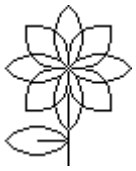
Welcome to the Permacomputing wiki!



permaflower

Permacomputing is both a concept and a community of practice oriented around issues of resilience and regenerativity in computer and network technology inspired by permaculture. ☊⌇☋☾ ☼☆ -☆:*´

There are huge environmental and societal issues in today's computing, and permacomputing specifically wants to challenge them in the same way as permaculture has challenged industrial agriculture. With that said, permacomputing is an anti-capitalist political project. It is driven by several strands of anarchism, decoloniality, intersectional feminism, post-marxism, degrowth, ecologism.

Permacomputing is also a utopian ideal that needs a lot of rethinking, rebuilding and technical design work to put in practice. This is why a lot of material on this wiki is highly technical.

Most importantly, there is no permacomputing kit to buy. See permacomputing as invitation to collectively and radically rethink computational culture. It is not a tech solution searching for a problem.

For more information:

- Permacomputing
- Principles for people making things
- Issues in today's computing

Resources:

- Getting Started - a starter's manual of some sort
- Library - texts and media about and around permacomputing
- Projects - selection of projects affiliated with or adjacent to permacomputing
- Assessments - review and analysis of existing and established pieces of technology
- Communities - permacomputing communities of practice and related
- Events - such as talks, workshops, meetings and seminars

1. Permacomputing

Permacomputing is both a concept and a community of practice oriented around issues of resilience and regenerativity in computer and network technology inspired by permaculture.

In a time where computing epitomizes industrial waste and exploitation, permacomputing encourages a more sustainable approach, maximizing hardware lifespans, minimizing energy use and focussing on the use of already available computational resources. Permacomputing asks the question whether it is possible to rethink computing in the same way as permaculture rethinks agriculture. Permaculture is the science and practice of creating semi-permanent ecosystems of nature. The resilience of any such ecosystem is equal to its diversity and interconnectedness. Permaculture design is a system of assembling conceptual, material and strategic components in a pattern which functions to benefit life in all its forms. It seeks to provide a sustainable and secure place for living things on this earth.

At first it may seems paradoxical to connect permaculture and computation. Indeed, an extractive technology that depends on a wasteful use of finite resources can hardly be permanent. Therefore, by making this connection, what we are truly asking is whether or not there can be a place for computer and network technology in a world where humans contribute to the well-being of the biosphere rather than destroy it? And if yes, how?

Permacomputing wants to imagine such a place and take steps towards it. It is therefore both utopian and practical. We want to find out how we can practice good relations with the Earth by learning from ecological systems to leverage and re-center existing technologies and practices. A radical reduction of wastefulness is a fundamental aspect of it: maximize the hardware lifespans, minimize the energy use. And this is not just about a set of technical problems to be fixed—the attitudes also need a radical turn. Understandability is aesthetics, virtual does not mean immaterial and doing things with less is not *a return to the past*. We want to investigate what a permacomputing way of life could be, and what sort of transformative computational culture and aesthetics it could bring forward.

The principles of permacomputing are:

- Care for life
- Care for the chips
- Keep it small
- Hope for the best, prepare for the worst
- Keep it flexible
- Build on solid ground
- Amplify awareness

- Expose everything
- Respond to changes
- Everything has a place.

Properties of permacomputing systems

The principles concretely manifest themselves in various forms so as to highlight the following properties:

- **accessible**: well documented and adaptable to an individual's needs.
- **compatible**: works on a variety of architectures.
- **efficient**: uses as little resources (power, memory, etc) as possible (minimization).
- **flexible**: modular, portable, adapts to various use-cases.
- **resilient**: repairable, offline first, low-maintenance, designed for disassembly, planned longevity, lifespan maximization, descent-friendly or design for descent

Some additional concerns are of indirect interest because they impose costs on the entire end-to-end process of software creation:

- it's bootstrapping from machine code without circular reasoning (bootstrappable builds)
- it's obvious what source code went into it (reproducible builds)
- it's easy to audit its source code, including all dependencies

2. Principles

These **design principles** have been modeled after those of permaculture.

These are primarily design/practice principles and not philosophical ones, so feel free to disagree with them, refactor them, and (re-)interpret them freely. Permacomputing is not prescriptive, but favours instead situatedness and awareness of the diversity of context. Said differently, its design principles can be as much helpful as a way to guide practice in a specific situation, as it can be used as a device to help surface systemic issues in the relationship between computer technology and ecology.

Also, this is a big work-in-progress :)

Care for life

This is the ethical basis that permacomputing builds on. It refers to the permacultural principles of "care for the earth" and "care for people", but can be thought of as the basic axiom for all choices.

Create low-power systems that strengthens the biosphere and use the wide-area network sparingly. Minimize the use of artificial energy, fossil fuels and mineral resources. Don't create systems that Jevons paradox.

Care for the chips

Production of new computing hardware consumes a lot of energy and resources. Therefore, we need to **lifespan maximization** of hardware components – especially IC, because of their low material recyclability.

- Respect the quirks and peculiarities of what already exists and repair what can be repaired.
- Create new devices from salvage computing.
- Support local time-sharing within your community in order to avoid buying redundant stuff.
- Push the industry towards **Planned longevity**.
- Design for disassembly.

Keep it small

Small systems are more likely to have small hardware and energy requirements, as well as high understandability. They are easier to understand, manage, refactoring and repurpose.

- dependency (including hardware requirements and whatever external software/ libraries the program requires) should also be kept low.
- Avoid pseudosimplicity such as user interfaces that hide their operation from the user.
- **Accumulate wisdom and experience rather than codebase**.
- **Low complexity is beautiful**. This is also relevant to e.g. visual media where "high quality" is often thought to stem from high resolutions and large bitrates.
- Human-scale: a reasonable level of complexity for a computing system is that it can be entirely understood by a single person (from the low-level hardware details to the application-level quirks).
- Scalability (upwards) is essential only if there is an actual and justifiable need to scale up; down-scalability may often be more relevant.
- **Abundance thinking**. If the computing capacity feels too limited for anything, you can rethink it from the point of view of abundance (e.g. by taking yourself fifty years back in time): tens of kilobytes of memory, thousands of operations per second – think about all the possibilities!

Hope for the best, prepare for the worst

It is a good practice to keep everything as resilient and collapse-tolerant as possible even if you don't believe in these scenarios.

- While being resilient and building on a solid ground, be open to positive and utopian possibilities. Experiment with new ideas and have grand visions.
- Design for descent.

Keep it flexible

Flexibility means that a system can be used in a vast array of purposes, including ones it was not primarily designed for. Flexibility complements smallness and simplicity. In an ideal and elegant system, the three factors (smallness, simplicity and flexibility) support each other.

If it is possible to imagine all the possible use cases when designing a system, the design may very well be too simple and/or too inflexible. Smallness, simplicity and flexibility are also part of the "small, sharp tools" ideal of the Unix command line. Here the key to flexibility is the ability to creatively combine small tools that do small, individual things.

- Computing technology in general is very flexible because of its programmability. Programming and programmability should be supported and encouraged everywhere, and artificial lock-ins that prevent (re)programming should be broken.
- Design systems you can gradually modify and improve while running them.

Build on a solid ground

It is good to experiment with new ideas, concepts and languages, but depending on them is usually a bad idea. Appreciate mature technologies, clear ideas and well-understood theories when building something that is intended to last.

- Avoid unreliable dependency, especially as hard (non-optional) dependencies. If you can't avoid them (in case of software), put them available in the same place where you have your program available.
- It is possible to support several target platforms. In case of lasting programs, one of these should be a bedrock platform that does not change and therefore does not cause software rot.
- **Don't take anything for granted**. Especially don't expect the infrastructure such as the power grid and global networking to continue working indefinitely.
- You may also read this as "grow roots to a solid ground". Learn things that last, enrich your local tradition, know the history of everything.

Amplify awareness

Computers were invented to assist people in their cognitive processes. “Intelligence amplification” was a good goal, but intelligence may also be used narrowly and blindly. It may therefore be a better idea to amplify awareness.

- Awareness means awareness of whatever is concretely going on in the world/environment but also awareness of how things work and how they situate in their contexts (cultural, historical, biological etc).
- You don’t need to twiddle with everything in order to understand it. balance of opposites emphasizes observation.
- It may also often be a good idea to amplify the computer’s awareness of its physical surroundings with things like sensors.

Expose everything

As an extension of “amplify awareness”: Don’t hide information!

- Keep everything open, modifiable and flexible.
- Share your FLOSS and design philosophies.
- **State visualization**: Make the computer visualize/auralize its internal state as well as whatever it knows about the state of its physical environment. Regard this visualization/auralization as a background landscape: facilitate observation but don’t steal the attention. Also, don’t use too much computing resources for this (updating a full-screen background landscape tens of times per second is a total overkill).

Respond to changes

Computing systems should adapt to the changes in their operating environments (especially in relation to energy and heat). 24/7 availability of all parts of the system should not be required, and neither should a constant operating performance (e.g. networking speed).

- In a long term, software and hardware systems should not get obsoleted by changing needs and conditions. New software can be written even for old computers, old software can be modifed to respond to new needs, and new devices can be built from old components. Avoid both software rot and retrocomputing.

Everything has a place

Be part of your local energy/matter circulations, ecosystems and cultures. Cherish locality, decentralization. Strengthen the local roots of the technology you use and create.

While operating locally and at present, be aware of the entire world-wide context your work takes place in. This includes the historical context several decades to the past and the future. Understanding the past(s) is the key for envisioning the possible futures.

- Nothing is “universal”. Even computers, “universal calculators” that can be readapted to any task, are full of quirks that stem from the cultures that created them. Don’t take them as the only way things can be, or as the most “rational” or “advanced” way.
- Every system, no matter how ubiquitous or “universal” it is, is only a tiny speckle in a huge ocean of possibilities. Try to understand the entire possibility space in addition to the individual speckles you have concrete experience about.
- **Technological diversity**, avoid monoculture. But remember that standards also have an important place.
- Strict utilitarianism impoverishes. Uselessness also has an important place, so appreciate it.
- You may also read this principle as: **There is a place of everything**. Nothing is obsolete or irrelevant. Even if they lose their original meaning, programmable systems may be readapted to new purposes they were not originally designed for. Think about technology as a rhizome rather than a “highway of progress and constant obsolescence”.
- There is a place for both slow and fast, both gradual and one-shot processes. Don’t look at all things through the same glasses.

3. Issues

There are several issues in today’s computing.

If the information and communications technology (ICT) industry was purposefully following a harmful agenda, these would probably be their design principles:

- disregard for life
- disregard for the chips
- more is better
- assume limitless resources
- keep it controlled
- outsource the problem
- amplify ignorance

- obfuscate everything
- destroy communities
- achieve monopoly

While the industry is rarely discussed in this way, there are increasingly well-documented issues in the dominant computational cultural paradigm which all together help better explain how such a harmful agenda can emerge explicitly or implicitly.

Additional topics to explore:

- **inaccessible:** greenwashing, Californian ideology, otherness, pseudosimplicity
- **incompatible:** vendor lock-in, proprietary
- **inefficient:** bloat, maximalism, cryptocurrency, calculation factory, cornucopianism
- **rigid:** monoculture, siliconization
- **failing:** silver bullet, planned obsolescence, wishcycling, software rot
- **extractivist:** attention economy, capitalism, Big Tech, neoliberalism

4. Getting started

What can I do?

This is a frequent question, and not an easy one to answer.

Permacomputing can take many forms, and every context and situation is different. For someone, getting started with permacomputing may be:

- helping a school to work with recycled computers
- learning how to repair and replace components in computing hardware
- discussing the use and impact of smartphones in the household, or data centers in the workplace
- working with local farms and collectives to develop low energy weather prediction
- researching how to provide less resource-intensive tools and systems for their lab or workspace
- getting involved with initiatives to create energy efficient and accessible local libraries of information
- engaging with politics and policy making to advance tech and enviromental regulation in their institution, town, or region

- helping artists interested to engage with ecological topics using tools and media in line with this intention
- writing their own FORTH for a chip reclaimed from e-waste

Each of these can mix and match, and are also examples from the following categories of action.

Participating

Join discussions in your institution, union, building, company, or town council, to figure out new ways to discuss the impact and regulation of the usage and re-use of computers.

Join a union, join an environmentalist group, join a citizen science lab, etc.

It's also important that users of hardware and software feel confident enough to voice their opinion, especially when the development of these projects is done in a relatively open way. For instance, issue trackers can be important activation sites to voice struggles beyond reporting technical faults.

Experimenting

Investigate the places in your life or work involving computer use. How much energy does it use, including accessed network resources? Can it be reduced, replaced, or removed? What impact does it have on your wellbeing and your community? Is there a common task you perform online which could be moved locally? How can computer use for a particular task be phased out?

Approach computer use and acquisition with longevity in mind, considering things like whether you need to buy new hardware, or could you repurpose an old computer or device instead? Does it need to use a computer at all?

When developing something new, what are you gonna use to ensure you make something that does not end up being harmful or wasteful? How will you measure the impacts of your project, like the resources used to create and run it including energy, fresh water, and waste heat? Can your new system be self-obviating system?

With computation and computer tech consumerism taking such a big space in so many activities, it is very likely that a domain, a common/everyday tool, or a practice, in which you have interest, professionally or not, could become your field of experimentation. Do you need to acquire new skills? How will you acquire these skills? Can you afford to learn such skills? Could you figure it out with the help of others?

Reading and learning

For non-programmers and infrequent computer users, some minimal knowledge of computing jargon and practices is recommended. However, the question of literacy in relation to computational culture is often reduced to staying in the loop with the "latest developments" the tech industry and acquiring technological skills, when we need to talk more about the other way around.

A lot of the radical thinking in computer science and engineering seems to be too often stuck on the same old 60-70s countercultural ideas from the United States. We think that it's important that people with a strong technological background start to catch up with decades of the various strands of computational critique discussed in feminist studies, gender studies, software studies, cultural studies, and also arts and humanities.

Organizing

Consider starting a local group around permacomputing. You don't have and should not try to work on this topic on your own! Talk to local cultural organisations, hackerspaces, squats, town councils, schools and universities to help organising some events, workshops, skill-sharing sessions, show&tell, etc. Try to bootstrap a small permacomputing community. Make use of our terms to get you started with questions of moderation, make use of the wiki, communicate on the existing lists, chats, or start your own!

Publishing

Regardless what you do, it will be very inspiring to others if you document it a bit, both successes *and* failures. It does not have to be extensive, but it can be a much more effective way to demonstrate how to activate critical practices in relation to computer techology. You can do that on your own website if you have one, you can make zines, something individual or something with others, and of course contribute! More generally publishing does not need to be only about the projects you are involved with directly, maybe it is about helping others writing a manual, a cookbook, a sampler, something relatable and accessible.

Breaking the monoculture

Like any other community of practice that emerged from contemporary computer tech circles, permacomputing suffers from very poor cultural diversity. How can we make this space more accessible and inclusive? Like, really, and not just empty words. How can the privileges that some of us have to be able to dedicate time on such topics can be generative of activities that can contribute to breaking the tech monoculture

and how can the permacomputing space, including this wiki, can become a platform for less privileged groups to be represented *and* supported?

Note: the first version of this document was motivated by, and in part inspired from, discussions and contributions from participants of the LIMITS 2023 workshop. THANKS <3

5. Library

this.is.a.work.in.progress :) complete reorganization and new addition is underway *

Texts

Permacomputing specific

- Viznut's original text, 2020.
- Discussion on related concepts, Viznut, 2021.
- Interpretation of permacomputing, Neauoire, 2021.
- Selected quotes on Permacomputing, Sejo, 2022
- Permacomputing Aesthetics: Potential and Limits of Constraints in Computational Art, Design and Culture, ugrnm, mr_ersatz, sandu, viznut, 2023.
- Permacomputing and the Dance of Repair Amid the Vestiges of Digital Obsolescence, sister0, 2023.
- The Oceanic provenance of Permacomputing and Computational poetics, sister0, 2024.

In relation to permacomputing

- Summary of related terms, l03s, 2021.
- In relation to ICT blogpost, Robert Engels, 2022.
- What might degrowth computing look like?, Neil Selwin, 2022.
- permacomputing in LeMonde, Nastasia Hadjadji 2024

Alternative perspectives

- Woods and Branlat, "Basic patterns in how adaptive systems fail", 2011

Damaged Earth Catalog

The Damaged Earth Catalog is a growing online catalog of the different terms in circulation, used by communities of practice, in relation to computing and network infrastructure informed by ecological ethics, degrowth, resilience, repair, and minimalism. It is currently developed by l03s as part of her PhD research.

https://damaged.bleu255.com

See also the DEC entry.

Films and moving images

These movies, documentaries and audiovisual works can be used to bring context for the underlying issues that are motivating both practices and discussions about permacomputing and related.

Non-Fiction

- Unser täglich Brot (2005)
- Nord-sud.com (2007)
- Waste Land (2010)
- Blood in the Mobile (2010)
- Plastic China (2016)
- Fais-le toi-même (2016)
- Death by Design (2016)
- Low-Tech (2023)

Fiction/Arthouse/Experimental

- Core Dump (2018-2019)
- Weitermachen Sanssouci (2019)
- Neptune Frost (2021)

Talks/Stream

- Vivre sa passion sur Tara Tari (2013)
- “Fake” Chips? (2020)
- From appropriate technology to permacomputing: a glossary of counternarratives and practices (2022)
- On permacomputing aesthetics (2023)
- An approach to computing and sustainability inspired from permaculture (2023)

6. Projects

This is a list of projects that share goals and/or values with permacomputing, either explicitly or because we think they do connect somehow Of course given how the permacomputing definition and principles leave some room for indidividual and collective (re-)interpretations, we are aware of the difficulty in framing precisely what may or may not fit precisely.

At time of writing there is no plan to go for ISO certification :) however, we do hope that the projects who make use of the term permacomputing can be articulated meaningfully and demonstrate a practice in relation to the permacomputing and principles.

This is why this section is also an opportunity to review and highlights strengths and pitfalls of these various projects, and inspire each other. Make use of the Discussion page of each project for comments and discussion.

The people behind the following projects may have used the term permacomputing to describe their work, or we see strong and/or potential relevance:

- Collapse OS
- Gemini
- Chifir
- Uxn
- Civboot
- Freewheeling Apps
- Mu
- Teliva

Projects whose relevance has not yet been fully assessed:

- DawnOS
- Solar Protocol

It is important to remember that these are attempts at drawing lines at the edge of one's computing needs, and personalized systems to address those needs, but are by no means "permacomputing products". Permacomputing is about finding these limits, and not their artifacts.

Historical

- PADI
- BBC Domesday Project
- CARDIAC, by Bell labs

7. Assessments

Permacomputing is also learning to deal with existing technologies, often trying to find the least evil among bad alternatives. In hardware we can't afford being too picky because we'll want to lengthen the lifespans of already existing pieces of hardware (even bad ones), but in software we usually have more choice.

When assessing software and hardware, we'll want to focus on technicalities such as resource use (especially dependency and documentation – mostly because many pieces of today's technology fail miserably in these areas.

See also:

- Hardware
  - Computers
  - single-board computers
  - Microcontrollers
  - Peripherals
- Software
  - Programming languages
  - Operating systems
  - Protocols
  - File formats

8. Communities

It's difficult to speak of Permacomputing as one single community. And why would we want to? We want to welcome and include as many people as possible in the discussion.

Discussion

At the moment several groups use the following means to exchange and stay in touch with each other to discuss permacomputing and related topics:

- **IRC:** #permacomputing channel on libera.chat - permacomputing specific discussions;
- **XMPP:** lowtech/permacomputing - permacomputing specific discussions;
- **XMPP:** moddingfridays newbie friendly mutual help for electronic repair and creative modding;
- **Forum:** permacomputing on the SLRPNK Lemmy instance;
- **Forum:** permacomputing on the SDF Chatter Lemmy instance;
- **Email discussion list:** permacomputing /at/ we lurk /dot/ org - permacomputing specific discussions;
- **#permacomputing**: obvious hashtag that we use on socials (mainstream and alt);

Note: the IRC channel, XMPP chatrooms, and the list are moderated and you must agree to our terms. The Forums are also moderated and have their own terms.

List of communities

The following includes communities that may share some goals, practices or ideas with permacomputing. For ideas and ideologies that don't have communities around themselves, see concepts.

Practical:

- Appropriate technology
- Collapse computing
- FLOSS
- Smallnet
- See also: projects

Academic:

- Computing within Limits
- Sustainable ICT

Activism, Policy Making and Legislation:

- offline first
- right to repair
- Green software engineering

Artistic:

- Holistic Computing Arts
- Small File Media Festival
- Algorave
- demoscene
- solarpunk
- Ecological Art and Design

9. Events

This section features workshops, talks, and other permacomputing connected activities. If you are planning to organize your own event and want to be featured or would like to link an existing permacomputing activity, please contact us!

Talks

- **2024/06/26** - The Oceanic provenance of Permacomputing and Computational poetics, at ISEA2024, Aus, with sister0

- **2023/12/02** - Practices of Digital Resilience & Permacomputing, at FIBER, NL, with brendan, cmos4040, l03s, luen, Marie Verdeil, ola, orx, Sunjoo Lee, ugrnm and unbinare
- **2023/09/21** - Approach To Computing and Sustainability Inspired From Permaculture, at Strange Loop 2023, USA, with neau
- **2023/06/14** - permacomputing aesthetics, at LIMITS23, US/International, with brendan, dusan, ugrnm and viznut
- **2023/02/22** - Permacomputing in the academy: how to problematise computer technology in art and design education, keynote at ETHO/ELIA, UK, with ugrnm
- **2022/10/28** - Introduction to Permacomputing, at Zine Camp, NL, with ugrnm and ola
- **2022/3/10** - permacomputing overview, at Natural Intelligence Lab (FIBER), NL, with l03s, ola and ugrnm

Courses, workshops and seminars

- **2024/07/18** - Decolonising digital culturescapes: permacomputing from an antipodean perspective, at EASST-4S 2024 Amsterdam: Making and Doing Transformations, with sister0
- **2023/11/30-2023/12/01** - Networking with Nature: Connecting plants and second-hand electronics, at Fiber Reassemble Lab 2023, NL, with brendan and orx
- **2023/11/17-2023/11/18** - Imagination above Productivity: Resurfacing and scaling the digital ecosystem around us, at Fiber Reassemble Lab 2023, NL, with ola
- **2023/10/13-2023/10/14** - The Cloud is Just my Old Computer: Creating a permacomputing server, at Fiber Reassemble Lab 2023, NL, with luen and ugrnm
- **2023/01/7** - permacomputing and Low-power Photography, at The Sustainable Darkroom, UK, with Felix Loftus
- **2022/04/01-ongoing** - permacomputing in the arts, at the Willem de Kooning Academy, NL, with ugrnm

Meetings and seminars

- **2024/12/03-09/07** - Digital Caretaking ☲fireside☲ Talks a series of 12 Online talks, every second Tuesday, with Shahee Ilyas, Nancy Mauro-Flude, Kate Rich, Denisa Reshef Kera, Jo Pollitt, Samara McIlroy, Jason James and ugrnm, hosted by sister0, Tasmania, Australia
- **2024/02/28** - Permacomputing Meet-Up, Offline.place in Berlin, Germany.
- **2023/09/20** - Permacomputing Meetup, Saint-Louis, Missouri, US
- **2023/06/22** - permacomputing seminar, at Royal Halloway University of London, UK, with Olga Goriunova, l03s, Dave Young and ugrnm
- **2022/08/20-2021/08/21** - permacomputing wiki edit-a-thon, at Varia, NL

- **2022-ongoing** - Regular permacomputing meetups at Iffy Books, in Philadelphia, PA, US

TODO: lots of stuff missing: Critical Infra lab launch, etc.

