Kurt's Demo 2023

MECHANICAL CAD: YESTERDAY, TODAY, AND TOMORROW

Most of the mechanical engineering community is tied to the GUI, so you'd also need a way to generate code from GUI interactions. This is quite similar to an HTML "point and click" GUI that generates code on the backend. This allows folks who want to script to script and others who want to click can click. Both worlds can be happy — code on the left side, render on the right, just like a markdown editor.

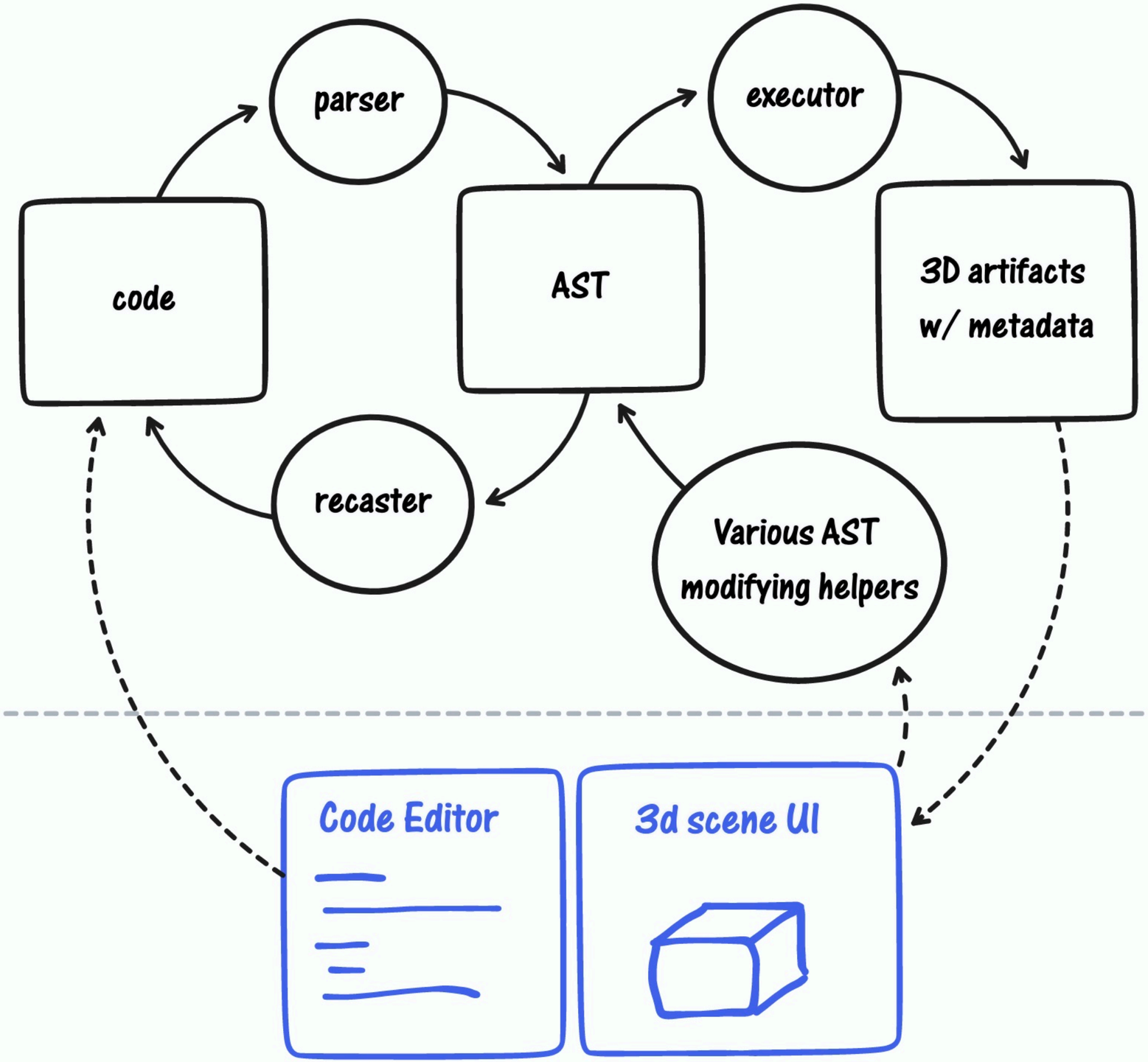
Jessie Frazelle

March 24, 2021

What Have I been building?

- Made a language/complier/interpreter (it's simple, and has plenty of bugs)
- Has recasting abilities, for the express purpose of aiding in code-gen
- Has a Fake geometry engine for the executor part of the language to hook into
- Built (simple and ugly) UI that treats the code as the source of truth.

Thanks to Hannah and Jess for being so supportive even though they have no idea what this was.


parser
executor
code
AST
3D artifacts
w/ metadata
recaster
Various AST
modifying helpers
Code Editor
3d scene UI

