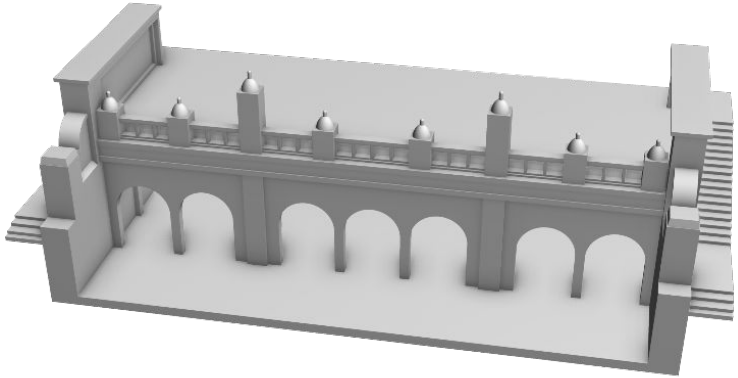
Fidget

Yet Another Implicit Kernel

Matt Keeter
July 3rd, 2024

What are we talking about?

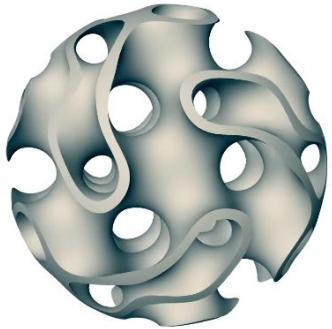
- Complex closed-form implicit surfaces
  - `f(x, y, z)`
  - Less than 0 → inside the shape
  - Greater than 0 → outside the shape
- Arithmetic and transcendental operations
- Hundreds or thousands of clauses



libfive

Home Studio Download Examples About

`libfive` is a software library and set of tools for solid modeling, especially suited for parametric and procedural design. It is infrastructure for generative design, mass customization, and domain-specific CAD tools.



