade between one image and the other. This fade is such that the sum of the squares of the opacities remains 1, so they add like

quantum amplitudes do to make probabilities. This app is to create a sort of quantum meme: a pair of images, aligned in some way, which a user swipes to make modulate between different ratios of superposition. To say this app is a quantum meme is another way of saying it is an artistic expression which rejects binary thinking about the Universe. Any time we wish to make a commentary on any failed dichotomy, which can load up two images, move them around, scale them, rotate them, and save the JSON file created to share with other people with whom we wish to share this particular attack on binary thinking.

Chapter 5

Scrolls

Scrolls are the text documents of Trash Robot. Think of this like the Microsoft Word of the Trash Robot ecosystem(with some drastic differences). Scrolls, along with maps, feeds, and symbols, form the basis of the user-facing system of documents which are shared on Geometron. They are used to document the system, to share ideas, post articles, create ads or lists of ads, or really any type of document one can imagine.

The format of scrolls can take some getting used to for people used to, as it is not WYSIWYG(What You See Is What You Get), but rather uses the Markdown language to create formatting with code. One of the first tasks of some enterprising new Trash Robot participant will be to create the fully WYSIWYG version of the scroll editor, but for now this is what we have. Part of the goal here

is to have no documents ever be in a format other than human readable. Even if Markdown is a little awkward to read, a real live human can always read the text and someone with very very basic understanding of code can immediately turn it into fully a formatted document.

Need to address the choice to use a markup language rather than WYSIWYG: harder to use, but easier to copy. Ease of copying is more important than ease of use. Bifurcation of users and Operators allows this to be smooth in spite of increase in difficulty of use.

Symbol for scroll:

Create a scroll

To create a scroll, go to the scroll editor at scrolleditor.html, and enter the name of the new scroll in the input marked "new scroll name". When you do this a blank document should appear with black background and green text(this is easy to change if you find it annoying). Just type out your document if it's just text, hitting enter twice between paragraphs.

For further information on using the Markdown language, see

- the wikipedia page
- the official web page
- The Markdown Guide

The most annoying thing about markdown is putting images in, which you do as follows:

also show an enumerated list, a bullet list, headings, links, italic and bold, embedding html

Edit a scroll

To edit an existing scroll in the scroll editor, click on it or enter its name in the new scroll name input(no need for the "scrolls" prefix).

copy a scroll from another place online

All things in Trash Robot self-replicate and scrolls are no exception. To create a replicator we use the program copy.php which is on every TR server. To this scroll is called "scrolls" and to replicate it we make a link to "copy.php?from=[some web address of a trash robot server]/scrolls/scrolls&to=scrolls/scrolls". If you run this on a server it will fetch this scroll and place it locally in the scrolls directory. Of course as with any replication this will overwrite the scroll on the local server so be careful, as any new edits on the new server will be lost.

To copy all the scrolls, as well as maps and other data, from another TR server, we use copydata.php. On whatever server you are on, make a link to "copydata.php?from=[the

url of the domain from which you're copying]. Note that for both copy.php and copydata.php the source domain can be the IP address of a TR server on your local network.

Also, the simplest way to copy a scroll is to open a new scroll on a new server in the scroll editor, then open the scroll to be copied in the scroll editor on the old server, and just select all, copy, and paste to the new one. This functionality is part of why having everything be in a human readable format like Markdown is important.

link to a scroll from a scroll

Links from a scroll to a scroll or from a map to a scroll can be realized in Trash Robot by having the target link be either "scrolls/scrollname" or "maps/mapname", and the code in the TR user page will convert those to local links. E.g. link to terminal scroll. Scrolls can also be linked to globally by using "user.php" with a scroll specified. For example to link to the Terminal scroll on trashrobot.org we link to `https://www.trashrobot.org/user.php?scroll=scrolls/terminal`.

mathuser.php

mathuser.html

Delete Scrolls

Everything on every instance of Trash Robot can be deleted quickly and easily and with no backups. When something is deleted it's really gone. Rather than backing things up or saving to "the cloud", in Trash Robot we replicate what we want to keep and whatever doesn't get replicated will probably eventually be deleted. To get this functionality, we have a delete scroll page called `scrolldelete.html`. To delete, click on a red X. But this is for keeps! Deletion really is deletion. If you see some bad stuff on a server, just delete it. If you want to post stuff and not have it deleted, replicate it to a quiet place where no one will see it where it can get replicated back to a live page later if it's deleted.

Some technical details and use of Math

The basis of the scroll software is the JavaScript library Showdown.js, which is great, and it converts from markdown to html. So scrolls are all in raw markdown but display as html. Use of HTML tags still work as well. By default it's commented out but by editing the code using `editor.php` it is possible to turn math on using the MathJax JavaScript library, making it the same LaTeX-like markdown that is used in markdown elements in Jupyter notebooks. This allows for rapid free self replicating math papers to be created and shared on the Net-

work.

Code structure

Showdown.js, scrolls/*, filesaver.php, fileloader.php, MathJax.js, dir.php, deletefile.php,

LaTeX workflow

address stability issues for large documents, alternative editors

mathuser.php

To convert a scroll to a tex document, copy the scroll into a new directory at the *nix command line. Then create a header and footer text file as follows:

header.txt =

and

footer.txt =

in the new project directory using your favorite text editor. Now be sure you have Pandoc installed, as well as pdflatex, and any stuff that needs to be installed for latex to work.

Now convert the scroll to a .tex file using pandoc as follows:

Then concatenate with header and footer using

And finally compile from text to pdf using

and if there are no errors in the tex code you will have a printable pdf document.

add full work flow to create a book with multiple chapters, this book. Articles. Links to more information. More details on installation and use of latex, workflow with latex editors to finish the project.

use cases

The number one application which can make money to keep the system replication is: operate a market, or a tree of connected markets, which people can pay to advertise on and you promise to back up and keep replicating their ads, which are designed to replicate freely and are not property. This is the most important thing. We can run scrolls as markets. An operator need only learn to copy/paste the most basic of formatting in markdown and to operate the scroll editor, then the rest of their job is physical communication: holding up a sign, spraying tags and stencils, putting up stickers, etc. As discussed in the Street Network chapter, this role is all about being someone who can make transactions happen on the street: any transactions. They can be asking for spare change with a cardboard sign, selling stuff, sharing ideas with people, giving out free food, or promoting something, as long as they understand and can navigate the dynamics of street-level marketing communication.

document how to do a thing with a sequence of images and instructions

write an article, news or otherwise

write math papers, CS etc, operate a local technical journal on a street corner

Chapter 6

Maps

6.1 Maps

Replication replication replication replication replication replication replication replication

replication replication replication replication replication replication

maps are designed for maximum replication of themselves, ease of sharing by text and copy paste. Ease of building apps to use and edit. And they are created to maximize the Geometron user's ability to replicate other technology.

figures:

map editor

Maps are a format in Trash Robot/Geometron which

are a generalized meme. They represent an ordered list of objects, each of which has a position in a rectangular area on the screen. Each element in the ordered array has an x and y position and width all normalized to the size of the square area, as well as an angle in degrees. The other properties each element has are a url for an image if they're an image, HTML text for both if they are not an image and for alt text if they are, and a link destination which can be either a url or a map or scroll link inside the geometron system. Maps can link to scrolls as well as other maps. Also, each element has a Boolean variable "maplinkmode" which is false if it is just a normal HTML link and true if it is a map or scroll link. Maps are all stored in the "`maps/`" sub-directory of each Trash Robot/Geometron instance. They are in JSON format.

Scrolls are all stored in the `scrolls/` directory. Links inside the Geometron system are identified as to whether they are scolls or maps by the full name of the file. For instance one would link to this scroll from anywhere in the system using the name "scrolls/maps" as the destination of either a link in a map element which has maplinkmode set to "true" or in a hyperlink in the markdown format of the Scroll.

Maps are defined with the JavaScript library "mapfactory.js" which is in the "jscode" directory at `jscode/mapfactory.js`.

Maps are created in Javascript by for example in a DIV element called "mainmap" with following code:

Maps are edited using the program `mapeditor.html`. Click on all the things at random to figure out how to use that program. Save often. Copy/paste JSON code from the text area to share maps across the Internet or privately with other users. You can email JSON code, store it, copy it etc, and anyone can import it with a paste into their Geometron instance and save it locally on their server. This generalized meme format replaces both meme making software and PowerPoint as well as a large number of HTML frameworks and formats. It allows for a generalized system for encoding information on an image, which can be critical to documenting self-replicating physical technology. The three pillars of all Geometron/Trash Robot software are the Map, the Scroll, and the Symbols which are created with the Geometron language. This "symbol" is generalized to include those made in all physical media, so that includes things like lab-on-chip fluidic circuits, hybrid upcycled electronic circuits, laser cut shapes etc. Once Geometron is used to encode all human language and all symbols and also all technology, it can drive the hardware which displays maps and scrolls. When all of this lives on fully upcycled hardware, the system if fully metabolized and we can build self-replicating technology that does not have any mining, money, or property, the ultimate goal of Trash Magic.

Deletion

Maps are deleted with `mapdelete.html`. Just click "delete" to delete. Be careful, there is no backup. Also on public servers this might break, as do all file creation and editing functions from time to time. It will work instantly on a Raspbery Pi Terminal.

Replication

When you create a new map, run `dnagenerator.php`, and the next time the whole tree is replicated that map will come along for the ride. To replicate a specific map, find the URL of that map and use copy.php. The syntax is

The "from" url can be anywhere on the Open Web or anywhere visible on the local network. For example, pastebin.com or a raw code link on Github

Map editor Icon Meanings

Go through in detail how to use the editor. Describe the specifications to build a better editor, plead with developers reading the book to write a better one.

Examples

Use cases.

annotated screen shot or image of geographic map. Example of location places on a photo of a tourist map with a DC subway exit map. Example of a screen shot from openstreetmaps.org

Location of a physical object in a photograph of a place, with link to file, page, scroll or map

navigation: simple links to other documents on the local Geometron system as well as to Geometron apps like feeds or symbol programs, and also links to other web pages.

Linking from a global page to a local terminal and vice versa, with photographs of the terminal uploaded to the terminal.

memes. Just regular old memes, but with edit capability and the ability to share

graph theory diagrams using geometron symbols combined with text, use of math via MathJax js library.

more generalized replacement for PowerPoint or Keynote, but free and open and readily replicated. Give example of replicating the whole deck of maps from one server to another using the standard code replication workflow, separate from replication of the whole system.

labels on a physical object to document that object. Labels can be links to further documentation of the object, which themselves have further zoomed in detail photographs of the object. Detailed, hyperlinked fractal documentation of physical objects can really help replicate those objects, which is what our whole system is for.

Geometric memes showing how geometric symbols fit over objects in the environment, connecting physical things to geometric abstractions like the pentagon or hexagon.

conclusions

Summary of use: all our media is designed for free replication. This means that it's easy to find a thing, easy to copy it, and easy to share it. Maps help us to locate things in physical space to share by annotating geographic maps as well as photos of places. They help us replicate technology by rapidly and freely creating documents of details of the object, with links to further documentation of finer and finer elements of the thing until all parts are sufficiently documented to enable replication. By replicating the pitch deck functionality of PowerPoint we further facilitate replication by helping people to communicate stories behind what we do, helping to convince people why they should replicate. Finally, the ability to make memes easily which can be edited and remixed we build a more dynamic social media based on memes than is possible with bitmapped graphics. This enables open brands to become virulent, getting more and more people invested in seeing our projects succeed and again furthering replication.

Chapter 7

Feeds

home

7.1 Feeds

A Feed is a sequence of elements. The elements don't have geometric structure like a Map. They can be text, links, symbols, or any other kind of media. They are generally stored in the "data" directory as JSON format files which end with ".txt" so that they can be read by humans in a browser.

The Feed is a general framework for building formats, but in the basic Trash Robot server we implement a few versions.

Global Image Feed

This is an array of image urls. This is a key component of how Icon Tokens are made. We often start by doing an image search on the Web for some symbol, logo, image, or icon. We then right click the image and "copy image location" to the clipboard. Then we drop the url in the input in the global image feed to add it to the feed. Click the red "x" to delete the image. Image feeds can be exported from the text area, copied, and pasted into the same window of any other Trash Robot, imported and used anywhere on the Network. Since this data is just text it can be sent via text message or email so that feeds can be privately shared. The local image feed is stored at `data/imagefeed.txt`

We can make global image links in this Feed by uploading images to www.imgur.com, then right clicking the image to get the url and putting that url in the image feed. This method is used to document much of the Trash Robot system or for general rapid information sharing.

local image feed must also be added, both of these linked to all the different places they are used, with description of how they are used in other apps.

Link Feed

This is a feed of "links" in a general sense which can be images, links, or just text. They are edited using the "operator screen", which should be in the link feed itself, and can be found at `linkfeededitor.html`. Each element has three fields: "href", "src", and "text", which are the url the link points to, the image if there is one, and the text. The data are stored on each Trash Robot at `data/linkfeed.txt`. As with the image feed, the whole feed can be copied, pasted, imported and exported using a text area, but in this case it is on the editor screen not the feed display. The input is used to put in urls of other link feed files. These can be anywhere on the Web. This can be used to make anonymous pastebin links which are link feeds which can display on any local Trash Robot without ever posting to a global server, for private exchange of link feeds. f ## Text Feed

The Text Feed is used for a number of Trash Robot applications. In spite of its name, it is not just a feed of text, but consists of three feeds(arrays): "text", "src" and "href". These really are what they sound like, three feeds in one. Users can add links, add images, add text, or delete any of them, and can copy and paste and share and import feeds. Text feed has a number of functions in the Trash Robot/Geometron system. It is used for the Map Editor as a source of links, images, and text which do not need to be entered in a keyboard. It is also used in

the Poetry Engine and Duality. These are documented with the poetry engine scroll and duality scroll.

Chaos Feed

Chaos Feed is a user friendly text feed. Type in the input to post. Hit red "x" to delete. Nuke the feed with the explode emoji. Reload with the arrow loop emoji. HTML works, so you can manually enter html for links and images, allowing a link out to be added. Chaos Feed can be set to be the top level of a Trash Robot Server for text feed sharing mayhem and fun. Chaos feeds are stored at `data/chaosfeed.txt`.

Icon Feed

This is a critical feed for the overall system work flow, as it is how we share the Token Icons which are printed into clay. See the workflow map for links to the elements of the process by which these are made. Here again is where the copying, pasting, importing and exporting of feeds is very important. Users can create a whole feed of icons locally on a private server, then send that via private message to other users anywhere in the world, who can then edit on their own private servers, without any data ever leaking to the public Internet, while still having no users and no databases on each individual server.

Symbol Feed

This is not really a feed in the strict sense above, but it behaves like a feed in the user interface. Every time a symbol is saved using `symbol.html` an SVG and PNG file are both created, and these are saved in a directory called `symbolfeed/`. These can be saved locally and then used for anything. The pairs of files are also used when programming the Dremel laser cutter to directly create laser cut acrylic geometry shapes. The SVG files alone, with different layers as different colors are used for the cut and etch layers when making laser cut shapes ordered from Ponoko.com. Clicking on an SVG file also loads it up into `symbol.html`, including the structural JSON information which sets styles and positions of the symbol.

Wall

The Wall is a feed of one element. It is just a text document, stored at `data/wall.txt`, which is edited and read by users. Type to edit. Delete to delete. There are no users, no databases and no logins. Just information freely shared.

Chapter 8

Symbols

Generalized symbols...

this is just an overview chapter of what GVM and hypercube and generalized symbols are, why we care, how they work, survey of the applications, separate from the various applications which all get their own chapters.

generalized symbol as a concept: pixels on screen, architecture, microfabrication, circuit fabrication, CNC, agricultural automation, biomedical automation,

GVM as an idea to address this maximally generalized symbol

Hypercube

detailed Hypercube structure, whole thing, possible extensions

Geometron is a geometric meta-langauge. That is it is a language for building geometric languages, which

is itself expressed entirely through geometry. The three main components of Geometron are the "Geometron Virtual Machine", or GVM, the Geometron Hypercube, and the Cursor.

The cursor is like a "turtle" in other geometry languages such as Logo, it is a collection of global geometric variable which can be acted on. These global variables might be the position of a cursor in an xy plane, of a cube in an xyz space, of a physical robot tool, or of any technology with geometry in it. The geometric actions discussed below are on this abstraction, which can also be thought as a "tool", which in some cases it literally is. However "tool" can be misleading since the state of the cursor is not simply a position but can include variables like "step size" which are abstract and not embodied in the physical state of a physical tool.

The GVM is an abstract "machine" of pure thought which carries out geometric actions. The actions of the GVM are arranged geometrically into two cubes(hence "hypercube") each composed of 8x8x8 cells. Each of these 512 cells has an address based on its geometric location. All addresses start with the number "0", and indexes count up from 0 rather than 1. One cube has addresses from 0 through 0777, and the other has addresses from 01000 through 01777. Each cell in the Hypercube represents either a geometric action or a list of addresses in the hypercube which the GVM will execute in order. The GVM is fed a "glyph" which is a sequence of Hyper-

cube addresses, and it executes the actions in the glyph in order. Some actions are therefore themselves glyphs, which in turn can be composed of sub-glyphs and so on. Infinite loops can easily be created this way.

The zero cube from 0 through 0777 is the "Action Cube", which represents actions themselves we wish to use the GVM for. Each action has a corresponding symbol in the symbol cube or 1 cube from 01000 through 01777, which is a glyph designed to communicate the meaning of the action to a human user.

The Action Cube is divided into layers, which organize the types of information into different categories.

0-07: left open for future use.