C

C

C

Code is data

Code is data

Code is data

```
const myVar = "str"
const sum = 5 + 2
log(sum)
```

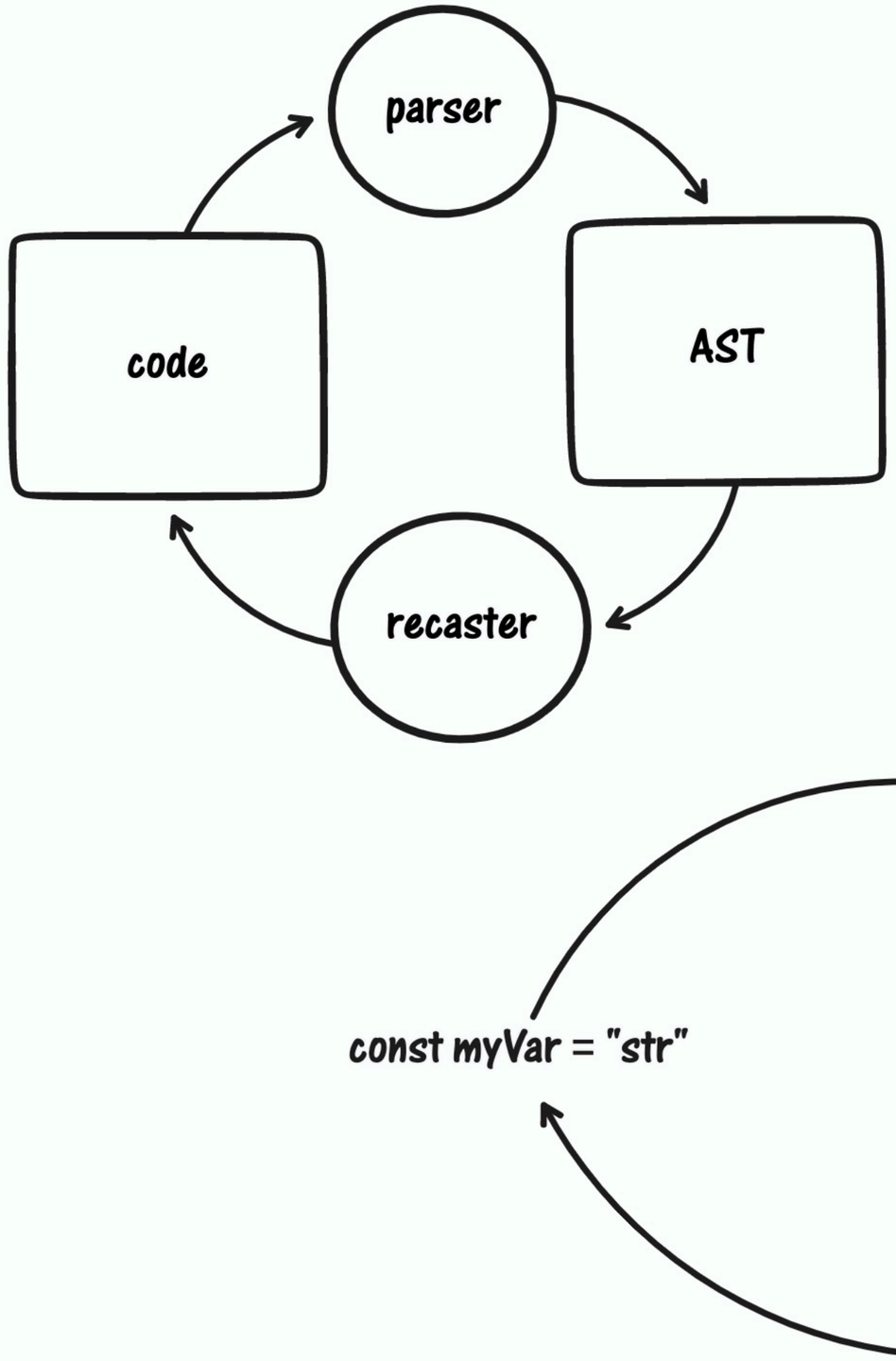

```
{
  type: 'VariableDeclaration',
  kind: 'const',
  declaration: {
    type: 'VariableDeclarator',
    id: {
      type: 'Identifier',
      name: 'myVar',
    },
    init: {
      type: 'Literal',
      value: 'str',
      raw: '"str"',
    },
  },
},
```

```
{
  type: 'VariableDeclaration',
  kind: 'const',
  declaration: {
    type: 'VariableDeclarator',
    id: {
      type: 'Identifier',
      name: 'sum',
    },
    init: {
      type: 'BinaryExpression',
      left: {
        type: 'Literal',
        value: 5,
        raw: '5',
      },
      operator: '+',
      right: {
        type: 'Literal',
        value: 2,
        raw: '2',
      },
    },
  },
},
```

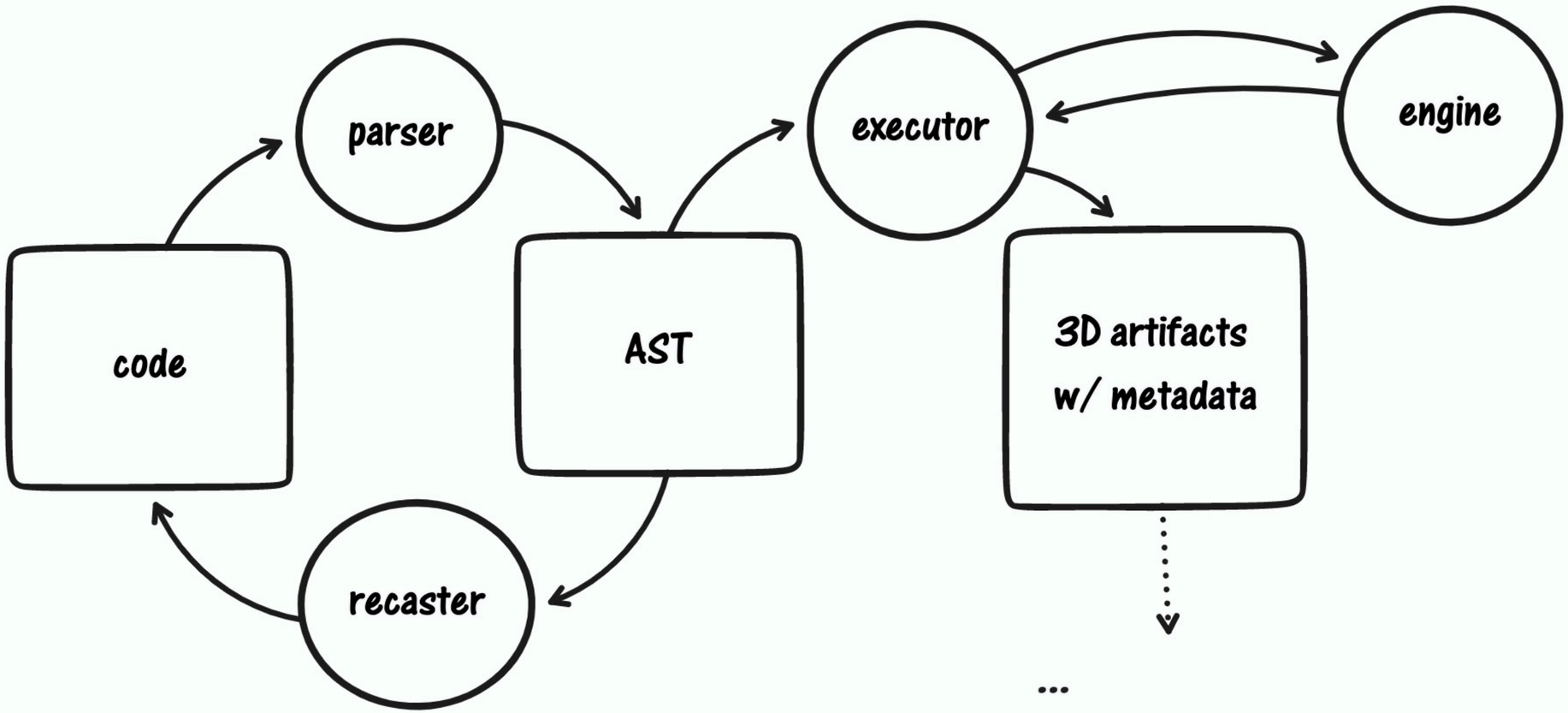
```
{
  type: 'ExpressionStatement',
  expression: {
    type: 'CallExpression',
    callee: {
      type: 'Identifier',
      name: 'log',
    },
    arguments: [
      {
        type: 'Identifier',
        name: 'sum',
      },
    ],
    optional: false,
  },
},
```

```
const myVar = "str"
```