10. Minimization

Minimization is used here to mean the limiting of the use of artificial energy for data storage and transfer. Compression is a form of minimization, but minimization does not necessarily means the optimization of data size through compression.

See also:

- Bandwidth minimization
- Media minimization

11. Design for descent

Designing for Descent ensures that a system is resilient to intermittent energy supply and network connectivity. collapse computing prioritizes community needs and aims to contribute to a knowledge commons in order to be able to succeed in case of infrastructure collapse.

12. Bootstrapping

Bootstrapping has many meanings in the computer world. The most common meaning is now more commonly referred to as **booting** (i.e. the process of starting a computer). Bootstrapping a software or a programming language, however, generally means making it available on a computing platform starting from a very elementary level.

Bootstrappability is important for permacomputing for ensuring that arbitrary software can be run in an indefinite future where computing environments can be very different.

Bootstrappability is often a problem with programming languages that have been implemented in themselves (like, a C compiler written in C, or a Rust compiler written in Rust). Usually, a language is made available on a new processor architecture by cross-compiling its compiler to it, but if that option is not available, bootstrapping is needed.

In case of ordinary software, bootstrapping means not only compiling the program itself but also compiling/bootstrapping all of its software dependency from the lowest level, including the operating systems. The amount of computing resources

(especially storage space and computing time) required for bootstrapping can be used to measure **bootstrap complexity**.

Binary executables compiled for simple virtual machines can be used to help bootstrapping. A good idea might be to have a simple, easily retargettable C compiler available as this kind of executable.

Classical, bare-hardware Forth environments have often been created in a bootstrapping-like way, where a simple memory editor gradually gains more vocabulary.

See also:

- Where did that prebuilt binary come from?

13. Permaculture

Permaculture is an approach to land management and settlement design that adopts arrangements observed in flourishing natural ecosystems. First formulated in 1978, it particularly stands in opposition against industrial agriculture. Permacomputing is based on the idea of applying permacultural ideas to computing (and "high" technology in general).

In particular, permaculture inspires permacomputing to:

- Recognizing the effects of computing to the biosphere, and trying to find ways to make these effects positive and regenerative.
- Turning waste into resources and constraints into possibilities.
- Explorative, imaginative and positive attitudes towards sustainable design, as opposed to "returning to the past" or "having to tolerate lesser resources".
- Opposition to the mainstream technological industry while offering a tangible alternative.

Permacomputing isn't the first attempt to bridge permaculture and computing. Earlier examples include:

- The Permaculture entry on WikiWikiWeb connects it with software design patterns but does not connect to the ecological reality.
- Kent Beck's talk "Programming as a garden: Permaprogramming" similarly drew inspiration from the philosophy to software design without the ecological aspect.
- Amanda Starling Gould's 2017 doctoral dissertation "Digital Environmental Metabolisms: An Ecocritical Project of the Digital Environmental Humanities" centers around the ecological aspect but concentrates on end-user activities.

14. Jevons paradox

Jevons paradox refers to the phenomenon where the increase of efficiency in the use of a resource leads to more use of the resource. Jevons originally noticed in 1865 that the development of more fuel-efficient steam engines resulted in an increased total use of coal: the falling cost of coal increased its demand and negated the gains.

See also:

- Wirth's law - a computing-specific variant of Jevons paradox

15. Salvage computing

Salvage Computing is the art of utilizing only already available computational resources, to be limited by that which is already produced. It's about figuring out how to make the best possible use out of the millions of devices which already exist.

Scavenge-friendly electronics are parts that are no longer manufactured, but that are available by the billions in landfills. Those who can manage to create new designs from scavenged parts with low-tech tools will be able to preserve electronics.

16. Planned longevity

Planned longevity is the opposite of planned obsolescence: the way of designing systems, especially hardware, so that it supports lifespan maximization.

Planned longevity is something that should ideally take place in the industry that produces the hardware. Sometimes, the shortcomings of the industry can be compensated by changing the firmware of the system or switching to a third-party software platform.

Chips should be designed open and flexible, so that they can be reappropriated even for purposes they were never intended for. Complex chips should have enough redundancy and bypass mechanisms to keep them working even after some of their internals wear out. (In a multicore CPU, for instance, many partially functioning cores could combine into one fully functioning one.)

Concepts that support planned longevity:

- Design for disassembly
- Open hardware
- Morseware

17. Design for disassembly

Design for Disassembly ensures that all elements of a product can be disassembled for repair and for end of life. This allows for and encourages repairs, with the result that a product's life cycle is prolonged; and it allows for a product to be taken apart at the end of its life so that each component can be reclaimed. Among other shifts in thinking and making, this means minimizing materials, using simple mechanical fasteners instead of adhesives, clearly labeling components with their material type, and ensuring components can be disassembled with everyday tools.

Unlike the nebulous goal of designing a sustainable product, designing a product for disassembly is a more concrete, quantifiable approach to ecologically sound making and to consumption.

18. Dependency

A **dependency** refers to another piece of technology (software or hardware) a technology depends on for using. In software, "dependencies" may refer to the entire network of dependencies or just the software part of it.

The dependencies of a normal computer application include the physical computer, the energy source (including the grid and all the economic dependencies needed to maintain it), the operating system, and a set of libraries (some of which may be bundled with the OS). An increasing number of computer programs also depend on an Internet connection and an arbitrary server often maintained by the corporate owner of the program.

Hard and soft dependencies

Nature is full of dependencies but most of them are **soft dependencies**: animals can usually find nutrition from a wide variety of other organisms instead of strictly depending on a specific species. In high technology, however, we are mostly talking about **hard dependencies**: computers and cars need to be built from very specific components produced in very specific factories. Programs often depend on specific versions of specific libraries, and changes to those libraries may software rot. Hard dependencies come with a low tolerance for faults and changes.

Documentation can be thought of as a soft dependency. Some kind of documentation is usually needed in order to fully utilize a program or a device, but this documentation can come from several alternative sources. It can also be omitted if the user has already learned the necessary information.

If only one compiler can compile the source code of a program, its compilation dependency is hard. If there's a large variety of different compilers that all succesfully

produce a working executable, the dependency is much softer. The same applies to websites and how compatible they are with different browsers. If only a few high-end browsers are supported, the dependency is quite hard.

Optional dependencies are soft, especially if they don't change the features or behavior of the program. If it is possible to run an application either natively or in a web browser, the monstrous dependencies of the web browser don't need to be taken in account when assessing the application.

Smallness softens: if a program is so short that it can be easily rewritten in another language by using the source code as a guide, its dependency on the original programming language softens.

Dependencies & permacomputing

Permacomputing calls for dependency-awareness, both regarding the material dependencies (the grid, the manufacturing infrastructure, etc.) and the digital ones. In both cases, the networks of hard dependencies are often several orders of magnitude larger than they should be, so permacomputing is concerned about shrinking the networks as well as moving the emphasis towards soft dependencies.

Software dependencies may sometimes include non-open-source pieces of software. These can usually be tolerated as long as there's a way to run them in an open-sourced emulator. Depending on an Internet connection in programs that are not essentially networking-related, however, should not be tolerated, especially if this means depending on an arbitrary server.

19. Pseudosimplicity

In mainstream computing, ease of use is usually implemented as superficial simplicity or **pseudosimplicity**, as an additional layer of complexity that hides the underlying layers. Meanwhile, systems that are actually very simple and elegant are often presented in ways that make them look complex to laypeople.

Feminist (cyberfeminist) critique of simple interfaces and invisibility of underlying infrastructure says this simplicity is just predisposition for mainstream or stereotype use.

See also:

Feminist server

20. Scalability

Scalability usually means upscalability: the property of a system to handle a growing amount of work by adding resources to the system.

In permacomputing, downscalability may be more important than upscalability. Systems in general are supposed to remain small, so they don't need to be designed with upscalability in mind. However, there may often be a need to implement an idea or an algorithm in a very restricted environments, which creates preference for downscalable ideas and algorithms.

21. Unix

Unix is a multi-user operating system whose development was started in 1969 by Ken Thompson and Dennis Ritchie, as well as an entire family of operating systems derived from the original Unix. Since Unix is also a trademark that only applies to specific products, terms like "*nix" are often used to refer to the entire family of Unix-compatible systems (including GNU/Linux). In this article, we don't bother to respect the trademark (Linux is a Unix for us).

Unix was originally a mainframe-like time-sharing operating system scalability to much smaller computers with much more limited processing power and storage space. In order to keep the system small, elegant and flexible, it was decided to have a set of "small and sharp" tools that can interoperate with each other via input/output piping. At later times, Unix gathered bloat, and from the 1980s microcomputer point of view it was already seen as a huge and complex OS for big computers.

The possibility to reimplement the system gradually, one tool at a time, was a major reason why Unix was chosen as the basis of the GNU project, even though Richard Stallman didn't particularly like it.

Historically, it may be interesting to compare Unix with Forth that was born at the same time for a somewhat similar purpose (bringing a "mainframe-grade" environment to a small computer), although Forth is a memory-oriented single-user system whereas Unix is a disk-oriented multi-user system. A major difference is that while Unix adopted a lot of ideas and principles from the mainframe world, Forth actively questioned them in order to get as small as possible. Also, Forth is a programming language to the core, while Unix consists of many separate tools that sometimes have a programming language built in.

Advantages of Unix from the permacomputing perspective

- The basic idea of having small, flexible and interoperable tools is close to permacomputing ideals.

- The use of a high-level language (C) has made it independent from specific processor and computer architectures. Programs tend to be source-code-compatible across Unix systems and often even with non-Unix systems.
- There are several independent but largely compatible implementations (classical Unix, GNU/Linux, Minix …), many of which are FLOSS.
- Unix-like systems may have relatively low hardware requirements, especially when talking about “barebones” systems mainly used with character terminals.
- Long history of use in a vast variety of different types of devices (embedded, workstation, server, supercomputer, etc.).

Disadvantages and problems

- Modern, “real-world” Unix systems, like most general-purpose operating systems, suffer from a lot of bloat and unnecessary complexity.
- A lot of this complexity is somehow related to legacy compatibility as are many weird quirks one can find in Unix. One might say that this has resulted from an excessive prioritization of accumulated tradition over system-level refactoring.
- Unix has reached such a dominant position in many areas of computing that it represents monoculture that narrows down technological diversity.
- Despite having a long legacy, Unix is far from a bedrock platform. Software often needs to be constantly maintained in order to keep it compatible with various libraries and other changing pieces of the environment.
- Binary compatibility between different versions of the same OS may be surprisingly bad; even C libraries (such as GNU) may change their ABIs in ways that cause incompatibilities and force recompilation.
- Inefficiencies and limitations that can be traced back to the pre-Unix mainframe ideals:
    - Preference for sequences of plain-text lines in input and output (as in 80-column IBM punched cards). Translating between plain-text formats and various internal representations causes overhead. Large plain-text files are often cumbersome to operate with.
    - There are good tools for defining a task (by writing a command line) but the chances to affect the running of the task are much more limited (as in the old batch-job culture). Possibilities of building interoperability between running programs are much weaker than between not-yet-running programs.
    - “Waterfall model” in software compilation, producing static monoliths that are very difficult to arbitrarily change especially when they are running. This has resulted in a plethora of scripting/configuration languages that compensate for the inflexibility.
- The original Unix shell was designed to be quite minimal, and programmability was added to it in various ad-hoc ways by later developers. Various scripting languages have come into use as a response to the messiness of shell scripting.

- Unix can be considered far too large and complex to many tasks it is currently used for (embedded systems, single-user mobile computers, etc.)

Unix-like operating systems and kernels

- BSD
- Minix
- GNU
- Linux
- Darwin
- Plan 9

22. Balance of opposites

Balance of opposites is important in many permacomputing contexts.

Many people have a tendency to form dichotomies where one side is somehow “the good one” whereas the other is the “bad” or even “evil” one. Sometimes, the good side is considered so good that it becomes a silver bullet, something that is supposed to be universally good in all cases.

Balance of opposites can be used to eliminating this kind of black-and-white oversimplification. There are very few things or ideas that are either “good” or “evil” in all possible contexts. Instead of bluntly stating that an idea or a piece of technology is “the best” or “just evil”, one should try to delineate the contexts where it works and where it does not.

Yin and yang

Yin and Yang (陰陽) are concepts from Chinese philosophy. Yang is active, controlling and expanding, while Yin is passive, yielding and contracting. They are not “good and evil” but complementary opposites that should have a balance, often via cyclic changes.

In permacomputing contexts, the Yin-Yang dichotomy is sometimes used to contrast different computing cultures. Modern technological civilization is disproportionally yang, and this yangness extends to the cultures of computer hacking: total control over systems (natural or technological) is praised, which easily leads to impoverished monocultures where a lot of energy is wasted on forcing things into narrow standards.

Too much yin, on the other hand, may lead to an excessive acceptance of the way how things are and “have always been”. It likewise easily leads to narrow norms, via traditionalism. The norms may be hostile to innovation, experimentation and

reappropriation. It may also lead to intellectual laziness, where rational analysis is not even attempted.

Yin and yang hacking

These concepts were introduced in the Permacomputing 2020 text.

In **Yang hacking**, a total understanding and control of the target system is valued. Changing a system's behavior is often an end in itself. There are predefined goals the system is pushed towards. Optimization tends to focus on a single measurable parameter. Finding a system's absolute limits is more important than finding its individual strengths or essence.

In contrast, **Yin hacking** accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation. The "personality" that stems from system-specific peculiarities gets more attention than the measurable specs. It is also increasingly important to understand when to hack and when just to observe without hacking.

Yang hacking is quite essential to computing. After all, computers are based on comprehensible and deterministic models that tiny pieces of nature are "forced" to follow. However, there are many kinds of systems where the yin way makes more sense (e.g. the behavior of neural networks is often very difficult to analyze rationally).

Transgression and immersion

Transgression and **immersion** are two oppositional ways to creatively relate to constraints, especially in the kind of digital art forms that appreciate constraints (chip music, demoscene, pixel art).

Transgression is yang: it attempts to "break" or "push" the boundaries; to get a system to do something it is not supposed to be able to do; to find new things by exploring the unexplored possibilities of a given platform. The characteristic sounds and looks of a system (such as the 1:1 square wave in chip music, or clearly visible pixel boundaries) are often considered unrefined and unwanted.

Immersion is yin: instead of breaking away from the typical and unrefined, it takes it as the basis to build on. The 1:1 square wave is now very much wanted. The individual characteristics of a system are appreciated and explored ever deeper.

23. FLOSS

Free Libre and Open Source Software (FLOSS) is an umbrella acronym used to refer to software development practices in which the circulation of software source code is enabled by licensing strategies that promote and simplify the re-use of the code that would otherwise be limited by the author driven doctrine of copyright laws.

The two major definitions of FLOSS are free software and open source software. They almost entirely overlap and follow the same principle of providing a definition and a set of approved licenses that match this definition. These licenses can be either compatible or incompatible with each others, making composite projects either very simple, or very complicated. They can however be split into two large families:

- (strong, weak) copyleft licenses. Such licenses impose the person making modification to copyleft material to share their modification under the same condition/license than the code they modified. The idea is to promote circulation and virality;
- permissive or copyfree/copycentre licenses. Such licenses have much more simple conditions for reuse, if any, making possible to use such source for closed source software and proprietary systems.

While a popular method for software production and distribution, FLOSS has been increasingly scrutinised for its underlying liberal, possibly ultra-liberal ideology that has been more useful to the for-profit software industry, than it has been useful to foster the much anticipated digital commons of public interest, as envisioned in the late 90s and 00s. This is because both free software and open source software proponents support the idea of permitting the (re)use of FLOSS source code for *any* purpose. As a result a growing number of post-free culture licenses have started to emerge in the late 10s and early 20s to address issues of ethics and exploitation found in the *for any purpose* take of FLOSS.

24. Decentralization

Decentralization refers to distributing activity away from a central, authoritative location.

In computing, a prominent example of offline first or peer-to-peer protocols.

Decentralization supports diversity (including technological diversity) and empowers users to own the technology and services they use.

Decentralization may also go horribly wrong in ways that work against permacomputing goals with radically increased energy use, etc. See the cryptocurrency/blockchain world for examples.

A useful reflection on the negative aspects of dencentralization is Problems of Decentralism from The Meaning of Confederalism by Murray Bookchin.

See also:

- file collection
- offline first
- information battery

25. Software rot

Software rot is generally thought of as degradation of obsolescence unless it is constantly maintained.

A better approach might be to talk about the reliability of the environment the software depends on. Would you build a house on a bog?

It is often necessary to build on “bogs” (i.e. “actively developed” platforms), but it might be a good idea to also be compatible with a bedrock platform whose specifications are static and solid.

Software rot is a big issue for cultures that constantly produce new programs (such as demoscene) that are not supposed to be constantly maintained after release. Programs written for classical platforms (such as DOS or NES) usually need no post-release maintentance at all, while those written for e.g. Linux will likely cease working in a decade or two. Sometimes, serious media archeology work (such as finding specific versions of old libraries) is needed to get a program to run again.

26. Big Tech

Big Tech is a name given to the largest technology companies in the United States, which dominate consumer computing technology around the world. The individual companies change, but the interests and practices of these companies largely remain the same, and are at odds with many of the principles of permacomputing.

- Big Tech software serves the needs and interests of the company, and their need for profit, first.
- As a result, hardware produced by these companies often follows planned obsolescence, in order to create more commodities to sell for profit
- Software created by big tech tends towards complexity. These are large, bureaucratic institutions with tens of thousands of employees, and knowledge about software produced by these companies is centralized within them, both legally and by virtue of its design

- Big Tech companies may destroy or abandon software that does not align with their current business goals
- Most Big Tech companies follow the model of software as a service.

27. Terms of service / Code of conduct / Privacy

This document is an adaptation of LURK's TOS.txt.

General agreements

The permacomputing wiki, IRC, and mailing list, as well as the LowTech/Permacomputing XMPP chatroom are used by people coming from a variety of cultural, ethnic and professional backgrounds. We strive for the permacomputing community to be welcoming to people of these various backgrounds and provide a non-toxic and harassment-free environment. This document is our iterative 3-in-1 Code of Conduct, Terms of Services and Privacy Statement.

By interactive with the permacomputing community you agree to the following:

- Be respectful towards others. This means that we will not tolerate homophobic, transphobic, racist, ableist and sexist slurs and content, even if intended as a joke, or as an ironic remark. That also includes demeaning, belittling or otherwise verbally intimidating communication. In short, any attitudes that may promote the oppression and/or exclusion of historically marginalized groups will not be accepted. Users who violate this rule may get warned once, or banned right away from using our services if their action were clearly ill-intended.

- We talk to and with people rather than about people and/or groups. That means no insinuating, unwelcome, or otherwise toxic comments regarding a person's lifestyle choices and practices, antagonizing and incendiary generalizations, flaming or edgy remarks at the expense of someone or another community. We understand and appreciate that for some, the internet has became a place to vent and dump their frustrations. While we welcome sarcasm, critique and the need to seek support and alliances while expressing defeating feelings, our services, as well as its local and remote users, are not meant to be your personal punching ball. Users who violate this rule will get warned once, and then banned from using our services.

- Absolutely no harassment, stalking or disclosure of others' personal details (doxing). Users which violate this rule will be unconditionally banned.

- Similarly, refusing to disengage during an escalating argument, or spamming users with private messages is a form of harassment. Users who violate this rule will get warned once, and then banned from using our services.

- Hate speech, such as, but not limited to: white supremacy, ethnostate advocacy, discussion of national socialism / nazism will not be tolerated. Users who violate this rule will be unconditionally banned.

- Mass-advertising content is prohibited. However we encourage you to share calls for projects/papers, new project announcements of yours, upcoming events and tasteful reminders to your followers of things like Patreon or websites where they can purchase/support your work.

- Media containing sexualized depictions of children (including lolicon) are not allowed. Users who violate this rule will be unconditionally banned.

Moderation

If you have trouble with someone violating these rules, contact us. Do not hesitate to reach out, and do not feel feel like you're being a nuisance when you do, on the contrary!

Privacy

By using our services you agree that a minimal amount of information about your connection (IP address) will be kept for debugging and maintenance purposes. You are welcome to use a VPN or Tor to connect to our services though. We do not actively store or archive this information beyond the default rotation time of the software we use. We welcome suggestions and practical information to limit data retention to a minimum! Contact us if you have good experience with that. Some of our services use cookies for remembering settings and preferences. We don't track our users, nor do we use analytical software (third-party or self-hosted).

USUAL CAPSLOCK BOILER PLATE

THE SERVICES USED IN THE PERMACOMPUTING COMMUNITY ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE ADMINS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SERVICES OR THE USE OR OTHER DEALINGS IN THE SERVICES.

28. Contribute

If you feel like you belong here, you are very welcome to contribute to this wiki!

You can about us by email for an account. Please:

- Present yourself briefly;
- Include URLs to work/project/socials/etc;
- Explain your interest in permacomputing and tell us what/why/how you would like to contribute to the wiki;
- Confirm that you have read the editing and agree to our terms.

Please cover these four points, we won't reply to vague or incomplete requests. Really, we won't. It may take a few days for us to get back to you.

29. sister0

ଘୀଓ ⏚ 🜃 🜄 🜁 🜂

salutations

Permacomputing Principles work in progress

Holistic Computing Arts, Citizen Science Fiction, and Somatic Exploration in the womb of the world

autoluminescence

position statement

collapsible.systems

🜃 🜄 🜁 🜂

30. DEC

Damaged Earth Catalog

About

The Damaged Earth Catalog is a growing online catalog of the different terms in circulation, used by communities of practice, in relation to computing and network infrastructure informed by ecological ethics, degrowth, resilience, repair, and minimalism.

It is currently developed by l03s as part of her PhD research. Entries on this page should point to the Damaged Earth Catalog wiki, check with l03s for suggesting new additions. Don't duplicate entries/work.

https://damaged.bleu255.com

Link to permacomputing

At the moment permacomputing is moving from a set of interrelated ideas and practices to a more coherent system of thought, as best exemplified with its principles. It's a work in progress and different people have different views to it, and how to best approach its openness/closeness.