010-037: "root magic", or actions which act on the Hypercube itself or the document being used to interact with Geometron.

040-0176: Printable ASCII. These are used to either place a ASCII character on the Word Stack(to be printed using the action at 0365) or to identify the action taken when a key is struck on a keyboard connected to a GVM

0177: do nothing

0200-0217: Fixed shapes, glyphs composed of sequences of actions which are left alone for general use, not edited frequently

0220-0277: General shapes. Glyphs used to create geometric languages such as schematic symbols, graph theory arrows, cross stitch patterns, flow charts etc.

0300-0377: Primary two dimensional geometric actions used on computer screens. These are further broken down into the Action Tablet documented below.

0400-0477: Machine actions. These include manipulation of global machine variables such as step size or speed of movement. They will generally be machine specific although we will document the specific values for the Token Printer below. IN actual Arduino code these are generally mapped to ASCII, as is the set below.

0500-0577: Shapes made up of sequences of actions in the 0400-0477 space. These can also include the entire rest of the Hypercube, and allow for a combination of 2d, 3d, and mechanical actions, so that for instance a cutting tool path can be saved as a 2d image in addition to the direct control of the machine using Arduino.

0600-0677: Shapes made up of sequences of actions in the 0700-0777 space. These can be used to build whole complex languages of three dimensional shapes for 3d printing design as well as VR and AR.

0700-0777: 3d actions. These actions are used to construct three dimensional abstract geometry. Specifically in the Trash Robot Geometron system they are used to build .x3d and .stl files which can be used for VR and 3d printing respectively. Learn more from 3d scroll. Or just try out the system on `voxel.html`.

This arrangement then maps to symbols which have meaning to humans pointing to the actions. Note that for the ASCII values this maps to a font of the printable

ASCII, which can be in any human language. Building more complex human languages like the CJK characters or fractal Arabic calligraphy, we can add whole cubes to the Hypercube. The range of characters between 01040 and 01176 are called a "font".

The software presented here allows us to use Geometron to make computer files in either the vector graphics .svg format or the bitmap format .png of all sizes, styles and shapes, save them, edit them, replicate and share them. These can be used as icons, as symbols in Maps, as figures in Scrolls, as art, or as patterns which can be directly transferred to a laser cutter to create all the laser cut acrylic shapes which are used for the arts and crafts projects presented in this work.

In addition, the software presented here uses the same Hypercube and GVM to create, edit, and replicate 3d files which are saved in .stl for 3d printing or x3d for VR and AR or games.

Perhaps most importantly, however, the software presented here allows us to create generalized symbols using Geometron which use physical machines to make physical matter with the symbols we create in Geometron. Extending the system to more cubes in higher address spaces can allow for a totally generalized methodology for creating geometric languages for any kind of fabrication, media, display or design technology.

Symbols are created using `symbol.html`. One edits a glyph which is displayed in a canvas element on the screen

by either hitting keys which correspond to actions or soft keys on the screen. Another canvas element displays the sequence of symbols corresponding to the actions in the glyph. The arrow and delete keys are used for editing by keyboard, and there are equivalent symbols on the soft key menu.

Glyphs are stored in software as strings formed from sequences of numbers separated by commas. As you edit a glyph the string will change in an input. That string can be copied to the clip board and saved, emailed or text messaged to someone who can paste it in their own Geometron software to copy the symbol.

Actions of symbol.html

Learn to operate Geometron with the Hello Geometron Scroll

read a book ish document of an obsolete version of Geometron in the Book of Geometron here

8.1 Geometron

The purpose of science and technology is maximize the ability of the human mind to interact with our physical environment. As our society becomes increasingly technological, we find that our power in that society is based on our ability to control machines. As machines become increasingly complex, the groups of people able to control them become more and more specialized, and most

people lose more and more control with each new more complex generation of machines. In this work we seek to push back against this trend by building a language for the natural control of machines by people. The intent here is to build not just a language but a linguistic framework which can be used in many context on many technologies for *all* people to wield more control over *all* machines. By gearing the operation of the language directly toward industrial automation machines it is our intent to build a pathway for ordinary people across Humanity to have direct control of the means of industrial production at the local level.

Geometron is a language for humans to control machines using the natural structure of both machine operation and the human mind. We consider three structures to be fundamental to how our minds organize thought which are mirrored in machine operation. Those are:

1. Cataloging or listing of things. One of the most basic things we do when we organize information is make lists of things: the periodic table of elements, the alphabet, the organization of the base 10 number system, division of light into discrete colors, the system of organizing mailing addresses etc. This is a structure mirrored in how modern computers are constructed: information is stored based on an addresses. Addresses in a modern computer also often contain other addresses, direction the operation of

the machine from one point in memory to another. While arbitrarily complex and varied types of information can be stored at a memory address, it is useful in both machine architecture and human language to impose a structure which is universal across many specific elements. For instance, the mailing address format can point to a vast estate, a PO box, an individual, a corporation, etc. but it's useful to have all these under the one unified system of addressing. Another example of this is library call numbers, which map all of human knowledge to a simple string of numbers and letters.

2. Naming of things. Perhaps the most fundamental form of abstract human language is using a word to point to a thing. In particular the ability of the human mind to chain the naming process together is fundamental to human thought. For example, we use names to point to people, but also use the word "person" to point to each individual person. That is, words are used to point from any kind of thing to any other kind of thing. The naming process creates complex networks of relations between things which we use to build up our whole view of the world. As with the cataloging process, this function is mirrored in the architecture of our machines. The languages we use to program machines are constantly using pieces of information, called

variables, which represent some meaning separate from the name of that information. For instance, in commonly used web software to have something called a "shopping cart" which is an abstract thing pointing to a list of things a person plans to buy. In addition to information technology we consider the mappings which take place in machines where something like a pull on a steering wheel can map to a whole sequence of things, from a computer to power steering, to the many machines which respond in concert to such actions. This is not exactly naming, but has a similar structure, where things are mapped to other things.

3. Geometry. In this work we assume that certain types of geometry are fundamental to the structure of both human thought and to how all machines we build operate. Most of the structure of the language presented here is based on either considering geometry which feels natural to the human mind like "left" and "right" and geometry natural to the operation of a machine like "distance between plants" in an agricultural robot or distance between pixels in a display.

The Geometron language uses all three of these elements to construct a geometric meta-language: a language for building languages. The language consists of

geometric *actions* organized according to an addressing system called the Geometron Hypercube. Each action is represented by a symbol which is itself drawn using the actions of Geometron.

In order to build a language that is as natural as possible for people to understand we begin with what geometry we all consider natural and work from there to a specific implementation. The most basic geometric action we consider are movement by one unit in the basic directions "forward", "backward", "left" and "right". These words in human language have meanings which depend on context, and that is how they work in Geometron as well. "Go forward" is a universal message which can be expressed in any human language, and yet its precise meaning depends on several unspoken assumptions. There is an implied direction-state of the speaker or listener. There is also an implied angle from which "left" and "right" deviate from "forward", and an implied unit which one might move. For example if giving driving directions in a city on a grid, we might say "drive 10 blocks forward along the street you're on, turn right, and go 5 blocks, then left and one more block". By abstracting this language we can generalize to something like "10 units forward, 5 units right, 1 unit forward" and this can apply to paces walked, pixels on a screen, or trees in an orchard. That is, the idea of a grid of points which we navigate with simple lefts, rights, forwards, and backs, is a *universal* idea, which does not depend on special

technical knowledge, but which we can expect anyone to understand even without written or technical language. All this also implies that there is a position of the listener/speaker who is depicting this sequence of actions.

To summarize the previous paragraph, we pose that there is a geometric "state" which all humans understand how to manipulate. This state consists of position, orientation, a basic unit like "blocks", "feet", "steps", "football fields" etc, and an implied angle by which we turn(usually 90 degrees unless specified otherwise). We can then create symbols to represent the basic operations of movement of this state as follows:

- go forward 1 unit
- go back 1 unit
- go left 1 unit
- go right 1 unit
- rotate left
- rotate right

We construct symbols for these which get as close as possible to using a universal language which all people will recognize, in this case arrows. These symbols are then:

These symbols are intended to be as universal as possible, avoiding both the selection of a language like English to draw from as well as machine-specific or technical mathematical language. Each of these symbols is itself

made up of geometric actions which will be documented below: changing of scales, rotations by various angles, and natural geometric constructions such as circles, arcs, line segments, and paths of joined segments. We immediately see the power of building a language for creating symbols which is written in itself. Just as human languages contain dictionaries consisting entirely of definitions written in the language being documented, we can create a language of geometric actions built entirely of symbols using those actions.

This is the most fundamental power of the Geometron language: to make arbitrary symbols which can then be used to program all sorts of other more general machine operations. Once we have the ability to make arbitrary symbols, we can immediately use those to construct commands for operating any machine. For example, if we have a simple robot that simply consists of a stage that can be moved in two directions with a tool over the top of it, we can label one with a red arrow and one with a green arrow, select some basic movement unit, say 1/100th of the total span of the stage, and start writing programs by using a sequence of these arrow-motion actions. In this language, a program that moves left on the red direction, then out in the green direction and back might look like this:

This simple concept can be very powerful when combined with direct controls of a machine. A user who starts moving labelled levers to control a machine can immedi-

ately enter a sequence of actions based on the experience of operating that machine to automate a process. This is the level at which automation can be truly controlled by the people of the world, and is a necessary condition for people to ever control their own means of automation.

With the goal in mind of being able to build a universal symbolic language for controlling machines, we may now proceed to documenting how that language is constructed. To reference the introduction here, there are three tools we will use: assigning addresses to things, assigning symbols to actions(in essence a naming of things), and the use of universal geometry of Nature. When we speak of "actions" in Geometron it is useful to have some sort of object which is carrying out those actions. That object is called the "Geometron Virtual Machine", or GVM. Depending on context, the GVM might draw on a computer screen, control a robot, create 3d structures, control the function of buttons and keys, encode writing in human languages and edit the structure of Geoemtron itself. In all cases, each action has a symbol, which is itself just a sequence of actions. Each action has an address which is made up of three numbers, each of which can range from 0 to 7. Types of action are organized by the first of the three numbers(e.g. machine movements vs. 2d geometry vs. 3d actions etc.). This system divides actions up into tablets which have an 8x8 array of "boxes" each of which maps to some kind of geometric action. The 8x8 array is an aesthetically pleasing way to

organize information which pushes the bounds of what is comprehensible to look at for a person while also fitting the convenience of using powers of two for interaction with computers.

Each action address is matched by a symbol address which adds a "1" to the left of the address. All addresses start with a "0" to indicate the format as a Geometron address, and to indicate for computer code that this is a base 8 system. Thus for example if the action for "draw a circle" is at address 0341, the address 01341 will contain the sequence of actions which draws the symbol representing the drawing of a circle. This ability of Geometron to pivot meaning radically based on context mirrors the power of human language in contrast to most computer languages.

The structure of addresses has its own geometry. A stack of 8 tablets, each represented as an 8x8 array makes an 8x8x8 cube. We have two of these cubes, the "action cube" and the "symbol cube". Again while this might seem cumbersome and one can get lost in the details of the exact functions of addresses, this is intended to create large scale structures which anyone can understand: every action is a location in one of two cubes of 8x8x8 cells. Ever action in the symbol cube consists of a sequence of actions which create the symbol representing the matching action in the action cube. That is, each cell in the 8x8x8 symbol cube contains a sequence of addresses in the action cube.

The action cube is divided into different types of function. At this point the need to give the language enough specificity to actually function requires that we dive in in some depth to the exact functions used. The address ranges in the Action Cube are as follows:

- 00-037: Actions on the functioning of Geometron itself, such as moving a cursor, deleting a symbol, or clearing a symbol. Taken together these types of action are called "root magic".
- 040-0176: ASCII. These numbers correspond to the printable characters standard on computers in the English speaking world. ASCII stands for "American Standard Code for Information Interchange" and is a universally recognized way to encode English characters on computers. The contents of these addresses are used to map key functions on a keyboard to anything in the Geometron Hypercube. For instance the address representing the letter lowercase 'a' is 0141, and in the default configuration that contains the address for "move forward by one unit", which is 0330. So when a user strikes the 'a' key, that adds the action "move forward one unit" to the sequence being edited. The symbols corresponding to these locations in the Action Cube are then symbols of the printable characters, which represent a font. That is, for example, the address in the Symbol Cube corresponding to the

lowercase letter ‘a’ is 01141. In the address 01141 in the Symbol cube we will find a sequence of actions describing how to draw a lowercase letter ‘a’. This creates an immediate way to handle all human languages and keyboard mappings, as we can simply edit the contents of the Geometron Hypercube to both change key functions and change all the printable characters in a set of 95 from space bar to tilde.

- 0177: do nothing
- 0200-0277: Shapes. Each of these addresses contains a sequence of actions. That is, rather than computer code directly doing something, when one of these actions is triggered, it maps to a sequence of actions stored at that address. This can lead to infinite recursive loops, and it is useful to add functions that break infinite loops or avoid them. Formally, the sequences in this range can reference any address in the whole hypercube but by *convention* they are taken to generally be two dimensional constructions out of which symbols are constructed.

- 0300-0377: Symbol action geometry. This is the heart of what makes the whole system work. These are the actions which are used to create symbols in the various two dimensional computer formats: canvas, svg, png and base 64 encoded bitmap. In the implementation presented here each of these

addresses represents a function in JavaScript which can both edit a canvas element in HTML and edit a string which can be saved as an SVG file. This tablet will be documented in detail below.

- 0400-0477: Machine Actions. These can be used to control any machine, and generally consist of direct physical actions like "move robotic stage left one unit", or "turn on motor for one unit of time". In the implementation here they are either functions in Python which control the GPIO pins of a Raspberry Pi or are instructions to a GVM implemented in Arduino.

- 0500-0577: Shapes made up of machine actions. These are simply sequences of addresses anywhere in the Hypercube but by convention are all either machine actions in the 04xx range or other elements of this range itself.
- 600-0666: Shapes made up of 3d geometric actions. These are again just sequences of actions, by convention being in the range from 0700 to 0777 which are 3d constructions.
- 0700-0777: 3d geometric actions. These are used to edit 3d files or do 3d geometric actions. In the implementation presented here, what is edited are x3d files(formerly VRML) which are used for virtual reality and augmented reality, as well as constructions with the THREE.js library which can

export to .stl files for 3d printers.

As stated above, the heart of the system is the symbol constructions in the 03xx tablet. We now start with showing the whole tablet as follows:

These are broken up by category. The fourth row, the range from 0330 to 0337, are motions: move forward, back, left, and right, rotate left, rotate right, shrink and grow by the scale factor. The second row sets the scales: the factor by which a unit is grown or shrunk. By default this scale factor is two: grow actions double the unit, shrinks halve it. Other scales are 3, 5, the square root of two, the square root of three, and the Golden Ratio. Rotations to the left and right are by an amount set by the symmetries: fourfold symmetry is 90 degrees, five fold symmetry is 72 degrees, and six fold is 60 degrees.

These are all related! The diagonal of a square is the square root of two bigger than the side of the square, the diagonal of a hexagon used to make a six pointed star in it, is the square root of three bigger than each side of the hexagon. The diagonal of a pentagon which is used to make a five pointed star is the Golden Ratio bigger than the side of the pentagon. These relationships between simple symmetries and these ratios are fundamental to the structure of the Universe as we perceive it. Every culture in the world uses these fundamental symmetries for art and communication. By using these scales and movements we can move our virtual "cursor" to any di-

rection, with any angle, in any location at any scale. The third row sets styles. There are 8 styles, each of which has a fill color, line color and line width. By default these are black, thick black, then the rainbow colors in order: red, orange, yellow, green, blue, and purple. However, they can be set to any values. Final actions in relation to symmetries is halving angles, doubling them, trisecting angles and tripling angles. Between these and the angle 36 degrees it is possible to get any number of degrees down to the single degree.

With all this, we can proceed to the constructions. The most basic constructions are circles, dots, line segments, and arcs.

Also, bezier paths.

0365 word, vs. font with 01xxx, paths closed an open,

02xx: cursors, arrows, special scales, square,

products: laser cut shapes, icons, 3d assets for vr and ar, 3d print objects, clay fabrication with nail, generic 2.5 d printing, agricultural robots, electron beam lithography, quantum processor programming, specific fonts: laser cut, clay pixels

how robot code works with all this, examples

root magic: how to edit, how it works, very brief

hello geometron, very brief with link

how the system works with actual implementation, but very brief

examples: circuit diagrams, quantum gates

what this can do and why you should care These are broken up by category. The fourth row, the range from 0330 to 0337, are motions: move forward, back, left, and right, rotate left, rotate right, shrink and grow by the scale factor. The second row sets the scales: the factor by which a unit is grown or shrunk. By default this scale factor is two: grow actions double the unit, shrinks halve it. Other scales are 3, 5, the square root of two, the square root of three, and the Golden Ratio. Rotations to the left and right are by an amount set by the symmetries: fourfold symmetry is 90 degrees, five fold symmetry is 72 degrees, an

Chapter 9

2d Web Graphics

9.1 2d Web Symbols and Icons

Style

Each instance of the Geometron Virtual Machine has a style object, which defines 8 layers, numbered from 0 to 7. Each style has a line color, line width, and fill color. The properties of the style object are stored in the JSON file data/currentjson.txt which is used by the app symbol.html to edit graphics which are used by the rest of the Geometron system.