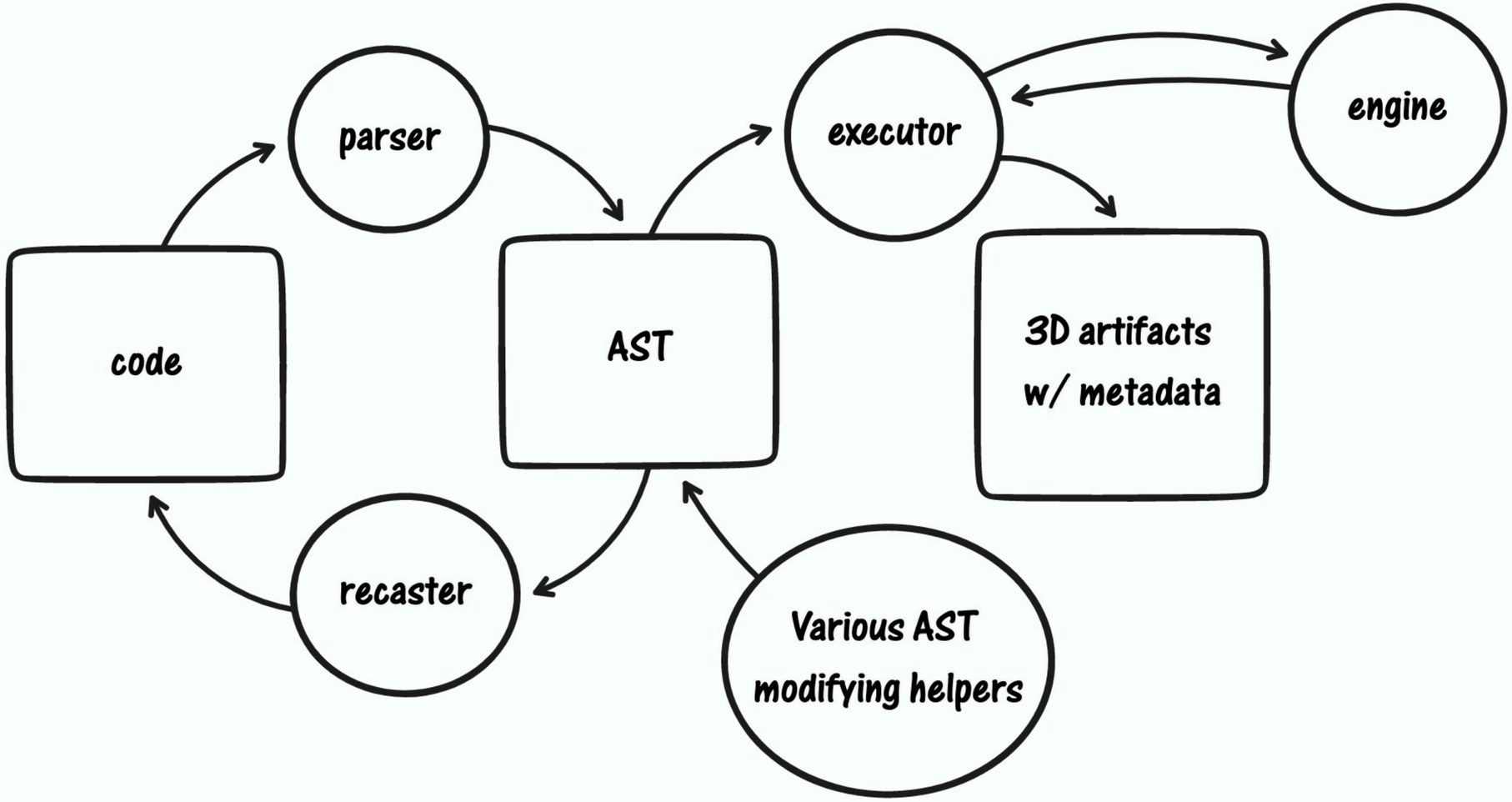

```
{
  type: 'VariableDeclaration',
  start: 0,
  end: 19,
  kind: 'const',
  declarations: [
    {
      type: 'VariableDeclarator',
      start: 6,
      end: 19,
      id: {
        type: 'Identifier',
        start: 6,
        end: 11,
        name: 'myVar',
      },
      init: {
        type: 'Literal',
        start: 14,
        end: 19,
        value: 'str',
        raw: '"str"',
      },
    },
  ],
},
```




```
{
  type: 'VariableDeclaration',
  kind: 'const',
  declaration: {
    type: 'VariableDeclarator',
    id: {
      type: 'Identifier',
      name: 'myVar',
    },
    init: {
      type: 'Literal',
      value: 'str',
      raw: '"str"',
    },
  },
},
```




```
{
  type: 'extrudeWall',
  geo: {/* data for the renderer */},
  sourceRange: [105, 132],
  // ^ non-negotiable metadata
  pathToNode: [body, 5, init, 2],
  location, quaternion ...
}
...
```