To be sure, many people and collectives have independently come up with similar ideas, so there's a lot of overlap between permacomputing and other concepts such as:

- Collapse Informatics
- Computing within Limits
- Degrowth
- [Frugal computing]
- Salvage computing
- etc.

And looking further there are also ideas and movements that have some similarities but maybe are not as closely related:

- Appropriate technology
- Begnin Computing
- Convivial computing
- Feminist Technology
- [Green software engineering]
- [Heirloom computing]
- Liberatory Technology
- Low-Tech
- [Permaprogramming]
- [Permatech]
- [Rustic computing]
- Small Technology
- [Sustainable ICT]
- etc.

Is it more fruitful to think about all these ideas as different viewpoints to the same thing rather than independent "movements" whose borders need to be

unambiguously defined? or on the contrary do they carry irreconcilable deviations and incompatible ideologies or world views?

This is the research and contextualisation that takes place at the Damaged Earth Catalog and help informs and bring perspectives on the constant becoming of permacomputing.

31. Collapse OS

Collapse OS is a Collapse computing. It is designed to:

- Run on minimal and improvised machines.
- Interface through improvised means (serial, keyboard, display).
- Edit text and binary contents.
- Compile assembler source for a wide range of MCUs and CPUs.
- Read and write from a wide range of storage devices.
- Assemble itself and deploy to another machine.

Collapse OS was also ported to the UXN virtual machine.

Dusk OS will be a similar OS geared at more powerful devices and aimed at general-purpose post-collapse uses.

See also:

- The official website of Collapse OS
- Collapse computing
- Civboot
- Dusk OS

32. Gemini

Gemini is an application-layer WWW protocol created by Solderpunk in 2019. Gemini and its document format (Gemtext) are somewhat similar to Gopher but add some features from modern WWW, such as mandatory encryption, hyperlinks and URLs. Due to the simplicity of Gemini, there is already a large amount of fully compliant client software.

It is of course possible to use HTTP(S)/HTML in a simple and restricted way along with limited web browsers to get similar technical characteristics. However, a separate protocol makes it possible to have a separate Smallnet space, **Geminispace**, where users are guaranteed to only encounter things that are compatible with their simple Gemini browsers.

Relationship to permacomputing:

- Low bandwidth, low complexity, low system requirements. These are partially dictated by the design of the format (no support for inline media, etc.)
- Coexistence with Gopher and HTTP/HTML with no intentions to replace either.
- The lack of inline media makes it easier to have file collection of Gemtext documents.
- Solderpunk has written about permacomputing, and at least one of the Gemini clients (Ariane) explicitly refers to permacomputing.
- However, Gemini does not address what can be thought as one of the basic problems of WWW. Documents are primarily addressed by referring to their servers, so impermanence and broken links can be expected especially as the typical Gemini server is small and private.

See also:

- Project Gemini FAQ (HTTPS/HTML)

33. Chifir

The **Chifir** computer is the definition of a universal virtual machine designed to host and archive projects. The example project to be hosted on the vm was Smalltalk-72, but its design did not reflect how this was to be done, and the lack of I/O made it unclear that it would even be a viable target for the system.

Relevance to permacomputing

The paper from which this specification originates considers issues of obsolescence.

- The Cuneiform Tablets of 2015
- Implementation, written in C

34. Uxn

Uxn is a simple virtual machine geared towards graphical applications, with features reminiscent of classic home computers.

Unlike most “fantasy platforms”, Uxn was designed with an implementation-first mindset with a focus on creating portable tools and games for salvage computing.

This stack-machine has 32 opcodes, and no registers. Given the stack a b c, the c item being the last to be added, and the first to be removed, a Program Counter(PC), a Memory(M), Devices(D) and a Return Stack(rs):

```
80 a b c M[PC+1] 08 a b?c             10 a b M[c8]      18 a b+c
01 a b c+1       09 a b!c             11 a {M[c8]=b}    19 a b-c
02 a b           0a a b>c             12 a b M[PC+c8]   1a a b*c
03 a c           0b a b<c             13 a {M[PC+c8]=b} 1b a b/c
04 a c b         0c a b {PC+=c}       14 a b M[c16]     1c a b&c
05 b c a         0d a {(b8)PC+=c}     15 a {M[c16]=b}   1d a b|c
06 a b c c       0e a b {rs.PC PC+=c} 16 a b D[c8]      1e a b^c
07 a b c b       0f a b {rs.c}        17 a {D[c8]=b}    1f a
b>>c8l<<c8h
2x a16 b16+c16   4x a b c {rs.b+rs.c} 8x a b c b+c
```

The implementation of the virtual machine is about https://git.sr.ht/~rabbits/uxn11/tree/main/item/src/uxn.c. A self-hosted assembler for the Uxntal assembly language is about https://git.sr.ht/~rabbits/drifblim/tree/main/item/src/drifblim.tal.

Assembled Uxntal applications such as text editors, drawing programs and livecoding environments are typically about 10-15kb in size.

Relevance to permacomputing

- Simplicity of implementation may make Uxn usable as a dependency other than the VM itself.
- The design characteristics call for small applications that use little computing power.

The general issues with virtual machines apply: running a virtualized program takes much more processing power than an equivalent native program would. It is therefore advisable to use VMs only for computationally simple applications.

Parts of the project are geared toward the specification of a Universal Virtual Computer, such as the Uxn sign language and writing system. It is capable of hosting Collapse OS.

See also:

- Uxn at 100r.co
- Uxn at xxiivv.com

35. Civboot

Civboot: a civilizational bootstrapper is a a project which aims to reduce the tools and knowledge necessary to bootstrap modern civilization as much as possible, with a

focus on computers. It was originally inspired by Collapse OS but is also an educational tool. As of 2022, it is still at a very early stage.

- Civboot's Git repository

36. Mu

Mu is a minimalist stack of languages built up from machine code. It requires a processor from the x86 family, and builds up to a memory-safe language without any additional dependencies (though it can also be bootstrapped from a C implementation).

Like Forth, Mu was intended to enable people to build interesting programs while being able to hold the entire system in their head. Unlike Forth, Mu does so while being well-typed and providing good guardrails that make programs easier to debug. 2/3rds of the LoC are devoted to automated tests.

Performance is not a priority; Mu achieves its goals by focusing on very basic hardware support. It only supports one screen resolution (1024x768 with 256 colors) and has only one font (albeit with extensive Unicode support). There's just one device driver for hard disk storage: ATA disks in the inefficient PIO mode.

Mu is different from more retro computers like Uxn in assuming lots of RAM and disk. It uses a 32-bit address space and avoids space-saving tricks in favor of straightforward code.

Mu is currently dormant. You can create graphical programs that run without an external OS (albeit only emulated on Qemu so far). You can also create text-mode programs for Unix-descended systems. It's currently stalled in need of expertise debugging real hardware and implementing networking. The mouse driver also needs work.

Since Mu depends on no features of x86 newer than SSE, it runs on any x86 processor built in the 21st century. Programs built on Mu are also expected to be resistant to bitrot since they require fairly basic emulation capabilities. It currently serves as a time capsule to test these hypotheses.

See also:

- An academic paper about Mu
- A summary of the Mu compiler on a single page. It's built in machine code so needs to be really simple.

37. Teliva

Teliva is a text-mode platform for disseminating sandboxing applications. In this respect it resembles a web browser (without markup or a DOM for documents). The sandboxing model is more flexible than web browsers.

Relevance to permacomputing

It tries to nudge computer owners to think about what permissions they should grant untrusted applications, and to learn a little bit of programming to do so. It also encourages owners to look inside application code and provides a trivial edit-run debug cycle to help them make changes to untrusted applications.

See also:

- A talk on Teliva

38. DawnOS

DawnOS is an entire operating system hosted on a One Instruction-Set Computer(SUBLEQ). The binaries for such a system break down to a much longer list of machine operations, but they take fewer transistors to run, and can be pipelined due to their uniform size.

dawnos.txt

DawnOS came with a text file, written by the author, lamenting the current state of computing:

> Imagine that software development becomes so complex and expensive that no software is being written anymore, only apps designed in devtools.
>
> Imagine a computer, which requires 1 billion transistors to flicker the cursor on the screen. Imagine a world, where computers are driven by software written from 400 million lines of source code.
>
> Imagine a world, where the biggest 20 technology corporation totaling 2 million employees and 100 billion USD revenue groups up to introduce a new standard. And they are unable to write even a compiler within 15 years.
>
> “This is our current world.”

See also:

- A Programming Language With Only One Command and the Anti-Imperialist Operating System Built on it

39. Solar Protocol

Solar Protocol is a project that connects solar-powered WWW servers from around the globe, redirecting traffic to a server that has solar power available (i.e. the part of the globe where the sun is shining).

A problem with Solar Protocol is that it either neglects the energy requirements of the Internet infrastructure that lies between the solar-powered servers and the users, or takes them as a constant that only depends on the amount of transferred data. However, we can be fairly sure that routing a packet across the world takes much more energy than routing it across a country.

If the power consumption of a server is small to begin with, its “greenness” may very well get negated by a bad routing decision. It may very well be “greener” to just route a user to a nearby fossil-powered data center than to a solar-powered server on the other side of the world.

Taking the network into the equation is difficult because even academic estimations of the power consumption of Internet routing have varied by several orders of magnitude. Still, there seem to be no mentions of this issue on the Solar Protocol website, even though it discusses the energy consumption in the server side and the browser side. Given the prominence of SP in the media, it is highly unlikely that the people involved have not heard about this kind of critique.

In 2015, it was estimated in a meta-analysis by Aslan&al. that moving a gigabyte across a national cable network took 0.06 kWh. This is the same figure as for running a 10-watt server for 6 hours. Even if the current figures are much lower, they cannot be ignored as irrelevant if the server network actually serves data instead of mostly remaining idle.

From the permacomputing point of view, Solar Protocol has a lot of the right spirit and may work very well as an educational or artistic project. Unfortunately, the project will remain somewhat problematic until it considers the effect of the Internet infrastructure.

See also:

- Solar Protocol website
- Why do estimates for internet energy consumption vary so drastically?

40. PADI

The National Library of Australia's Preserving Access to Digital Information (PADI) initiative aims to provide mechanisms that will help to ensure that information in digital form is managed with appropriate consideration for preservation and future access.

Its objectives are:

- to facilitate the development of strategies and guidelines for the preservation of access to digital information;
- to develop and maintain a web site for information and promotion purposes;
- to actively identify and promote relevant activities; and
- to provide a forum for cross-sectoral cooperation on activities promoting the preservation of access to digital information.

See also:

- website, on Wayback Machine
- PADI's Notes on emulation, on Wayback Machine

41. BBC Domesday Project

In 1986, the BBC launched an ambitious project to record a snapshot of life across the UK for future generations, but 16 years after it was created, the £2.5 million BBC Domesday Project was unreadable. The special computers developed to play the 12" video discs of text, photographs, maps and archive footage of British life had become obsolete.

By contrast, the original Domesday Book, an inventory of eleventh-century England compiled in 1086 by Norman monks, is in fine condition in the Public Record Office and can be accessed by anyone who can read and has the right credentials.

It has been cited as an example of digital obsolescence on account of the physical medium used for data storage.

42. CARDIAC

CARDIAC (CARDboard Illustrative Aid to Computation) is a learning aid developed by David Hagelbarger and Saul Fingerman for Bell Telephone Laboratories in 1968 to teach high school students how computers work.

The kit consists of an instruction manual and a paper computer.

The computer operates in base 10 and has 100 memory cells which can hold signed numbers from 0 to ±999. It has an instruction set of 10 instructions which allows CARDIAC to add, subtract, test, shift, input, output and jump.

```
INP(Input): take a number from the input card and put it in a
memory cell.
CLA(Clear&Add): clear the accumulator and add the contents of a
memory cell to the accumulator.
ADD(Add): add the contents of a memory cell to the accumulator.
TAC(Test accumulator): performs a sign test on the contents of
the accumulator; if minus, jump to a specified memory cell.
SFT(Shift): shifts the accumulator x places left, then y places
right, where x is the upper address digit and y is the lower.
OUT(Output): take a number from the specified memory cell and
write it on the output card.
STO(Store): copy the contents of the accumulator into a specified
memory cell.
SUB(Subtract): subtract the contents of a specified memory cell
from the accumulator.
JMP(Jump): jump to a specified memory cell.
HRS(Halt&reset): move bug to the specified cell, then stop
program execution.
```

Relevance to permacomputing

This project, and others like it, were offering a conceptual computer that could be understood in its entirety by a single person.

See also:

- On Wikipedia

43. Hardware

Hardware refers to the material parts of computing equipment, in contrast to software.

When assessing hardware, we should pay attention to possible sustainability problems preventing repair, reuse and reprogramming:

- Insufficient documentation
- Unavailability of obsolescence
- DRM locks preventing the running of homebrew software
- Intellectual property and closed-source firmware

- Not design for disassembly

The world is full of abandoned computer hardware, therefore we shouldn't be too picky, and find creative ways to work with even the lousiest pieces of hardware. We shall approach their problems as something to be fixed with hacking, reverse-engineering and activism. Reuse of already existing or old hardware can ease the stress on energetic and mineral-mining impacts of new production (with some exceptions for power-demanding or toxic devices).

Efforts to create new hardware components in biosphere-compatible and/or local ways are worth supporting (although that goal is still far away for microchips). We are particularly interested in what it requires to build specific types of component with minimal industrial dependencies without destroying the biosphere. Links to succesful DIY projects are welcome.

Types of hardware components:

- ICs
    - processors (including SoCs)
    - memory
    - FPGA
- Single-board computers
- displays
- batteries (including supercapacitors)
- human input devices
- storage devices
- radio devices
- sensors and active peripherals
- energy sources

How is permacomputing hardware different from IoT?

IoT devices are not specifically produced with sustainability in mind. Often IoT is a by-product of corporate business models and extractivist attitude and used in the same way. In permacomputing, hardware, peripheries and energy sources are balanced together to create supporting networks for ecosystems, designed and grown with critical care to every part, human-nature surroundings and commons.

-~

See also:

- bedrock platform (related to software portability and preservation)

44. Peripherals

A peripheral is an auxiliary device to a computer, it could be used for storage, printing, etc.

- Printers
- Connectors
- Power Plugs
- Batteries

neau: This entire domain seems to be full of products with obsolescence. It might be worth listing things by ease of repairability.

45. Software

When assessing software, we should pay attention to:

- Is it free from blackbox dependencies such as arbitrary external servers when running, compiling or installing it? Does it tolerate a lack of network connectivity?
- Is it legally possible to copy, modify and fork the software? (i.e. is it FLOSS?)
- What kind of libraries, programming languages and other software components does it depend on? How mature are these components (i.e. how much software rot can be expected due to changing interfaces etc.)? How large is the dependency network?
- How much resources does it require to 1) run the software, 2) modify the software (including recompilation from scratch) and 3) bootstrapping the smallest possible environment (including the operating system) that can be used to run and develop the software?
- Are there other software that do the same job? How easy would it be to transition to one of them?
- How simple and clearly-defined is the core functionality of the software? How long would it take to write an equivalent software from scratch?

Software Freedom

Software freedom is the freedom to run the program as you wish, for any purpose, to study how the program works, and change it, to redistribute copies and your modified versions so you can help others.

If the new software no longer runs on old hardware, it is worse than the old software.

Types of software

- operating systems
- editors (for media formats)
- compilers/interpreters (for programming languages)
- networking clients/servers (for protocols)
- …

Twee Editors

A twee editor is a [text-editor|text editors] that is the minimum size for a functional editor, without compression. Twee editors are usually very compact, but at the cost of accessibility.

46. Programming languages

When assessing programming languages, we should pay attention to:

- How complex is the language? How long would it take to learn all the syntactical details? How long would it take to implement a compiler/interpreter from scratch?
- How mature is the language? Do changes to the specification often break backwards compatibility? How much hacking does it require to compile and run a decades-old program in current implementations of the language?
- Are there several alternative implementations of the language? (It is generally a good sign if there are)
- What are the bootstrapping them?
- What kind of platforms do these implementations target? Is it possible to port a program to a very small and/or obscure device without switching to another language?
- How fast and compact is the generated code? What are the overheads and mandatory dependencies like? Does the hello world require bytes, kilobytes, megabytes or gigabytes of memory if all the dependencies are included?

Asking “what is the most suitable programming language for permacomputing?” is akin to asking what is “the most suitable plant for permaculture”. The entire question contradicts itself.

There is a high diversity of possible tasks and programs, and different programming languages suit them in different ways. Not all software needs to last for decades, run efficiently or be ultra-secure. However, it is still good if the language does not prevent this.

programming languages

- C
- Forth
- Lisp
- Lua
- Nim
- Hare
- Zig
- Smalltalk, see The Cuneiform Paper
- Go

See also:

- Drew DeVault's blog post about benchmarking compilers by Hello world size
- Blog post about a research in energy efficiency of programming languages

47. Operating systems

- Plan 9
- Oberon
- Collapse OS
- DOS

48. Protocols

When assessing networking protocols, we should pay attention to at least:

- Simplicity of implementation
- Resource use (e.g. how much bandwidth is used compared to the actual amount of transferred information)
- Offline tolerance (Does it depend on constant connectivity? How well does it work in a local network that has no world connectivity?)
- Decentralization (How well does it tolerate small servers/peers that are not always online? Does the design encourage large centralized servers?)

Some protocols:

- WWW
    - HTTP/HTML
    - Gopher
    - Gemini
- Offline-tolerant messaging protocols
    - Email

    - Usenet
    - Fidonet
- Offline-intolerant messaging protocols
    - IRC
    - XMPP
    - ActivityPub
- Peer-to-peer
    - BitTorrent
    - IPFS
- Encryption
    - SSL
- Remote access
    - ssh
    - mosh
    - VNC
    - X11
- Wired device-to-device communication
    - USB
    - RS232
    - Hayes command set
- Character terminal connections
    - ANSI X3.64
    - RTTY
- Packet networking
    - Ethernet
    - TCP/IP
- Wireless networking
    - Wi-Fi
    - GSM and its successors (xG)
    - AX.25

49. File formats

- iff

50. Concepts

Concepts and **ideas** that are needed to discuss permacomputing:

Practical concepts

- bootstrapping
- decentralization

- digital preservation
- documentation
- emulation
- minimization

Concepts related to design principles

- design for disassembly
- design for descent
- emotionally durable design
- planned longevity, or lifespan maximization
- personalities of the people interested in permacomputing
- self-obviating system

More theoretical or generic

- aesthetics
- artificial intelligence
- algorithm complexity
- automation
- awareness amplification
- balance of opposites
- communication complexity
- kolmogorov complexity
- dependency
- ethnomathematics
- focality
- games
- information and energy
- otherness
- permatechnology
- public domain
- regenerativity
- reuse
- scalability
- sustainability
- technological diversity
- unconventional computing

Phenomena of mainstream computing world

- attention economy
- Big Tech

- bloat
- calculation factory
- Californian ideology
- capitalism
- cryptocurrency
- cornucopianism
- greenwashing
- Jevons paradox
- maximalism
- monoculture
- Moore’s law
- neoliberalism
- obsolescence
- postdigital
- pseudosimplicity
- retro
- siliconization
- silver bullet
- always online
- software rot
- utilitarianism
- virtualism
- wishcycling

51. Collapse computing

Collapse computing or **Collapse informatics** is the study, design, and development of sociotechnical systems in the abundant present for use in a future of scarcity. Civilizational or technological collapse is an extreme example of such a future. The term “Collapse informatics” was coined by Bill Tomlinson in 2013.

A major project in Collapse computing is Collapse OS, an operating system and set of tools for restarting computer technology after collapse.

Many Computing within Limits papers from the early years are about Collapse informatics.

While many may not agree with the collapse scenario or even a future of scarcity, the practical results of collapse computing are in line with the goals of permacomputing. Studies of the longevity of hardware components are relevant to lifespan maximization even without a collapse. Collapse computing is also relevant to resilience, self-sufficiency and the simplification of technology.

See also:

- Tomlinson's 2013 article DOI:10.1145/2493431
- Unplanned Obsolescence: Hardware and Software After Collapse (PDF, LIMITS 2017) DOI:10.1145/1235
- Collapse computing at XXIIVV wiki

52. Smallnet

Smallnet or **SmolNet** (also known as Small/Smol Internet/Web, etc.) is a movement of small-scale Internet networking that emphasizes the smallness of the servers (including low system requirements) as well as that of the user communities. It can be regarded as a counter-movement to the centralization and bloatedness of WWW.

Examples of Smallnet: * Gemini * Public Unix servers (see also time-sharing) * BBS communities

53. Computing within Limits

Computing within Limits (or **LIMITS**) is an annual workshop concerning the role of computing in human societies affected by real-world limits. As the official website states, "As an interdisciplinary group of researchers, practitioners, and scholars, we seek to reshape the computing research agenda, grounded by an awareness that contemporary computing research is intertwined with ecological limits in general and climate- and climate justice-related limits in particular."

The workshop was started in 2015 as a response how Sustainable ICT circles reacted to ideas such as planetary limits or possible technological collapse. This is perhaps why many of the papers from the early years are connected with Collapse computing. In later years, the focus shifted towards "computing systems that support diverse human and non-human lifeforms within thriving biospheres", which is very much in line with what Permacomputing stands for.

Concepts originating from LIMITS include:

- Benign computing
- Regenerative computing

See also:

- The official website

54. Offline first

Offline first, like its name implies, is the design of design for descent.

According to Leslie Lamport (1987), “A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable”.

55. Right to repair

The **Right to Repair** movement seeks to ensure that OEMs provide tools, documentation to their consumers in order for them to perform their own repair.

56. Holistic Computing Arts

Holistic Computing Arts is an artist-centred focus on computing arts that acknowledges the problem of the mesmeric qualities of vast computations and their immeasurable possibilities alongside the troubling ethics of participating in the extractive violence of computing (where over 50 different minerals needed for computer chips are extracted from the earth, often from unceded lands).

Holistic Computing Arts calls for a reconsidered position on low-emission computational practices by artists working culturally, socially, and critically with increased skills in computational media empowered by this awareness of limitations, which, conversely, increases the potential for creativity.

Holistic Computing Arts workshops focus on how artists can contribute their expertise to the field rather than being co-opted by the dominant technological and economic discourses that generally inform and drive the conversation.

The discussion reflects on a computer as being a material in the manner of conventional art materials; able to influence, modulate, transform and to be our artistic productions.

Navigating personal computing infrastructure and the incumbent knowledge and conventions that lead to personal expectations, ranging across customs around sending and receiving emails, the provenance of hardware, file formats, directories/ folders, the minerals and vessels we keep data in, and the space and lands these may occupy.