While the style app edits the data file currentjson.txt which applies to the whole Geometron object used for symbol editing, the importing and exporting of data for sharing with other users only includes style information,

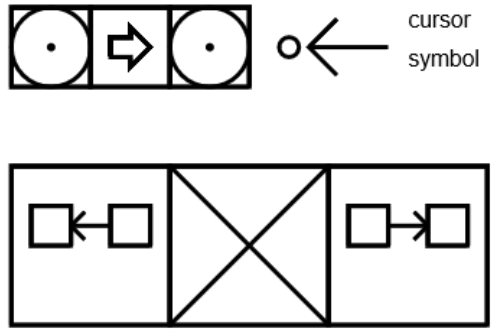


Figure 9.1: cursor edits.

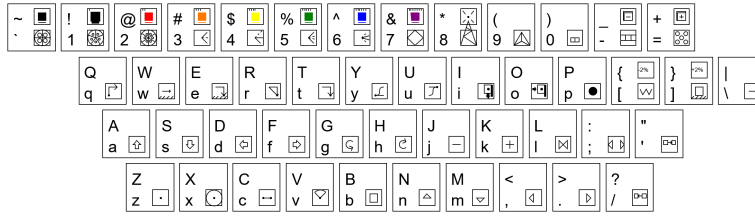


Figure 9.2: keyboard.

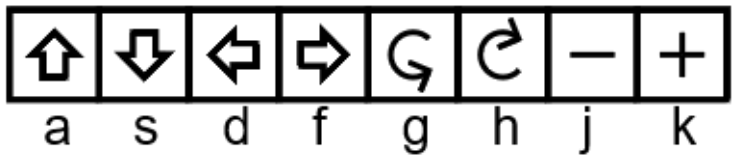


Figure 9.3: Movements. Arrows move along directions of the lines in the cursor. Rotation is by the unit indicated by the cursor wing angles. Scale actions are by the current scale value as shown by the dot positions on the cursor. Letters shown indicate the keys which map to these actions on a QWERTY keyboard with the default settings.

without the rest of the JSON data. This allows styles to be separated from the rest of the information for the purposes as usual of building a robust remix culture where Geometron users can constantly be sharing each piece of the system. The EXPORT button will always post the current style JSON in the window in the lower left of the screen. IMPORT will import the data, and RESET returns it to a default state. Try creating your own new style with unusual line widths and colors, then exporting

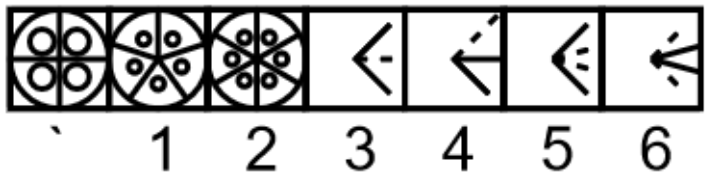


Figure 9.4: Angles described by symmetry glyphs. This also shows the actions to bisect, double, trisect and triple angles, and what keys are used to activate each geometric action.

it and saving it offline, sharing it with other users, etc.

Colors are in the format of HTML/CSS/JavaScript, and can be either names of colors like "red" or RGB color values like "#00ff00". This last format is a number in base 16 which has three 2 digit numbers in it(numbers between 0 and 0xFF), where the three numbers are values of red, green, and then blue. So black is #000000 and white is #FFFFFF. Any value where all three numbers are the same, like #808080 will be a shade of grey. Colors can be partially transparent by adding a fourth hexidecimal number which represents opacity. So fully opaque red is #FF0000FF, and red with half transparency is FF000080(80 because 8 is half of 16, this is actually 128

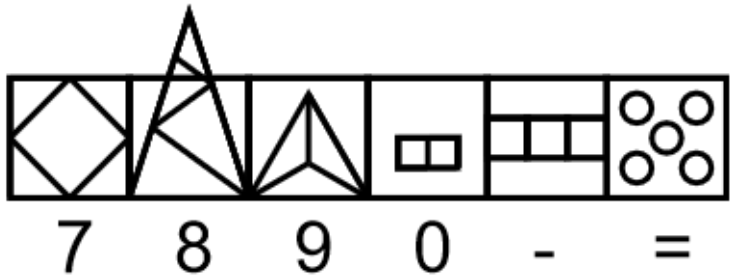


Figure 9.5: Scales, along with keys used to map to them in default configuration. There is no relation between the numbers on the keys and the mathematics of the scales. The scales shown are, from left to right, the square root of 2, the Golden Ratio, the square root of 3, 2, 3, and 5.

in decimal).

Graphics Setup

The next section of the JSON we want to know how to edit in order to be able to make useful graphics is the setup, edited in the app setup.html. Setup edits five numbers, all of which are in units of pixels: x0, y0, unit, width and height. Width and height are the width and height of the graphics file currently being edited or created. When a Geometron glyph is drawn with a given GVM, it starts with x and y equal to x0 and y0. Set-

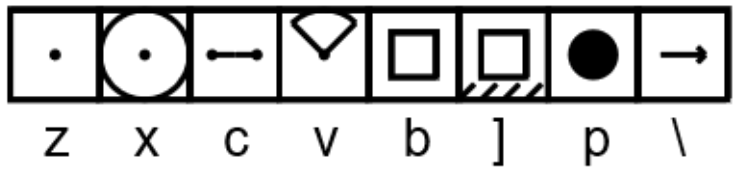


Figure 9.6: Basic drawing actions, along with keys used in default configuration to activate them. From left to right the actions are: draw dot, draw circle of unit radius, draw line segment of unit length, draw arc between cursor wings, draw a square, draw a filled square, draw a filled circle, and draw a line segment while moving forward one unit.

ting these two values is therefore effectively setting the horizontal and vertical offset of the field of view of the symbol. When we activate a pan function within the symbol.html app what we are really doing is modifying the values of x0 and y0 in the JSON file. These are done manually in this app. Finally, unit describes the initial unit value of the GVM. This is essentially the scale factor. So again when we activate the zoom functions in any other symbol editor what we are really doing is making changes to the variable unit in the global JSON file.

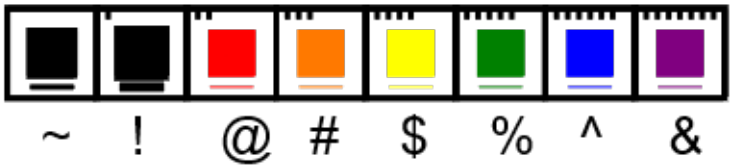


Figure 9.7: Layers. Each layer has a line color, line width, and fill color, all of which are set with the Style object using the Style editor app.

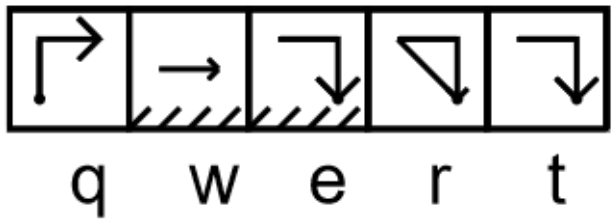


Figure 9.8: Path actions, with keys used to activate them in default state. From left to right, actions are: start path, draw line segment in path, close a filled path, close an unfilled path, and terminate a path without closing it.

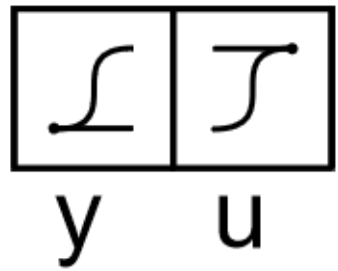


Figure 9.9: Start a Bezier Path and terminate it with the y and u keys.

The app setup.html has five fields in which to enter the numbers for the five values. There is also a reset button to restore default, with a 600 by 600 pixel square and 80 pixel unit centered in the center.

The Shape Stack

The final section of the global JSON file which defines the settings of the symbol app is the shape stack. This is a subset of the hypercube, and is stored both in the JSON file data/hypercube.txt and also data/currentjson.txt. When vector graphics files are saved, this shape stack is stored inside them so that they can be reloaded with the whole stack.

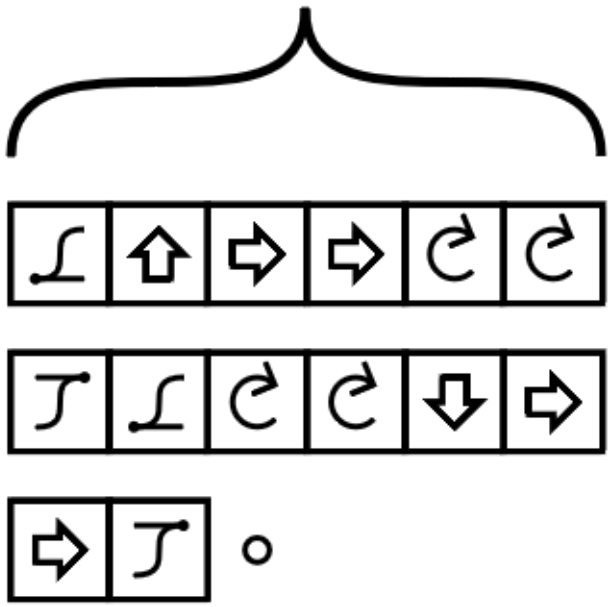


Figure 9.10: Demonstrating the power of Geometron to make useful symbols with Bezier paths quickly and easily: a twiddle bracket.

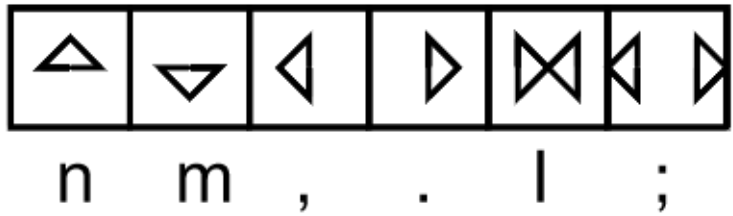


Figure 9.11: Pan and zoom the field of view.

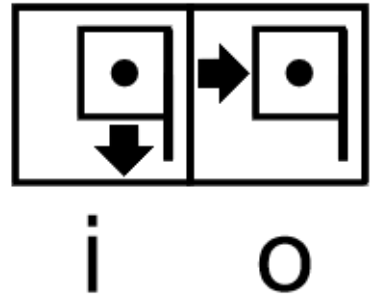


Figure 9.12: Drop a flag, return to flag.

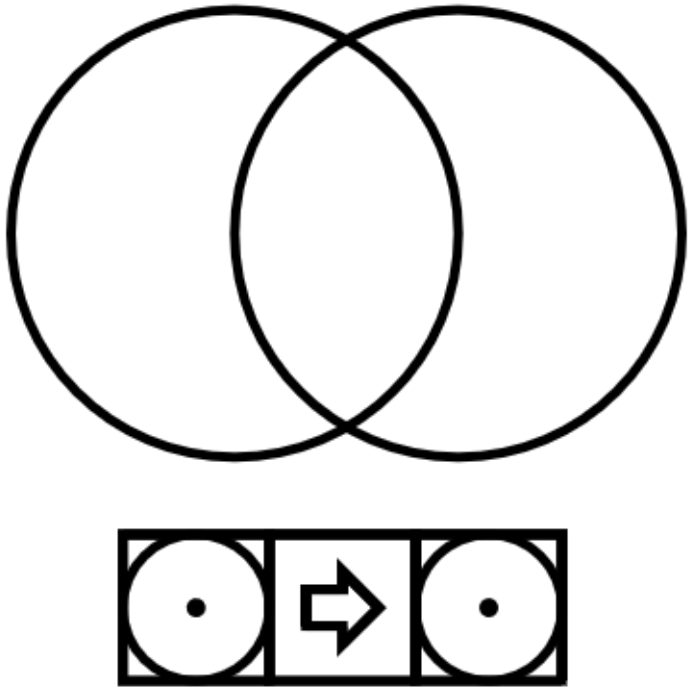


Figure 9.13: The "hello world" of geometric programming, the Vesica Piscis.

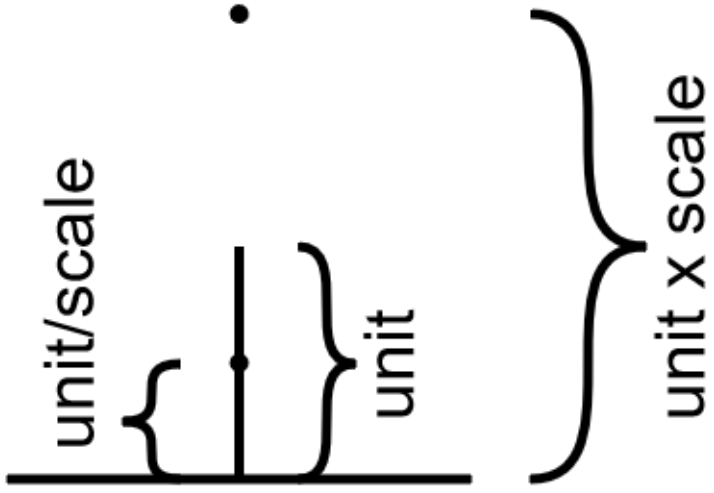


Figure 9.14: Cursor scale.

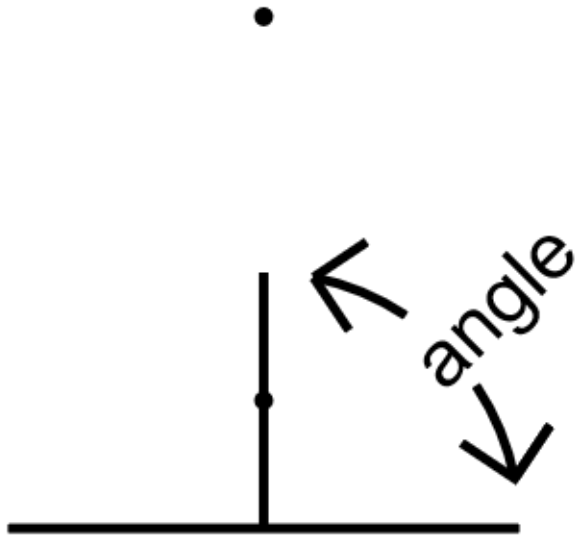


Figure 9.15: Cursor angle.

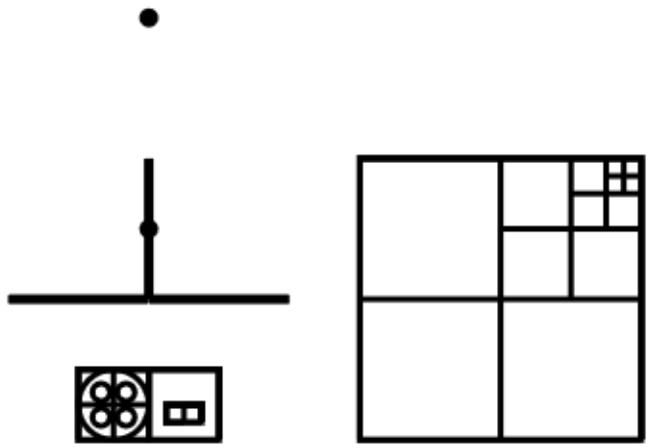


Figure 9.16: Cursor square.

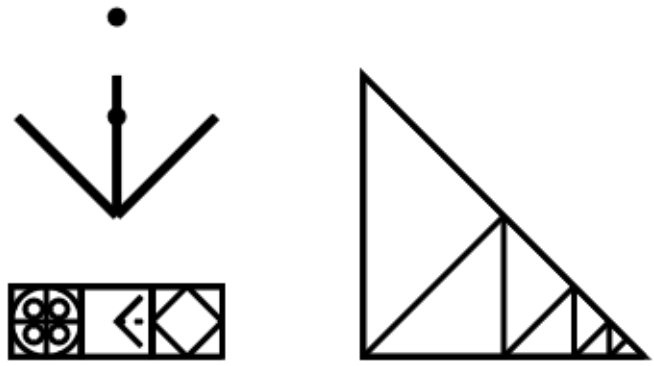


Figure 9.17: Cursor root2.

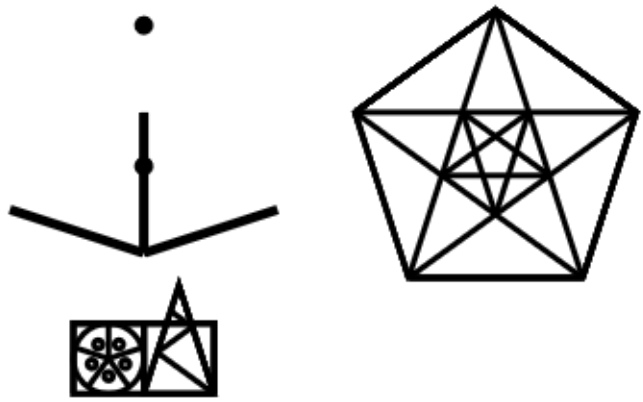


Figure 9.18: Cursor golden ratio.

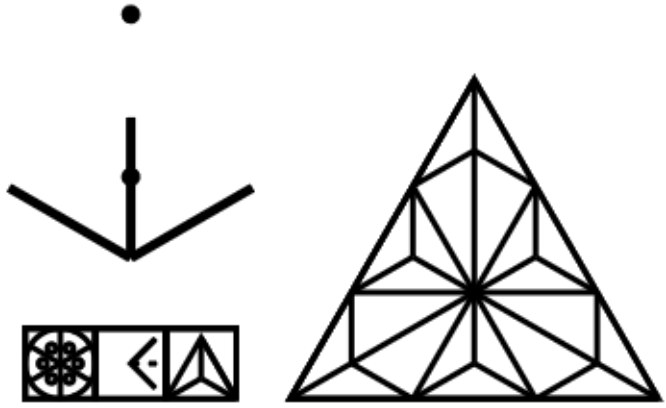


Figure 9.19: Cursor root 3.