parser
executor
engine
code
AST
3D artifacts
w/ metadata
recaster
Various AST
modifying helpers

const myVar = 5

parser

```
const oldAst = {
  type: 'VariableDeclaration',
  kind: 'const',
  declarations: [
    {
      type: 'VariableDeclarator',
      id: {
        type: 'Identifier',
        name: 'myVar',
      },
      init: {
        type: 'Literal',
        value: 5,
        raw: '5',
      },
    },
  ],
}
```

goes and executes

blah blah

```
const newAst = clone(oldAst)
newAst.declarations[0].init = {
  type: 'binaryExpression',
  operator: '+',
  left: newAst.declarations[0].init,
  right: {
    type: 'Literal',
    value: 1,
    raw: '1',
  },
}
```

modify AST

recaster

const myVar = 5 + 1

```
{
  type: 'VariableDeclaration',
  kind: 'const',
  declarations: [
    {
      type: 'VariableDeclarator',
      id: {
        type: 'Identifier',
        name: 'myVar',
      },
      init: {
        type: 'BinaryExpression',
        left: {
          type: 'Literal',
          value: 5,
          raw: '5',
        },
        operator: '+',
        right: {
          type: 'Literal',
          value: 1,
          raw: '1',
        },
      },
    },
  ],
}
```

const myVar = 5

parser

```
const oldAst = {
  type: 'VariableDeclaration',
  kind: 'const',
  declarations: [
    {
      type: 'VariableDeclarator',
      id: {
        type: 'Identifier',
        name: 'myVar',
      },
      init: {
        type: 'Literal',
        value: 5,
        raw: '5',
      },
    },
  ],
}
```

why not

```
const newCode = oldCode + " + 1"
```

??

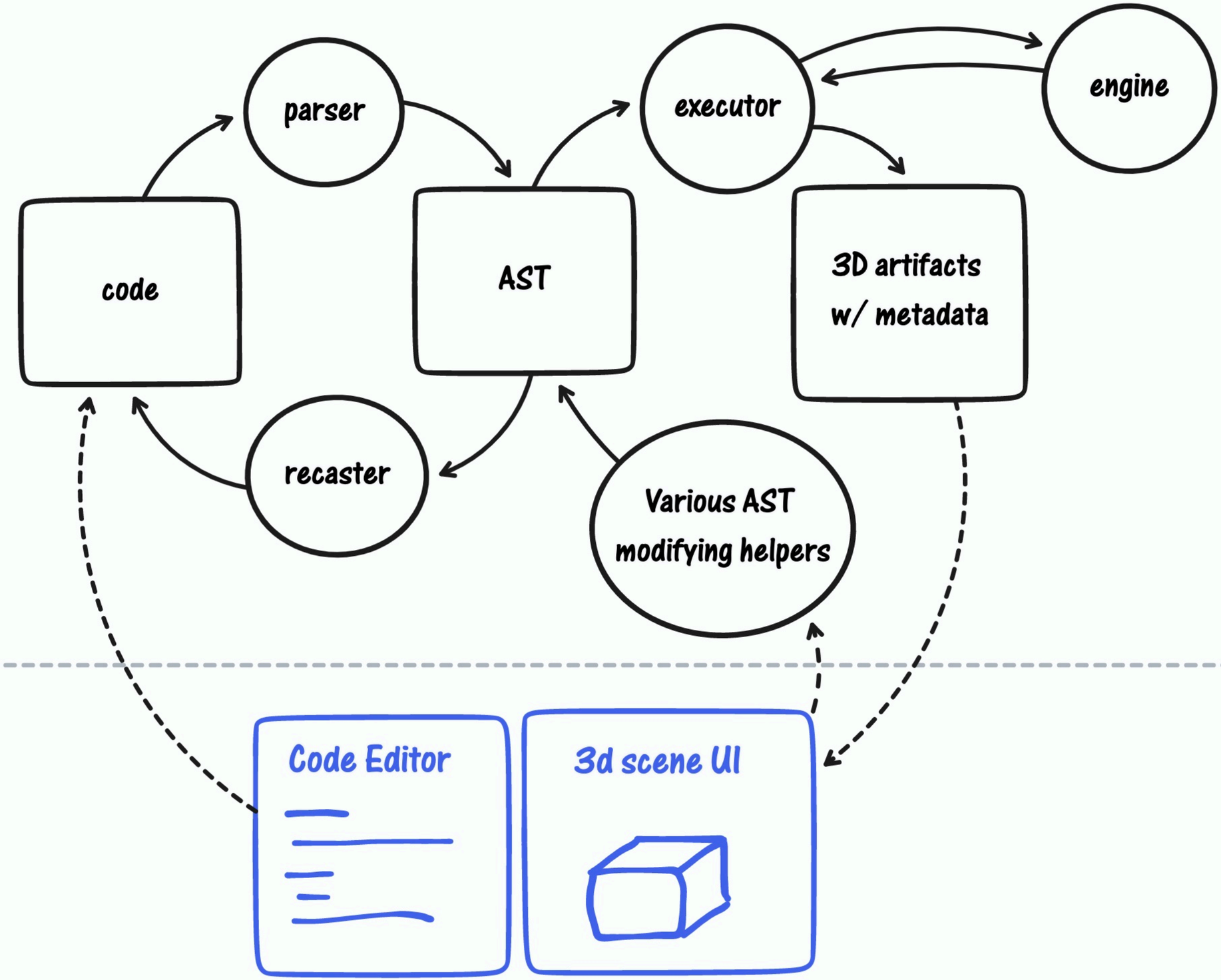
```
const newAst = clone(oldAst)
newAst.declarations[0].init = {
  type: 'binaryExpression',
  operator: '+',
  left: newAst.declarations[0].init,
  right: {
    type: 'Literal',
    value: 1,
    raw: '1',
  },
}
```