Discussions examine the computer as material media (physical, substantial and instrumental) capable of permeating and influencing all segments and layers of contemporary life via our communications, polity, activism, economics, psychological, etc., and reflects on the necessity of understanding the distinction between experience with computing arts as a medium (rather than as an editor for text, audio or visual materials, or third-party social media tools), where the computer performs as a cultural apparatus, where the partial content and action of the artwork are generated.

Depending on the experience of participants, for the uninitiated, typically included in the commentary is:

- intimate experiences with computing-inscribed social experiences (family photo archives)
- an exchange of the cultural/agential properties inherent in materials/materiality of the computing mediums through which we work and archive our ephemera;
- working with networks and systems from within carries awareness that we operate in an interconnected cultural and ecological sphere in which the computational plays a formative and extractive role; and
- appreciating computing as an art medium brings an awareness of ecological constructions and the social, cultural, and political issues that hover, enabling a tendency to analyse the systems and dynamics that belie people's artistic outputs.

See also:

- Holistic Computing Arts Conversation 2023, sister0, 2023.
- The Oceanic provenance of Permacomputing and Computational poetics, sister0, 2024.
- VvitchVVave post digital aesthetics symposium, 2019.
- Networked Art Forms: Tactical Magick Faerie Circuits, a series of events inspired by computer culture, artists, programmers and thinkers from the frontline of the maker aesthetic, devised by sister0, 2013.

57. Small File Media Festival

The **Small File Media Festival** is a film festival in Canada, focused on short films with file sizes less than five megabytes.

According to their website, SFMF works in defense of the tiny image. Size matters, and small is better, tiny is best, which is not merely to argue for a different aesthetics or narrative structures (that too) but also for an understanding that all media is media ecology – and as such, directly related to infrastructures with environmental costs.

- Official website, sadly, they have a poorly optimized website that fails almost entirely to display on slow-bandwidth and old browsers. The fact that the main page is more than half the size of a five-megabyte film somewhat ruins the point of their activism.

58. Algorave

Algorave is a livecoding event where people dance to music generated from algorithms.

Show us your screens

The tagline of the festival has relevance to permacomputing, as it promotes a culture of openness, where the computer musician may not be the main point of focus for the audience and instead attention may be centered on a screen that displays the process of writing source code, so the audience can not just dance or listen to the music generated by the source code but also observe the process of programming.

It also promotes the creation of file formats for preserving musical compositions, which may help against obsolescence.

See also:

- Demoscene

59. Demoscene

The **demoscene** is a computer art subculture that creates programmed audiovisual art in all kinds of programmable devices, often within very tight constraints. Aspects of the demoscene that may be relevant to permacomputing include:

- Media minimization.
- Small and optimized program code where bloat and superfluous dependency are shunned.
- The use of limitations and peculiar hardware characteristics as creative inspiration.
- The relationship to old technology. New and old platforms coexist, with old platforms having been supported as a continuous tradition rather than having been retro. New things are constantly discovered even on classic platforms.
- It is an example of a pre-siliconization subculture that still holds on some of the pre-siliconization values.

See also:

- Commodore 64
- Amiga
- ZX Spectrum
- pixel art

60. Solarpunk

Solarpunk is a movement centered on using and being affected by the use of renewable resources with a focus on decentralization, community activism, social justice and civic empowerment. A recognition that economic, social, and ecological injustices are all deeply inter-connected.

Embracing approachable, personal technology and envisioning a world in which the detritus of consumer culture is appropriated and repurposed toward the reconstruction of a devastated ecology.

Solarpunk imagery, with a lot of plantlife in a carless urban environment, is getting more and more recognizable. Since permacomputing is concerned about many difficult topics such as resource use minimization, scarcity and even collapse, it may be a good idea to compensate for this by allying with Solarpunk and its bright and hopeful outwards esthetics.

When embracing Solarpunk, however, there are some pitfalls. First, it is important to separate it from greenwashing technofuturism (which looks more sterile and corporate but is nevertheless often called Solarpunk). Second, one should perhaps not look too closely into the technological details of Solarpunk works (that may have standard sci-fi tropes such as screens projected in the thin air) but rather take it as a general mood and mindset.

See also

- A Solarpunk Manifesto

61. Contact

The Permacomputing wiki is facilitated by:

- ugrnm
- viznut

Its content is written by (in order of registration)

- neau
- thgie
- akkartik
- aw
- orx
- dusan
- katía

- luen
- suj
- cmos4040
- giz
- pixouls
- wakame
- decentral1se
- clwil

You can reach all of us at once by sending an email to **permacomputing / a t / bleu255 dot c o m**

Do you want to contribute something? Great, check editing!

62. neau

Half of Hundred Rabbits, a solar-powered studio operating from aboard a sailboat.

63. ugrnm

hello, I'm one of the admins.

more info + contact: https://bleu255.com/~aymeric

64. ola

dfgrdrgf

65. Bandwidth minimization

Bandwidth minimization is directly connected to the minimization of the use of artificial energy for networking, so it is of interest to permacomputing.

Ideas related to bandwidth minimization:

- Media minimization, including the use of more minimal styles
- Offline first design of applications
- Peer-to-peer rather than server-mediated communication
- Decentralization

66. Media minimization

Media minimization refers to the minimization of the kind of media that is usually high-bandwidth, such as images and videos.

Media optimization refers to the reduction of file size while keeping the appearance as close to the original as possible. This often involves twiddling with the optimization parameters of the compressor and a moderate use of filters to reduce unnecessary detail.

Another approach starts from stylistic and technical choices that aim at low or moderate file sizes. For instance, low-color styles with large solid-colored areas compress rather well (see posters or black-and-white woodcuts for inspiration). Repetitive pixel patterns may also work well in formats like PNG. High-color pixel art, however, does not compress that well.

If it is reasonable to run arbitrary code, procedural generation and algorithmic art provide a wide variety of stylistic choices.

In the midway between media optimization and style-first approaches are extreme media optimization techniques that lead to particular styles. One possible style is "ditherpunk" where a very small color palette is used in combination with automated dithering.

There is still room for a lot of research in automatic extreme media minimization. Ideally, the result of media minimization should resemble careful artisan work and look better than the original.

See also:

- Aesthetics
- Demoscene
- Small File Media Festival

67. Virtual machine

A **virtual machine** is an implementation of a computer on top of another. An **emulator** is specifically a virtual machine that simulates a different type of computer that also exists as real hardware. If the "emulated" computer is similar to the one that runs the emulator, the commonly used term is **virtualization**. Virtual machines that have no real-hardware counterpart (such as the Java Virtual Machine) are usually just called VMs, but their machine languages are usually referred to as **bytecode** rather than machine code.

Virtual machines can be used to ensure compatibility of a software – both across different types of hardware and in constantly changing software environments (i.e. to avoid software rot). Very simple virtual machines (ones that may be very slow but can be implemented in a very short time) are sometimes suggested as a means for very-long-term software preservation.

Virtual machines designed to run applications usually have the same kind of dependency and obsolescence problems as modern computer platforms in general. So, instead of ensuring compatibility, things like the Java Virtual Machine are more likely to add just another unreliable layer of dependencies. In order to avoid this, the design of the VM should be frozen and discourage software dependencies.

A general problem with virtual machines is the virtualization overhead, especially if the instruction set differs from that of the host computer. An emulated program may require several times the computing resources of an equivalent native program. JIT translation can be used to reduce this overhead. Static binary translation could be used to remove (nearly) all of the overhead, but is far more difficult to implement automatically.

Fantasy platforms are a class of virtual machines designed to be superficially similar to classic home computers, including immediate and worriless programmability with a simple programming language. A common design mistake in otherwise constrained fantasy platforms (such as TIC-80) is **uncapped speed** that allows the host computer to run the programs as fast as it can. This encourages programs that "push the limits" of the VM while actually taking advantage of the speed of the host computer and wasting a lot more energy for doing the same thing.

Virtual machines:

- Uxn
- p-code
- Chip8
- TIC-80
- WebAssembly

Languages that are usually compiled into virtual machine bytecode instead of native code:

- Forth
- Java
- Most scripting languages (Lua, etc.)

68. C

C is a general-purpose programming language created in the 1970s for the system programming needs of the Unix operating system.

The main benefit of C is that it is ubiquitous and quite mature. There are C compilers for nearly any imaginable processor architecture, and relatively old code often compiles quite well.

Compiled C code is generally quite resource-efficient. The speeds of compiled languages are often compared to C.

There are also languages whose compilers can produce C code to be compiled by a C compiler. These languages thus benefit from the optimization features and platform support of the C compilers:

- Nim
- V

Interpreted languages implemented in C:

- Lua
- Perl
- Ruby
- Python

As a language, C has many problems that subsequent languages have tried to fix with varying degrees of success. Examples of such languages:

- C++
- Objective-C
- D
- Go
- Rust
- Nim

Compilers:

- GCC
- Clang
- Zig has a C/C++ compiler that produces much smaller binaries (even static ones) than the mainstream GCC and Clang toolchains.
- Open Watcom is a C/C++/Fortran compiler usable for targeting legacy x86 operating systems such as DOS.
- Tiny C Compiler is a small (100+ KB) standalone C compiler for “modern” x86 and ARM targets (i.e. Linux but not DOS).
- vbcc is an optimizing C99 compiler particularly suitable for some legacy targets such as 68000 and 6502.
- cproc and other compilers based on the QBE compiler backend.

69. Forth

Forth is a stack-based programming language created in the 1970s by Chuck Moore.

The main benefit of Forth is that it is very small. A classical Forth system takes the roles of the compiler, the editor and the operating system while completely fitting in a memory space of less than 20 kilobytes. The smallness also makes it possible to implement a Forth system from scratch in a weekend.

Forth is often run with a two-stack virtual machine, which makes it much slower than compiled code, although native-compiling Forth systems are also common. Classical Forth systems typically also include an assembler that can be used to implement speed-critical parts of the program.

Most programmers see Forth as quite esoteric in comparison to other languages. Forth also doesn't "protect" the programmer from its inner peculiarities – even though it is possible to create abstractions, it is not advisable to forget what lies under those abstractions.

Forth has been standardized, but Moore himself hasn't cared so much about standardization. The "Redo from scratch" ideal is quite strong in the Forth culture and exemplified by Moore's own quest for an optimal set of language elements.

See also:

- Thinking Forth (the classical Forth book)
- Forth - the Early Years (by Chuck Moore)
- Standard Forth system for Uxn

70. Obsolescence

Obsolescence takes place when something is no longer maintained or required, even if it could still be usable. **Planned obsolescence** takes place when obsolescence is actively designed and initiated by the manufacturer/maintainer.

The concept of obsolescence is generally add odds with technological diversity and often also wasteful especially in the case of planned obsolescence.

Types of planned obsolescence

- Obsolescence of desirability: When designers change the styling of products so customers will purchase products more frequently due to the decrease in the perceived desirability of unfashionable items.
- Obsolescence of function: When an item is produced to break down or otherwise become non-functional in an abnormally short period of time.
- Obsolescence of compatibility: When a product becomes obsolete by altering the system in which it is used in such a way as to make its continued use difficult. Common examples of planned systemic obsolescence include not accommodating forward compatibility in software.

- Pseudo-obsolescence of desirability: When planned obsolescence appears to introduce innovative changes into a product, but in reality does not, often forcibly outfashioning an otherwise-useful product.

Examples

- Non-user-replaceable batteries: Some products, such as mobile phones, laptops, and electric toothbrushes, contain batteries that are not replaceable by the end-user after they have worn down, therefore leaving an aging battery trapped inside the device.
- Phoebus cartel: The cartel conveniently lowered operational costs and worked to standardize the life expectancy of light bulbs at 1,000 hours, down from 2,500 hours, and raised prices without fear of competition.

71. Lifespan maximization

Lifespan maximization is the extension of hardware lifespan by the users. It may be supported by planned longevity from the manufacturer's side, but it rarely is.

IC requires large amounts of energy, highly refined machinery and poisonous substances. Because of this sacrifice, the resulting microchips should be treasured like gems or rare exotic spices. Their active lifespans should be maximized, and they should never be reduced to their raw materials until they are thoroughly unusable.

Broken devices should be repaired. If the community needs a kind of device that does not exist, it should preferrably be built from existing components that have fallen out of use. Chips should be designed open and flexible, so that they can be reappropriated even for purposes they were never intended for.

Chips that work but whose practical use cannot be justified can find artistic and other psychologically meaningful use. They may also be stored away until they are needed again (especially if the fabrication quality and the storage conditions allow for decades or centuries of "shelf life").

Use what is available. Even chips that do "evil" things are worth considering if there's a landfill full of them. Crack their DRM locks, reverse-engineer their black boxes, deconstruct their philosophies. It might even be possible to reappropriate something like Bitcoin-mining ASICs for something artistically interesting or even useful.

Minimized on-chip feature size makes it possible to do more computation with less energy but it often also means increased fragility and shorter lifespans. Therefore, the densest chips should be primarily used for purposes where more computation actually yields more.

72. Documentation

Tutorials are lessons that take the reader by the hand through a series of steps to complete a project of some kind. They are what your project needs in order to show a beginner that they can achieve something with it.

- Gets the user started
- Allows the user to learn by doing
- Ensures the user sees results immediately
- Focuses on concrete steps, not abstract concepts

How-to guides assume some knowledge and understanding, and take the reader through the steps required to solve a real-world problem. They are recipes, directions to achieve a specific end - for example: how to create a web form; how to plot a three-dimensional data-set; how to enable LDAP authentication. How-to guides are quite distinct from tutorials. A how-to guide is an answer to a question that a true beginner might not even be able to formulate.

- Solves a problem
- Focuses on results
- Allows for some flexibility

Explanations can equally well be described as **discussions**. They are a chance for the documentation to relax and step back from the software, taking a wider view, illuminating it from a higher level or even from different perspectives. You might imagine a discussion document being read at leisure, rather than over the code.

- Explains a decision
- Provides context
- Discusses alternatives & opinions

In some cases, a discussion that is superficially about a single system may contain deeper insights that make it closer to **wisdom literature**. This type of literature may help pass design wisdom from a generation to another. **Textbooks** may also come close to wisdom literature.

Reference guides are technical descriptions of the machinery and how to operate it. They are code-determined, because ultimately that’s what they describe: key classes, functions, APIs, and so they should list things like functions, fields, attributes and methods, and set out how to use them.

- Describes the machinery
- References material should be austere and to the point.
- Structure the documentation around the code
- Do nothing but describe

Reference guides are particularly crucial for hardware. In software, reference information can be (laborously) derived from the source code or even the binary executable, but similar analysis for microchips requires specialized equipment and can be considerably more complex.

73. Feminist server

TODO: Insufficient introduction/context, please expand

A situated technology. It has a sense of context and sees itself as part of an ecology of practices.

See also:

Feminist Server – Visibility and Functionality

History of Anarchaserver and Feminists Servers

A feminist server ….

Trans*feminist servers…

74. Operating system

An **operating system** is a piece of software that most crucially handles the interoperability between hardware and software as well as between different software programs.

In general, an OS is required in order to run other programs. A program that is written to use the hardware directly does not require a separate OS, but this severely limits the possibility of using any other programs in the hardware. Even embedded systems that are only supposed to run a single fixed application often run on top of some kind of an OS.

See also:

- Operating systems - list of various OSes to be assessed for permacomputing

75. Bloat

Bloat refers to something wastefully and unnecessarily large. In software, it may refer to the presence of program code that is perceived as unnecessarily long, slow, or otherwise wasteful of resources, but a considerable amount of bloat originates from external dependency.

One formulation of bloat is Wirth's law, a variant of Jevons paradox: software is getting slower more rapidly than hardware is getting faster.

76. Character terminal

Character terminals have been a prominent way of using computers since the early years. At first, paper-based teleprinters were the most common computer terminals, but CRT-based video terminals started to replace them in the 1970s. In the 2000s, the character terminal is most often a **terminal emulator** program running on a graphics-capable computer.

Character-based terminals and interfaces from a permacomputing point of view:

Advantages:

- A VT100-compatible terminal emulator is quite simple to implement and its basic hardware requirements are quite low (a couple of kilobytes of RAM are enough for representing the screen contents).
    - Even simpler terminals can be usable in the Unix world provided that a Terminfo/Termcap entry is available. For "VT100-only" software, Screen or Tmux can be used as a compatibility layer.
- The hardware requirements for a server are even lower than for a client. It requires very little from a computing device to send and receive characters over a serial connection; even the smallest microcontrollers can do this.
- The simplicity (no long packets etc.) makes all the possible failures relatively easy to notice and diagnose. Even a bad physical connection will show up as random noise characters unless there's an error-correcting layer.
- Nearly any device with a keyboard and a display can be used as a terminal emulator as long as it has some kind of a serial or network connection. VT100-compatible terminal emulators are also available on just about any classic or modern personal computer platform.
- The basic bandwidth requirements are very low (you mostly send and receive individual characters as individual bytes).
- The limited nature of the character-based screen sets a practical upper limit to the fanciness of the interface, and this helps keep the bandwidth and system requirements low. The simplicity and manageability of text terminals make many people prefer terminal-based software even when working on powerful graphical workstations.
- Simple interactive programs are more straightforward to program for character terminal I/O than e.g. web browsers.

Problems:

- "Dumb" remote terminal connections set quite high requirements for the responsivity of the network and the remote computer. The user feels uncomfortable if there's even a slight delay between the keypress and the visible response. Even WWW is far more tolerant to network lag.
    - Local echo used to be a common terminal feature that could be used to reduce this discomfort. Seeing the typed character immediately also made it easier to spot and correct mistakes. Local echo is useful for simple and rudimentary command line interfaces, but more complex features (such as tab completion) reduce its usability.
    - Mosh is a protocol that extends SSH with some more tolerance for unreliable and laggy network connections by trying to predict how the typed text will look like on the terminal before the server sends its actual response.
    - Block-oriented terminals such as the IBM 2260 and 3270 addressed these problems already in the 1960s. In a sense, web browsers can be thought of as successors of block-oriented terminals.
- The potential of the hardware used as a text terminal is nearly always underused. Even "dumb" terminals are usually not so dumb on the inside. The VT100, for example, is basically a 8080-based microcomputer with far more potential than just running the ROM-based terminal program.
    - Unlike many later terminals and terminal emulators, the VT100 does not have features such as client-side copying, filling or highlighting, or even arbitrary scroll zones (only full-width zones are supported). Since the VT100 became the "gold standard", even today's terminal-oriented software such as Tmux are forced to implement their fancy windowing features by constant character-by-character refreshes.
    - In general, a lot of relatively basic stuff requires far more bytes than it ideally should. When using a text editor over SSH over TCP/IP, every single keypress translates into quite many protocol bytes. Even a website with a text edit box may be much easier for the network.
- Textmode rendering comes from a world where pixels are cheap but memory is expensive. It is therefore not very good for small screens that could be used more efficiently with direct pixel addressing and non-monospaced fonts.
- Many terminal-based programs make the assumption that the screen is at least 80 characters wide. This makes them impractical on small displays.
- Since keypresses are transmitted as individual characters, it is generally not possible to create UIs that depend on the arbitrary pressing and depressing of keys (e.g. cursors that move constantly when an arrow key is down and stop moving exactly when the key is released).
- Likewise, it is not possible to support arbitrary combinations of keys (ctrl+A and ctrl+shift+A both transmit the same byte). The limited key combination support

has made it difficult to implement some common CUI/TUI standards (such as using shift+arrows to mark blocks of text) on VT100-compatible terminals.
- Nearly all ways to display even rudimentary graphics have been somehow crippled in mainstream text terminals:
    - Unlike many contemporaries (such as Teletext, Videotex and many early personal computers), the VT100 has no pseudopixel characters, only box-drawing and curve-drawing ones. Some pseudopixel characters have been adopted into Unicode, but their support in fonts has been unreliable.
    - VT220 and later VTs support user-definable characters but this feature has been rarely implemented in emulators (no surprise, since the character matrix size varies by terminal model and display mode).
    - Some special-purpose VT terminals feature bitmap and vector graphics ("Sixel" and "ReGIS"), and some terminal emulators (xterm) actually support them. However, Sixel graphics was defined so that each individual color needs to be separately transmitted (no proper bitplanes), so it makes sense only for very low-color images.

Some technical vocabulary:

- **ANSI X3.64** is a ~1976 control sequence standard for character-based video terminals. It was criticized for being too complicated for "dumb" hardware and thus requiring a microprocessor. One of the complications was the use of ASCII decimal numbers as command parameters (earlier terminals had used single bytes instead).
- **VT100** and **VT102** are physical, microprocessor-based text terminals manufactured by Digital Equipment Corporation in around 1978-1983. They became the de-facto reference implementation of the ANSI standard.
- **ANSI.SYS** on the IBM PC is another influential implementation of the standard. Notably, it supports the cell-specific foreground and background colors of the CGA-compatible display controllers. "Ansi art" therefore usually refers to art made for the IBM PC textmode, including the non-standard pseudographic characters featured in the IBM PC character set.
- **ECMA 48** (the last version is from 1986) is a later standard that extends on the X3.64. It defines ANSI.SYS-style colors but also some potentially useful features that are very rarely supported in today's terminal emulators (such as a support for multiple pages and editable areas).
- **Text mode** is a hardware feature of a display controller, allowing the representation of the screen as character indexes rather than individual pixels. The pixel matrices of each character are fetched from a separate character memory (which can be ROM or RAM or both). Physical text terminals, most 8-bit microcomputers and IBM PC compatibles have a hardware textmode. In the game console world, equivalent display modes are referred to as "tile-based".

Sometimes, text mode is simulated in software (e.g. the framebuffer console on Linux)
- **CLI** (command-line interface) is based on typed commands. CLIs have been around since the teleprinter days.
- **CUI** and **TUI** (character-based UI, text-based UI) are a more graphical type of character-based interface. They may have things like status bars, cursor-navigable menus and even windowing or mouse support. A lot of work and standardization on CUI/TUI took place in the IBM PC world. Unfortunately, many of the common CUI/TUI approaches are somewhat prohibitive on low-bandwidth character terminals.
- **Telnet** is a protocol for non-encrypted terminal connections over a TCP/IP network. **SSH** is a protocol for encrypted ones (and therefore recommended in nearly all circumstances). The easiness to sniff the network for Telnet passwords was the major reason why SSH was adopted.
- **Pseudographics** refers to characters that have been intended for drawing crude graphical images rather than representing text. Some of these are intended for drawing of box frames (e.g. U+2500..U+257F in Unicode), while others are for simulating low-resolution pixel framebuffers (e.g. U+2580..U+259F). UNSCII is a font that attempts to implement Unicode and legacy pseudographics as completely as possible.

See also:

- Character sets
- ANSI X3.64
- BBS

77. Bedrock platform

A **bedrock platform** is a hardware platform or a universal virtual machine that can be expected to remain compatible with any software that has ever been written for it. Bedrock platforms can be used to prevent software rot.

Note that this is **not** a “shopping list” or a list of “allowed” hardware. Bedrock platform support is simply a way of maximizing the probability that a program can be run in an indefinite future and a way to keep its dependency reliable. The concept of bedrock platform is **not** relevant to e.g. embedded-system-type projects that are designed for a very specific hardware, or the kind of software that is known to have a short lifespan.

