Wasm Blocks

Procedural Modeling with Micro Webassemblies

Sean Isom
Adobe

Disclaimer: The views and opinions expressed in this presentation are those of the author and do not reflect any products, views, or positions of any employer or other entities they represent.

Software = Code + Data

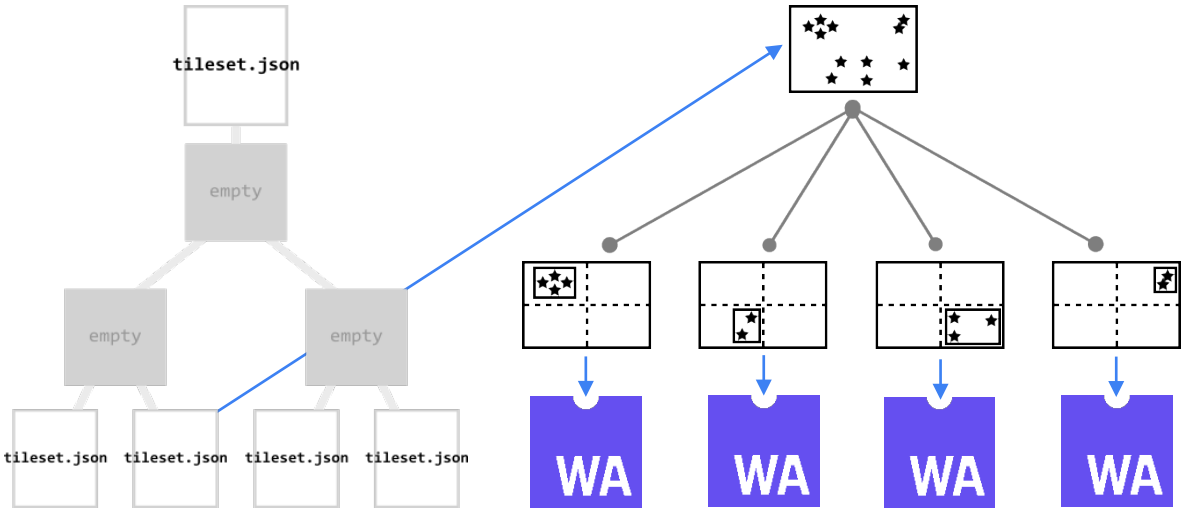
- Someone, Some Year

What if we could procedurally replace portions of our data with code for greater efficiency and fidelity?

What is Procedural Modeling?

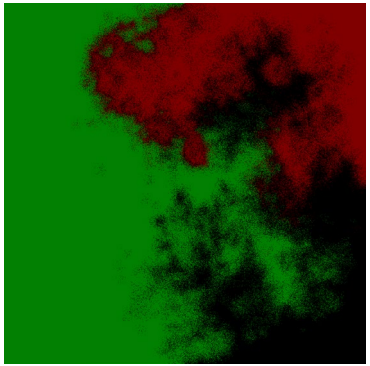
- Create data programatrically instead of manually
- Examples: Formal grammars, L-systems, fractals, generative
- Add artistic variation at massive scale
- Incorporate randomness with heuristics
- Integrate with high-quality manual data
- Efficient use of space

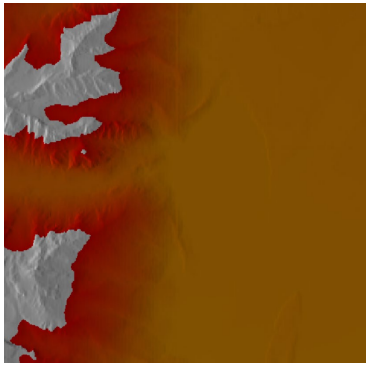
What does this have to do with Wasm?

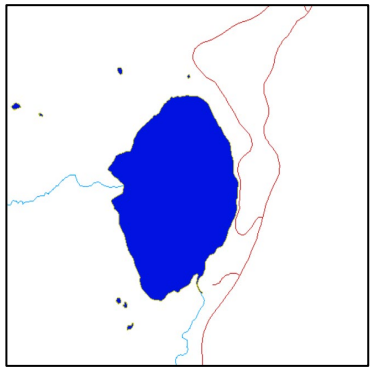


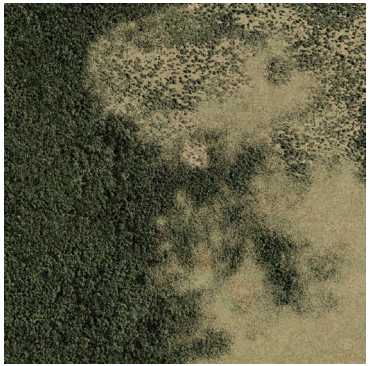
- Thesis: Code more efficient than data (binary size)
- High performance
- Portable
- Dynamic compilation
- Fast startup
- Secure by default for arbitrary code
- Support User-Generated Content