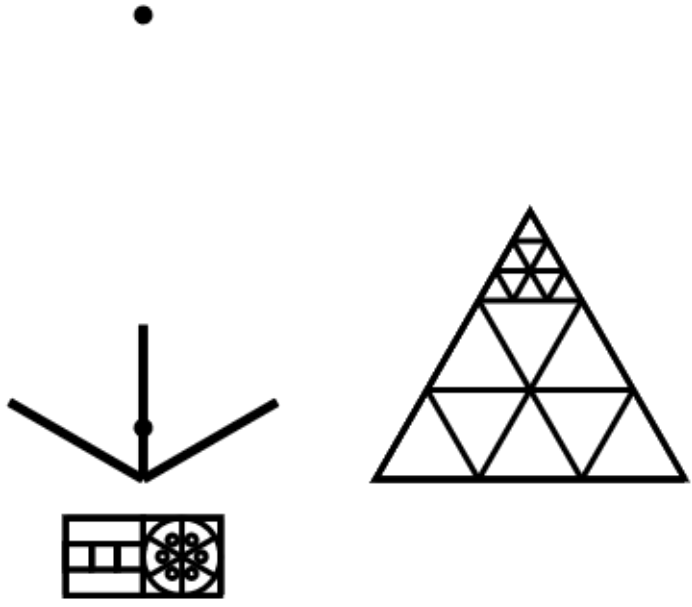


Figure 9.20: Cursor 3.

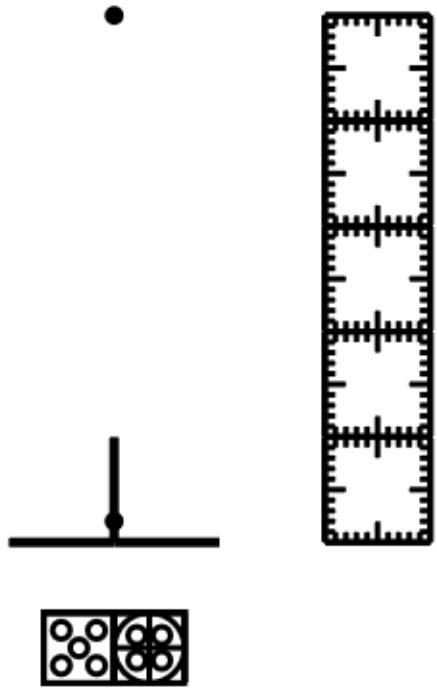


Figure 9.21: Cursor 5.

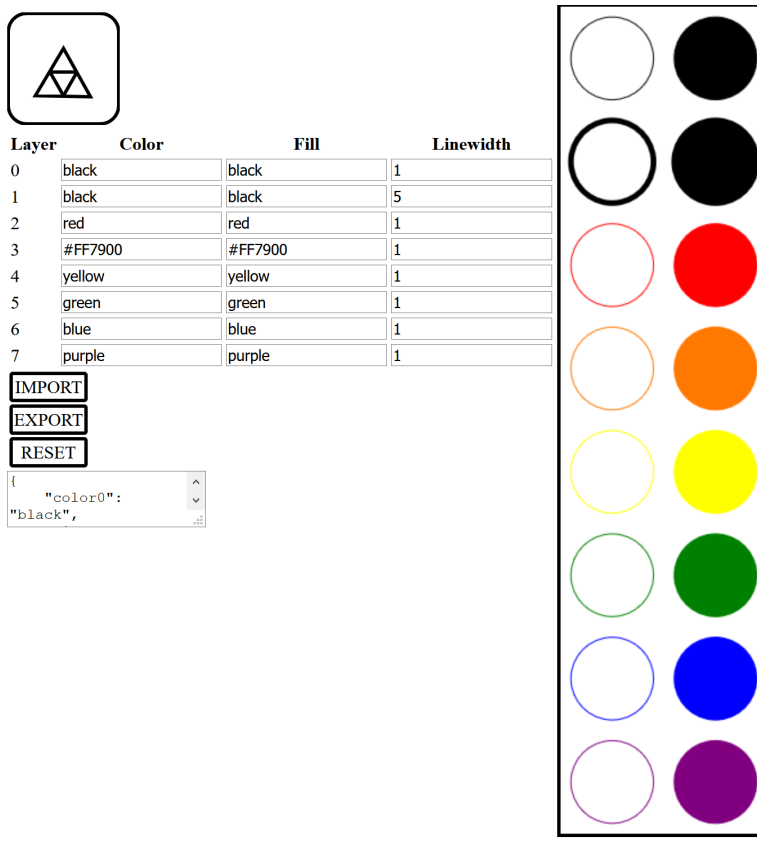


Figure 9.22: Screen shot of the style editor app at styleeditor.html. The display on the right hand side of the screen shows an unfilled circle and filled circle of each layer's style. The text area in the bottom left of the screen is used to import and export style data, which can be saved offline and shared with other users via text message, email, etc. The RESET button resets the style to a standard setting, which will erase any changes made to the existing style. Enter new values into any field to immediately change it.

Exchange of shape stacks. Manual retrieval from .svg files. uploaded svg files and click to load the whole json.

control panel

hypercube editor for manual direct control

font editor, font.html editing data/hypercube.txt

keyboard editor

symbol trace, saves to symbol feed, which can get loaded, links to globalimagefeed, localimagefeed

work flow with glyph strings, copy paste from all the inputs to all the inputs

laser cut designs

- hello world vesica piscis
- symbols, how they work with hypercube,
- editing, cursor, keyboards, control panels, modes
- symmetries and scales, different methods of geometron(AG)
- cursor,movements, basic constructions(segment, circle, arc, dot)
- layers, colors, lines, style json, working with styles, transparency in hex colors, finding colors
- bezier curves
- paths
- character stack
- fonts
- flags
- tracing symbols from images

- editing the hypercube and shape table, sharing them, import and export of hypercube, sharing of byte-code
- canvas,svg/png/base64 workflow, laser cut shapes production, practical graphics for manuscripts and web, iconsymbols, usage in jupyter notebooks, how the JSON embeds in the SVG, how the symbol feed works, how the setup of JSON works,
- control panels, softkey interfaces, writing geometron apps, how to replicate in other systems from scratch
- examples of using the GVM in JS, documentation of geometron.js, how to use just as a js library to build whatever you want, a list of things someone needs to build to make all this a lot better, how to do that

Chapter 10

Shapes and Fonts

10.1 shapes and fonts: examples

this chapter is only examples. the whole workflow and structure is in the previous chapter, including how to make a font, edit, share, examples of very elementary symbolic languages. This is just a gallery of examples with links to the actual files for users to copy/paste and use.

- editing shape stack, workflow, how to share, upload, download, send and store, same with fonts, connect to hypercube
- basic shapes built into 0200 thru 0217, how it's all connected to 01xxx
- pixel fonts

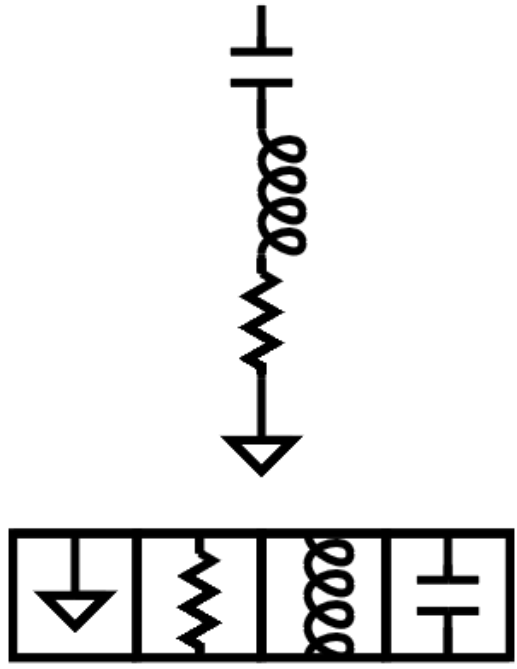


Figure 10.1: RLC circuit.

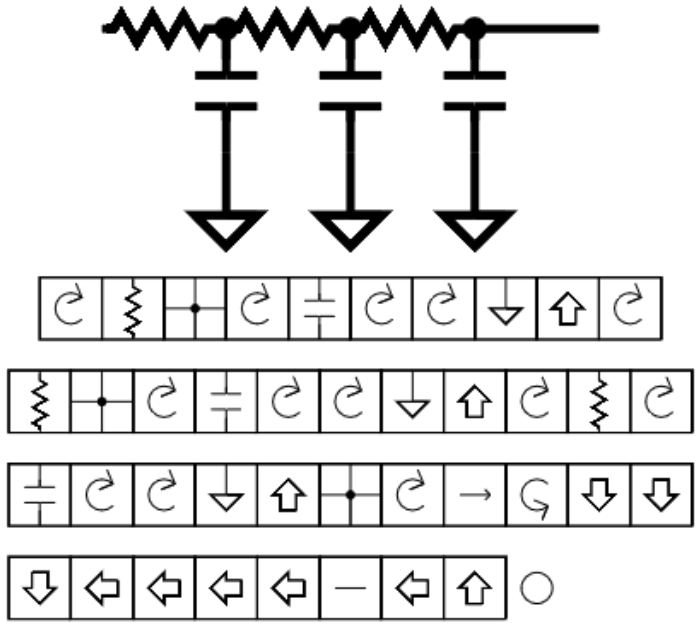


Figure 10.2: RC line circuit.

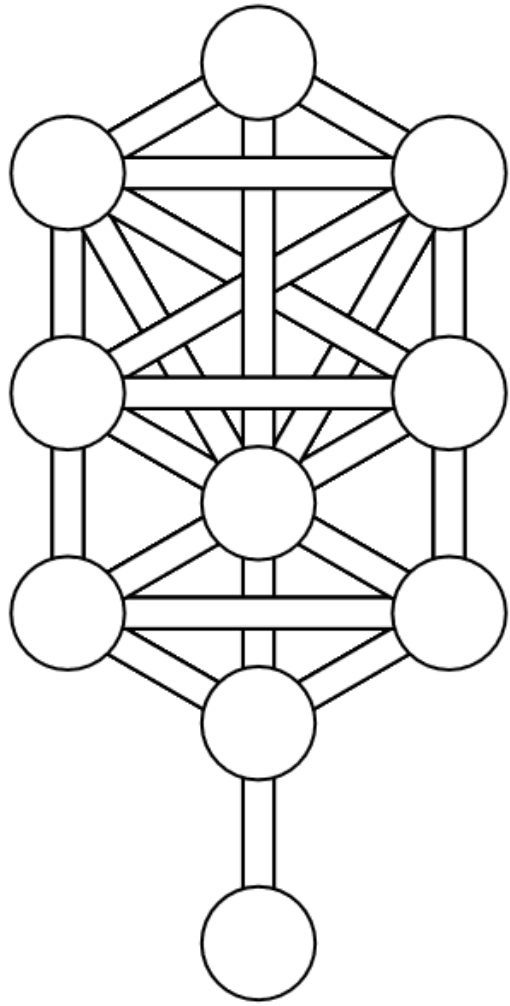


Figure 10.3: The Tree of Life from Jewish mysticism.

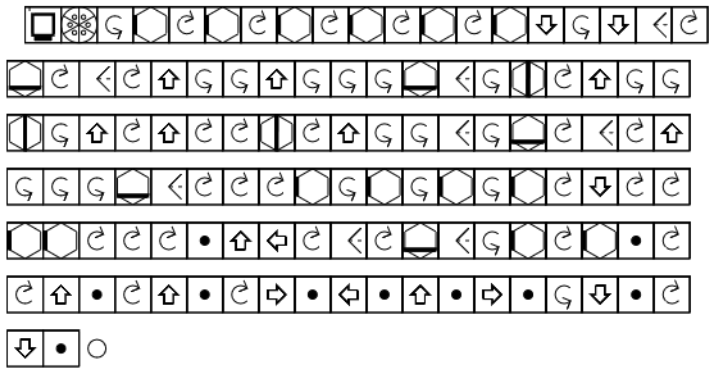


Figure 10.4: Symbol glyph spelling of Tree of Life. What makes things like this easy to make is building up building blocks like the cross pieces of all different scales, and the use of universal symmetries and scales(6 fold, 12 fold, and the square root of 3 and 2).

- laser cut fonts
- signal flag font
- general circuits
- quantum circuits
- organic chemistry diagrams
- graph theory, with digression into how to operate Maps with mathjax formatting for fully tex compatible graph theory figure creation
- quantum logic gates
- classical logic gates
- standardized icon design for geometron system
- katakana
- hebrew
- laser cut shape set shape stack
- laser cut ruler
- laser cut protractor
- tree of life from western occult practice
- penrose tiles, fun with golden ratio, fivefold symmetry
- cross stitch design

Chapter 11

Machine Control

11.1 Machine Control

democratization of machines

We should have as much control as possible of every machine we interact with and every machine which affects our lives and the lives of our communities. We currently live in a time where our ability to control our own machines is decreasing very rapidly, and it is the intent of this work to build technology which reverses this trend. For the last few hundred years, as machines have become more powerful and complex there has been a constant conflict between machines' ability to reduce labor and their ability to also increase the power and control of whoever controls the machines. In the early days of the

Industrial Revolution we started to see battle lines being drawn around the forces of mechanization which rage as hot today as they did then.

The basic conflict of the Industrial Revolution was between laborers who saw machines as replacing their role in society and those who saw the machines as having an overall benefit to society at large by producing more products faster and more efficiently. This conflict stemmed from the way we have structured labor and work during the time and place that these technologies came to power, namely the wage labor system. Under wage labor, the more "labor" someone does, the more money they get to survive. This creates an incentive for automation for the capitalist class, and create the opposite incentive in the laboring classes. Marxists took this structure for granted and argued that the laboring classes could liberate themselves through political control, but took the basic premise of "work" or "labor" for granted as ideas.

The time we find ourselves in today differs so vastly from that of the industrial revolution that we must pause and re-evaluate some assumptions. The most fundamental assumption of the time of the Industrial Revolution which we must dispose of is the need for a constant stream of mined materials and materials extracted from far away places. This element of industrial production is often hidden from the consumer but is far more fundamental than people often recognize. Every early industrialization process, involved some type of empire-

building over large land masses in order to build stable supply chains to maintain a flow of materials from mine to factory to consumer. The more deeply one studies industrialization the more clear this becomes. Some countries industrialized without building their own empires, but they did so by leveraging powerful trade relationships with existing empires. In all cases, the power of the physical network of raw materials was what made or broke the process.

But today the world we live in it materially different in this fundamental way: all the materials of a mechanized civilization are now readily available everywhere in the world, without exception. The global consumer industrial capitalist system has consumed the mineral wealth of the last 300 years of imperialism and industrialization remarkably evenly through the world by pushing the same consumer products everywhere, all of which form localized waste streams of highly processed materials.

Just take aluminum as one example. Aluminum is one of the most powerful tools we have ever discovered to create technology, with a very wide range of uses. However, before we can even shape it we have to reduce the bauxite ore, which is a very energy intensive process. I strongly recommend that the reader spend some time choosing random elements like aluminum or phosphate and learning where they come from. All the minerals we use take vast amounts of energy and complex mechani-

cal processes. And yet, all of them are now free! If we want to use aluminum in a technology today not only do we not have to reduce the ore to pure metal, we don't even have to machine it. We can find fully processed and coated aluminum shaped into already-useful shapes from beer cans to light poles. If we build our civilization on this waste stream, all the assumptions of politics and economics from the last 300 years of Western thought must be abandoned.

Today, we can build machines directly from the materials we find in our environment, without the need for a global supply chain we control. Even if the existing supply chains collapse, the existing cache of waste we find in our immediate environment should be sufficient to live technologically advanced lives. Therefore, under these new material conditions, machine control is direct, physical, and built into the design of the machine, rather than based on "power" in relation to large organizations like nation-states and corporations.

[all the above paragraphs should move to chapter 1, then just get referenced here as we dive into the following list of principles]

If our goal is to have the most control possible over our own lives in a technological society, we now aim to design that control directly into the machines. Democratization of this control(avoiding building a power elite of technology experts) then becomes a task of user interface design. If we look at the last few hundred years of build-

ing labor-saving machines, we find some good things and some bad things. In this work, I aim to create a framework for making those choices and for building technology which follows the principle that the more direct control the individual has over their technology the better, and will now delineate those principles.

Direct control before automation. This means every operation of a machine has a physical control which a user can use to directly control that operation. If a tool can be moved up, down, left, right, forward, and back, we will always have a controller which enables them to directly carry out that action without any further automation. This controller should be as physical as possible(rather than through software and touch screens). Unlike the automation discussed below, it will generally be as continuous as possible, allowing a user to directly move small or large distances with feedback being directly trough observation of the machine outside of any control technology.

Controllers are sized for humans. We want controllers which are as closely matched to how humans think, feel, and act as possible. This means things will be much bigger and more redundant than they need to be by the absolute minimal requirements of control. Buttons and levers will be as large as they have to to allow the absolute maximum range of human users to operate it without difficulty. We will not have defaults be small buttons and have a special case with large buttons, but

will always have buttons sized for people with limited sight or dexterity.

Controls should use geometry to be obvious. A lever next to a hoist should have pulling it up move the hoist up, and down be down. A pair of buttons should have the top one move up and bottom one move down. We also shape the controls to have geometric meaning as much as possible, for instance shaping a button as a physical arrow pointing in the direction of the motion it causes.

All actions should be convertible to automation. This means if we can use a controller to move a tool along some path, we can build a program which repeats that action with a single button press. Thus a controller will have, in addition to the direct controls, a "go" button which will initiate the automation program. The previous two principles dictate that this button be large, green, and round.