A simple bedrock platform guide based on the IBM PC line:

- Can you compile and run the program in FreeDOS?
  - If yes, you have the bedrock support (just make sure that the compiler and other needed tools are archived somewhere).
  - If not, can you create an x86 operating system image that compiles and runs the program without accessing any external resources?
    - If yes, you have the bedrock support (just archive the image and/or everything you've put in it).
- If you can only run it in these environments but not compile it, it is still far better than nothing.

Some possible criteria for bedrock hardware:

- The hardware has been popular and commonly available at some point of history (and preferably remains that way).
- Every detail of the hardware is well-known and fully documented. (Having a 100% compatible open-source emulator can be considered full documentation)
- There have been several independent manufacturers for each component.
- Multiple emulators for the platform are commonly available for many different environments.
- There are no copyright issues in regards to the hardware design, firmware IP, etc.
- It is possible to pinpoint a "standard configuration" that is supposed to run all the programs and can be used for testing software compatibility (usually this means the original version of the hardware or the de-facto most popular variant).

Candidates for bedrock hardware:

- IBM PC: Widely cloned, remains ubiquitous, every type of common component has had multiple manufacturers in different parts of the world (with the exception of OPL2/OPL3 common in classical soundcards). Can be emulated by open-source software such as QEMU, Dosbox or Bochs. There are also several different DOS-compatible operating systems, including FreeDOS. Standard configurations may be difficult to pinpoint.
- NES/Famicom: Widely cloned especially in China/Taiwan without any of the original Japanese components. Huge amount of available emulators, and running a ROM file with one is very straightforward. No software dependencies (as there's no internal firmware ROM).
- ZX Spectrum: Simple design that was easy enough to duplicate in Eastern-block countries even with 100% non-Western components. Clones are still manufactured, emulators are widely available.

- MSX: Standardized platform, every chip used in MSX-1 computers has had both U.S.American and Japanese manufacturers. (MSX-2 on the other hand depends on specific Yamaha chips). Emulators widely available. The firmware ROMs may pose issues as long as Microsoft exists. Also, there's no obvious standard configuration.

Raspberry Pi is an example of a platform that fails the criteria. It depends on a single-manufacturer SoC chip (Broadcom BCM2835) that doesn't have full documentation available. QEMU emulates some versions of the platform to some extent but this emulation does not cover the undocumented parts of the chip (e.g. running the GPU firmware code).

For virtual bedrock hardware, the main criterion is that the specification is simple enough that it can be implemented in a small effort for commonly available computers, and that the specification is unambiguous and frozen.

Even though optimal resource use is not a major point in bedrock platforms, having that as well would be ideal. Not only should it be possible to make minimal-overhead emulators, but to read, static code analysis, decompilation and recompile the code in order to use it on arbitrary future computers. Writers of bedrock-platform software may want to make sure that it will be easy enough to e.g. separate code from data even when analyzing binaries.

The Maxwell's Equations of Software

Alan Kay has famously described Lisp as the "Maxwell's equations of software". He describes the revelation he experienced when, as a graduate student, he was studying the LISP 1.5 Programmer's Manual and realized that "the half page of code on the bottom of page 13 was Lisp in *itself*.

> Yes, that was the big revelation to me when I was in graduate school—when I finally understood that the half page of code on the bottom of page 13 of the Lisp 1.5 manual was Lisp in itself. These were "Maxwell's Equations of Software!" This is the whole world of programming in a few lines that I can put my hand over.
>
> I realized that anytime I want to know what I'm doing, I can just write down the kernel of this thing in a half page and it's not going to lose any power. In fact, it's going to gain power by being able to reenter itself much more readily than most systems done the other way can possibly do.

See also:

- You have made your bedrock, now lie in it.

78. Plan 9

Plan 9 is a fully-featured, clean, compact and uniform operating system built as a successor to Unix.

Plan 9 is not a product, it is an experimental investigation into a different way of computing. The developers started from several basic assumptions: that CPUs are very cheap but that we don't really know how to combine them effectively; that *good* networking is very important; that an intelligent user interface is a Right Decision; that existing systems are not the correct way to do things, and in particular that today's workstations are not the way to go.

Relevance to permacomputing

The Plan 9 system is likely one of the fastest and most complete operating systems for the Raspberry Pi devices. In comparison with the various compatible Linux and BSD distribution, it comes already bundled with all the software and documentation required to understand and expand its inner workings.

"An argument for simplicity and clarity."

— Rob Pike

79. Silver bullet

Silver bullet is a metaphor for an effective and universal solution for any problem. In computing, the metaphor was notably used in Fred Brooks' 1986 essay "No Silver Bullet".

The belief in silver bullets is quite popular in computing. Many computing-oriented people are vocal fanatics of a single programming language, a single operating system or a single design paradigm, that is better than its "competitors" in every context and will eventually make them obsolescence.

Since technological diversity is an important idea in permacomputing, silver-bullet type of thinking should be avoided. Instead of putting ideas or pieces of technology in a universal linear order of superiority, one should be aware of different contexts. The strengths in one context may be weaknesses in others. The "best practice" of a specific context may be an "antipattern" in another context.

Permacomputing is not a silver bullet either.

80. Monoculture

Monoculture is an agricultural term referring to the practice of growing only one species in a field at a time, but the term has also been adopted to computing. Identical computers that run identical software are an example of an **algorithmic monoculture**. Monocultures tend to be vulnerable because all their elements (be they plants or computers) share the same vulnerabilities and other weaknesses.

While a strict definition of algorithmic monoculture may require identical computers and software, looser definitions are also possible. Any kind idea, design concept or piece of software that is so dominant that it is difficult to envision alternatives may form a monoculture.

Technological monocultures can be counteracted with technological diversity.

81. Permacomputing 2020

Permacomputing 2020 is a shorthand to Viznut's essay that introduced the concept of permacomputing in July 2020. Its original title was "Permacomputing (some early notes)".

- The original text at viznut.fi

82. Cryptocurrency

Cryptocurrency is a form of decentralization digital currency that uses strong cryptography for security and verification.

Proof-of-work cryptocurrencies such as **Bitcoin** are extremely problematic from the permacomputing point of view. Their market value is closely connected to the amount of energy used for "mining", which is currently a huge amount of wasted energy (IEA estimates ~100 TWh global consumption for 2020; in comparison, all of the world's data centers consumed 200-250 TWh that year). The embodied energy from the production of the vast amounts of computing hardware used exclusively for "mining" is not included in this figure, and the material life cycles of the hardware will of course need to be considered as well.

What about only using renewable surplus energy and otherwise useless hardware for mining, or participating in the PoW-crypto market without participating in mining at all? It can argued that this is problematic as well, since mere participation in the market (even by actively discussing the topic and thus keeping the hype up) may help maintain the value system.

Proof-of-stake is the main alternative to the proof-of-work model. It has not yet been assessed from the permacomputing point of view. (If you have studied this topic, please consider assessing it and expand the page accordingly)

Blockchain is the type of data structure used by cryptocurrencies. They may be useful for some purposes of decentralization, but it is doubtful whether they are an optimal or even good solution for any problem. (If you have studied this topic, please consider expanding the page)

83. File collection

A **file collection** is a set of computer files that is deliberately maintained by someone. In today's computing, most non-personal files can be considered temporary: they have been downloaded from somewhere and can always be redownloaded if needed again. The practice of actual file collection is becoming ever more marginal, but it has a place in permacomputing for increasing resilience and reducing network dependency.

Before the ubiquity of the broadband Internet, users of personal computers usually had files of all the software they used (and usually a lot of software they never used), often on physical floppies or CDs. These collections were cared for, and even the decision to delete a rarely played PD game could be painful. While people had their private file collections, there were also public file collections, such as BBSes, which often served as repositories of commonly needed PD software and much more.

While broadband networking is probably the most important reason for the marginalization of file collection, another reason may be "software rot if they are not constantly"updated". Keeping a computer offline for a long period of time often brings up large batches of automatic updates; experiences like this may contribute to the illusion of"file rot".

Today, many people have small personal servers or websites, but they are rarely used to share any files other than those directly related to the maintainer. However, they could also easily serve larger collections of files, including copies of all kinds of online resources the maintainer considers important. A lot of resources are available with licences that allow unlimited redistribution, but this right to redistribution seems to be quite underused.

In a file collection culture, nothing that is published is supposed to only stay in one place. A resource stays in one place only if it is too obscure or uninteresting to anyone else, maybe even the author. Even the kind of files that in the WWW culture could be called blog posts or social-media posts get spread to multiple places if they are considered of any interest at all. From the permacomputing point of view, a file that is only available on one server has a hard dependency on that server, and hard dependencies are supposed to be avoided.

In a permacomputing world, servers that host file collections would be just as common as public libraries. People would primarily use the servers that are geographically close to them. They would contain all the commonly used software and documentation (and their complete dependency) along with large amounts of other freely distributable media (books, entertainment, reference databases such as Wikipedia, etc.)

84. Information battery

An **information battery** refers to the "storage" of surplus energy in the form of information (such as intermediate results of computation) that can be used later to lessen the need for computation.

The concept was introduced in the context of supercomputing, but it has also been discussed in the context of degrowth computing in 2022. It mirrors the permacomputing idea of using surplus energy for slow, computing-intensive tasks.

The concept emphasizes the storage of rather temporary information on a rather short-term basis. An example of a more long-term use of surplus energy for more permanent results would be the gradual optimization of software: when initially compiling a program, the compiler would only do rather lightweight optimizations, and the heavier and more computing-intensive ones would be postponed to times of abundance.

See also:

- Original 2021 paper by Switzer and Raghavan

85. About

This website is a wiki dedicated to permacomputing.

We want to encourage discussions presenting an inspiring future as well as research and projects on how they can become reality. Permacomputing starts with questioning and reducing computer use, a focus on repair, and then designing future computing with the biosphere and human life in mind.

Is this for you?

Are you looking to understand the effects of computer use in your community, experiment with projects, create art about alternative futures, and/or join or organize local groups? Then yes! And there's more about these ideas in getting_started.

At the same time, we're not here to help you greenwashing your new "sustainable" technology startup, green AI, ecological web3 funding, circular crypto carbon credit

program, and other nonsensical technosolutionist forms of climate denial and white [re]centering privilege. For these cases, please go elsewhere :)

Contact us!

The Permacomputing wiki is facilitated by ugrnm and viznut. Its content is written by (in order of registration): neau, thgie, akkartik, aw, orx, dusan, katía, luen, suj, cmos4040, giz, pixouls, wakame, decentral1se, clwil, brendan, sister0, kattrali

You can reach all of us at once by sending an email to:

permacomputing / a t / bleu255 dot c o m

HTTP vs HTTPS

It is possible to consult this wiki on port 80, that is to say using `http://` instead of `https://`. The reason to keep providing such access is to allow modest but capable computers, as well as simple/legacy browsers, to access the content. Note that when registering, authenticating or editing, you will be redirected to port 443, `https://`.

Local/Offline copies

We use ikiwiki as wiki compiler. This wiki exists in fact as a collection of plaintext source files that are automatically turned into static HTML files whenever an edit is made to these source files.

Because these plaintext source files and HTLM files are very portable and easy to copy, you are encouraged to have local copies in order to be less dependent on WWW or individual server installations.

See cloning to learn how to make local/offline copies.

Install Notes

If you are curious how this was all installed and configured, we took some notes on installing.

86. Editing

Style

Permacomputing wiki is not Wikipedia, so being neutral or encyclopedic is not among our goals, and original research is encouraged. However, it is a collaborative project, so if you want to express an opinion the other editors may not agree with, please use

the relevant Discussion page, and make sure to sign your comment with your handle. Go to the Discussion page, in the top menu of this page for an example.

Copyediting recommendations

1. explain the abbreviations before using it (ex. Operations Systems (OS) are amazing. The OS are actually shit).
2. make sure the references are explained before using them. (ex. “Unix is a multi-user operating system whose development was started in 1969 by Ken Thompson and Dennis Ritchie, as well as an entire family of operating system derived from the original Unix.” First, explain the original Unix, or rearrange the sentences where you explain what unix is, then explain that there are multiple versions, an original and others). Same for people mentioned, quotes, etc.
3. explain the relevance of a quote.
4. transition between paragraphs, quotes etc. is key. similar to the rule above, make sure the paragraphs are transitioning from one point to another.
5. avoid jargon. if a technical/academic/etc term needs to be used, maybe it needs its own page as well. (i.e. “it refers to a very specific kind of ‘digital’, a highly technoprogressivist and industry-defined kind that became prominent in the ‘digital revolution’ hype of the 1990s.” technoprogressivist is not a common word that could be understood easily, either provide a short explanation or make a page for it)
6. check if there are other pages you can crosslink in your text (i.e. you mention hardware, link the hardware page there)
7. avoid using brackets (as much as possible), it breaks the reading (and comprehending) flow (quite a bit) (right?).
8. avoid passive-aggresive language. rather, explain why some concepts are wrong/ didn’t work/are bad. (i.e. “digital revolution” hype of the 1990s. why was it a hype?)
9. make sure all formatting choices are unified in your text. (i.e. https:// permacomputing.net/Principles/ here, the sub-principles are written as a paragraph first, and then as list. should be the same style).
10. use formatting sparingly and with purpose.
11. avoid repetation and redundant words, be concise, simple, clear.
12. don’t forget to save! :D

On attribution, quotes and footnotes

At the current stage of maturity of this wiki, it is often not advisable to write a comprehensive articles about a topic if someone else has already done it elsewhere. Put in a link to that external resource instead. (In the future, we will perhaps want to host copies of all these "dependencies" in the local repository as well, but not yet.)

When introducing a new term, please try to include a proper and non-biased definition of the topic before proceeding to the permacomputing-specific points of view. You can use Wikipedia or other sources for this, **just make sure you properly attribute and quote** (we are CC0, Wikipedia is CC-BY-SA, so you can't just copy-and-paste even from there).

If you rely on other sources for the writing of a section, do not be lazy or mindlessly copypaste from other sources. In general, **do not invisibilize people from which you took inspiration and/or learned something from**. Take the opportunity of contributing to this wiki to also point to their work and research. You must properly attribute your source. You have 4 options:

- **Hyperlinks:** Sometimes it's enough to point to external reference as an hyperlink if there is not much to discuss. For instance Ursula K. Le Guin has an interesting take on technology.
- **Footnotes:** Can be handy to drift a bit[1] but also to give a proper footnote reference when paraphrasing and referencing the thoughts of someone else. For instance Ursula K. Le Guin has been critical of a specific usage of the work *technology* when misused to only refer to the most recent developments, which also happen to be the most problematic[2].
- **Inline quotes:** Use this with footnotes when you want to quote something short inline. For instance Ursula K. Le Guin offers to understand technology more broadly as "the active human interface with the material world"[3].
- **Block quotes:** Finally, you may want to quote entirely a part of someone else's writing, in which case, use the block quote formatting, with a footnote for the reference. For instance here is how Ursula K. Le Guin suggests to reconsider how we use the word *technology*:

  > Technology is the active human interface with the material world. But the word is consistently misused to mean only the enormously complex and specialised technologies of the past few decades, supported by massive exploitation both of natural and human resources. This is not an acceptable use of the word. [4]

Limitations of the footnotes

This it not biblatex/biber. So as you can see in the examples above, you cannot reuse an existing footnote, and there is not elegant handling of repetition (no Ibid.).

Reference style

When referencing, please use the Notes and Bibliography version of The Chicago Manual of Style. However, this is not an academic paper, don't overthink it or spend ages on it, try to make it work as best as you can, it's just to have some overall consistency. No sweat :)

Acceptable content

Basically any topic is allowed as long as it can be discussed from a permacomputing-relevant point of view and do not break the terms.

Licensing

While editing the Permacomputing wiki, you agree that your contribution will be published and made available under the CC0 Waiver. If you use images, photos, from other sources, please make sure to give full credit and if available the license/tersm under which the image is made available.

87. Dusk OS

Dusk OS is a 32-bit Forth operating system similar to Collapse OS, but more ambitious in its targets and capabilities. Dusk OS is designed to operate on modern, reclaimed machinery in situations where the production of new computers is disrupted. It prioritizes extreme simplicity, currently only about 10,000 lines of code. It is designed to port existing software written in C, using its own DuskCC "Almost C" compiler.

Dusk OS is currently a work in progress, but is under active development.

- Dusk OS project page
- Dusk OS Mailing list

88. WWW

The **World Wide Web** (**WWW** or simply the **Web**), invented in 1989, is the dominant way of using the Internet and also the world's dominant software platform. It is based on files located on servers and linking to each other by URLs that usually indicate the protocols, server address and the path to the file. The protocol is usually HTTP(S) and

the document format is HTML, but other protocols and document formats may also be used as long as they support URL-based linking.

Problems

Problems of WWW from the permacomputing point of view:

Even though HTTP is a simple protocol and the basic HTML is a simple format, the current standards are bloated and only a few web browser engines are able to support them at the commonly required level.

A URL only points to a single server (or a cluster that looks like one), so its usability depends on the availability of that server. When a server is gone, so are all the links that point to it. If a server is temporarily offline, there's no backup plan for locating the files elsewhere.

The design thus discourages small and temporary servers while encouraging large and centralized servers that maximize their availability. The design also discourages file collection (that used to be a common practice when FTP was big).

Attempts to fix the problems

Internet Archive (which is another centralized service) partially solves the problem of broken links and vanishing files by sometimes providing usable copies of the vanished files.

The Smallnet movement advocates simpler formats and alternative protocols (such as Gopher or Gemini) as well as small and personal servers, but hasn't yet done much to address the URL problem that makes those small servers unappealing.

Web3 is supposedly an attempt to decentralize the web by using dubious technologies such as blockchain. The concept/buzzword has so far received a lot of criticism.

IPFS is a peer-to-peer protocol that makes it possible to put files in a decentralized space with no reference to specific servers. The URLs are monstrous but at least the protocol makes decentralized peer-to-peer websites technically possible.

Web & Permacomputing

Overall, the web is not very compatible with permacomputing, particularly due to its low offline tolerance and its server-specific way of referring to resources.

However, its dominant status often justifies its use at least as a way to ensure some level of cross-platform compatibility and accessibility in today's computing world. An

application that only supports a few platforms (such as iOS and Android) is usually an even worse idea than a web-only application. (For using a web browser as a target platform for arbitrary software, see web browser)

Some questions to ask when building something on the web:

- Is this project something that actually needs the web or networking? If it can run locally, support running it locally.
- What does the site require on the client side? (The size and system requirements of the smallest browser it can be used with, the energy requirements of the client-side scripting (if used), bandwidth requirements)
- What does the site require on the server side? (Bandwidth, disk space, the memory/cpu requirements of the server-side scripting (if used))
- How easy is it to use the resources offline? (If the site consists of multiple static files, think about giving the option of downloading the content as a single file)
- Are you fine with the idea of people sharing your content on their own sites? (If yes, put it available under a free content licence and encourage people to spread it)
- Are there ways to use the site with a non-web protocol/client? (Permacomputing wiki can be edited via Git, for example)
- What are you planning to use as a server? Purchasing of dedicated hardware should be avoided, but using something your community already has is a good idea. Using a time-sharing account to a small, local webhotel is always a better idea than buying “cloud computing” from a multinational corporation (even if you’ll only get 99.9% uptime instead of 99.999%).

Software

A list of moderate-sized web browsers:

- Dillo
- Links
- Lynx
- W3C

A list of moderate-sized web servers:

- lighttpd
- …

Common types of web applications:

- wiki
- …

89. Web browser

A **web browser** is a client program for accessing the WWW. Its main feature is the loading of HTML pages over HTTP(S) connections and rendering them. There are moderate-sized web browsers as well as huge and bloated ones.

Today, many WWW sites are entire applications that depend on complex client-side programmability with Javascript or WebAssembly. This makes the modern web browser the most common application software platform. Supporting it may therefore be useful, even though exclusively supporting such an unreliable target is not that wise from the permacomputing point of view.

Programs written in C, C++ and some other languages can be compiled into web browser applications with LLVM-based compilers such as Emscripten and Zig. This makes it possible to minimize web-specific code and to use the same source code for native applications.

90. Internet

The **Internet** is the global network of computer networks that use the TCP/IP protocol suite to communicate. Sometimes, even wider definitions have been used; for example, any computer that is reachable by Internet email could be said to be “on the Internet” even without an IP address.

The Internet routing technology was originally based on military requirements, i.e. nuclear attack survivability. The possibility for non-hierarchical and decentralized routing makes the network quite resilient.

However, many other key aspects of the Internet are somewhat centralized (e.g. the tree-like way of dividing the global IP address space or coordinating the DNS hostnames). The currently dominant resource access technology, the WWW, is particularly oriented around centralized servers and thus lacks the resilience of the underlying network.

Matter and energy

The energy and material requirements of the Internet-related infrastructure are of particular concern to permacomputing.

TODO: Include information about research on the energy requirements of data transmission etc.

Other aspects of the Internet

- history and culture
- services and protocols
- alternative ways of networking, both historical and current

91. Paper computer

The design for a simple portable computer that only requires a pen and a piece of paper.

The computer consists of a sheet of paper that contains both the program as well as a number of data registers, that will be used to represent the contents of the registers.

To begin, the pen, representing the program counter, is positioned at the line 00 of a program. The instruction in that line is then processed by the user by either moving the pen(program counter), modifying the value of a data register or by checking if a data register has become “empty”(zeroed).

Historical

- CARDIAC
- WDR
- etc..

92. DRM

Digital rights management is a way to protect copyrights for digital media.

When standards and formats change, DRM-restricted content may become obsolescence. When a company undergoes business changes or bankruptcy, its previous services may become unavailable.

93. IC

An **integrated circuit** (**IC**) or a **microchip** is a set of electronic circuits (usually consisting of MOSFET transistors) on a small flat piece of semiconductor material (silicon). It is currently the dominant technological basis for computers.

Moore’s law and **Rock’s law** have succesfully described the history of semiconductor integration for decades: while it has been possible to double the transistor density every two years, the cost of building a state-of-art fabrication plant has doubled every four years. This has centralized IC fabrication and distanced it from local and small-scale production.

Types of ICs:

- processor
- memory
- ASIC (nearly any complex IC that's not general-purpose can be called "application-specific")
- logic chips (e.g. the 4000 series and the 7400 series)
- TTL
- FPGA
- …

Making

There are hobbyists who have succesfully produced ICs in garage-like conditions. In 2021, Sam Zeloof produced twelve 1200-transistor chips at home (the first microprocessor had 2300 transistors), although some second-hand special purpose equipment such as a maskless photolithography stepper were used.

