Welcome !

This document summarizes featured projects and provides links to more information, including source code.

Cryptography

THORIUM / CESIUM

This symmetric cryptosystem uses a square matrix of bits as a key. The rows and columns are "circular shifted," so that the matrix is topologically a torus (not exactly but *somewhat* like Rubik's Cube.) Cesium is generalization of Thorium to an arbitrary base. It's easy with Cesium to just use base 27 (the alphabet and an underscore), so that one can exchange messages right away in English. I hope to write up a version in Javascript so that I can provide the encoder on a static website for actual use. I teach math, and I'd like to give my students an example of a complex but still invertible function.

`https://k0ntinuum.github.io/th0rium/`

CONE

This symmetric cryptosystem is based on a triangular implementation of a ternary elementary cellular automata. The key is 27 ternary symbols which represent a rule for a ternary cellular automaton with a neighborhood size of 3. The rule is then also used as a seed (along with a symbol of plaintext) in order to get the ciphertext symbol.

`https://k0ntinuum.github.io/c0ne/`

FORTEX

This system, like Cone, features a triangular key. Like Cesium, the base used is adjustable. Rows of the triangle are rotated, and a central column functions like an escalator (all the shifts are circular). The modular sum of elements in this central column is added to plaintext symbols, and the apex of the pyramid is used to represent this changing value in console "textgraphics" demonstrations.

`https://k0ntinuum.github.io/f0rtex/`

PRE

Pre or Prefix is a symmetric cryptosystem featuring prefix or instantaneous codes. One input symbol might become several output symbols. One output symbol might condense several input symbols. Each state of the machine writes a prefix code, but the union of prefix codes (the total output of the machine) is not generally a prefix code. This means eavesdroppers will have trouble even tokenized the ciphertext to begin with. The reading or input mode of the machine uses prefixes in a different way, checking for multi-symbol inputs to compress in a first-come first-serve basis. This reading mode twist isn't necessary for the system, but it amplifies and emphasizes the theme of variable length outputs.

`https://k0ntinuum.github.io/revolver-jl`

`https://k0ntinuum.github.io/prefix-jl`

`https://k0ntinuum.github.io/refix-jl`

Elementary and Continuous Cellular Automata

FFLO

Fflo implements a continuous cellular automaton, allowing (in most versions) the user to *play* the automaton like a visual-musical instrument. You can try a stripped-down version in the browser here : `https://k0ntinuum.github.io/kont-ts-min/`.

I've written Fflo in C, Rust, JavaScript, and Go. I've written it to use a typical graphics library (Raylib), and I've written it to use "textgraphics" — to run in the console, using ANSI escape codes to get pixels. In one C version, a terminal *raw mode* allows for a interactive textgraphics experience. I've also written Rust/WebAssembly versions design to play well with Wasi/Wasmtime. Some versions allow the user to save "cartridges" (filter settings), so that promising parameters can be explored again. Some very rich patterns can emerge from simple rules. Some variants are just different enough to have their own names (e. g. Tornado).

`https://k0ntinuum.github.io/fflo-mn-C`

`https://k0ntinuum.github.io/fflo-ia-C`

https://k0ntinuum.github.io/fflo-go

https://k0ntinuum.github.io/fflo-tg-rs

https://k0ntinuum.github.io/torn-rs/

STARSHIP

I wrote the first version of Starship as an undergrad studying elementary cellular automata in a summer research program. The program (really family of programs) provides a visual exploration of computational space, in particular the space of ECA. Wolfram focused (at least most famously) on binary automata, but Starship generalizes this to an arbitrary base.

https://github.com/k0ntinuum/starship

Artificial Neural Networks

SCOPE

This program visualizes the interactive real-time learning of an artificial neural network. The user can vary the architecture of the network, adjust the learning rate, reset parameters, and so on. I wrote the first version in C while studying deep learning in grad school. Other versions followed in Rust, Go, and Javascript. The JS browser version (https://k0ntinuum.github.io/scope-js/ is extremely simplified (the user can choose between 3 buttons).