All automation actions can be stopped at any time. The big red "stop" button must also be universal. Pushing this button immediately stops the automating action. This is different from the "EPO" or Emergency Power Off button which can also be a good idea, which shuts down a machine completely. A stop button simply stops whatever action was initiated by the go button.

Discrete geometric actions of Geometron, using geometric symbols enable programming with no specialized technical knowledge. This is the heart

of where Geometron enables democratization of machine control. The idea of "move one unit to the right" is something humans can understand without any special training. The idea of making a symbol for that action which consists of an arrow or equivalent is also universal enough to require to special training and also eliminates the language barrier one gets from using word-based human language. These discrete motions sound limiting, but as with the earlier chapters of this work, if we allow for operations which change the unit of movement, we can make arbitrarily complex and precise motions built up from combinations of different units of movement.

Symbol hierarchy. Symbols are used describe action sequences which are built up to make more complex actions. So an action like "move right, lower drill press, raise drill press" can be condensed into "drill a hole", which has its own symbol, and then "drill a row of holes" has its own symbol, which is made up of a simple sequence of the "drill a hole" symbols. Those rows can be terminated with a "move back to start" action by chaining "move left" actions, along with a "move towards" action. When these are put together into a bigger sequence we can build up to a symbol which means "drill an array of holes", symbolized by just an array of dots in the standard Geometron convention of a square with symbols in it along with movement to the position of the next symbol.

Upcycled hardware. This is the principle used

throughout this work, but must be re-emphasized here because it is so important and because it is so easy for controllers. All we need is buttons and mechanical structure. The main body of controllers can be made from cardboard trash, as can all the surfaces touched by the user. Using simple geometry for all these parts allows them to be replicated effectively using social media without sending any files from user to user, just copying a geometric construction using simple shapes and symmetries. Buttons themselves can be found in any of numerous consumer waste products, often in fairly large number. An old stereo or printer can have dozens of buttons sometimes, and since the basic controllers we will build generally have about 8 buttons, this means one scavenged electronic junk item can yield multiple controllers in some cases. Ribbon cables and connectors can also be scavenged, but they are so cheap and easy to buy that for the time being buying them from consumer off the shelf sources can be more efficient.

Structure

The control of machines using Geometron follows the pattern of the rest of Geometron as documented in earlier parts of this manuscript. We assign addresses in the Action Cube in the Geometron Hypercube to geometric actions. These actions all have symbols in the Symbol Cube of the Geometron Hypercube. The Geometron Hyper-

cube, as described earlier, is a pair of cubes, each with 8x8x8 = 512 cells. One cube represents actions, and one cube represents symbols. Each cell in each cube contains a sequence of addresses which describe either a geometric action or a symbol of an action. Layers of the action cube are divided up by what kind of action they represent.

Actions which represent direct control of machines are assigned to the addresses in layer 4 of the Action Cube. All addresses in the Geometron Hypercube begin with a zero to indicate that they are in a range from 0 to 7(base 8). The addresses representing machine actions are of the form 04xy, where xy are numbers from 0 to 7. Most of this set of actions is empty in the version of Geometron presented here, leaving a large space open for future development, potentially adding vast complexity if that is useful. The way actions in the 04xy block are performed varies by the specifics of what hardware is used. Design of symbols and programs can be carried out entirely in a Web browser using the main system of Geometron. No hardware control is needed at all to design these systems. It can be totally self-contained in the browser, building up custom languages of symbols in the Symbol Cube of the Hypercube in the address space defined by 014xy, where x and y are the same coordinates between 0 and 7 as in the Action Cube. Again, sequences of addresses form glyphs which are just text which can be copied and pasted and shared instantly throughout the global network by private text message or public text file sharing

and replication as described in the code chapter.

Actions representing combinations of basic actions, like the "drill an array of holes" action mentioned above are put in the

2 and a half d printing and the Trash Robot icon token printer

The wall robot

Microfabrication and Nanofabrication

- introduction, the democratization of automation
- structure of hypercube, workflow, general principles
- trash robot printer
- wall robot on large building
- agricultural robot prototype/description
- electron beam lithography
- hacked 3d printer to make 2.5d printer

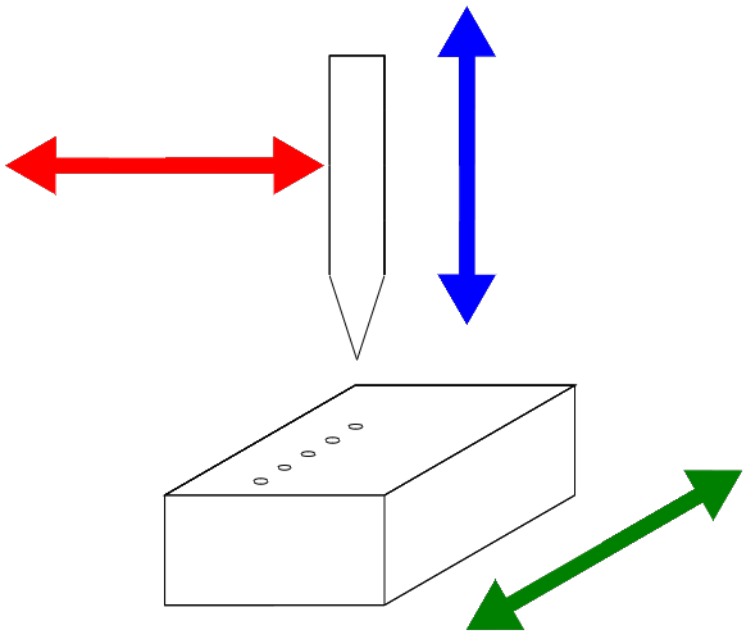


Figure 11.1: A probe move over a sample in either the x or z direction, and the sample moves along the y axis. Repeated poking of the sample prints dots a simple but very versatile tool.

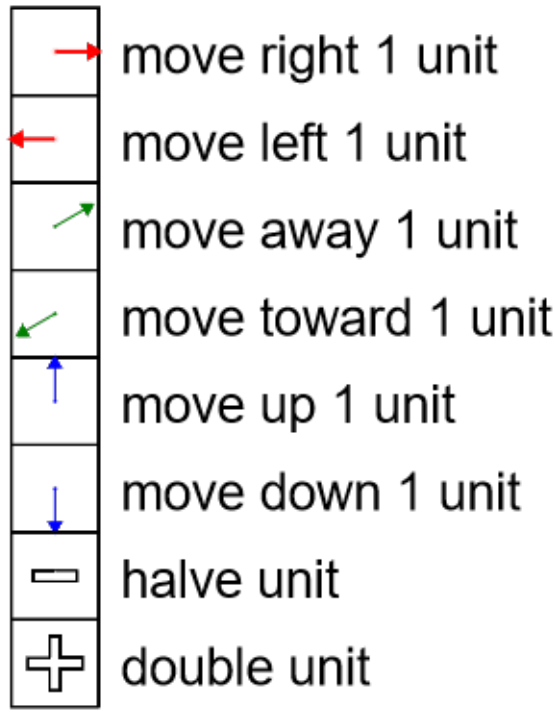


Figure 11.2: Basic geometric actions of machine control for an arbitrary machine that moves along three perpendicular axes.

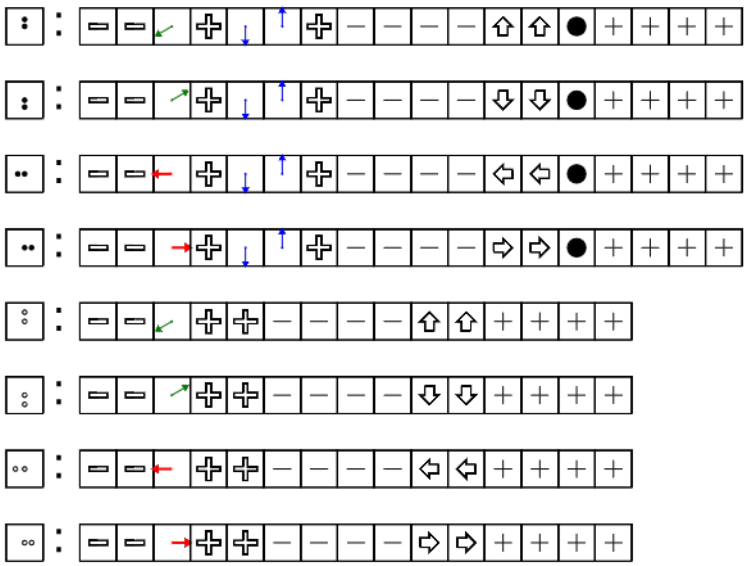


Figure 11.3: Dot actions from which symbols are constructed.

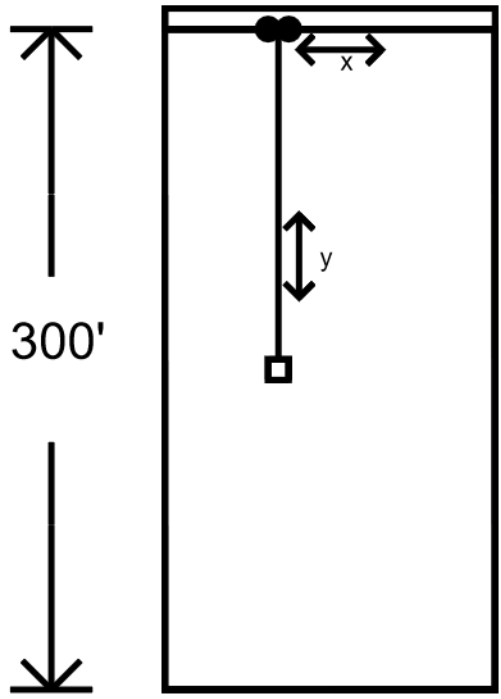


Figure 11.4: A hoist run along a rail going across the edge of a roof of a building can make a simple robot which can move to anywhere along the wall.

Chapter 12

Geometron in 3d and Beyond

12.1 Geometron in 3d and Beyond

- where it exists in hypercube, concepts of movement and construction, cursor
- file formats and software: THREEjs, .x3d, how they are used, 3d printing
- fonts: examples of pixel font of standard English letters and braille with hemispheres
- rotation angles of many part robotic arm tools
- quantum geometron: rotations in hilbert space, representation of hypercube in state of quantum pro-

cessor, direct mapping to protein folding without gates

- writing 3d geometron apps, how to use, how to expand and rewrite
- detailed documentation of the code

Chapter 13

Ontology

13.1 Ontology

1. the branch of metaphysics dealing with the nature of being.
2. a set of concepts and categories in a subject area or domain that shows their properties and the relations between them.

The purpose of our whole structure of knowledge is to create a civilization in which technology freely replicates. We will choose terminology and methodology entirely based on this criterion. The breaking down of things into sets composed of elements which are themselves things is based on this. We always choose that which maximizes the probability of replication and the

freeness of the thing being replicated. To this end, how do we define a "thing"? Topology of replication, meaning the connection in space and time between people, actions, places, materials, energy, information and other things which are required to replicate the thing. Set structure of replication, meaning the way we break the thing down into elements and those elements into sets with more elements to maximize the ability of the human mind to understand how to replicate the thing. And geometry, the specific geometric constructions that make up the physical replication of the thing. To create a Geometron Thing is to build whatever media we need to create and document the topology, set structure, and geometry required for replication.

We consider things in the most abstract sense. From things we create sets, which are also things. This section is a more formal mathematical treatment of the material in the "magic" chapter. It is more technical than that chapter, and does not reference magic, but does reference our specific implementation.

This is where the Theory of Self replicating sets is described. We point to the failures of the project of set theory in the 20th century, how it was a dead end, and how we can navigate out of the dead end by placing the mathematicians ourselves in the sets we describe. The whole failure of the old models of self replicating machines(perhaps this gets moved back to organic media chapter).

set notation, departure from ZF-C paradigm. New way of dealing with foundations of mathematics: the mathematician is part of the sets considered. And we continue this through the whole construction of math.

Foundational math in the paradigm of the 20th century was from ZF-C set theory to arthmetic, to algebra, to geometry, etc... but the whole edifice is based on sets which follow axioms, but those sets do not contain their makers. We create a foundation of mathematics in which the creators of the math are always in their own calculations, starting at the primitive level of what an "object" is and what a set of objects is. We consider not only ourselves to be elements of the sets in our universe, but every aspect of ourselves, our communities, our minds, and our ecosystems. For example, looping back to SRS, the desire to replicate a set is part of the set. A set might be

{ - powerpoint slide about self replicating sets - the desire to tell people about self replicating sets - the persuasiveness to induce others to desire to tell people about SRS - the means to replicate the powerpoint to the next user(e.g. posted on slideshare) }

From these sets we construct geometry which is used to make generalized symbols, which are used to make all the things in our technological complete set, which is again a part of our new approach to interacting with abstract sets.

Random notes:

unlike normal set theory, sets can be elements of each other(we call it “element”, not “subset”, to erase the distintion between set and some hypothetical non-set thing)

The structure of sets, the elements we choose to describe a set, are completely determined by what will be the most useful for replicating the set, for a human operator.

example Things:
The Terminal
The Geometron Terminal
The Pyramids
Audion
ArtBox
Trash Robot Icon Printer
Skeletron
laser cut shape set
laser cut ruler
custom laser cut shapes and stencils

13.2 On Self-Replicating Sets

1. Computer Science and the Theory of Self-Replication

Throughout the history of modern computing machines, people have contemplated the idea self-replicating machines. At the dawn of the information age, John Von

Neumann in particular devoted thought to the subject, creating a blueprint that people continue to use both for understanding hypothetical self replicating machines and for understanding the architecture of existing computing machines. At the same time, Alan Turing developed a similar toy model for how generalized computing machines work, which is taught in basic computer science classes to this day. We will not delve very deeply into these models, but will instead present a very crude sketch of them in order to discuss the assumptions made by computer scientists in the models they build to understand their world.

The Turing model of a computer consists of an infinite tape of 1's and 0's along with a machine to both read and change the state of the numbers in the tape. The string of numbers describes instructions for the machine, which follows those instructions by moving the tape back and forth and changing numbers from one state to another. Turing was able to show that this toy model of an abstract computing machine could be proven mathematically to be equivalent to any other abstract toy model of any other kind of computer, including the complex machines built today(hence the continued use of this model in teaching and scholarship). This is considered to be one of the most important results in theoretical computer science.

Before discussing this model's limitations we must say a word about the nature of scientific models. When we

investigate a thing using the scientific method we have in principle the entirety of science knowledge to call on, built up from a vast number of models in different fields and sub-fields. For example, if we are presented with a rock to analyze, a physicist might ignore everything but the crystal structure of some prominent material in the rock and bring the modern understanding of crystals to bear on it. The micro-biologist will only be interested in the aspects of the rock that interact with the organisms on the surface, while the ecologist will be interested in that but primarily how the rock regulates the flow of water through the ecosystem of which it is a part. As scientists we may agree that models of surface chemistry for microbes, models of how atoms arrange in crystals, and how water flows through rocky soil are all "good science", but in any given context the model we choose to focus on depends on that context.

It is our assertion that while the model of Turing and his contemporaries is not wrong, that it is deeply misleading because in most cases it does not describe the most relevant aspects of the machines we call "computers". Computers do, of course, compute. That computation is described by the mathematics of computation. They also create heat, described by thermodynamics, and are we therefore to call them heaters? They keep time with extremely fast clocks, do we simply call them clocks? No. But what actually are they? What is the model for a modern computer which is most useful when trying to

understand them? In this era, in the year 2020, the most useful model for a computer is the one that describes how they have totally changed all aspects of global society in a relatively short time. For this we must expand the models available, and in particular we must focus on a specific shortcoming not just of mid-20th century computer scientists, but of most scientists in the "modern" era, namely our refusal to put ourselves and the societies of which we are a part into the systems which we study. In some abstract sense, one can argue that we put ourselves into the models with the role of the observer in quantum mechanics for instance. But we do not put the role of something like university politics into the models, even though these forces clearly influence the science we use and hence the conclusions we draw. It is the assertion of this paper that this blindness has become so critical in the understanding of computers as to be wrong in a way that has real world consequences.

So what is the problem with a Turing machine? If we look at it naively as a physical thing, not as a mathematical toy, we see a number of things that are not realistic, such as the infinite tape and the lack of any meaningful human readable input or output. But these are typical of useful scientific models: while the tape is understood to never be really infinite, the results you can get from the machine with "such a large memory that it doesn't matter" and infinite mostly don't matter, so our toy model still works. But the critical flaw of the machine which

gives us an incorrect understanding of how computers function in society is that it has no origin, no purpose, and no destiny. Who built this machine? Why? How long can it run before it breaks? What happens when it breaks? When it ceases to function, what replaces it and why?

We now live in a world where a large fraction of the computing machines that exist live on a trajectory from a mine to a landfill of less than 2 years. During this brief journey from mine to landfill, they are mostly used for communication, and much of that communication is marketing information the primary purpose of which is to lead to further consumption of similar machines. That is, *in their current state* the *primary* action of these machines is rapid replication.

This is the reason we must shift the *primary* scientific model of computing machines from the Turing model to a more biological understanding. If we seek to study a new species of life that is taking over an ecosystem rapidly, we will always try to focus on the models that allow us to understand that phenomenon because it is what affects outcome. If a farmer asked us to evaluate a new crop blight and we came back with a deep study of how carbon chemistry works because all life contains carbon they would be very disappointed in the results even if they are technically correct. Likewise if we are to deal with the explosive changes to our physical world caused by computing machines we must focus on the means of

replication. We will give an analysis of these means in the next section. However before doing that we must turn to the history of self-replication as an idea in mainstream computer science.