DIY project information:

- Sam Zeloof
- HomeCMOS project

The materials that end up in the finished products don't have to be exotic or destructive (e.g. silicon, oxygen and aluminum for the bare chip, and plastic/ceramic and some metal for encapsulation). However, fabrication processes usually use dangerous substances such as hydrogen fluoride (HF) to keep the chip clean. Trichloroethylene (TCE) has poisoned groundwater supplies several times, including in 1974 when a leak took place in the MOS Technology Pennsylvania plant. Finding alternatives to these substances may be more urgent than developing local and small-scale production.

Zeloof has been able to eliminate some aggressive chemicals such as sulfuric acid from the process. HF is still mentioned, but CF4 and CHF3 are listed as alternatives for it.

94. Lua

Lua is a general-purpose high-level language created since the 1990s at PUC Rio de Janeiro.

Lua provides a sweet spot along three axes:

- high-level dynamic syntax with anonymous functions and closures
- reasonably high performance by compiling to a virtual machine's bytecode

- small and approachable implementation with no dependencies beyond C

Lua also provides a framework for graphical applications called LÖVE. LÖVE depends on the host OS for graphics. On Unix-descended systems, that is approximately the SDL2 library.

95. Smalltalk

Smalltalk is an object-oriented, dynamically typed reflective programming language. Kay says their Smalltalk virtual machine for the 8086 was 6 kilobytes of machine code.

- ST-72 on the SqueakWiki

Relevance to permacomputing

While the language itself might not apply every aspect of its own design philosophy, the guidelines drafted out in Design Principles Behind Smalltalk, might in some way or other align with the values of Permacomputing.

- human-scale: If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual.
- bedrock platform: A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible; and all parts of the system should be held in a uniform framework.

96. Go

Go is a general-purpose programming language created in 2007 designed primarily for the needs of server-side networked application programming.

Go is often favoured for its portability, relative simplicity, ease of concurrent programming, performance and fast build times.

There are several major implementations: gc, gccgo/gollvm, GopherJS and TinyGo.

Go has a published language specification and a v1 compatibility promise which is useful when considering long-term program maintenance.

The loading of program dependencies is based on URLs. This implies always-on internet connectivity and typically centralised git repository hosting. A work-around for this is to use vendoring which supports downloading and bundling all program dependencies into the source tree.

Go can be bootstrapping with Go itself, or C (via `gccgo`/`gollvm`). However, with the introduction of generics, not all new versions of Go can be bootstrapped with C.

`gccgo` supports up to version 1.18. It's also possible to bootstrap from Go 1.4, the last version written in C.

There are language design decisions which make Go unsuitable for embedded systems. The TinyGo project aims to provide an alternative.

Go binary sizes are generally acknowledged to be bloat. For example, a binary size of 1.3 MB for 6 lines of code. The design of Go prioritises performance at the cost of memory, embedding considerable amounts of runtime information in produced binaries. Passing flags such as `-ldflags="-s -w"` to `go build` can aid in size reduction.

Similar to Rust, Go is largely dependent on Big Tech for resources and funding.

97. TODO

To keep track of things TODO before launching the wiki:

Server

- migrate pmc wiki to new server
- modify cron zip script to run only if changes made to wiki

ikiwiki

- strategy for handling images / decide on policy for images (ditherpunk?)
- look into the file upload plugin for images and stuff
- tag index plugin
- A more wikiwiki layout (top and bottom menu)

Content

- section for curriculum postdoc thing (wip)
- section on workshop outline "The Cloud is Just My Old Computer"
- new properties structure from katía/crunk (wip, will be spread over pmc long page and the issues with tech page)
- laypeople-accessible introduction to permacomputing
- switch to index3

98. DOS

Disk Operating System (**DOS**) is the name of many operating systems starting from 1960s mainframe OSes, but most commonly it refers to Microsoft's MS-DOS and compatible systems running on IBM PC compatible hardware. FreeDOS is an actively

maintained DOS-compatible operating system that supports things like USB memory sticks. DOSBox is a popular emulator for running DOS-based IBM PC programs in modern operating systems.

DOS is designed to only run one program at a time with minimal external dependencies. It is thus quite immune to many problems of more modern OSes, such as software rot. It may well be used as a bedrock platform or as a “bare-bones” system that may work for basic tasks even when everything else fails. The main compatibility issue in DOS is that a lot of hardware is commonly used by “banging the bare metal”, which makes e.g. old sound applications incompatible with modern hardware.

See also:

- CP-M
- Collapse OS

99. IRC

The **Internet Relay Chat** (**IRC**) is a multi-user chat protocol created by Jarkko Oikarinen in 1988. It was inspired by a similar service called “Relay” on the Bitnet network.

The “official” permacomputing IRC channel is #permacomputing on the Libera.chat network.

Permacomputing assessment

Pros:

- The protocol is relatively simple (although slightly wasteful on bytes, e.g. the characters “PRIVMSG #” are repeated for every public message).
- It is rather lightweight to the client and the server alike.
- There is a large amount of different implementations for a vast variety of platforms.
- It is open and has no owner.
- There have been no major changes that would have jeopardized the backwards compatibility between the servers and the various client implementations. Client programs from the 1990s can still be used.

Cons:

- It is highly dependent on constant unbroken connections on both the server side and the client side.
    - Breaking of inter-server connections cause “splits” where messages from some servers never reach the other servers.

    - Similarly, the clients only receive those messages that were sent when they were connected.
- While the lack of a server-side message storage makes the servers more lightweight, it encourages individual users to maintain persistent connections to the servers and to keep logs of their own. Thus, the total need for computing resources ends up being higher than what it would be with server-side persistence.
- The setting up of a constant client connection is often more complicated for a user than using a website or an “app”.
- The nature of purely realtime feeds and the flat structure of the messaging make it difficult to maintain several discussions on the same channel at the same time.
    - The difficulty of participating in “old” discussions easily contributes to the “fear of missing out” that is a major element of IRC addiction.

Alternatives to IRC

Every now and then, there have been attempts to fix the problems of the IRC, often by launching entirely new chat services. Examples from the 2010s include Slack and Discord. These are usually centralized and non-open services owned by companies, so they aren’t very interesting from the permacomputing perspective.

Perhaps the most promising alternative to IRC at the time of writing is Matrix.

100. iff

Interchange File Format is a generic container file format originally introduced by the Electronic Arts company in 1985 (in cooperation with Commodore) in order to facilitate transfer of data between software produced by different companies.

Relevance to Permacomputing

IFF helps minimize problems such as new versions of a particular program having trouble reading data files produced by older versions, or needing a new file format every time a new version needs to store additional information.

It also encourages standardized file formats that aren’t tied to a particular product. All of this is good for endusers because it means that their valuable data isn’t locked into some proprietary standard that can’t be used with a wide variety of hardware and software. Above all else, endusers don’t want their work to be held hostage by a single corporate entity over whom they has no direct control.

IFF helps to break this needlessly proprietary stranglehold that developers have exerted upon endusers’ works.

101. Digital preservation

Techniques

Migration: Periodically convert data to the next-generation formats.

Emulation: Mimicking the behavior of older hardware with software, tricking old programs into thinking they are running on their original platforms.

Encapsulation: Encase digital data in physical and software wrappers, showing future users how to reconstruct them.

Universal Virtual Computer: Archive paper copies of specifications for a simple, software-defined decoding machine; save all data in a format readable by the machine.

Useful Concepts

Bitstream Copying: is more commonly known as “backing up your data,” and refers to the process of making an exact duplicate of a digital object.

Persistent Media: a medium like a gold disk, may reduce the need for refreshing, and help diminish losses from media deterioration, as do careful handling, controlled temperature and humidity, and proper storage.

Technology Preservation: is based on preserving the technical environment that runs the system, including operating systems, original application software, media drives, and the like. It is sometimes called the “computer museum” solution.

Digital Archaeology: includes methods and procedures to rescue content from damaged media or from obsolete or damaged hardware and software environments. Digital archaeology is explicitly an emergency recovery strategy and usually involves specialized techniques to recover bitstreams from media that has been rendered unreadable, either due to physical damage or hardware failure such as head crashes or magnetic tape crinkling.

Analog Backups: an analog copy of a digital object can, in some respects, preserve its content and protect it from obsolescence, while sacrificing any digital qualities, including sharability and lossless transferability. Text and monochromatic still images are the most amenable to this kind of transfer.

Replication: In each case, the intention is to enhance the longevity of digital documents while maintaining their authenticity and integrity through copying and the use of multiple storage locations.

Normalization: is a formalized implementation of reliance on standards. Within an archival repository, all digital objects of a particular type (e.g., color images, structured text) are converted into a single chosen file format that is thought to embody the best overall compromise amongst characteristics such as functionality, longevity, and preservability.

Canonicalization: is a technique designed to allow determination of whether the essential characteristics of a document have remained intact through a conversion from one format to another.

102. Emulation

Emulation is a digital preservation technique involving mimicking the behavior of older hardware with software, tricking old programs into thinking they are running on their original platforms.

See also:

- bedrock platform
- virtual machine

103. Emotionally durable design

Emotionally Durable Design is a framework brought forward by Jonathan Chapman. It proposes attributes that should be considered in designing things. The aim is to enable us humans to form stronger bonds with the designed things.

The attributes brought forward in the Design-Nine paper are:

- Relationships
- Narratives
- Identity
- Imagination
- Conversations
- Consciousness
- Integrity
- Materiality
- Evolvability

Certain attributes are clearly coming from design, treating things as can be expected. Others, for example *Conversation* and *Consciousness*, are more philosophical. These attributes are treating things as non-human beings, essentially flattening the ontological hierarchy between humans and non-humans. This approach is common in indigenous epistemologies. There is no direct equivalent in western (EU- and US-

centric) practice, but traces of it can be found in philosophy, for example in the book Vibrant Matter by Jane Bennett.

-

My, thgie, personal opinion: The book by Jonathan Chapman is difficult at times, going into formulations that can be read as classist. I appreciated the practical/designerly approach to think about non-human ontologies.

Bibliography

Chapman, J. (2015). Emotionally durable design: Objects, experiences and empathy.

Haines-Gadd, M., Chapman, J., Lloyd, P., Mason, J., & Aliakseyeu, D. (2018). Emotional Durability Design Nine—A Tool for Product Longevity. Sustainability, 10(6), 1948. https://doi.org/10.3390/su10061948

104. Personalities

Permacomputing Personalities

This non-exhaustive list of archetypes tries to discover and identify different perspectives for permacomputing. These are “pure”, “extreme” and stereotypical descriptions, a real person can likely identify with several of them in different degrees.

The descriptions are also very “opinionated”, especially assumptions about their motivation. Similar to De Bono’s Six Thinking Hats, they can be used to discuss a topic from different perspectives.

If you feel that something is missing, you are very welcome to contribute.

Archetypes

The Vintage Computing Enthusiast (VCE)

The Vintage Computing Enthusiast is very interested in hardware that is at least two decades old. They like to tinker with and restore or computers, maybe they are recreating older devices on modern hardware using emulators or by putting modern hardware into older-looking enclosures.

VCEs are motivated by caring, preservation and nostalgia.

Opinions:

- O1: Older hardware should be kept in working condition for as long as possible.
- O2: Simpler devices are better than complex ones, because they are easier to repair, maintain and rebuild.
- O3: Older technologies contain knowledge that needs to be kept alive.

The Post-Collapse Prepper (PCP)

The Post-Collapse Prepper believes that current civilization will collapse in one way or another in the next decade and that some form of computing technology needs to be kept alive nonetheless.

(see http://collapseos.org/why.html)

Opinions:

- O4: Building electronics from scavenged parts will become more important in the future.

The Cosplay Wastelander (CW)

The Cosplay Wastelander is inspired by a post-apocalyptic, mostly destroyed earth, with a certain degree of “fictionality”. Examples for the aesthetics include the “Mad Max” movies or “Fallout” computer games.

They are interested in mobile, ruggedized or dirt/damage-resistant hardware, likely with an “exposed wires and PCBs” look.

Opinions:

- O4: Building electronics from scavenged parts will become more important in the future.
- O5: Ours is just one of many worlds, many others are possible.

The Solarpunk Tinkerer (ST)

The Solarpunk Tinkerer wants to solve the problems of the future from the comfort of their workshop. They believe that technology and computing will become essential in maintaining the solar-powered farming machines, automated transport vehicles and small-scale medicine laboratories that will be used to produce food, materials and medicine locally instead of relying on global supply chains.

They see logistics, overproduction and growth-oriented economies as current problems that need to be solved in order to create a sustainable future. In their opinion, electronics/computing is required so that humanity and nature can coexist.

Opinions:

- O6: To build a proper computer, you should need no more than what an average workshop provides.

The Computer Abolisher (CA)

The Computer Abolisher believes that much of modern technology is a necessary evil that needs to be overcome. Necessary, because in a future shaped by degrowth, modern technology is still needed to implenent far-reaching changes to the world. In the long-term, they want to decrease humanities dependence on computing to nearly or almost zero.

Opinions:

- 07: Technology is a means to an end, not a goal in itself.

The Neo-Luddite (NL)

The Neo-Luddite believes modern (computing) technology to be the latest invention to drive inequality and keep means of production out of the hands of the common person. While not against technology in principle, they see our current dependence on huge, expensive factories for computer hardware, large companies who write our code and technology trends like “cloud computing” as deliberate decisions to concentrate power.

The NL wants to democratize hardware and software development, including technologies that make it possible to create hardware locally, to write complex programs with little (or distributed) manpower.

Opinions:

- O8: Computing does neither belong in the ivory towers of research nor a corporate headquarter.

The Permacomputing Artist (PA)

The Permacomputing Artist takes inspiration from different aspects of Solarpunk, Permacomputing, Demoscene and related areas. This can take the form of constraints, from lower resolution displays to energy-saving computing, from file size restriction to color palette aesthetics. Or maybe describing or visualizing a future in which the currently researched concepts have found their way into practice.

Opinions:

- O5: Ours is just one of many worlds, many others are possible.

The Computer Counter Culturist (CCC)

For the Computer Counter Culturist, vintage computing, permacomputing and related fields offer the opportunity to make statements that question current "majority culture". Using older hardware instead of buying the newest device, using devices consciously instead of photographing and posting every meal, writing Interactive Fiction (IF) that can be run on fourty year old computers instead of creating games that require the latest hardware. The focus of the CCC is to call consumerism, wastefulness, capitalism and similar topics into question.

Opinions:

- O9: It is important to show what could be, to inspire people to change things.

Opinions

Based on the archetypes, different "opinions" can be identified. These are statements that a person can agree/disagree with (to varying degrees).

These statements can also serve to create a map (or grid) of potential research areas / areas of inquiry.

- O1: Older hardware should be kept in working condition for as long as possible.
- O2: Simpler devices are better than complex ones, because they are easier to repair, maintain and rebuild.
- O3: Older technologies contain knowledge that needs to be kept alive.
- O4: Building electronics from scavenged parts will become more important in the future.
- O5: Ours is just one of many worlds, a lot of others are possible.
- O6: To build a proper computer, you should need no more than what an average workshop provides.
- 07: Technology is a means to an end, not a goal in itself.
- O8: Computing does neither belong in the ivory towers of research nor a corporate headquarter.
- O9: It is important to show what could be, to inspire people to change things.

105. Aesthetics

Aesthetics is relevant to many aspects of computing. Here, we are mostly concerned of the superficial visual appearances and their technological bases.

A predomidant aesthetic in mainstream computing is **maximalism**, that is based on the idea that increased detail, complexity and "fidelity" are the key to "better graphics". This kind of preference is highly problematic from the permacomputing point of view because it creates a preference for energy usage maximization (when

uncapped). Permacomputing should therefore strongly prioritize non-maximalist aesthetic approaches.

The dominant approach on the demoscene is optimalism, or “capped maximalism”. It often proves that mass appeal can be reached despite tight limitations, but the aesthetic basis is still maximalist – fitting a maximal amount of content within the limits, the more the better.

Ideally, the low complexity itself can be a source of beauty: things can look good *because* of their smallness, not *despite* it. If this succeeds really well, even the most mainstreamy viewer won’t be longing for more resolution or detail.

In user interfaces, the ideal of low complexity may easily lead to the now fashionable oversimplification, where the internal details are hidden from the user. This is not what we want. We should rather find ways to keep users aware of what is going without overwhelming them with the information.

Media minimization techniques sometimes lead to styles such as “ditherpunk” that require acquired taste and are still more likely to belong to the “despite” category than the “because” category.

Another example of acquired taste is “Unix brutalism” that uses a lot of monospaced fonts, program code and other elements typical of character terminals. It should be noted that despite its “hardcore low-level vibes” it is often a suboptimal way of using display hardware.

The characteristics of electronic paper (slow update speeds, low saturation, no flashing, bookiness) may be used as an antithesis for the psychologically intensive mainstream computer aesthetics – regardless of what kinds of screens are actually used. Elements may grow in rather than scroll in (more like plants than cars). The semblance of printed media alludes to a world that is slower and more thoughtful than the mainstream Internet.

See also:

- Permacomputing Aesthetics paper, 2023
- Slides from the presentation of the paper

106. Artificial intelligence

Artificial intelligence (**AI**) refers to machine actions that are perceived by humans as intelligent. **Machine learning** is a subset of AI where the “intelligence” is not programmed in but learned from data by a machine learning model. In current mainstream usage, these terms are nearly synonymous, but AI in general is a very vast field of different approaches, most of which are quite obscure to non-experts.

Dumb and smart programs

Humans instinctively relate to machines and tools either as body extensions or as autonomous creatures. It is generally not a good idea to require a user to use both approaches simultaneously. “Smartness” is not wanted in applications that are supposed to be wielded as tools or instruments, if it makes them more complex and less predictable. If there are “smart” functions in a tool, they should be clearly separate from the “wielded” portion, with the option of disabling them completely.

Among the most important software, compilers are programs that are generally supposed to be rather smart in order to produce efficient code for the target platform. It is also where a large amount of resource use can often be justified by the energy that is saved by the efficiency of the produced code.

Green AI

The research, training and deployment of very large machine learning models takes a radically increasing amount of energy and dedicated hardware in today’s world. This is why “Green AI” has become a thing.

The tinyML foundation is concerned about small machine learning models that run on very low power when trained (but may still require a lot of resources to train).

(Please include interesting information/resources about low-power AI/ML, if you have studied this topic)

See also:

- Schwartz&al’s Green AI paper from 2019
- Playing Atari with Six Neurons

107. Automation

In **automation**, the essential question is how much human effort the automation saves in comparison to the requirements of the automation technology.

Mere laziness does not justify automation: modern households are full of devices that save relatively little time but waste a lot of energy. Automation is at its best at continuous and repetitive tasks that require a lot of time and/or effort from humans but only a neglectable amount of resources from a programmable device.

Permaculture wants to develop systems where nature does most of the work, and humans mostly do things like maintenance, design and building. A good place for computerized automation would therefore be somewhere between natural processes and human labor.

108. Awareness amplification

Awareness amplification is an important purpose of computing as well as a design principles of permacomputing. In general, it refers to technologies that make the world (including the local environment) more observable to people, but it also includes making computers more aware and sensitive of their surroundings.

Observation can be either active or passive. The user may specifically focus on a specific part of the world or have a background awareness of what is going on in general. This is a major difference from intelligence amplification that emphasizes an active and narrow focus on specific objects of information.

When amplifying the user's background awareness, it is crucial to not steal their attention. Many user interface designs fail this; a common mistake is to use intrusive notifications for events that are unrelated to the user's current focus of attention. Soundscapes and landscape-like visualizations could offer a non-intrusive alternative.

The more complex a technology is, the more important it is that it dedicates a portion of this complexity to explain itself. Simple tools such as hammers or bicycles are pretty much self-explanatory due to their simple and easily observable designs, but computers need to amplify their own observability in order to be observable at all.

Examples of awareness amplification within core computing:

- A non-intrusive visualization of everything the computer knows about its environment (energy sources, network traffic, data from any additional sensors the computer is connected to)
- A computer that adjusts its operation to what it knows about its environment.
- A user interface that makes the computer's inner workings and material-energetic conditions easily observable to the user.

More applicative ideas:

- A sensor system that increases the observability of the ecosystem the computer and its users belong to, including the cultivated and constructed parts thereof. This may help people notice changes and details that beyond the reach of normal human senses.
- Simulations of ecosystems, economic systems and other "subsystems of the world" can be used in planning as well as in gaining deeper insight to how these systems work.
- Many intelligence amplification applications may also amplify awareness: automatic information retrieval, mind-mapping and note-taking systems, co-operative telecommunications environments, etc.

109. Kolmogorov complexity

The **Kolmogorov complexity** of an information object, such as a piece of text, is the length of the shortest computer program that produces the object as output. It is a measure of the computational resources needed to specify the object, and is also known as algorithmic complexity.

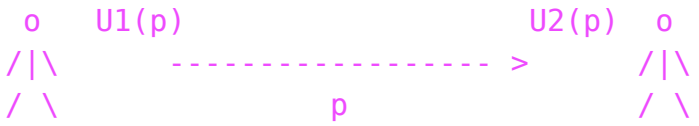
```
KU( x) = min { |p| | p ∈ Σ*0, U( p) = x }

U - Universal Turing machine
Σ - tape alphabet
p - program
x - information object (text)
```

The program (also called minimal description) received in language grammar system common for both sides triggers generative action on receiver side producing the same language information as by transmission originator.

```
Historically, the first scientific concept of information
originates from Shannon (1949) and defines the ammount
of information in the object as bit length of transmission
needed to choose the right object from predefined set of
elements.
This is used in prefix-coding schemes, where the most used
symbols
are encoded with least bits and least used symbols with more
bits.
```

Kolmogorov computational principle from 1963 can be related to association in psychology and art. With association, previous experience is recalled from memory by a short impulse or pretext. Vladimir Boudnik in Explosionalism (1949) states that image is built layered on previous influences, memories and experiences. Artwork is a shot which explodes in people's heads (like infinite stream of associations).



Chaitin complexity is a minor modification of Kolmogorov complexity, which was discovered independently. We presumed that universal turing machine with alphabet Σ0, works with blanks on the tape to recognize end of programme. It can be a

problem, because we cannot chain programmes or put data on tape without other delimiters etc. Chaitin complexity, also called self-delimiting complexity HU(x) of string x in universal machine U is a length of shortest self-delimiting program p, which in U produces x.

```
HU( x) = min { | p | | U( p) = x}

p is self-delimiting programme, that means the end of the
programme
can be recognized by reading only all of its symbols and nothing
else.
As a consequence, such a programme has to be non-prefixed.
```

Chaitin complexity leads to the definition of algorithmic entropy and information complexity.

110. Ethnomathematics

Ethnomathematics is the study of the relationships of mathematics and culture, with a specific focus on the mathematical thinking of indigenous or “non-literate” peoples. **Ethnocomputing** is an offshoot of ethnomathematics that does the same thing with computing. In practice, both ethnomathematics and ethnocomputing are most often connected with education, with the belief that using familiar concepts from one’s own cultural background will lead to better learning results.