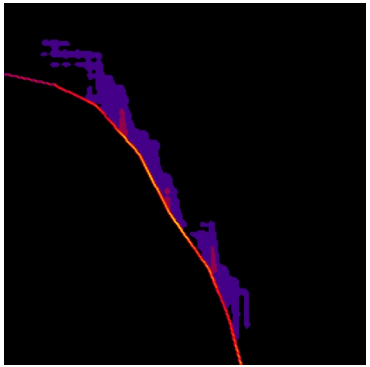
EXAMPLE 1: Vector Maps

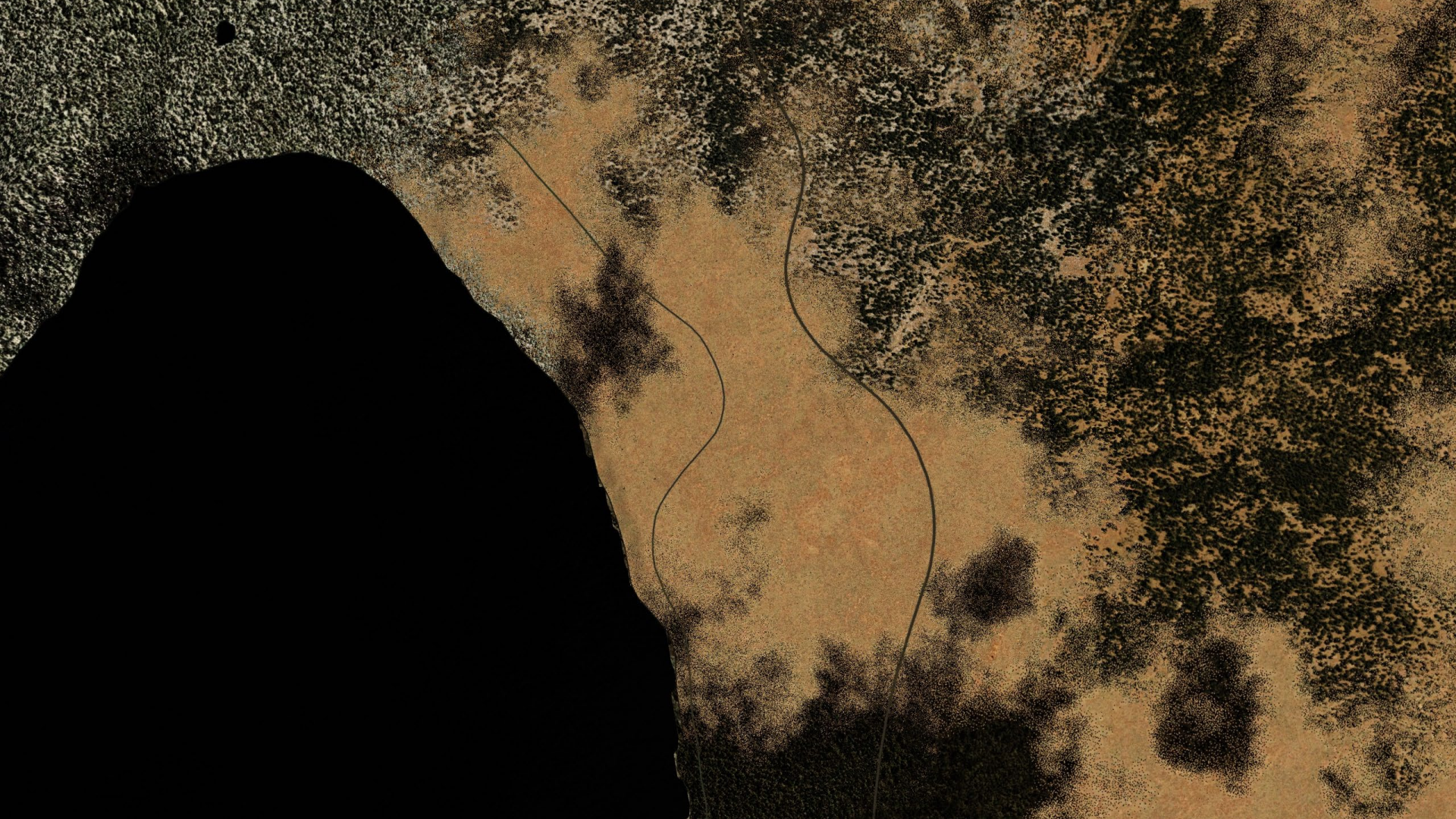










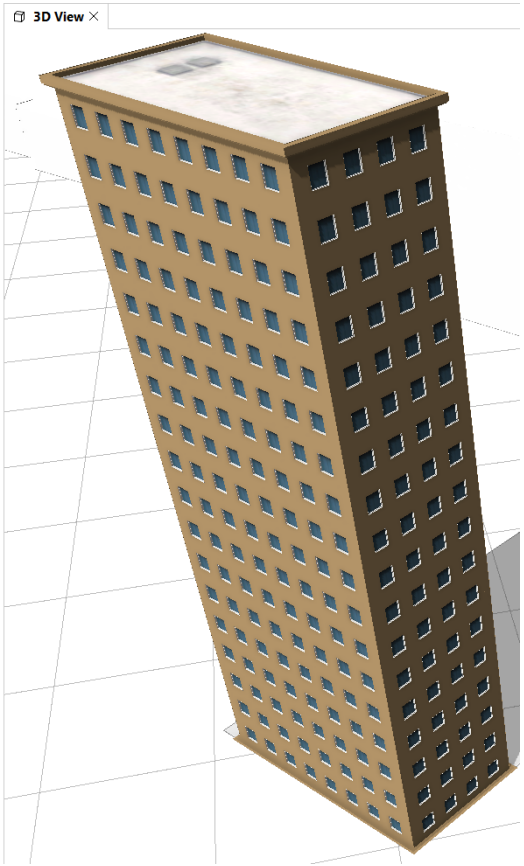


EXAMPLE 2: 3D Building Models

oro.cga | *Scene

```
// Grammar 0

attr depth = 7.95272636
attr height = 38.7053642
attr roof_height = 0.406891108
attr roof_overhang = 0.346042305
attr width = 15.07317114


@StartRule
Start-->
//translate(rel, world, 0, 0, 0)
s(width, 0, depth)
extrude(height)
comp(f) { bottom : Base | side : Facade | top : Roof }

Roof-->
offset(roof_overhang*0.8)
comp(f){ inside: TopFaceOffset }

TopFaceOffset-->
extrude(roof_height*0.2)
comp(f) { bottom : RoofLedge | side : RoofLedge | top :  TopFaceOf

TopFaceOffsetTop-->
offset(roof_overhang*0.2)
comp(f) { inside: TopFaceOffsetTopOffset }

TopFaceOffsetTopOffset-->
extrude(roof_height*0.8)
comp(f) { bottom : RoofLedge | side : RoofLedge | top : TopFaceOff

TopFaceOffsetTopOffsetTop-->
offset(-roof_overhang) comp(f){ inside: TopFaceOffsetTopOffsetTopIn

TopFaceOffsetTopOffsetTopInside-->
extrude(-roof_height*0.5)
comp(f) { side : RoofLedge | top : reverseNormals() RoofTop }

// Grammar 1

attr p00_GH = 0.317185193
attr p01_FH = 2.13179612
attr p02_AH = 0.0158432871
attr p03_SW = 0.417930305
attr p04_TW = 1.77921629
attr p05_GW = 7.11686516
attr p06_WT = 0.629350841
```