There are two main intellectual threads in this story: cellular automata and self-replicating robots. Interestingly, it is the former of these that actually have a vast experimental component and the latter which is entirely theoretical. Cellular automata are in some sense a generalization of the Turing model: they are sets of rules for multidimensional(usually 2 dimensional) grids of numbers, generally 1's and 0's, which follow some set of rules. These systems are simulated on real computers, and as time advances in a program we can see fascinating patterns of what appear to be naturally oscillating structures moving around in a visual display of dots on a two dimensional canvas. If the rules are created correctly, these structures can be made to replicate themselves. The literature on cellular automata is vast and complex and continues to be a very active field. Nonetheless, it suffers the same flaw as the Turing model: it exists in a vacuum, with an assumed infinite time, and no reason to exist(at least this reason is not part of the theoretical framework used to describe them), no origin and no ultimate destiny when it breaks. There is beautiful pure math to be found in these systems, but no illumination of how computers function as machines that copy themselves.

The second thread of self-replicating machines is the

study of theoretical "robots that build robots". This has attracted some truly wild speculation. What is generally imagined is a totally automated system without human intervention in which a computer controls robots that mine minerals which are used to build both more computers and more robots. This is then imagined to be so self-contained that it can be used to expand out into the universe outside of Earth, eventually consuming all things into this vast, automated, technical ecosystem which does not need any living thing to grow. Technologists have imagined these systems, then promised that they can be a fantastical utopia of free things, but also warned that they can destroy the world by consuming everything in site. For decades, theorists have written very detailed descriptions of such systems, delving into metallurgy, synthetic chemistry and the like to try to prove that such a thing can be built. Most recently, the work of K. Eric Drexler and Ralph Merkle in the 1990s pointed to a system like this built from precise positioning of atoms in matter–essentially treating physical matter as just another Turing machine. These theorists constructed very detailed imaginary worlds where atomically precise machines manipulate atoms to both compute and create, replicating themselves on a global scale. Perhaps their day will come at some future time and such machines will be built, but right now these models are of no help in understanding technology as we find it today. The rest of this paper is devoted to developing both a theory

of self replication in modern technological society and in actually *using* this theory, just as the pioneers of modern computer science used Turing's model early on, to build new things based on the new model.

2. Self-Replication in Human society

First, a word on self-replication in biological systems that exist outside of human society. "Natural" biological organisms never replicate themselves in a vacuum. On Earth as we find it today, all living systems replicate as part of larger ecosystems, and the parameters of those ecosystems are *fundamental* to the overall replication. No animal can live long enough to reproduce without a constant flow of oxygen from plants needed for respiration. Conversely plants need the carbon dioxide we produce in order to live long enough to replicate. No one would dispute that a tree is a self-replicating thing, and yet trees only replicate in collaboration with a large number of other organisms, generally other plants, fungi, animals, all working in concert to make the overall forest system replicate itself, of which the trees are only a part. What we find in spite of this, however, is that scientists outside of biology put much more restrictive rules on self-replication, saying a thing does not "really" replicate itself if it replicates as part of a larger system. Hence the flaw in computer models: if humans replicate machines, those machines are not called "self-replicating" because

for the overly restrictive definition of the "self" of the machine they do not spontaneously replicate. But this type of isolated spontaneous replication does not exists in nature even for purely biological systems, so applying it to systems outside of biology will give results that are at odds with how the biological world works. Again, we must distinguish between models that are "right" and models that describe the primary characteristic of a system under study.

In the pre-industrial societies, *all* technology is self-replicating. Historically, people make technology using the materials found in their environment, generally from other organisms like trees(wood) and animals(bone) or found objects of which there is a plentiful supply. People then reproduce and teach the system of constructing such technology to the next generation, along with enough understanding of that technology that the young can in turn pass along the information when they age and are passing it along to yet another generation. This type of self-replicating technological system can exist in stable dynamic equilibrium with existing ecosystems. Trees can grow, be converted to boats to hunt in, which lead to cooking technology to cook the animals killed in the hunt, and trees and game animals can then re-grow as future generations replicate their technology.

If we take this broader view of self-replication we don't even need to restrict ourselves to humans to see self-replicating technology. The beaver dam self-replicates

quite easily and indeed before the rise of modern society one can imagine the dam itself as a self-replicating entity which uses the beaver as a vector, but which transforms the landscape vastly in excess of what one might imagine possible for a single small animal. To make sense of a landscape transformed by beavers it does not make sense to either study just beavers or just dams, we much consider the self-replicating set of dam-and-beaver as a single system. The same is true with human technology.

Moving into the very early reaches of our history as we learn it today based on written records, the next self-replicating systems are religions and empires. A religious text is perhaps one of the best prototypes for self-replicating technology which can shed light on the current state of affairs. Religious texts such as the Torah or the Koran describe both a world view which gives people a "why" to what they do which includes the replication of that text. They also include the description of "how" to replicate the text, building up complex structures of education which teach the next generation of humans what they need to know to keep replicating the next down through the generations. The other main replicating structure is that of the military bureaucratic empire. An empire replicates by expanding to incorporate more and more people into the actions of further replication. This is generally done by force, which can keep growing by taking more land for more mining resources and also more people to continue to gain power to continue to ex-

pand, consuming other systems and turning them all into that one central imperial system.

The entirety of human written history can be looked at through the lens of these self-replicating systems, where the means of replication is the primary descriptor of the systems. The history of what used to be called "Christiandom" can be seen entirely through this light. The Torah was replicated by Jews for thousands of years, and was limited in its replication by Jews' only replicating it to other Jews, so the growth was limited by the biological reproduction of the people who did the replication. Then, when Christianity appeared, the same text was suddenly being replicated by the people of one of the vastest military empires every built, Rome. Ultimately this replication consumed Rome and became the Holy Roman Empire which among other things was a vast replication machine for the scripture. Then, due to technological advancements, it became possible to replicate that scripture mechanically in bulk with the printing press, and we see another explosion of change in that world where the press and how it replicated religious scripture caused some very radical change. As Western capitalism developed, the King James version of the Bible, printed in bulk on mechanical presses defined the beliefs of the military empires that then went on to conquer the globe(singling out the British Empire, followed by the American as the most powerful of the lot). Nothing in our world today makes any sense without this story of evolving self-replication.

So now this brings us at last to the discussion at hand: self-replication in regards to modern computer technology. How do computers replicate? Just as an analysis of beaver dam replication requires understanding the trees from which sticks are harvested and the rivers which feed the beaver ponds, this analysis must include externalities that are ignored by theorists, such as investors and marketing. Modern computers exist as creations of a combination of mass market consumerism, venture capital investment, and government research and development mostly focused on the military. Every company that makes computer hardware and software today is the creation of a very specific process whereby an entrepreneur pitches a company to investors, who in turn pitch their fund to larger institutional funds like pensions. After they get funding, they grow using a very specific type of worker, the modern tech worker fully indoctrinated in a certain culture. That growth is made possible by a system of mass media that transmits the information to consume the products to the masses outside of "tech". Those masses of people both put money into the products of this creation process and also put investment capital into the financial system that funds the venture capital that creates all this. Nominally all this is enabled by the "money" system which used to be based entirely on mining of minerals, but is now based on some complex system of faith more loosely based on mining. The computer systems which out-evolve their competi-

tors are the ones that replicate the fastest. They are the ones that convince people to consume more and faster, and put more money and time into the system. The venture capitalist David Horowitz has explicitly said that in the future they are building everyone will either be forced to obey the media on these systems or will become one of the people building them. And indeed this is the logic of the self-replicating machine. People like David Horowitz have to exist in order for the machines to out-replicate other machines.

This picture presented here is of course a vast oversimplification and the product of a fairly brief and superficial analysis. It is not the purpose of this paper to create a full model of replication of modern "tech", but rather to convince the reader that *such a model is needed*, in the hopes that people will develop more accurate models that we can use to try to gain some control over this system and ultimately over our lives.

In summary, the simplest model for computers that I think we should consider now is not that of the Turing machine but of the advertising machine which exists for the sole purpose of convincing people to consume more advertising machines. This might take the form of presenting PowerPoint to investors, using computers to train the workforce to build the technology, or spreading an article in the tech press about some new technology, but in most cases it is just direct advertising to the consumer. But in all cases the primary characteristic which

determines form is replication. This is why evolution of machines has favored more and more of the machine being screen, with the highest possible pixel density and color contrast, rather than maximal computation power. Pixels are what sell pixels.

We must also distinguish between viral replication and independent replication, although the line is blurry. Viral replication assumes a fixed system in which the information replicates. For instance, information can replicate itself within a commercial social media platform like Facebook or Twitter just as influenza can replicate in a host human, but this does not replicate the *system*. The means of replication of a system such as Facebook is in fact not replication of content, but the whole system including venture capital, technical labor, media to sell it, the legal framework to enforce the power of the company, etc. It is this full system replication that we are concerned with here, not the replication of information within such systems, known colloquially as "memes".

3. The *Potential* Power of the Open Web for Self-Replication

What is the Web? The Web is not the Internet. The Internet is a network that connects almost all computers in the world, both physically and with some software protocols. This network traces its origins to the network

of military and academic(but military-funded) computers that emerged from ARPA back before the modern commercial Internet, going back to the end of the 1960s. The Internet can in principle be used to exchange any information and treats information in the same abstract sense as in the models of theoretical computer science discussed above.

The World Wide Web was initially the creation of one person: Tim Berners Lee. It was initially created as a directory for the large science institution Lee worked for(CERN). The Web works beautifully on the Internet and the Internet is what made it huge, but it is not the same thing. The Web is a system for encoding information for and by humans to communicate with other humans. It includes human readable code designed to create universal documents which link to each other and can contain images of all kinds and text in all languages, creating a sort of universal document in a universal language in which all of humanity can communicate. While the modern Web is commercialized and increasingly not open to all users by default, this is a choice we have made that can be un-made. In principle any computer *can* be both a web server and a web browser. If we call the "open web" the collection of all web files which are openly viewable to anyone connected to the Network, the open web can in theory physically grow to include every computer in existence using the existing physical telecommunications network.

Let us now estimate the size of this potential Open Web network. We suppose that given the multiple billions of smart phones, laptops, servers in server farms, embedded systems, supercomputers, etc., that the total number of potential web servers is of order 100 per person. We then round human world population up to 10 billion, and estimate that there are 1 trillion total potential servers on the Open Web. Given that even a modest cell phone has a few gigabytes, but many servers and deep storage computers have terabytes, we can round up a little and say the average server can host 10 gigabytes of data. If a file like this one(this paper, the one you are reading rightnow) is 100 kilobytes to 1 megabyte, let's round to 1 meg and say there are 10,000 documents like this one(they could in theory all be math papers) for each server. So the total number of documents per person is 1,000,000. But we as a society *share* these documents, so in some sense what we all have is not 1 million but 1 million times 10 billion or 10 quadrillion documents(10,000,000,000,000,000). The informational universe in which we construct this new mathematics consists of this network of 10 quadrillion linked documents.

This universe of information exists on web servers which can in general be made to run code that edits and replicates the documents. Thus *every* document in this universe of information can self-replicate and be edited in situ. If this is all on the Open Web with code that can be edited by anyone on the Web, all users can constantly

edit all documents, so potentially we have 10 billion people all simultaneously editing 10 quadrillion files all of which are able to instantly self-replicate from any node on the Network to any other Node. This vast network effect can create power in the same what that billions of brain cells with massive interconnection, creating a document of greater power than any that has ever been possible before. The power of such an open system will be so vast that it will make no sense to have any private data. Without any property on the Open Web, things can replicate freely, and the increased value will be so great that it will consume private property online. This evolution will be physical as the value to the physical caretaker of a physical web server becomes greater to participate in the Open Web than to keep it in the commercial web. Note that if we try to simply write down the number of ways that these documents can point to each other to self-replicate, since each document can replicate from *any* other document in the collection of documents(and any number can replicate from any one other document), the number of ways they can be structured is the number of documents to the power of itself. This is 10 to the 16 to the power of 10 to the 16. 10 to the power of a few hundred is already exceeding the number of estimated protons in the known universe. So one can make similar arguments about this system as people make in regards to quantum computation systems: we can even for a very small network build something that is totally impossible

to simulate on a classical computer.

The power of a fully self-replicating and evolving Open Web on this scale is that documents can describe the replication of *physical* things, and the replication of the documents can include replication of the things. If things we use in our lives are replicated rather than purchased or mined, it changes the basic assumptions about what value is. Note that like "set", "thing" is used in a maximally general sense to include things like "a feeling of awe at the largeness of a tree" or "the tendency of cats with white fur to cause a change in the appearance of black clothing". The word "thing" is used here as a placeholder for *anything* which human language can be made to describe or point to.

In order to build these documents we must first define the idea of what exactly a self-replicating document is, and how it fits into more general concepts of self-replication. To that end we will take an excursion into the math known as set theory, which is the next section of this paper.

4. Self Replicating Sets/Documents

Motivation and definitions

Set theory, is the mathematical study of sets. Sets are simply "collections of objects". The idea of a set as a collection of "objects" considers the idea of the "ob-

ject" at a level of generality perhaps shocking to non-mathematicians. "Object" here can mean *anything*. Mathematicians generally mean by "any object" any object which a mathematician might talk about. However in principle it can be anything that anyone might talk about(as we seek to generalize these ideas beyond mathematics). For the purpose of this work we well define a generalized object to be anything which human language can possibly describe. Any word, symbol, or collection of words or symbols which point to something–that something is an "object". And a "set" is just a collection of such general objects.

The notion of a generalized object is familiar to modern computer programmers who use the idea of "object oriented programming" to create generalized objects which are used to build linguistic handles in human language on computer programs. Thus a modern programmer might define something abstract like "shopping-cart" for an e-commerce website, and then that object will have properties like "list of objects" and "total price". We choose to take the path taken by foundational mathematics and have our basic concept from which all other concepts will be built be the collection of objects(which are themselves objects) be the fundamental idea.

In order to understand the motivation of this work it is necessary to trace very briefly the history of set theory. Through the end of the 19th and beginning of the 20th century there was a vast effort by some of the

most brilliant mathematicians in the world to construct a universal mathematical theory base on the theory of sets and symbolic logic. Axioms were proposed, used to prove things, argued about, and re-written. The goal was to base *all of mathematics* on the axioms of set theory, and to go from there to a universal system of truth in which statements may all be proven to be true or false.

People like Bertrand Russel pointed out that such systems can create paradoxes that make it impossible to create a self-consistent system of logic/truth/math. This paradox can be summarized by considering the "list of lists that do not list themselves". The list defined here is a list of lists. Is this list on itself? If it is, it cannot be by the definition of itself. If it is not, it must be by the same reason. In spite of having publicly stated this paradox, Russel and his co-author Alfred North Whitehead wrote what is now considered a seminal work in mathematics *Principia Mathematica*(not to be confused with a book by Isaac Newton of almost the same name), which attempted to create such a universal basis of mathematics. While the achievements of 20th century set theory, logic and analysis are fantastic and useful, they ultimately failed in their goals, and this was proven mathematically by Kurt Goedel in 1931.

In the post-Goedel world we should take for granted that no universal logical construction can be built which defines truth and falsehood without contradiction. Goedel's proof presented a fork in the road intellectually. We could

have used this as the sign to go back through mathematics, accept contradiction as part of life, and build a math based on desired outcomes. In some sense this is what society did by mostly ignoring the work of most post-war mathematics(with some very narrow exceptions like number theory for cryptography). While professional mathematicians took the opposite fork, building increasingly complex systems based on each other, where a vast tower of ideas linked by formal logic built up from the axiomatic set theory of the early 1900s to create a bridge to nowhere. The sole purpose of most mathematical concepts and theory today are to advance the career of working mathematicians. We forget, both outside and inside mathematics, that people used to believe ideas in math actually *mattered*. We also forget that mathematics has for thousands of years been one of the most powerful tools the human mind has for understanding and interacting with our world, and indeed mathematicians have traditionally played a central role in the largest power structures throughout history.

Having proved that a universal truth machine cannot exist, we may now abandon the project of early 20th century mathematicians such as Russel and Whitehead and proceed to reconstruct axioms of set theory based on a *desired outcome*. This system will not be judged on its ability to prove theorems, eliminate logical contradictions, or get tenure for math professors. It will be judged *only* on its ability to improve the human condition. It

is time, finally, after almost a full century, to inherit the true legacy of Goedel.

Right now all of humanity is locked into one giant self-replicating set which has as elements all of industrial society. The purpose of this work is to create a set theory which enables people to construct sets which create the maximum amount of human freedom. To that end, we seek to make sets have elements that are defined as generally as possible and also which always have the *desires* of the creator of the set as an element. When we move forward replicating these sets in the new society we build, each act of replication involves also replicating this desire. We therefore only replicate that which transmits a desire that we consent to and agree with for some reason.

Let us make some statements here about the sets we want to define. We define self-replicating sets as sets which contain as elements the means to replicate themselves. We also state that since our goal is to create the objects of our desire with our mathematics that whatever that desire or intent will always also be an element of the set. We maintain the tradition of both formal set theory and object notation on computer science and define sets in writing by listing elements separated by commas and contained in "curly braces" "{" and "}".

We also state that in general the sets we will construct will have a primary element, the replication of which is the purpose of the set. There might be many other elements and subsets which are needed for replication, but

the *primary element* is defined as the element the replication of which represents the *primary intent of the creator of the set*. We thus make the human will, desire, or intent a fundamental element of our entire set theory. To distinguish our set theory from that of mathematics as it exists today we coin the term "set magic" to be the theory of sets which contain both the desires of the creator of the set and the means to replicate the entire set. This is loosely based off of the quasi-secular use of the word "magic" from both chaos magic and Thelema magick to indicate the attempt to impose the human will on the world we find around us.