modify AST

recaster

const myVar = 5 + 1

```
{
  type: 'VariableDeclaration',
  kind: 'const',
  declarations: [
    {
      type: 'VariableDeclarator',
      id: {
        type: 'Identifier',
        name: 'myVar',
      },
      init: {
        type: 'BinaryExpression',
        left: {
          type: 'Literal',
          value: 5,
          raw: '5',
        },
        operator: '+',
        right: {
          type: 'Literal',
          value: 1,
          raw: '1',
        },
      },
    },
  ],
}
```


parser
executor
engine
code
AST
3D artifacts
w/ metadata
recaster
Various AST
modifying helpers
Code Editor
3d scene UI

```
fn3(fn2(fn1("butts")))

fn3(
  fn2(
    fn1("butts")
  )
)
```

```
const result1 = fn1("butts")
const result2 = fn2(result1)
const result3 = fn3(result2)
```

```
let result3 = fn1("butts")
  |> fn2(%)
  |> fn3(%)
```

Demo

- "selecting 3d feature"
- editing LineTo with identifier
- sketch on face (somewhat complex AST modification)

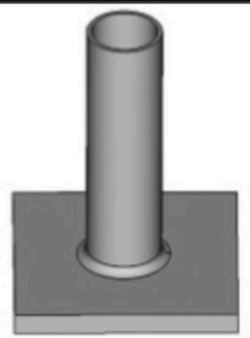
Code is the ultimate in "intent capture"

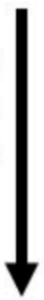
Code is the ultimate in "intent capture"

Kurt 🍩
@IrevDev

This one image from Mathur et al says so much.

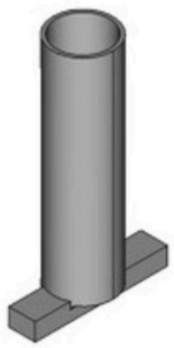
Let's talk about expressing intent in parametric CAD models and heuristics 👇





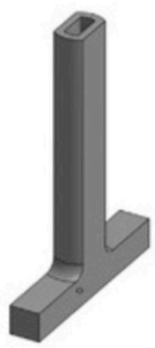
(a) *Initial design.*

(b) *After perturbation:* CADQUERY.



(c) *After perturbation:* FREECAD.

(d) *After perturbation:* AUTODESK FUSION 360.



(e) *After perturbation:* ONSHAPE.

Clicking around in a GUI is producing data

Code is data

Code is the ultimate in "intent capture"