Ethnocomputing and ethnomathematics are relevant to permacomputing particularly from the point of view of technological diversity. How we currently conceptualize computing is a result of specific historical and cultural conditions, and the cultural basis is actually getting narrower due to siliconization. Ethnomathematics and ethnocomputing can be used to reveal this narrowness as well as to help imagine a greater diversity of options. They may also help envision deeper history roots to algorithmic, computational and mathematical thinking – they’re much older and much more universal than commonly thought in the eurocentric techno-progressivist narrative.

NOTE: While cultural appropriation is usually not a big concern in theoretical computer science topics, it is possible to use ethnocomputing in problematic ways that make it a concern. One should be careful and respectful when using and representing computational or mathematical concepts from different cultures.

Some interesting examples:

- Many traditional divination systems (I Ching, Geomancy, Ifá) use binary combinatorics, i.e. give meanings to 3-, 4-, 6- and 8-bit binary sequences.

- The quipu/khipu recording system of Andean peoples, based on strings and knots, has been studied as an example of a complex indigenous data structure.
- Fractal-like recursion and self-similarity are very prominent in African arts. This is a central theme in Ron Eglash's seminal ethnomathematics book "African Fractals: Modern Computing and Indigenous Design".

111. Games

Games have been made for computers since the early years. Playful and non-utilitarian uses are an important aspect of any technology – for example, playing with hunting bows led to the invention of the musical bow and other string instruments.

Computer games are commonly framed as 1) entertainment (rather than e.g. art or life-changing experiences), 2) something produced by an "industry" (in the same way as there is "the film industry"), and 3) something that has high demands on the hardware resources. Many of the common conceptions are at odds with permacomputing. One should therefore look into the "margins" (i.e. "independent" games) for a more diverse view of what computer games can be.

Before siliconization, it was a common idea that any computer is suitable for playing games – and the most limited computers were actually considered unsuitable for anything else. This is something worth returning to in permacomputing: target those platforms that are available to anyone (either directly or via emulation), and especially those that are no longer preferred for utilitarian purposes. Supporting a bedrock platform increases the likelihood that the game can be revisited in an indefinite future.

Maximalist aesthetics is a major driving force for obsolescence and ever higher hardware requirements. It is a better idea to stick to a less demanding style (e.g. pixel art) than to adhere to the mainstream gamer idea of "up-to-date graphics".

In the non-digital world, a "game" is often more like an immaterial idea than a product (think about the variety of games that can be played with playing cards). The same approach can be applied to computer gaming as well: Tetris is not really a specific program/product, but a game that any programmer can implement in a moderate amount of time. If the logic is defined strictly enough, the different implementations can be compatible in regards to scoring, competition and skill acquisition. The games that can be fully described as sets of rules have the longest potential lifespans.

Computer games often blur the distinction between a game and a virtual environment. Game-playing is a goal-oriented activity, but in a virtual environment it is not necessary to reach for anything specific. An important aspect of simulated environments is that their representations of the world can be used to gain insights to the actual world. This is how games can be used to awareness amplification.

Common problems

Game programming is often quite careless, as games are traditionally “allowed” to use all the available resources. Thus, it is common to find things like CPU-hogging busyloops even in turn-based games, or ridiculously high system requirements in modern pixel art games.

Running of games often requires emulation. Cycle-exact emulation may be orders of magnitude more resource-hoggy than “just enough” emulation, so it may be a good idea to support the fast but inaccurate emulators when targeting classic platforms.

Specific games

These games may be interesting from the permacomputing perspective, either technically or otherwise.

Another World

There is very little code in Another World. The original Amiga version was reportedly 6,000 lines of assembly. The PC DOS executable is only 20 KiB. Surprising for such a vast game which shipped on a single 1.44 MiB floppy. That is because most of the business logic is implemented via bytecode. The Another World executable is in fact a virtual machine host which reads and executes byte-sized opcodes.

Adventure games from Infocom, Sierra and Lucasfilm are also based on bytecode virtual machines.

See more:

- The Polygons Of Another World

112. Information and energy

Related to: Thermodynamics

Information and entropy relation to energy

We can distinguish 2 types of information in relation to abstract and physical worlds:

- Unbound information - possible symbols or states are understood as purely abstract and not related to any physical system. That is information used in mathematics.
- Bound information - possible symbols or states are identified as microstates of some physical system

Information to be recorded, transmitted or processed has always some material carrier and has to be reflected or mapped to some signal. With processing such a signal the spread of matter or energy in some thermodynamic system is changing. So any process of computation of bound information is necesarilly followed by change in thermodynamic entropy.

Information in thermodynamic units as information bound to states of some thermodynamic system is negative entropy (negentropy).

Getting some information (by observation or measuring), the entropy is going down. But this observation needs some energy and this energy taken is increasing the entropy. From second thermodynamic law this observation or getting information needs always more energy than is the amount that resulting information contains.

Thermodynamic system is a continuous space containing a large amount of particles in interactions with each other. The rest of physical world is system surroundings.

The thermodynamic system can be one of the next 3 types:

- Isolated system, which does not exchange any matter or energy with surroundings
- Closed system, which can exchange energy, but not matter
- Open system, which exchanges both energy and matter with surroundings.

First two are only abstract or temporary states of the systems. We even cannot get any information about state of the isolated system. All computational systems are open, as computational devices are material, and material is being mined, formed, assembled, disassembled, so any computation, no matter how abstract and symbolic, is bound to the material and energy exchange.

When the abstract mathematical or symbolic processing is done in pure mind of the human, there is an energy and material exchange needed by related biological processes in the brain.

Energy needed for signal modulation

Landauer's principle asserts that there is a minimum possible amount of energy required to erase one bit of information, known as the Landauer limit: $E=k_B T \ln 2$ (k_B is the Boltzmann constant and T is the temperature). For T equal to room temperature 20 °C, we can get the Landauer limit of 0.0175 eV (2.805 zJ) per bit erased.

The equation can be deduced from the Boltzmann's entropy formula $S=k_B \ln W$, considering that W is the number of states of the system, which in case of a bit is 2, and the entropy S is defined as E/T. So the operation of erasing a single bit increases

the entropy of a value of at least k_B ln 2}, emitting in the environment a quantity of energy equal or greater than Landauer limit.

It puts a fundamental ceiling on the increase in the number of computations per joule of energy dissipated. Until recently, this increase has been exponential (doubling every 2 years), so by 2048 we would reach Landauer's limit. Probably the slowdown already increased the doubling to 2.6 years, which means there will be more limited increases in performance per Watt.

113. Permatechnology

Permatechnology refers to permacultural approaches to technology. Permacomputing can be thought of as a subset of permatechnology, which in turn can be thought as the technological aspect of permaculture. The concept of permatechnology can therefore be used as a linking piece that provides a larger context for permacomputing.

Computing is somewhat peculiar when compared to e.g. transportation technology: airplanes are not many orders of magnitude faster or more energy-efficient than sailships, but computers can have this kind of contrast to manual calculation and data processing. Computing therefore constitutes an interesting extremity of the technological possibility space.

Another difference is that practically all production of computers is currently based on somewhat destructive and highly centralized processes, so a lot of rethinking and rebuilding is required for the type of technology in general.

All of these peculiarities give permacomputing somewhat unique challenges compared to most other types of permatechnology.

114. Regenerativity

Regenerative design, Regenerative computing

The design of digital technologies (hardware and software) should be determined by democratic and participatory processes and help regenerate natural ecosystems and promote social cohesion. [5]

Regenerativity principles disrupt and surpass technology design associated with power accumulation, a polarisation of opportunity and environmental inequalities, consumption and profit maximalisation.

- Regenerate natural ecosystems and social cohesion.
- Democratic, co-creative and sustainable.

- Creative problem solving with wide perspective and planet encompassing focus with humans as integral part.
- Evolution from designing objects to designing material flows and systems serving the common good.
- Design process should be as open, participatory and transparent as possible.
- Integrate diverse views, needs and issues (not just those of predominantly highly-educated, middle-class males in urban centres), co-design principles with active participation from all users are essential.

See also:

Permacomputing research fields and methods;

115. Reuse

Reuse means the use of a piece of technology for new purposes, often ones that were never envisioned by the original designers. **Reusability** refers to design that makes reuse possible.

In **hardware**, reusability is of utmost importance. In order to maximize component longevity, it should be possible to reappropriate them to different purposes.

In **software**, the question of reuse and reusability is more complicated. Software can be constructed and discarded without waste (it's just patterns of electrons), so it can't be compared to hardware in this case. You don't need to feel sorry for programming something redo from scratch, because the replaceability of software is what computers are all about.

Excessive reuse easily leads to bloat and multiple layers of legacy, as in case of Unix. Monoculture may also appear if a lot of programming is framed as the reuse of a single silver bullet. Too much concern for reusability may also make the piece of software unnecessarily big and complex.

Minimal reuse, on the other hand, may easily lead to "Not Invented Here" kinds of problems.

The reuse of **ideas** is less problematic than the reuse of software. Wisdom and ideas can be accumulated and refined without the risk of bloat or sedimentation.

Keeping ideas simple makes it easier to implement and reuse them. Having a rich layer of different ideas at the bottom of the dependency pyramid may also help with creating technological diversity.

116. Technological diversity

Technological diversity is the opposite of technological monoculture. It refers to the diversity of artifacts (software and hardware) as well as that of ideas, philosophies, paradigms, techniques, languages, mental models, etc.

Diversity comes from the mutually strengthening co-existence of different kinds of things, including ones that are generally considered “obsolete”, “useless” or “fringe”.

Understanding of history (and not only the mainstream history written by the “winners”) helps grasp the spectrum of possibilities and see beyond what is currently fashionable in the mainstream.

The purpose of **standards** is to facilitate co-existence and co-operation of different pieces of technology, not to limit their diversity. So, they are more like common languages than norms. Linguistic diversity is richness, but it is still good for everyone to master a couple of common world languages. (Monolingualism or a single prestige language, however, easily leads to impoverished monoculture.)

Gut reactions against technological diversity are often based on mental models where competition, obsolescence and narrowly defined efficiency are key concepts. Models where the default purpose of something like a new programming language is competition against pre-existing languages with the goal of making them obsolete.

Diversity is particularly needed on the low levels of the pyramid, the elementary technologies and ideas that everything else is built on. A wide variety of different ways to build computers and design systems may help make the entire pyramid more resilient.

See also:

- unconventional computing
- ethnomathematics

117. Unconventional computing

Unconventional computing, also known as **alternative computing**, refers to computing with unusual methods. An unusual method may be e.g. an unusual theoretical model or an unusual physical basis. The term “unconventional computing” was coined in 1998.

Permacomputing is interested in expanding the lowest layers of the technological possibility space, especially in order to develop computer technology that better integrates with natural processes. This, along with the strive for a greater

technological diversity, makes unconventional computing techniques interesting from the permacomputing point of view.

Fluidics, using liquids or gases in place of electricity, is probably the most mature alternative technological basis for computing. A fully working fluidic digital computer, Flodac, was already built in 1964. Its performance class was similar to relay computers (tens of cycles per second), but it was mentioned in the paper that clock speeds up to 250 Hz could be reached with similar but more compact circuitry.

Flodac-like logic gates are based on how fluids move within static structures. So, unlike mechanical and relay computers, Flodac had no moving parts. These structures could probably be printed with rather rudimantary etching techniques or 3D printing, unlike semiconductor manufacturing that requires extreme purity of the material and the etching process.

Optical computing is also quite mature. It has often been envisioned as a way to stick to the Moore's law after the limits of silicon microchips have been reached. The material constraints are different to those of semiconductors, which may also make it an interesting option from the DIY point of view.

Many unconventional computing technologies such as DNA computing are still at a very early stage of development (as in "the addition of two small numbers was succesfully demonstrated"). And even those who aren't are often unsuitable for conventional digital computers.

Some unconventional computing techniques use living organisms. The use of Physarum slime molds has been studied for a long time, and they can e.g. solve shortest-path problems.

Quantum computing is probably the most hyped type of unconventional computing because of the ability of a quantum computer to do an operation "in millions of parallel universes at a time".

Today, integrated circuits are so dominant that even historically important component technologies can be considered "unconventional". These include fully mechanical parts (like those in mechanical calculators or the Zuse Z1), relays (Z3), electron tubes (most 1950s computers), discrete transistors (most early-to-mid-1960s computers), parametrons (some Japanese computers mostly from the 1950s) and Symmag (the French computer CAB500 from 1957).

See also:

- FLODAC - A pure fluid digital computer (Gluskin&al, 1964, PDF)

118. Calculation factory

A **calculation factory** is a computer that works like a factory: it has a relatively static material and energy requirements with very little concern or flexibility in regards to the environment they are part of.

Mainframe computers in particular have been thought as factories since the 1950s, including the idea of minimizing their idle time by having their operators work in multiple shifts. This kind of thinking carried over to the Internet server world.

Even small computer systems tend to inherit some characteristics from calculation factories. They often have very poor means to adjust their operation to the changes in physical conditions. This kind of environment-reactivity is an area permacomputing is interested to develop.

119. Californian ideology

Californian ideology refers to the dominant ideology of Silicon Valley, as described by the British media theorists Barbrook and Cameron in their 1995 essay "The Californian Ideology". The ideology is very influential because of the ubiquity of Californian technology products, as well as because siliconization has made it prominent in computing industries all around the world.

Californian ideology combines growth-oriented and individualistic neoliberalism with elements from the U.S. counterculture of the 1960s-1970s. It embraces ideas such as cyber-utopianism, techno-determinism, transhumanism and maximalism. Notably, it often attempts to constitute holistic world views where computing (as well as economic growth) has an essential role down to the metaphysical level.

From the permacomputing point of view, Californian ideology can be regarded as a kind of anti-example – how not to build a holistic world view around computing and high technology. While some of its root ideas are good (including the use of computers for self-improvement), they have been banalized by the growth-obsessive techno-capitalism.

External links:

- The 2007 revised version of the original essay

120. Cornucopianism

Cornucopianism refers to **cornucopia**, "the horn of plenty". A cornucopian is someone who posits that there are few intractable natural limits to growth and believes the world can provide a practically limitless abundance of natural resources.

121. Greenwashing

Greenwashing refers to false claims that a product or activity is environmentally friendly, especially when the claims are made by corporate PR and marketing divisions. Unintentional greenwashing may take place when an unjustified belief of environmental friendliness is actively maintained by well-intended people.

Permacomputing tries to base itself on physical reality rather than beliefs or positive social signals. Therefore, it is important to avoid being affected by greenwashing, be that intentional or unintentional. Claims should be questioned, and all the effects of every dependency should be accounted for. A lot of problems are related to buying new things (sometimes even if those things are solar panels), so a lot of greenwashing checks should be made before making purchases or advocating a hardware product.

122. Maximalism

Maximalism is, generally speaking, the idea that more is better. It often incorporates the idea that qualitative improvements are dependent on quantitative increases – that "progress" is not even possible without quantitative growth. Maximalism is prominent in many areas of modern technological civilization but is particularly so in computing, thanks to Moore's law.

Maximalist aesthetics strives for the maximization of resolution and detail. This leads to increased screen sizes and growing bandwidth requirements.

Maximalism leads to an increasing use of artificial energy and other limited resources, even when the energy efficiency is dramatically improved (see Jevons paradox). This is extremely unsustainable, so permacomputing prefers to take an anti-maximalist, "small is beautiful" type of stance that emphasizes qualitative improvements and technological diversity.

Extropianism, as advocated by the Californian ideology philosopher Max More (sic), is an extreme philosophical stance based on maximalist ideals. It dreams of things such as human immortality, quantitative intelligence maximization and an endless expansion of the maximization-oriented "civilization" into the outer space.

123. Postdigital

Postdigital or **post-digital** is a somewhat misleading concept but may be useful for understanding some phenomena in today's world.

The "digital" in "postdigital" is to be read a bit like the "modern" in "postmodern": it refers to a very specific kind of "digital", a highly technoprogressivist and industry-defined kind that became prominent in the "digital revolution" hype of the 1990s. This

is also why even some profoundly digital things such as pixel art and digital glitches have been labelled “postdigital” in artistic circles.

When distancing itself from this kind of “mainstream digitality”, “Postdigital” also distances from many of the destructive ideas whose abolishment is also relevant to permacomputing (maximalism, virtualism, obsolescence, etc).

“Postdigital” is also a reaction to the digital oversaturation. Computers are no longer “inherently cool” (like many people born in the 1970s and 1980s found them), but now the same kind of coolness can be found in non-digital things such as physical and analog artforms – or things that are digital in alternative and countercultural ways (like permacomputing is).

124. Retro

Retro is a Latin word meaning “backwards” and “before”. In computing, it generally refers to a kind of “time-capsule” computing that tries to re-enact a historical time period when a hardware platform was “still alive”. Its central driving force is nostalgia.

The concept is problematic from the permacomputing point of view because:

- It affirms the industrial definition of “platform death” and that there can be no genuinely new uses for a platform when it is “dead”.
- It separates the current time period from the “old times”, thus creating an artificial mental boundary.
- While historical re-enactment and time capsules have their definite places and hardware obsolescence).

The concept of **Zombie media** has a similar problem with affirming the industry-defined concept of media “death”.

Heirloom computing is another form of time-capsule computing but one that designs a static artifact that future generations may return to.

Examples of non-retro uses of old computers can be found in the demoscene and hacker cultures.

125. Siliconization

Siliconization refers to how local technocultural practices all around the world have been replaced by an imported “Silicon Valley” model, especially since the history.

This phenomenon was particularly prominent in Eastern European countries after the fall of the USSR. The term was originally used in Romania to refer to how their local

practices (“șmecherie”) were obsoleted at this time. It is also a near-anagram of “colonization”.

Similar developments also took place in many other countries at this time. In Western Europe, this era is often connected to the marginalization of the earlier home computer cultures by a “Wintel” monoculture and the normalization of constant hardware “up”grades.

The spread of Californian ideology and the technology startup culture are closely connected with siliconization.

In order to understand the effects of siliconization it is important to remember and study the pre-siliconization computer cultures – particularly regarding how the mindsets and values differed from those dominant today.

External links:

- Corruption, Șmecherie, and Siliconization: Retrospective and Speculative Technoculture in Postsocialist Romania

126. Wishcycling

Wishcycling is when people place non-recyclable items in the recycling and hope those items will end up being recycled.

127. BBS

Bulletin Board System

A **Bulletin Board System** (**BBS**) is an online community service based on character terminal connections – traditionally over landline telephone modems, but currently most BBSes use Telnet or SSH over the Internet instead. There are also packet radio (AX.25) BBSes used by radio hams.

BBSes came to existence in the 1970s but were preceded by time-sharing services of the 1960s. A major difference between a BBS and a time-sharing server (such as a public Unix system) is that BBS software is usually user-friendlier and specifically designed for messaging and file-sharing. Also, BBSes have generally been run on much smaller computers, usually microcomputers with a single modem. Multi-user BBSes were not uncommon in the “golden years”, but most BBSes only allowed only one online user at a time. In many places, BBSes declined quite rapidly especially when broadband Internet connections became available.

Some notable features of the BBS culture from the permacomputing perspective, especially in contrast to the Internet:

- Locality. Since long-distance telephone calls are more expensive than local calls, most modem users have preferred servers that are geographically close to them. This helped create local variation to the BBS scene and prevent excessive centralization.
- Personality. BBSes generally look and feel like their owners, and visiting a BBS feels more like visiting someone's home than a public service. On the Internet, small messaging forums and Smallnet may sometimes create the same kind of feeling.
- Limited availability. There used to be much more BBS users than BBS nodes, so it was quite common to get a BUSY signal instead of a connection. Automatic dialling lists that repetitively called several BBS numbers until a connetion was made were common. Also, BBSes could become temporarily or permanently unavailable due to various personal reasons. So, BBS users generally felt joy on succesful connections, something that does not happen on the Internet where everything is expected to be constantly available at all times.
- Intensity. When online time is limited (and possible even charged per minute), it needs to be used actively instead of "just idling around". Non-multitasking operating systems amplify this. Offline is the default state of mind, and online is an exception.
- Scarcity and sharing. When connection time is a limited resource, users are generally expected to contribute to the systems by message-based discussions and/or uploads instead of just passively "leeching" files. Most BBSes implemented an upload/download ratio system where files were used as "currency".
- File collection. BBSes typically host all kinds of software and other files useful for computer users. Also, before the broadband Internet, it was not feasible to redownload the same files every time they were needed, so it was also important to curate local, personal file collections.
- Decentralization. Because of the file collection culture, nothing that is published in one place stays in that place. For message forums, there are networking systems (particularly Fidonet) for creating shared forums between several BBSes, although system-specific ("local") messaging cultures are also considered important.

128. Pixel art

Pixel art is the practice of producing bitmap images by consciously deciding the position of each individual pixel. Pixel art images often have bandwidth minimization because of preference for small resolutions and limited color palettes. In modern computers, pixels tend to be very small, so pixel art is usually upscaled to a larger

resolution. This also makes modern pixel art less dependent on a specific physical resolution.

Pixel art has a long prehistory that makes it centuries or millennia older than computers (textile techniques such as cross-stitch can be considered pixel art). Despite this, it is very “digital”, with each pixel directly corresponding to a group of bits in the storage. The simplicity of the concept also makes it easier to learn than e.g. 3D modelling or even 2D vector art.

From the permacomputing point of view, pixel art can be a good choice because of its low system requirements, long tradition and aesthetics anti-maximality. Alternatives that are worth considering include low-complexity vector art and generative art based on short computer programs.

129. viznut

http://viznut.fi

130. thgie

Hi, I’m Adrian. Find me here:

- https://post.lurk.org/@thgie
- https://thgie.ch

131. aw

Hello world

132. wakame

Hi, I’m wakame.

https://tech.lgbt/@wakame

133. decentral1se

🤸 permanent gezelligheid 🤸

moar @ my webshit

134. Chip8

CHIP-8 is a virtual machine created by RCA engineer Joe Weisbecker in 1977 for the COSMAC VIP microcomputer. The Chip-8 language is capable of accessing up to 4KB(4096 bytes) of RAM, from location 0x000 to 0xFFF(0-4095). The first 512 bytes, from 0x000 to 0x1FF, are where the original interpreter was located, and should not be used by programs.

The original implementation of the Chip-8 language includes 36 different instructions, including math, graphics, and flow control functions.

All instructions are 2 bytes long and are stored most-significant-byte first. In memory, the first byte of each instruction should be located at an even addresses. If a program includes sprite data, it should be padded so any instructions following it will be properly situated in RAM.

The computers which originally used the Chip-8 Language had a 16-key hexadecimal keypad.