Thus a way to express the most general possible self-replicating set from our newly defined set magic is:

```
set = {
    desire,
    object of desire,
    means to replicate this entire set
 }
```

Note also that in order for the whole set to replicate, the desire of the creator must also replicate. This is what in our existing system is called "marketing" and "sales". Without first convincing another mind to share our desire for replication, it will not happen. The power of the open web is thus not just about replicating documents but replicating the desire to replicate documents–in mod-

ern parlance, just marketing. The commercial web has proven excellent at this, and the open web has the potential to vastly improve on that.

We further state from the outset that all self-replicating sets have externalities. These will be elements or subsets of elements which draw on the resources outside any given "closed" system in order to replicate. Thus rather than attempting to build a conceptually closed system and then finding that it is not closed as did earlier set theorists, we accept that we are building a less formal construction that will always have an externality and that we must describe what this is in order to properly define a set. The number of degrees of freedom of this externality is what limits the overall degrees of freedom of a system. For example, if an element of a set is "text reading system" that can be on any of many different technologies. Whereas if something has an externality "lithium ion polymer batteries", the entire system is reliant on that one technology and any threats to large scale extraction of lithium from the Earth are threats to the whole system. We thus seek to constantly struggle to improve on the externalities, with the ultimate goal for them to be in equilibrium with the living Earth.

A self-replicating document is an example of a self-replicating set(a "document" is just a collection of symbols, hence a set whose elements are symbols). This document is created as itself a self-replicating document according to the prototype we propose for the whole sys-

tem. We now set forth to define the set which is this paper in theoretical terms, then to describe all the subsets which together make for a self-replicating set which can be evolved into other self-replicating sets. We now proceed to formally define the set which is this document.

This Set

The prototype self-replicating set we define in this paper is itself this paper, and is formally defined as follows:

```
On Self Replicating Sets[This Paper] = {
    The desire of the author to describe self-replicatin
    A description of self-replicating sets,
    The means to replicate this set
}
```

The first of these will always be a part of the system of sets we are constructing here: desire. All sets are defined with an element which maintains the desire of the author/creator/artist, and this is maintained as sets replicate to ensure that sets only replicate with the intent/consent of someone. Furthermore, we separate the thing being replicated from the means to replicate it. Elements are generally themselves sets which are broken down into more finely defined elements. This paper is part of the "description", but what formally is "this paper"? We seek to define it as a set, or a collection of ele-

ments contained in "description of self-replicating sets". This includes the following elements:

```
Description = {
    narrative structure,
    definitions,
    digital text document,
    pdf document
}
```

Now again we break off the replication methods from the thing being replicated. Replication is to happen on the Open Web. That is, it must have self-contained and self-replicating code that can copy itself from any web server to any other, and be edited after being copied, then copied again from the new instance so that the information is totally decentralized and evolves naturally as it's copied and edited. We also need this document to include instructions in human language(English in this particular instance) on how to replicate the whole system, by either buying a domain and setting up paid web hosting or building a physically local web server to server the files. This document will contain that information. Furthermore, the replicator set must include the other media that are used to replicate the whole set, including content on commercial social media.

```
Replication = {
```

```
    code replication,
    server replication,
    pdf replication,
    in-person pitches and classes and discussions,
    media: email, post cards, posters, signs, commercial
}
```

Code and replication of code

The elements of the self-replicating code that can propagate this document across the Open web are as follows:

- replicator.php
- dna.txt
- filesaver.php
- fileloader.php
- `README.md`
- editor.php
- `index.html`
- `pageeditor.html`
- dnagenerator.php
- text2php.php

All of this code can be edited using the program `editor.php` which runs on any server that has php installed(most web servers). The file dna.txt represents this list of files, which replicator.php uses to fetch them and copy them to a new server(see below). README is

the text of this document itself, and the name of that file is set by the standards used on Github so that by default any self-replicating document that's put up on Github has its content readable immediately. The format of the README file is by default Markdown. The save and load scripts are required to edit files on the server, and pageeditor is the page that uses these files to edit the main manuscript README.md. All php files are stored as .txt files so that they can be readable and easily accessed from the open web. The program text2php.php copies all the files in the php directory to the main web directory and changes the file extension from .txt to .php so that they can run. The file editor.php edits the copies in the /php directory, and then running text2php makes those programs executable.

Server replication

In order for replication to take place from server to server we need the ability to "colonize" new web servers with this code. This is done in any of several ways. Right now the main way is to buy a domain(usually about 10 dollars for the first year), pay for web hosting(5-20 dollars per month), then put the replicator program on the new server and run it. The second method used is to put a web server on a Raspberry Pi, a computer that can be bought for as little as 35 dollars and fit in a pocket which is excellent for serving web files over a local network.

This allows for a grey area open web that is open to anyone on a local wifi network, and can see the rest of the Open Web but cannot be accessed by users outside that wifi network. In either case, the replication of the server consists of placing the file "replicator.php" in the main web directory of the new server, then pointing a web browser to [your new servers web address]/replicator.php. This will then cause the program to run, copying the rest of the system and linking to the main page which will display the newly replicated document.

The technical details of this process must be described briefly here in order for this document to truly self-replicate. We recommend buying a domain and getting shared hosting on dreamhost because they have proven to work with this code and are affordable and not a scam. One can also use any of several free web hosting options, including 000webhost. In both cases you will find a file editor screen(this can be a little frustrating to find but always exists) which will allow you to create new files and edit them. Use this to create the file replicator.php and copy the code from here into it, save it and close it. On the Raspberry Pi, replication starts by making a new flash card with the operating system on it. One must then install the web server and php language using this set of instructions. After that one can move to the main web directory, copy all the files, change permissions, delete the existing page, and run the replicator:

```
cd /var/www/html
sudo rm index.html
sudo curl -o replicator.php https://raw.githubusercontent.com
php replicator.php
sudo chmod -R 0777 *
hostname -I
```

Once the whole thing has been copied, to have the *next* copy be a copy of this copy and not its predecessor, use the code editor to edit the file replicator.php so that it points to the global url for the dna on your new page, not the previous one. This is done manually for now. So if you do not do it, the next copy will be not of the new document but of its predecessor. Note also that once this system is installed anyone can run any code on your server, so no private or personal data of any kind should ever share a server with this system. This system assumes that no private data exists unless it's on a physically isolated wifi network, and it must remain separate from the private Internet(especially anything with e-commerce, as stealing financial data would be trivial if that is ever done).

Another good practice as one works with files on the open web is that since we must assume all files might be deleted at any time, but we want the current copy to remain readable not just by us but by all users on the Network, we constantly back up text to free and open but non-editable paste sites like pastebin.com. To

fork the whole code, one can edit on a live Open Web server, back up to a pastebin, and copy. But one can also create a github repository, copy the code locally to your hard drive, edit it on there, run dnagenerator.php to make a new dna file, then point your replicator to your github repository so that many copies can be made without the original being corrupted(using a hybrid between the password-protected space of Github and the Open Web which copies files from there). Note that while one can use any code editor for editing the local copy, we can keep a consistent system with the Open Web servers by running php -S localhost:8000 at the command line(I assume people who fork the code in this way know what this means) and connecting a browser to http://localhost:8000 to edit *in situ*.

Pdf document

As useful as the Open Web is, it is also useful to have documents in a traditional format which is compatible with physical paper printers to make copies we can carry outside of digital readers. To do this we have several options. We can print from the browser(which will look bad), we can convert to Microsoft Word which will corrupt the file and also look bad, or we can use the LaTeX system which will look great but take more work. For this initial instance we focus on the LaTeX version. As this document replicates and evolves hopefully more

skilled users than the initial author will make smoother systems for conversion but right now a combination of the Haskell library "pandoc" and manual editing of the output file are the easiest way to convert from markdown to LaTeX. A document produced in this way is included in the replicator of this set.

```
pandoc -o paper.tex README.md
pdflatex paper.tex
```

5. Generalization and Social Implications

All ideas which we desire to propagate on the Open Web need all the typical means of use of media to try to convince others to replicate the document. In some cases we seek to explain the whole thing in a tiny capsule of information, as with the "elevator pitch". In other cases we seek to show by example the power of these ideas. The media we expect will be used in the spread of this system include *all* media in the most general sense, including physical things like machines which carry various writing on them. Any physical thing can have a domain name on them which points to a document which describes how to replicate the physical thing. This extremely generalized definition of self-replicating document(as a type of self-replicating set) means any physical object can be thought of as a self-replicating document. Combined with the

"Open Web" defined above, this can create an entire universe of useful things we can use to make up the fabric of our lives, building sets to live in which are independent of the existing industrial order.

Ultimately if we can build sets that we have the capability to evolve based on our desires, we can push that evolution toward what we call a "technological complete set". That is a set which describes a full self-replicating system that we can live in, which can exist in equilibrium with its environment just as pre-industrial societies did before the whole world was consumed by one very destructive set. In a complete set, the people who live in that set(we place ourselves conceptually into the set) have everything they need for a good life such as medicine, abundant food, clean water, the ability to live in a comfortable temperature etc. In addition, a set is not complete if it is out of equilibrium: if a set requires constantly destroying things and not replacing them to exist it is not complete. As we look to the future this is possible in a way that is totally different from what was possible before the rise of industrial society. A future complete set based society has no reason to mine, since the quantity of material that has been extracted from the Earth and processed into very ordered structures is more than enough for a large human population to live on indefinitely.

This Technological Complete Set does not need to be designed and built by any one person or group. All that

is needed to achieve such a set is to build sets which people have the capacity to evolve based on desire, and to impart into a group of people the desire to achieve this set(given the assumption that such a set is physically attainable). This document is meant to describe such a set, but also to serve as a seed which the author hopes will evolve in such a direction. By itself it is probably insufficient to build a large complex system, but it provides a prototype for a number of other self-replicating sets which we will construct and replicate over the Open Web. It is the nature of such sets that as they are created and released into the wild, they will all build on each other with network effects and that a small amount of exponential growth early on can create very large effects as the system evolves.

What is next in this program? The self-replicating sets which are currently being created and released are largely about replicating symbols. Symbols and logos play a very powerful role in how our minds process the world around us. The ability to create a self-replicating symbol which has some intention imposed on it is one of the most powerful forces in our world today. This is the power corporations wield with their brands, logos, and marketing messages. By building systems for very rapidly creating symbols, giving them meaning, and replicating them, we empower the masses with this same power. The media for the symbols includes artistic tools for physical media creation(stationary, wall art, post-

cards, signs) and digital media creation(vector graphics of simple geometric logos). From this power, we hope to build a fire that consumes the media landscape and transforms the nature of human existence on this planet.

Chapter 14

Trash Robot

14.1 Trash Robot

What is Trash Robot?

Everything here refers back to Ontology. We build this thing, show how to replicate it, show how replication can benefit the replicator, which stimulates further replication. Build things and sell them. Build things and use them. Build things and share them for mutual aid and benefit.

Metabusiness. SRS. Thing. Organic Media. Geometron vehicle. Maker swarm. Methodology. Trash Magic(self replicating media from trash, seed of full complete set).

Some roles in Trash Robot:

Printer Operator. You carry a printer with you, and print coins on demand.

Symbol Artist. Build sets of things, assign them symbols, find those symbols in image form, trace into robot code, save as sharable data, share with people, participate in symbol markets with other people around the world.

Token replicator. Carry bags of tokens, stamps, prints, use them to print replications of themselves as well as the tools to further replicate them and sell them or exchange them with other people, again as a global network of self-replicating media.

The Street Network can be used to sell a huge range of goods and services. Trash Robot is a specific collection of self-replicating technologies which can also be sold directly via the Street Network. It includes a list of things which will be described here, which readers are encouraged to copy and sell. But more importantly it provides a prototype for a new type of product which can be built at scale using the decentralized craft mode of production. As soon as this book is released and we build a swarm to start living Geometron and selling the elements of Trash Robot and bartering for the services provided by the Network we will also start building totally new products using this method. We will work to build products which have not exact equivalent on the market in the consumer economy and which would be very difficult to replicate in that space for various logistical reasons of how tech-

nology scales. IN particular we need to explore in this chapter the economics of enclosures and how cardboard and duct tape and geometron built by crafters and immediately sold at a profit on a one-at-a-time basis differs from injection molding of enclosures at scale.

The Open Brand

rainbow and googly eyes. Soft black textiles. Felt. Geometric constructions from Geometron. Use of geometry in cardboard fabrication, HDPE sheets. Modularity. Things carried in cloth bags. Blocky geometric font. duct tape and cardboard and sticks. Things built from trash. Images of the things: box, flag, shirt, pants, bags with robots. The fashion brand.

The ArtBox

As with everything in Trash Robot the ArtBox is a self-replicating set. The elements are the Trash Ties, the Tape Snake, a purse which is the box itself, and a collection of standard art supplies which sit in the box and are used to make another box. Trash Ties are 6 foot long sections of cotton clothesline, with the ends duct taped. We buy a whole package of 100 feet of clothesline, and cut into 6 foot sections, duct taping where the cut will be before cutting so each section has both ends taped. We then buy a collection of colored duct tapes which

includes black, red, orange, yellow, green, blue, purple, and pink, or one more if it is available. We thread the Trash Tie through all of them twice, and join it to itself with a square knot. This is the Tape Snake. We carry this around with us to anyplace where we are working on replication.

The contents of the box are box cutters, sharpie markers, a laser cut acrylic shape set as documented elsewhere in this chapter, and a bag of googly eyes(which can be purchased in bulk online for very little money), scissors, and a ruler, also documented elsewhere here.

This is a nice purse! If you make one and have the supplies you can go make one with a friend and they can make more and so on. And as you all make them and grow the swarm you can use them to carry around art supplies for other art projects as a super stylish art supplies purse until people notice and want to buy it, then a market will exist and we will sell them to whoever wants them. These are so striking and nice to use and cost so little to make and are so easy to copy that they can be a big part of what replicates the whole Trash Robot system which in turn helps replicate all of Geometron and the Street Network.

We will make millions of these to give away, share, or sell. The can be freely shared in our Network without money or even barter and just used as needed for other Geometron constructions, but can always be sold to people not in the Network.

Figure 14.1: Elements of the ArtBox and a Tape Snake.

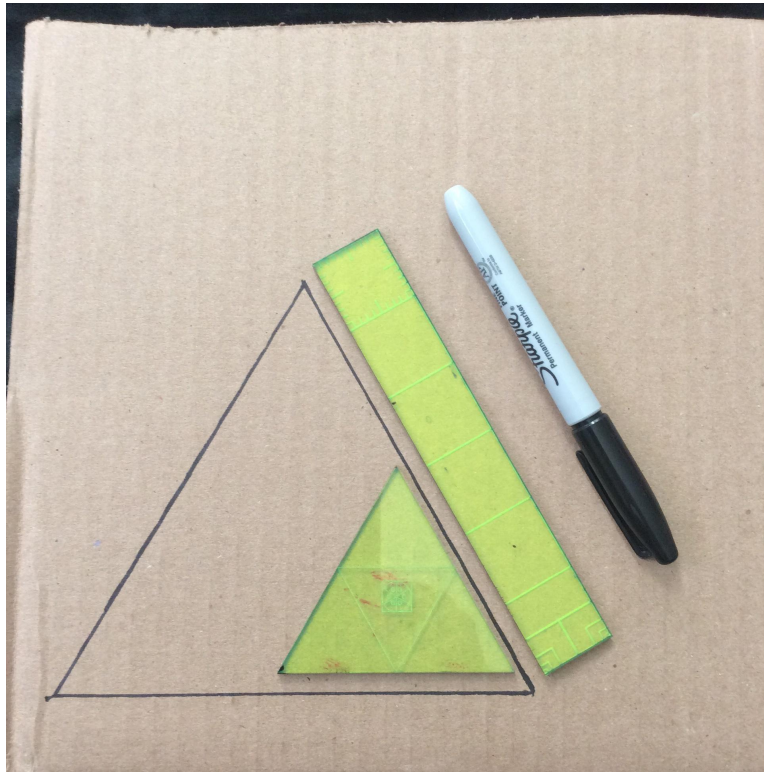


Figure 14.2: Cut out a 6 inch equilateral triangle from thick corrugated cardboard.

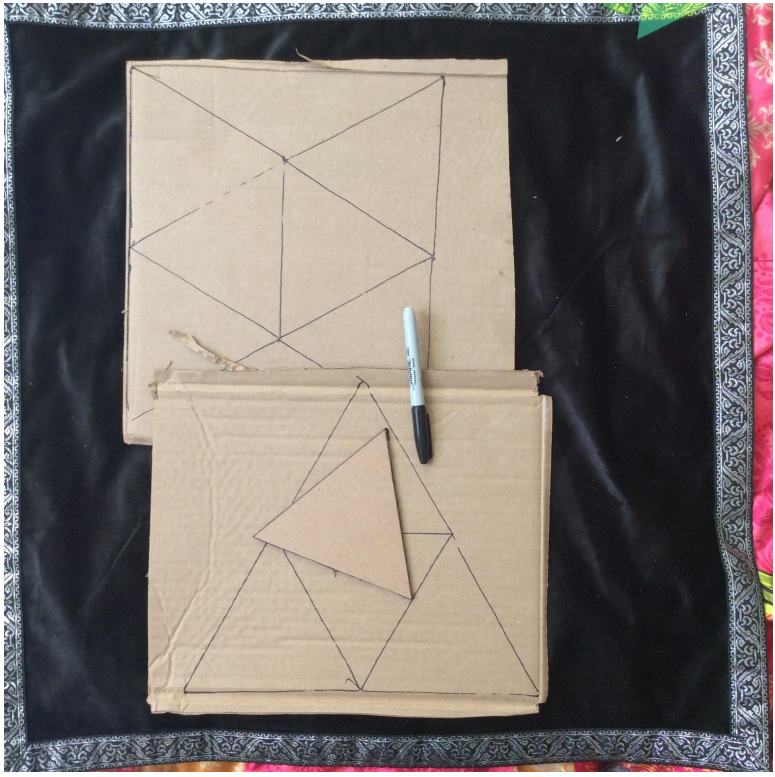


Figure 14.3: Copy so that you have a total of 10 triangles.