Therefore clicking around in a GUI should produce code

Show them the data they've produced in a way they can reason about

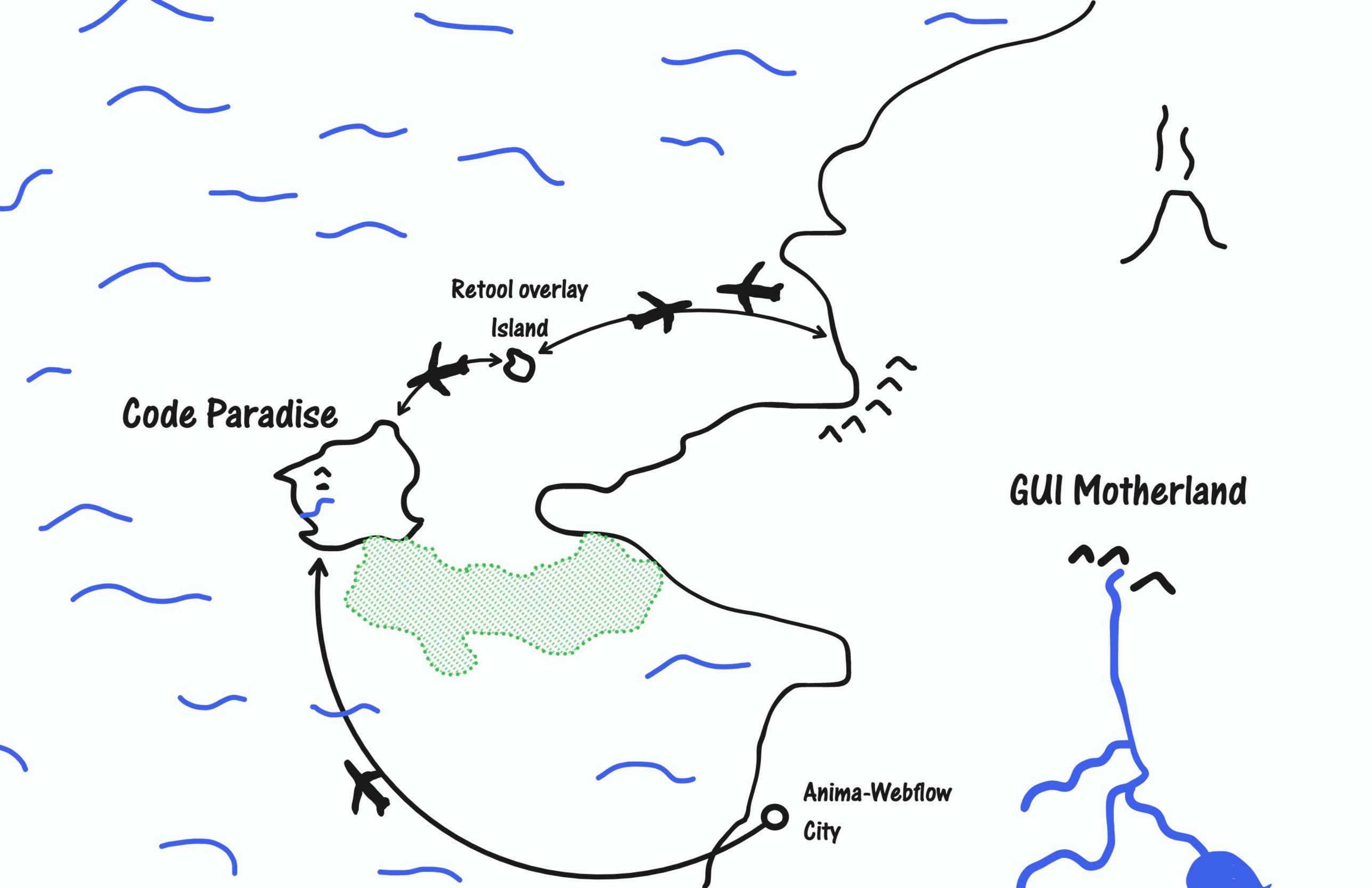
Code Paradise

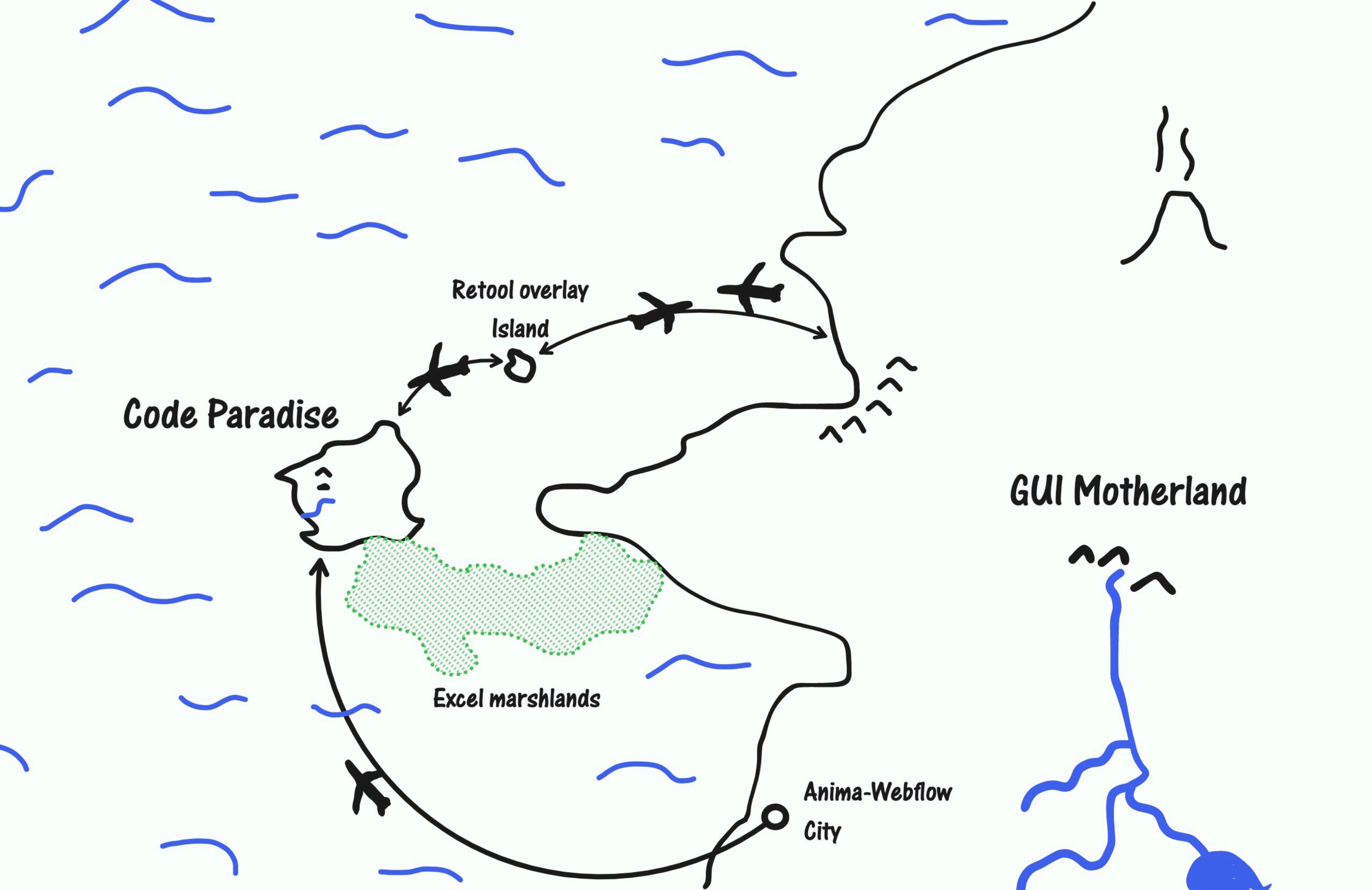
GUI Motherland

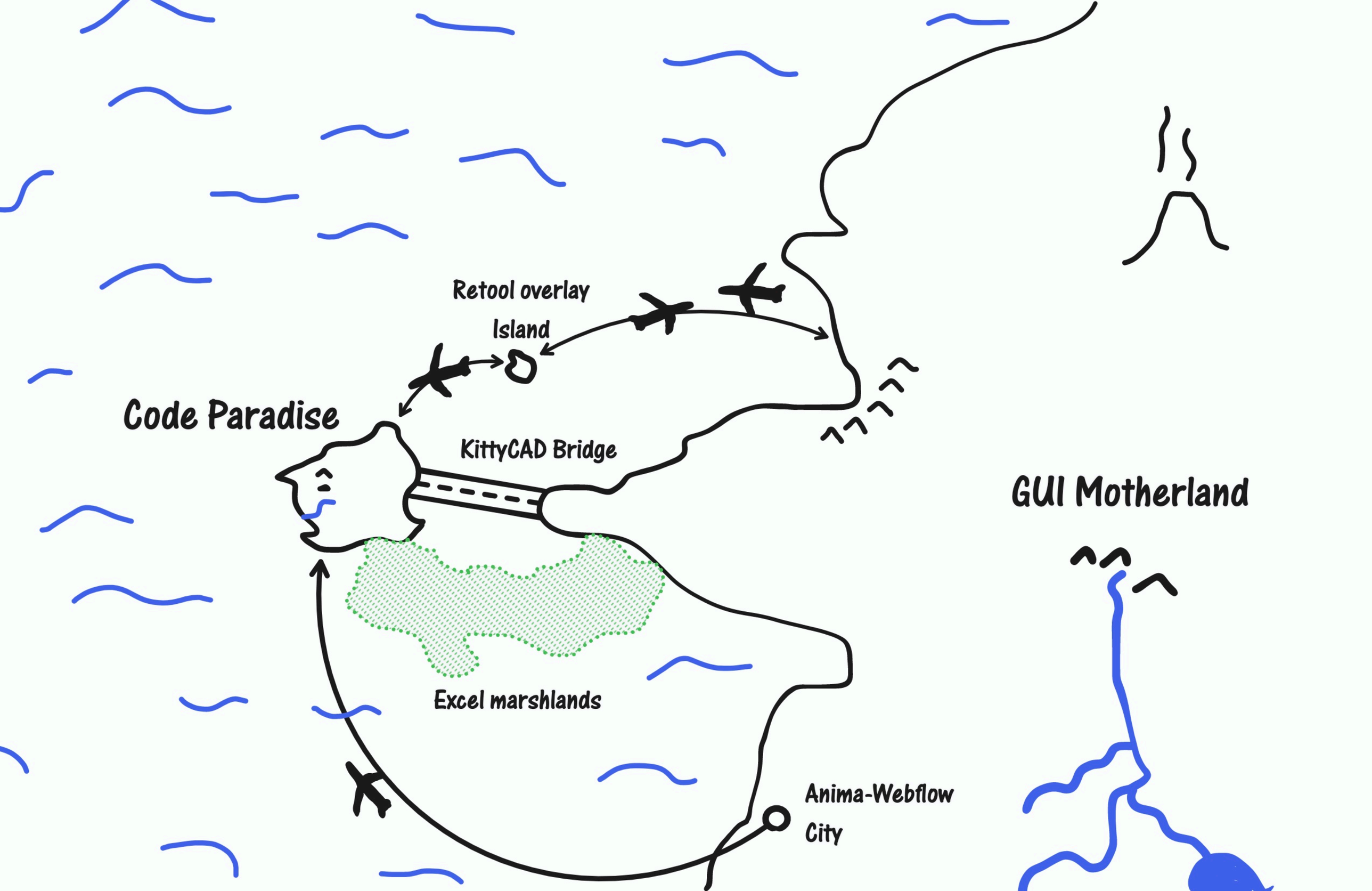
Code Paradise

GUI Motherland

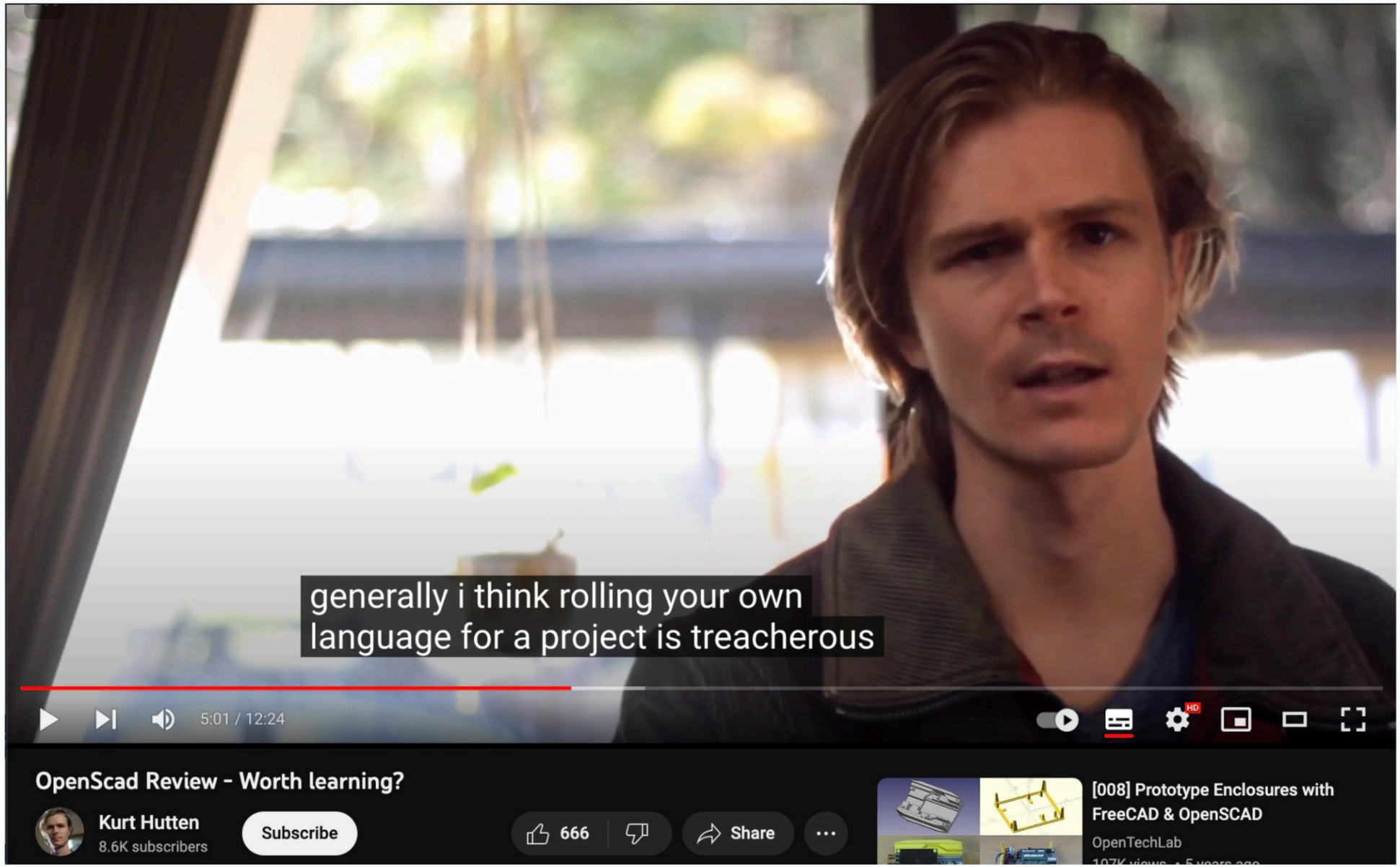
Anima-Webflow
City


Retool overlay
Island
Code Paradise
GUI Motherland
Anima-Webflow
City


Retool overlay
Island
Code Paradise
Excel marshlands
Anima-Webflow
City
GUI Motherland


Retool overlay
Island
Code Paradise
KittyCAD Bridge
Excel marshlands
Anima-Webflow
City
GUI Motherland

Why a new language?

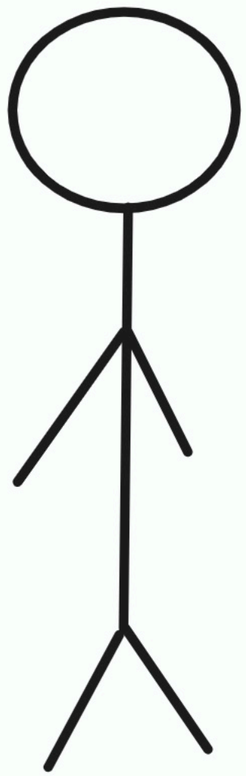


- Code Gen is hard

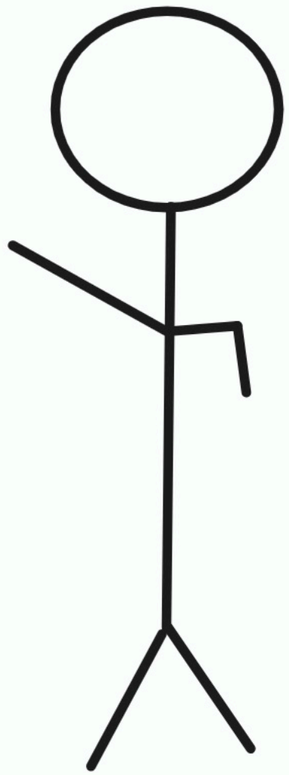
- We should give ourselves as much of an advantage as possible by controlling the language and tailoring it to code-gen
- i.e. locking it down

- if we used Python, our AST modifier would have to understand the whole of Python and stay up to date.

What's your best impression of a functional programmer?



Blah blah immutable, blah blah pure, blah blah pure immutable, immutable pure. Pure pure immutability.



Data 😇😇

Calculations 😇