135. Rust

Rust is a programming language created in 2010. Rust is a statically typed, multi-paradigm general purpose language developed by Mozilla. It has applications for both “high level” programming tasks (such as web applications) as well as low-level systems programming code. The first stable release was in 2014. Rust has a number of advantages over C:

- Memory safety
- Modern semantics and language features

Rust has a number of disadvantages from a permacomputing perspective:

- Currently only has a single fully-featured implementation
- Limited platform support
- Highly complex language and toolchain requires a lot of computing resources, unsuitable to run on many platforms
- Largely dependent on Big Tech for resources and funding

136. Redo from scratch

Redo from scratch is an idea that is antithetical to reuse. Instead of modifying an existing program to fit a new purpose, a completely new program is written from scratch.

The philosophy of Chuck Moore, the author of Forth, has very strong RFS elements. Instead of deciding a set of standard pieces to build software from, Moore was ready to rethink just about any previous decision. What came to be the time-tested core of the Forth language was based on constant and obsessive rethinking, experimentation and redoing-from-scratch.

In practical purposes, RFS often has huge risks and problems, starting from the bugginess typical of new programs. However, it can be heartily recommended as an artistic or educational practice, as a way to exercise one's programming skills, or as a research method for experimenting with completely new ideas.

137. IBM PC

IBM PC is a computer released in 1981 by IBM, as well as an entire series of computers compatible with it. It is still the dominant computer platform in desktops, laptops and servers, even though (and maybe because) IBM lost its grip of it by the 1990s. Modern x86 processors remain software-compatible with the Intel 8088 processor of the original IBM PC, even though it is no longer possible to run all the software natively. The operating system of the original IBM PC was Microsoft's DOS (PC-DOS, MS-DOS) which was surpassed in popularity by Microsoft Windows in the 1990s.

The ubiquity of IBM PC compatibles as well as the lack of a single corporation that controls the platform makes it a good candidate for a bedrock platform.

The **Wintel** platform (consisting of IBM PC compatible hardware and a Windows operating system) represented a technological monoculture especially in the early 2000s, when non-x86 computers had been marginalized, ARM-based mobile computers were not yet common, and most applications weren't yet being targeted for web browsers.

Sometimes, "PC" is used as a shorthand for "IBM PC compatible", while sometimes it refers to a "personal computer" in general. This may lead to misunderstandings and misconceptions, such as regarding the original IBM PC more revolutionary than it actually was in the early years.

138. Raspberry Pi

Raspberry Pi is a popular family of inexpensive single-board computers. They are often used in electronics/computing hobbyist projects including ones whose goals somewhat align with those of permacomputing.

While the Raspberry Pi is often among the best alternatives for many purposes, a particular problem with it is that it is designed for disposability rather than longevity. The cheapness and smallness of the boards may also be deceptive and make it too appealing to purchase new ones.

Chips

Most of the models are based on Broadcom SoC chips that are insufficiently documented, even though the large and active user base somewhat compensates on the problem of closed hardware.

An exception is the Pico whose microcontroller chip (RP2040) was designed by the foundation itself and has apparently a rather complete register-level documentation. An interesting feature of RP2040 is its programmable IO (PIO) that is general-purpose rather than tied to specific protocols and interfaces, while being powerful enough to e.g. generate video signals.

Alternatives

There are many different single-board computers, some of which based on fully documented or even open-source hardware. See single-board computer.

139. ikiwiki

!meta robots="noindex, follow" This wiki is powered by ikiwiki. [[!if test="enabled(version)" then="(Currently running version !version .)"]]

What made us choose this wiki software:

- It stores the pages as flat files that can be fully managed via version control (Git) and can therefore be easily copied. (No hard dependency on WWW or a specific server)
- Ikiwiki compiles static html pages out of markdown and can be used to do this even without a web server.
- Relatively light-weight. (Although it depends on Perl and a bunch of Perl modules)

Some documentation on using ikiwiki:

- ikiwiki/formatting
- ikiwiki/wikilink
- ikiwiki/subpage
- ikiwiki/pagespec
- ikiwiki/directive
- ikiwiki/markdown
- ikiwiki/openid
- ikiwiki/searching
- templates

140. Cloning

This wiki is meant to be cloned so that you can consult it while being offline. There are several ways to do it depending on the system you use and the tools you have access to.

You have access to git

Cloning

It's as simple as:

```
git clone https://git.bleu255.com/repos/permacomputing.git
```

Markdown source files

Once you have done this you can:

- open the wiki pages in their native plaintext forms as markdown files;
- search through the wiki with plaintext file search tools like grep, rg or equivalent on your OS;
- potentially navigate from pages to pages with the help of plugins in some text editors and or pagers that support markdown files.

HTML rendering

With the markdown source files you can also easily turn them into a local HTML static web site for offline browsing. To do that you need to install ikiwiki.

```
ikiwiki --rebuild --wikiname permacomputing --verbose /path/git/
permacomputing /path/permacomputing-html
```

But that's not all!

If you call ikiwiki directly to generate HTML files to browse locally you will be disapointed as ikiwiki links points to folder in which an index.html will be found. This is fine for an HTTP server as most will default sending the index.html to the browser upon folder URL request. However, if you browse locally, the web browser will display the content of the folder instead. Making navigation super annoying. This script to be run everytime a new static version is built will fix all the href to actually point to the index.html instead of pointing to the folder.

```
/path/git/permacomputing/_tools/fix_local_href.sh /path/
permacomputing-html
```

Now you can point your browser to `/path/permacomputing-html/index.html` and enjoy read only offline access to the wiki.

You don't have access to git

If you do not have access to `git` on your operating system, you can download a `zip` file that contains both the markdown source files and the generated HTML files, with the paths fixed. The `zip` file is generated once a week.

```
# pick your fav download tool...
wget http(s)://permacomputing.net/permacomputing.net.zip
fetch http(s)://permacomputing.net/permacomputing.net.zip
curl -O http(s)://permacomputing.net/permacomputing.net.zip
```

141. Installing

These install notes make the following assumptions: Debian OS, nginx, fcgi-wrap, stagit. Also these are notes, not all the steps are provided, such as restarting nginx, enabling https, etc.

Base installation

```
sudo apt install ikiwiki
sudo mkdir /var/www/damaged.bleu255.com-src /var/www/
```

```
damaged.bleu255.com
sudo chown $USER:$USER /var/www/damaged.bleu255.com*
echo "hello" > /var/www/damaged.bleu255.com-src/index.mdwn
mkdir ~/ikiwiki-cfg
ikiwiki --verbose /var/www/damaged.bleu255.com-src /var/www/
damaged.bleu255.com --url=https://damaged.bleu255.com --dumpsetup
~/ikiwiki-cfg/damaged.setup
ikiwiki --setup ~/ikiwiki-cfg/damaged.setup
```

Configuration changes

IMPORTANT, any changes to the wiki configuration must be followed by this to reflect changes:

```
ikiwiki --setup ~/ikiwiki-cfg/damaged.setup
```

Enable CGI

```
# ~/ikiwiki-cfg/damaged.setup
cgiurl: 'https://damaged.bleu255.com/ikiwiki.cgi'
cgi_wrapper: '/var/www/damaged.bleu255.com/ikiwiki.cgi'

# /etc/nginx/sites-available/damages.bleu255.com
server {
  listen 443;

  server_name damaged.bleu255.com;

  root /var/www/damaged.bleu255.com;
  index index.html index.htm;
  access_log /var/log/nginx/damaged.bleu255.com-access.log;
  error_log /var/log/nginx/damaged.bleu255.com-error.log;

  location / {
    try_files $uri $uri/ =404;
  }

  # Max size of file upload
  client_max_body_size 10m;

  location /ikiwiki.cgi {
    gzip off;
    fastcgi_pass unix:/var/run/fcgiwrap.socket;
```

```
    fastcgi_index ikiwiki.cgi;
    fastcgi_param SCRIPT_FILENAME /var/www/damaged.bleu255.com/
ikiwiki.cgi;
    fastcgi_param DOCUMENT_ROOT /var/www/damaged.bleu255.com/;
    include /etc/nginx/fastcgi_params;
  }
}
```

Enable git

```
ikiwiki-makerepo git /var/www/damaged.bleu255.com-src /var/www/
git.bleu255.com/repos/damaged.bleu255.com.git

# ~/ikiwiki-cfg/damaged.setup
rcs: 'git'

ikiwiki --changesetup ~/ikiwiki-cfg/damaged.setup

# ~/ikiwiki-cfg/damaged.setup
git_wrapper: /var/www/git.bleu255.com/repos/
damaged.bleu255.com.git/hooks/post-update
git_wrapper_background_command: git push

ikiwiki --setup ~/ikiwiki-cfg/damaged.setup
cd /var/www/damaged.bleu255.com-src
git config pull.rebase false
```

stagit hooks and stagit diff URLs

```
cd /var/www/git.bleu255.com/repos/damaged.bleu255.com.git/hooks
ln -s ../../../update_single.sh post-receive

# ~/ikiwiki-cfg/damaged.setup
diffurl: https://git.bleu255.com/damaged.bleu255.com/commit/\[\
[sha1_commit\]\].html
```

Limited web access

There are 2 ways to deal with user:pass for web access (if we rule out external auth like openid):

- outsource it to httpd-auth, implies that someone has to maintain the list of users and their passwd manually
- or add a password prompt during account creation, something we'd share with only trusted people, or people showing interest, etc. We choose this approach

  # ~/ikiwiki-cfg/damaged.setup account_creation_password: pa55w0rd ikiwiki –setup ~/ikiwiki-cfg/damaged.setup

Support for sending emails

Useful to reset passwords, etc.

```
sudo apt install libmail-sendmail-perl

# ~/ikiwiki-cfg/damaged.setup
adminemail: yolo29383@hotmail.com

ikiwiki --setup ~/ikiwiki-cfg/damaged.setup
```

Stronger passwor hashes

```
sudo apt install libauthen-passphrase-perl

# ~/ikiwiki-cfg/damaged.setup
password_cost: 16
```

Disable OpenID and emailauth

```
ikiwiki --setup ikiwiki-cfg/damaged.setup --disable-plugin openid
ikiwiki --setup ikiwiki-cfg/damaged.setup --disable-plugin
emailauth

# ~/ikiwiki-cfg/damaged.setup
disable_plugins: [emailauth, openid]
```

Default git commit message when none provided

Ikiwiki and a recent enough version of git allow for empty git messages (the “Optional description of this change” while editing on the web), that’s nice but it makes stagit history impossible to browse because it uses such messages as links. To make ikiwiki provide a default commit message when non given, you can do this:

```
diff --git a/git.pm.old b/git.pm
index 2bc2500..2198207 100644
--- a/git.pm.old
+++ b/git.pm
@@ -680,21 +680,8 @@ sub rcs_commit_helper (@) {
        $params{message} =
IkiWiki::possibly_foolish_untaint($params{message});
        my @opts;
        if ($params{message} !~ /\S/) {
-               # Force git to allow empty commit messages.
-               # (If this version of git supports it.)
-               my ($version)=`git --version` =~ /git version
(.*)/;
-               if ($version ge "1.7.8") {
-                       push @opts, "--allow-empty-message", "--
no-edit";
-               }
-               if ($version ge "1.7.2") {
-                       push @opts, "--allow-empty-message";
-               }
-               elsif ($version ge "1.5.4") {
-                       push @opts, '--cleanup=verbatim';
-               }
-               else {
-                       $params{message}.=".";
-               }
+               # Force a message to commit if none given.
+               $params{message}.="empty web commit";
        }
        if (exists $params{file}) {
                push @opts, '--', $params{file};
```

142. wiki

A wiki is a user-editable website. The concept was developed initially by Ward Cunningham and the first implementation was used for the 1995 WikiWikiWeb site.

> The program wants everyone to be an author. So, the program slants in favor of authors at some inconvenience to readers[6]

Wikis have been instrumental in making use of hypertextuality and online web editing through early programmable http interfaces, to form ever expanding collaborative writing resources, or personal knowledge repositories. Today they come in many shapes from minimal WikiWikiWeb clones, to variations on the same principles, to much more complex and bureaucratic editorial platforms like Wikipedia.

143. History

History refers to the study, documentation and narrating of the past. Knowing and understanding of the past is essential when envisioning, planning and building the future. The way how history is told affects the ways in which future can be imagined.

Since permacomputing envisions a long-term future of the computing that, it needs to tell the **history of computing** in ways that make permacomputing and similar alternative ways of thinking more relevant.

Problems of mainstream computing history

A lot of things get eliminated from the mainstream narrative for various reasons:

- It is essentially "winners' history":
  - The developments in the US are overemphasized in comparison to what happened in the rest of the world, sometimes even eliminating prior art: Vannevar Bush's "Memex" vision was not that original (see Paul Otlet), Douglas Engelbart was not the first one to invent a mouse (see Telefunken's Rollkugel), etc. etc.
  - The history of computer networking is told in ways that eliminates non-Internet networks, some of which were still quite prominent in the 1980s. BBSes are sometimes mentioned as a side curiosity because they are part of the "consumer history", but what about BITNET, DECnet, etc.? Even Minitel is scarcely mentioned even though it had millions of users already in the 1980s, maybe because it was French and therefore irrelevant.
  - The history of personal computing very much centers around a Californian ideology where the young enterpreneurs (such as Steve Jobs) were the heroes who liberated the world from the evil mainframe culture. This is sometimes intertwined with a more general "hacker mythos" even though its approach to liberation was often largely non-commercial.
- It is also very much "consumer history" especially from the 1980s onward. Consumer-grade hardware and their applications (especially games) get a lot of love, and even a lot of obscure platforms from small countries are documented.

However, it is often very difficult to even find mentions of prominent institutional or scientific projects, strange non-US hobbyist subcultures, etc.
- There have been conscious attempts to make earlier developments irrelevant or obsolete, especially in Internet history:
    - The “Internet years” idea in the late 1990s (“one year in cyberspace is equivalent to ten years in meatspace” etc.) was perhaps invented because researchers did not want to do their homework. The pre-WWW Internet was so long ago in Internet years that it was in a different era that didn’t need to be studied.
    - “Social media” was defined in a way that made it possible to start its history from the 2000s (again, eliminating BBSes, Usenet, IRC, etc.)
    - In general, each new user generation wants to pretend it invented more things than it actually did.
- Local histories are understudied, especially those of non-Western countries. The same applies to minorities, women, lower social classes and many other group that don’t get to be as loud as the affluent white Californian males.

Problems in how the story is told:

- The overarching story is that of economic growth and maximization. Hardware systems are divided into “generations” (that may sometimes be only a few years in length) that are intended to obsolete the previous generation. Big companies and their business “achievements” (such as the establishment of monocultures) get a lot of praise.
- This “chain of obsolescence” narrows the technological history down to a one-dimensional “highway of progress” where there are only two possible directions (“forward” and “backward”). This makes it difficult to envision other directions and represent them in ways that don’t sound “backward”.
- The concept of “retro” is used to separate some parts of history and technology into a different world that is only relevant to personal memories. This world is also the place for “backwards” ways of thinking (such as the appreciation of small and efficient program code that can’t be justified from business perspectives, or the acknowledgement of the benefits of earlier communications systems in comparison to modern social media).

Ideas and examples

- **Siliconization** is a concept that is used in Romania to refer to how their local technocultural practices (“șmecherie”) were replaced by an imported “silicon valley” model in the 1990s.
- Eriksson and Pargman have suggested the use of **counterfactual history** as a tool to imagine computing futures. It is often difficult for students and other people to even imagine a computing world that is not built around Moore’s law,

so for example imagining how computing might have evolved in a low-coal world can be helpful at making this kind of conceptual leap.

144. Universal Virtual Computer

The specification of an hypothetical computer designed to decode and preserve some form of computation over time.

Candidates

- Lambda calculus is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution. See also, Church Numerals.

145. Intelligence amplification

Computers (and many of their predecessors such as mechanical calculators and tabulating machines) were invented to assist humans in cognitive tasks such as calculation and data processing. **Intelligence amplification** takes place when computer interaction assists the human user in conceptual thinking by e.g. improving access to information. A lot of today's common computer applications (including the WWW) can be regarded as IA, even though the concept became unfashionable in the 1980s.

For permacomputing purposes, IA can be regarded as a subset of awareness amplification.

The use of punched cards and mechanical devices for "enhancing natural intelligence" was already suggested in 1832 by Semyon Korsakov and in the 1910s by Wilhem Ostwald. In these suggestions, each punch card would contain an idea or a "micro-thought", and a mechanical device would assist in finding and connecting them. Emanuel Goldberg (1930) and Vannevar Bush (1945) combined this concept with a microfiche viewer/searcher.

In Douglas Engelbart's Augmentation Research Center, these ideas evolved into a user interface prototype that supported a form of human-computer symbiosis. Hypertext and many modern GUI concepts originate from Engelbart's project.

146. Research fields and methods

Research methods

Livinglabs

Sustainability research method, experimental/development environment and ecosystem.

Local initiatives, pragmatic and efficient, balancing social, environmental and economical considerations. Livinglabs are places to overcome narrow scientific or engineering specialization and theoretical assumptions and test consequences of technology next to living ecosystems. All actors - human, animal, plant or microbiome and researchers are on the same level to resolve positive or negative impacts. Instead observation of subjects there is principle of co-creation, experiential learning, prototyping involving communities, inclusive social space for designing and experiencing their own future.

Artistic research

Creative inquiry, practice as research.

Practice led research is a distinctive feature of the research activity conducted by arts and humanities researchers, it involves the identification of research questions and problems, but the research methods, contexts and outputs then involve a significant focus on creative practice. This type of research thus aims, through creativity and practice, to illuminate or bring about new knowledge and understanding, and it results in outputs that may not be text-based, but rather a performance.

Scientific critique, interdisciplinarity

Interdisciplinarity means not only multiple disciplines involved (informatics, biology…), but also multiple domains of inquiry.

- *Interpretivist:* meaning, constructivist, network, dialogue, interdiscipline
- *Empiricist:* exploratory, conceptual, reflective, discipline-based
- *Critical:* positionality-change, contextual, transdiscipline, perspective, question
- *Art practice:* meta-theoretical, practical, reflexive, post-discipline, visual systems

Art practice also come in coupled with critical, interpretivist and empiricist domains in theory dimensions:

- Create-critique, Meaning-making, Enact-explain

Research fields

Meta-disciplines of trans-discipline of sustainability:

Ecosystems and computational conditions of biodiversity

Sustainability and toxicity of computation

Biodigitality and bioelectric energy

Table of Contents

32. Gemini
33. Chifir
34. Uxn
35. Civboot
36. Mu
37. Teliva
38. DawnOS
39. Solar Protocol
40. PADI
41. BBC Domesday Project
42. CARDIAC
43. Hardware
44. Peripherals
45. Software
46. Programming languages
47. Operating systems
48. Protocols
49. File formats
50. Concepts
51. Collapse computing
52. Smallnet
53. Computing within Limits
54. Offline first
55. Right to repair
56. Holistic Computing Arts
57. Small File Media Festival
58. Algorave
59. Demoscene
60. Solarpunk
61. Contact
62. neau
63. ugrnm
64. ola
65. Bandwidth minimization
66. Media minimization
67. Virtual machine
68. C
69. Forth
70. Obsolescence
71. Lifespan maximization
72. Documentation
73. Feminist server
74. Operating system

75. Bloat
76. Character terminal
77. Bedrock platform
78. Plan 9
79. Silver bullet
80. Monoculture
81. Permacomputing 2020
82. Cryptocurrency
83. File collection
84. Information battery
85. About
86. Editing
87. Dusk OS
88. WWW
89. Web browser
90. Internet
91. Paper computer
92. DRM
93. IC
94. Lua
95. Smalltalk
96. Go
97. TODO
98. DOS
99. IRC
100. iff
101. Digital preservation
102. Emulation
103. Emotionally durable design
104. Personalities
105. Aesthetics
106. Artificial intelligence
107. Automation
108. Awareness amplification
109. Kolmogorov complexity
110. Ethnomathematics
111. Games
112. Information and energy
113. Permatechnology
114. Regenerativity
115. Reuse
116. Technological diversity
117. Unconventional computing

118. Calculation factory
119. Californian ideology
120. Cornucopianism
121. Greenwashing
122. Maximalism
123. Postdigital
124. Retro
125. Siliconization
126. Wishcycling
127. BBS
128. Pixel art
129. viznut
130. thgie
131. aw
132. wakame
133. decentral1se
134. Chip8
135. Rust
136. Redo from scratch
137. IBM PC
138. Raspberry Pi
139. ikiwiki
140. Cloning
141. Installing
142. wiki
143. History
144. Universal Virtual Computer
145. Intelligence amplification
146. Research fields and methods

About

This publication comes from the Permacomputing wiki https://permacomputing.net, 19 September 2024.

The order of appearance of the pages unfolds from the index page, recursing to the entire wiki. Script by Delisa Fuller:

```
# run from the root of the repository
from os import listdir, curdir, path
import re

# find each wiki link, which is formatted as `[[page_name]]` or
```

```
# `[[some text|page_name]]`
# https://permacomputing.net/ikiwiki/wikilink/
# capture groups:
# 0 - link text (if any)
# 1 - page name
# 2 - anchor name (if any)
INTERNAL_LINK_PATTERN = re.compile(r'\[\[(.+\|)?(.+)(#.+)?\]\]')
# identify all pages in the wiki, which are some combination of
alphanumeric
# and underscores, ending in ".md" or ".mdwn"
PAGE_PATTERN = re.compile(r'^\w+\.md(wn)?$')
# all files in the wiki
ALL_PAGES = [f for f in listdir(curdir) if path.isfile(f) and
PAGE_PATTERN.match(f)]

# match link text to a page filename
#
# >>> find_page("Getting Started")
# 'getting_started.mdwn'
def find_page(link: str) -> str | None:
    link = re.sub(r'\W', '_', link)
    return next((f for f in ALL_PAGES if
f.lower().startswith(link.lower() + ')), None)

# list of files traversed, appended to as new files are
encountered
pages = ['index.mdwn']
# current filename to read from pages
index = 0

while index < len(pages):
    filename = pages[index]
    print(filename)
    try:
        with open(filename, 'r') as page:
            for link in
INTERNAL_LINK_PATTERN.findall(page.read()):
                page = find_page(link[1])
                if page is not None and page not in pages:
                    pages.append(page)
    except Exception:
        pass
    index += 1
```

1. parenthesis could be used as well, sure, or long — em dashes, but if you're going to fork the discussion to something that's too long to fit in the flow of the main text, and that does not need its own page, then a footnote can be quite handy.↩

2. See Ursula K. Le Guin, "A Rant About 'Technology'," 2004, http://www.ursulakleguinarchive.com/Note-Technology.html.↩

3. Ursula K. Le Guin, "A Rant About 'Technology'," 2004, http://www.ursulakleguinarchive.com/Note-Technology.html.↩

4. Ursula K. Le Guin, "A Rant About 'Technology'," 2004, http://www.ursulakleguinarchive.com/Note-Technology.html.↩

5. Digital Reset Report, TU Berlin, 2022;↩

6. Ward Cunningham, "MoreAboutMechanics," 1996, https://web.archive.org/web/19961129192942/http://c2.com/cgi/wiki?MoreAboutMechanics↩