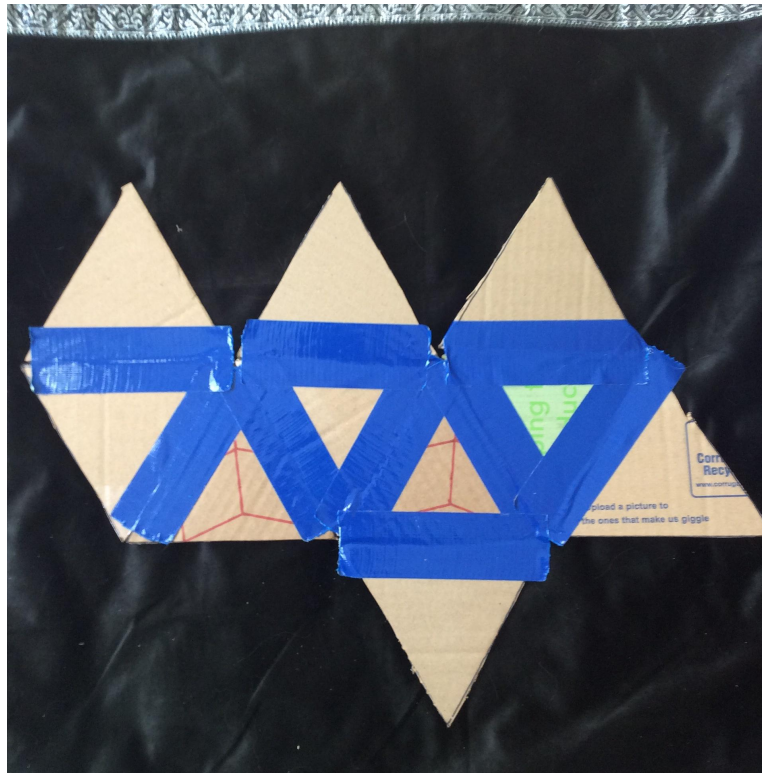


Figure 14.4: Stitch together the net pattern as shown.

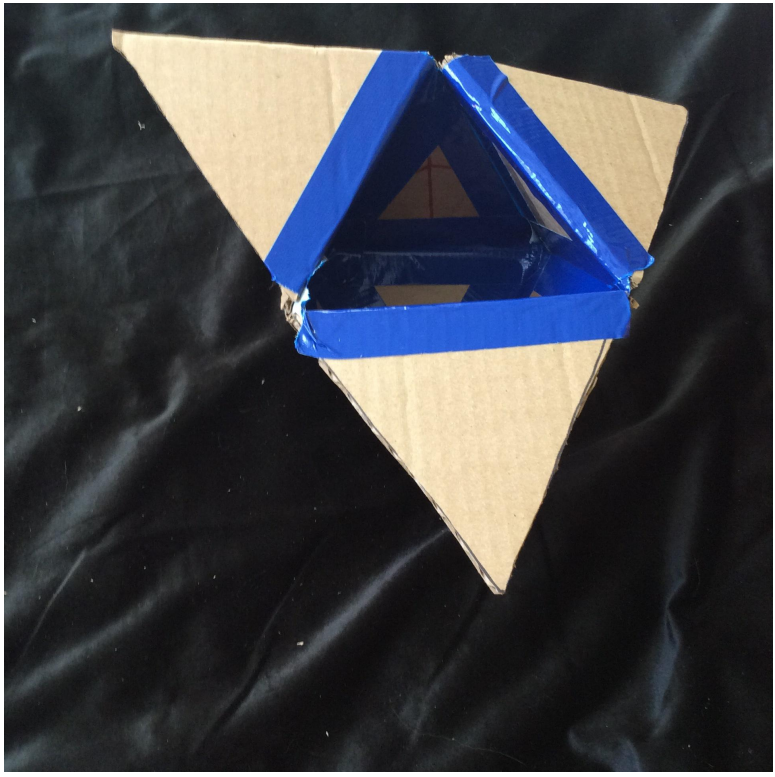


Figure 14.5: Assemble into octahedron with three top triangles meeting in a tetrahedron top the octahedron.

Figure 14.6: Use all colors of duct tape to create rainbow skin effect as shown. This is the prototypical Trash Robot rainbow open brand after we add the mouth and googly eyes.

Figure 14.7: Stab holes as shown in each of the top three triangles at the upper tips.

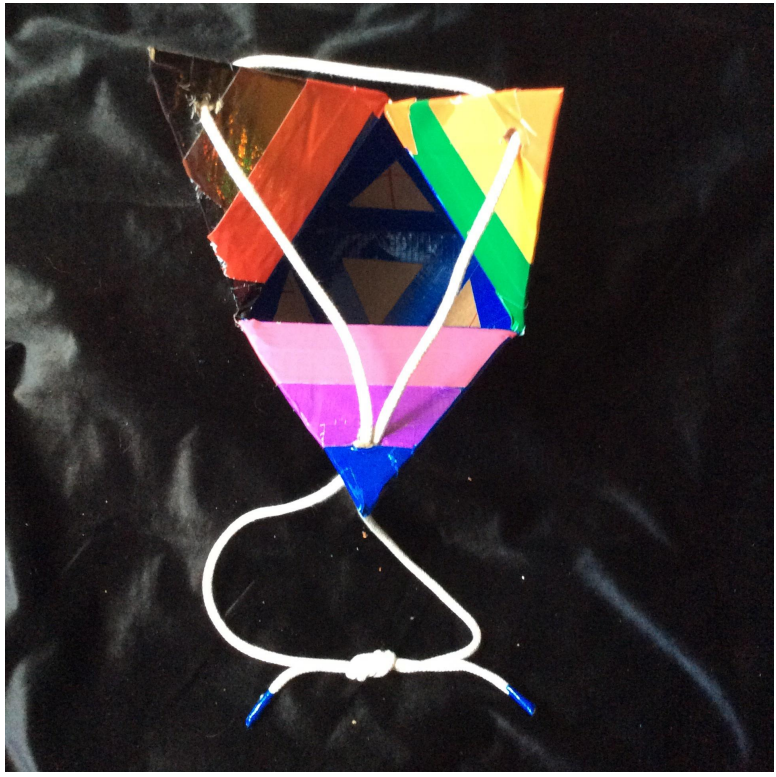


Figure 14.8: Thread a Trash Tie, as described above, through the the holes as shown and tie a square knot.

Figure 14.9: Complete the box design with a black duct tape piece for the mouth and crookedly placed asymmetric googly eyes. Multiple faces can all have the face design repeated. Now we replicate all the contents, finding another sharpie, another box cutter, etc. and put a whole kit required to make another box in our box to use to make more boxes. We also replicate the Tape Snake so that the new box has a Snake for replication, and pass some extra Trash Ties along for the next replication as well.

Laser Cut Acrylic

The Geometron workflow for creating .svg and .png files is designed among other things to make it as easy as possible to create files to use in commercial laser cutters. These have been used both in the kind of laser cutter available in maker spaces and from the professional service at Ponoko.com(these require slightly different files).

Laser neon green acrylic shapes in the basic Shape Set and the 6 inch Geometron ruler are key elements of our system, and replicate along with the rest of the ArtBox. Making custom or specialized geometry tools like this is a very good metabusiness for Trash Robot. We can sell the standard Shape Set, sell the Golden Triangle, the 6 inch Ruler, the Geometron Protractor, Penrose Tiles, custom spray stencils using the laser cut stencil font, and just custom arbitrary laser cut shapes for design clients.

A good design target for custom spray stencils is the domain name which points to the local Geometron Raspberry Pi Server on the Street Network.

Show each design. Link to global files. How to use Dremel laser cutter, how to use Ponoko.

skeletron

poles, geometry, tetrahedron, octahedron, icosahedron, tripods, flag poles, S Hooks, photos of constructions

textiles

the font. flags. Bags. Clothes. Methodology. Fashion business on the street. all the layouts of the designs. Waving the flag with the pole. Power of the flag to direct traffic to pages in the Network over the cardboard sign, legitimacy via the open brand.

The icon token printer robot

Build the brain.

Build the controller.

Build the mechanicals.

Workflow: image feeds, align, trace, share, print, bake, stamp, replicate, paint and sand, build into sets, share and sell, make pendants, stitch into clothing and accessories, make jewelry. Make more robots. Donate, share, and sell robots.

The symbolic currency, creating sets, the sigil boards, sets as fashion, sets as symbolic currency which can mean *anything*. This symbolic currency goes back to the previous chapter, where we discuss the general idea of self-replicating sets. For any given set of generalized "things" in the sense of the previous chapter, each thing can be assigned a symbol. That symbol can be traced into the robot code which drives the printer. All those symbols can be stored as a token feed which is saved and shared with other users, who can then print them all out. So an

abstract set can always be represented by a collection of Geometron icon symbols shared over a network and also a collection of clay painted icon tokens, carried in a bag as described in the Textiles section of this chapter, and replicated using more clay without use of a robot. So we can represent any self-replicating set with a set of symbols shared over both a local and global network and *also* a collection of physical tokens in a bag which are like coins in a coin purse. Attractive and convenient coin purses which contain symbols representing a self-replicating set which themselves are self-replicating media can form the basis of a very powerful symbolic economy based on sets, networks, and symbols rather than numbers.

The creation of a self-replicating symbolic currency which has both physical and digital versions is a fundamentally different type of currency than money. Because it all self-replicates, the idea of property simply does not make sense, because the value of replication is reversed. If I own a plot of land or a car, its value is based on how hard it is to replicate. If it were possible to freely replicate a billion acres of the best land imaginable with warp drive to get to the best location, your million dollar plot of land will be worthless. If someone builds a free car that grows naturally in a forest and runs on the sun, your 40,000 dollar gas car will be worthless. But if you build a robot from trash to fabricate useful things, the more people copy the robot the more value there is in the whole network. If a billion people copy my robot, I can

get a million of the cleverest makers in the world all improving the design, and what I end up using a few years from now light years beyond anything I could have even dreamed of. So the more people replicate a free thing the *more* that thing is worth to me, even as the creator of the thing. This makes the way value works the opposite of property. Value goes up with abundance rather than scarcity. If we make the choice to value abundance, meaning the freeness of things, then that is what will end up with value. But this replication also means that any numerical unit of value will very quickly break as the supply of value exceeds the total currency available again and again(essentially deflation).

Some notes on the pi version, how this can be a pi driven robot.

Arduino Generic Shield

This is a stub. It expands into many technologies but we document just the board and most basic of codes. Future technologies will reference this.

```
Trash Robot = {
    ArtBox,
    Trash Tie,
    Tape Snake,
    Token Printer,
    Terminal,
```

```
    Brand,
    Textile,
    Shop,
    skeletron,
    constructions
}

constructions = {
    duct tape,
    cardboard,
    trash ties,
    HDPE sheets
}

shop = {
    ArtBox purse,
    Trash Robot branded clothes,
    laser cut acrylic,
    token printer kits,
    tokens,
    pendants,
    printed bottle caps,
    terminal install
}

laser cut acrylic = {
    golden triangles,
    penrose tiles,
```

```
    full set,
    ruler,
    protractor,
    custom shape,
    spray stencils
}

printer = {
    brain,
    controller,
    mechanicals,
    workflow
}

    printer workflow = {
        build, share, sell, use printers, following instructi
        use Geometron server to follow the rest of this workf
        image feeds,
        aligner,
        trace,
        share feed with other users, save, copy, paste, share
        load code into Arduino, print in clay tablet, bake it
        use print to create stamp, sell or give away or use,
        use stamp to create both coin-like tokens and pendant
    }
```

Chapter 15

Full Stack Geometron

15.1 Full Stack Geometron

- end goals
- hybrid fabrication technology
- clockless operation, hardware GVM
- image stack, hardware map processing
- roctal, storage hardware, scaling, read/write/operate
- large projections in Skeletron booths, Geometron station, fully immersive VR/AR at 2 meter scale

Goals

Our end goal is an information technology which has zero mined materials. Every material input in the creation of any technology in our system should come from either a

waste stream or from the living world in a closed loop(e.g. naturally harvested sticks and tree sap, goose guano).

Our initial goal has been to rely on the most open and lowest cost off the shelf hardware possible, which right now is the Raspberry Pi. As the system expands we will want more and more ways to fully wipe hard drives and sim cards and replace them with Raspberry-Pi-like systems running some form of Linux, with web servers installed with PHP running the same sofwtware. That will be the next step after the Pi phase. But after that we will try to break up the system into modular parts which can be scavenged from more and more destroyed electronics.

We see this as a continuous transition where in the beginning we are just running an app on an existing phone, then we wipe the phones and run our own OS, and then we start taking screens, batteries, and other components out and mixing and matching them. It is in this mixing and matching where the next phase of development will happen.

As this mix-and-match upcycled Linux system evolves, the Trash Robot will in parallel evolve more elaborate ways to fabricate electrical connections at the millimeter scale. This will eventually lead to more and more complex circuits being built up from scratch, connecting various upcycled components at the individual level(a single resistor, capacitor or MOSFET for instance). When these circuits can be fully integrated in three dimensions

into collections of upcycled components, and when we have some degree of automation in this process, we can start to say we have a fully upcycled system, but it will still be the same software stack, with Geometron running on PHP, Apache, and Linux.

This hybrid interconnect technology can be based on immersion of circuits in an electrolyte solution with a robotic probe which applies current to deposit metal, as well as techniques where the robot probe is modifying clay, and a series of clay fabrication steps are carried out in layers just as conventional micro-lithography adds layers.

In parallel to all of this we need to be developing battery technology using aluminum from beverage cans with carbon from charcoal made from organic material like burned dung. We also need energy production based on ultra small scale generation using found materials. So our goal will be to find a reliable waste stream like fans from broken computers, turn that into a generator of electricity which can run on very small scale water or wind at the level of from 100's of watts down to single digits of watts. Using the aluminum air cycle with organic waste and beer cans from the trash it should be possible to have a zero-mine closed loop energy production and storage system at the level of IT platforms. Building IT infrastructure as a shared resource in some local public place which stays fixed in a location removes the severe demands currently placed on battery technology for net-

worked devices which have to fit in a pocket. If servers also serve as the information terminal, with a giant screen for all passerby to see, a battery could be a 6 foot high stack and it would have no real impact on usability. This relaxed restriction combined with using waste aluminum cans and locally sourced burned dung should make aluminum air cycle batteries practical in a way they have not been traditionally.

Another technology which has to be developed as part of the Trash Robot road map is printing in permanent materials at the micron or sub micron scale for long term information storage. This has to be developed in parallel with readout technology which uses upcycled camera chips from trashed smart phones as sensors in our hybrid fabrication technology which can read the information back out.

Developing our own information storage system is also a part of a much more radical shift in information technology, where we abandon the whole operating system and build hardware for direct Geometron-based document processing. To do this, we re-adjust what a machine does. The only goal of our machine is to freely share documents over a network. When a machine is not operating something mechanical like the Trash Robot, all it does is display pixels on the screen. Those pixels only change if a user engages with the system. We therefore totally replace the model of a Turing machine doing arithmetic on ones and zeros with a model which

just uses the Geometron Virtual Machine to draw vector graphics in the screen pixels(this can include all text in all human languages, which can all be drawn using Geometron fonts) and a system for displaying bitmaps in geometric positions as described in the Geometron Map format. To do this, we will need an image stack, a memory system which records an array of bitmaps of images, and a Geometron Virtual Machine. We will also need a network of switches, where user interaction with keyboards, pointers, buttons etc. can trigger behavior like scrolling or editing a scroll, zooming on a map, drawing a symbol, or clicking on a link which goes to some other file.

All this does not need a constantly running clock. This is one of the biggest differences between traditional computers and a full stack Geometron machine. When there is no user interaction it simply does nothing. It doesn't just have minimal operating system services running in the background, it doesn't even have a clock. No aspect of the system changes state in any way. It is just a collection of pixels getting enough power to stay in whatever state they are in, until the user engages, and then it only changes state in direct response to each user interaction.

The data we store in the system can be either short term or long term. Short term data can be stored in a modularized memory unit based on easily found memory chips from trashed phones, but with standardized

interconnects added by the Trash Robot fab technology so that they can be easily plugged in and replaced after they are upcycled into our system. Long term memory is designed to be *very* long term. We can use scaled down versions of the Trash Robot clay printer presented here to print in clay at a much smaller scale. As with the larger clay symbol tokens, this can also be self-replicating, as the depressions in the clay can again be used to stamp out inverse images which can be used to make more. So with no automation or IT at all someone with no tools or even electricity should be able to clone data storage media.

The format for long term storage is in arrays of squares or dots which are either in a 1 or zero state, filled in or empty. If possible, the data all have redundancy, with Arabic numerals written along with each Geometron bytecode byte, as well as in some cases an actual print of the symbol being represented, and alignment marks for that individual byte. This redundancy and alignment mark system should make the data maximally robust against decay and future technological limitations, as if it were absolutely necessary, the