Actions 💩

Not the next big general purpose language

Performance is not critical

Can build inter opt with other languages through our SDKs

Its two jobs are only to:

- Support code-gen well/robustly
- Convey intent

In defence of little languages

Make XSLT Inform yaml ANTLR CFML SWIG IDL Emacs Lisp Jinja JSON VIM-Script Lex Sed
Bash Batch Mustache AWK CSS HTML Scsh XML yacc SQL Tcl Bison Guile ASP.NET

```
model User {
  id       String  @id @default(uuid())
  userName String  @unique // reffered to as userId in @relations
  email    String  @unique
  name     String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  image                 String?
  bio                   String?
  Project               Project[]
  Reaction              ProjectReaction[]
  Comment               Comment[]
  SubjectAccessRequest SubjectAccessRequest[]
}
```

You, 1 second ago | 1 author (You)

```
model Comment {
  id     String @id @default(uuid())
  text   String
  user   User   @relation(fields: [userId], references: [id])
  userId String
  project   Project   @relation(fields: [projectId], references: [id])
  projectId String

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

```
},
"/async/operations/{id}": {
    "get": {
        Try it
        "description": "Get the status and output of an async operation.\nThis endp
        "operationId": "get_async_operation",
        "parameters": [
            {
                "description": "The ID of the async operation.",
                "in": "path",
                "name": "id",
                "required": true,
                "schema": {
                    "type": "string"
                },
                "style": "simple"
            }
        ],
        "responses": {
            "200": {
                "content": {
                    "application/json": {
                        "schema": {
                            "$ref": "#/components/schemas/AsyncApiCallOutput"
                        }
                    }
                },
                "description": "successful operation",
```

	Creative Teams	Mechanical Teams	Software Teams
Tools	- Blackbox - Series of mouse clicks - Short term productivity		- Semantic description - Long term productivity - Verification each iteration - Flexiable pipelines
Product	- One-time release - Needs to look right	- Complex - Long-term - Iterative