3D View



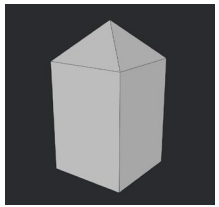
Implementation

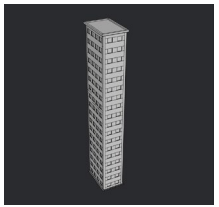
```
void AddGrammarColor(std::vector<cga::Grammar>& grammars)
{
    using namespace boost;

    cga::Grammar grammar;

    grammar.addAttr("", "facade_color", Attribute("facade_color", "#a3875e"));
    grammar.addAttr("", "facade_color0", Attribute("facade_color0", "#a3875e"));
    grammar.addAttr("", "facade_color1", Attribute("facade_color1", "#373226"));
    grammar.addAttr("", "facade_color2", Attribute("facade_color2", "#8c7a74"));
    grammar.addAttr("", "facade_color3", Attribute("facade_color3", "#60635e"));
    grammar.addAttr("", "facade_color4", Attribute("facade_color4", "#bbeaff"));
    grammar.addAttr("", "facade_color5", Attribute("facade_color5", "#f3c89d"));
    grammar.addAttr("", "facade_color6", Attribute("facade_color6", "#282425"));
    grammar.addAttr("", "facade_color7", Attribute("facade_color7", "#f0cda8"));
    grammar.addAttr("", "facade_color8", Attribute("facade_color8", "#808080"));
    grammar.addAttr("", "facade_color9", Attribute("facade_color9", "#808080"));

    grammar.addRule("", "DoorGlass");
    grammar.addOperator("", "DoorGlass", make_shared<ColorOperator>("#202020"));

    grammar.addRule("", "LedgeFace");
    grammar.addOperator("", "LedgeFace", make_shared<ColorOperator>("facade_color"));

    grammar.addRule("", "RoofLedge");
    grammar.addOperator("", "RoofLedge", make_shared<ColorOperator>("facade_color"));

    grammar.addRule("", "RoofTop");
    grammar.addOperator("", "RoofTop", make_shared<SetupProjectionOperator>(axesSelector:AXES_SCOPE_XY,
                                                                            texHeight:Value(Value::TYPE_RELATIVE,
    grammar.addOperator("", "RoofTop", make_shared<TextureOperator>("roof2.png"));

    grammar.addRule("", "Wall");
    grammar.addOperator("", "Wall", make_shared<ColorOperator>("facade_color"));

    grammar.addRule("", "Wall0");
    grammar.addOperator("", "Wall0", make_shared<ColorOperator>("facade_color0"));
```

- Grammar compiled using C++ / clang
- Opportunity to go direct to wasm
- Parameterized input (WASI)
- Pipe stdout with obj/mtl files
- Stream *.wasm tiles in quadtree
- Grammar(s) per building
- SIMD – Performance within 68% x64
- Code size challenges – 589kb
  - Split off host functionality
  - Group buildings / LOD?

Performance

- 200 Buildings – Low complexity vs High complexity (and cached compilation)





What's Next?

- Component Model
- Separate Host Functionality
- Graphics APIs
- Binary Size Optimization
- Compilation Performance
- Recompilation

Input `*.wit`

```
interface grammar {
  record matrix4 {

  }

  record shape {
    name: string
  }

  record rectangle {
    shp: shape,
    x: u32,
    y: u32,
    pivot: matrix4,
    scope: matrix4
  }

  get-stack: func() -> list<shape>
  push-stack: func(shp: shape)
  pop-stack: func() -> shape

  record operator {
    rule: string
  }

  record operator-comp {
    op: operator,
    name: string,
    values: list<tuple<string, string>>
  }

  add-rule: func(rule: string)
  add-operator-comp: func(rule: string, comp: operator-co
  get-operators: func() -> list<operator>
}

default world terragen {
  import derive: func(start-rule: string)
  import generate-geometry: func()
  import api: self.grammar

  export run: func(params: string)
}
```

Generated bindings

Language: C · File: terragen.c

```c
#include "terragen.h"


__attribute__((import_module("api"), import_name("get-sta
void __wasm_import_api_get_stack(int32_t);

__attribute__((import_module("api"), import_name("push-st
void __wasm_import_api_push_stack(int32_t, int32_t);

__attribute__((import_module("api"), import_name("pop-sta
void __wasm_import_api_pop_stack(int32_t);

__attribute__((import_module("api"), import_name("add-rul
void __wasm_import_api_add_rule(int32_t, int32_t);

__attribute__((import_module("api"), import_name("add-ope
void __wasm_import_api_add_operator_comp(int32_t, int32_t

__attribute__((import_module("api"), import_name("get-ope
void __wasm_import_api_get_operators(int32_t);

__attribute__((import_module("terragen"), import_name("de
void __wasm_import_terragen_derive(int32_t, int32_t);

__attribute__((import_module("terragen"), import_name("ge
void __wasm_import_terragen_generate_geometry(void);

__attribute__((weak, export_name("cabi_realloc")))
void *cabi_realloc(void *ptr, size_t orig_size, size_t or
  void *ret = realloc(ptr, new_size);
  if (!ret) abort();
  return ret;
}

// Helper Functions

void api_shape_free(api_shape_t *ptr) {
  terragen_string_free(&ptr->name);
}

void api_operator_free(api_operator_t *ptr) {
  terragen_string_free(&ptr->rule);
}

void api_tuple2_string_string_free(api_tuple2_string_stri
  terragen_string_free(&ptr->f0);
  terragen_string_free(&ptr->f1);
```

THANKS

Sean Isom

@theisomizer

https://github.com/seanisom/WasmBlocks