Studio
Desktop design tool

Bindings
Scheme and Python

Standard library
Shapes and geometric operations

C API
`#include <libfive.h>`

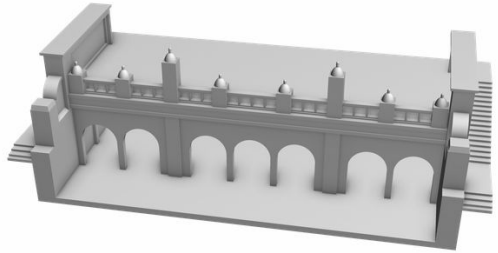
`libfive`
C++ library

libfive.com

Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces

Matthew J. Keeter, independent researcher

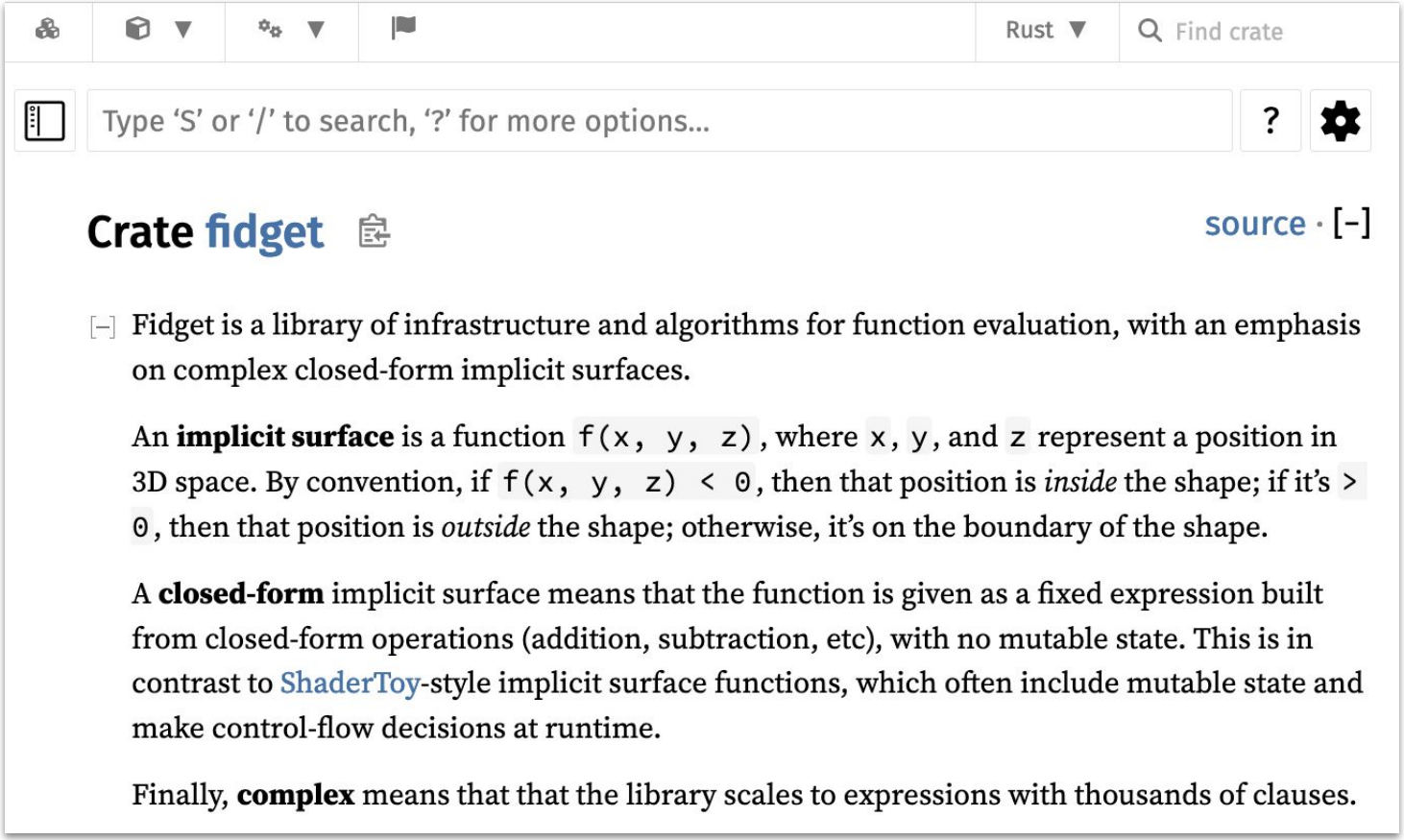
ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2020



mattkeeter.com/research/mpr

Why build something new?

- Find the “right APIs” for a high-performance implicit kernel
- Experiment with native (JIT) compilation
- `libfive` is hard to hack on
  - 40 KLOC, mostly C++
  - Building is annoying (CMake, Python, code generation)
  - Writing C++ is hard
- “Prefer Rust to C/C++ for new code”
  - Safety, performance, correctness
  - Built-in package management and cross-platform builds
  - Easy to target WebAssembly

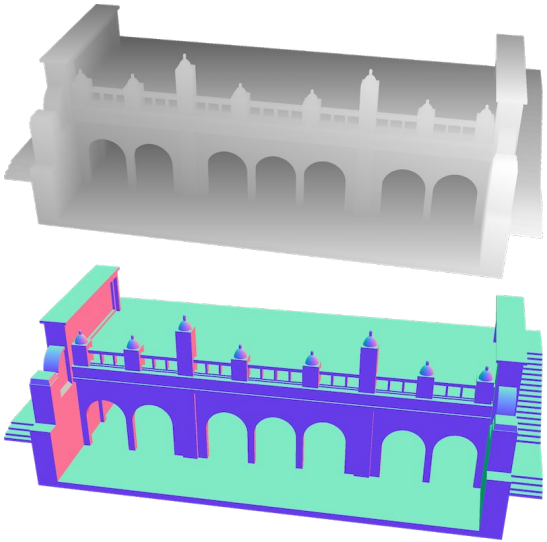


docs.rs/fidget

What's included?

- Shape construction
  - Trees, graphs, tapes
- Fast evaluation
  - Point evaluation (single and array)
  - First derivatives (forward-mode automatic differentiation)
  - Interval arithmetic
- What algorithms can we build on top of fast evaluation?
  - Rendering (2D, 3D)
  - Meshing (manifold dual contouring)
  - Constraint solving (Levenberg-Marquardt algorithm)

Benchmarking rasterization



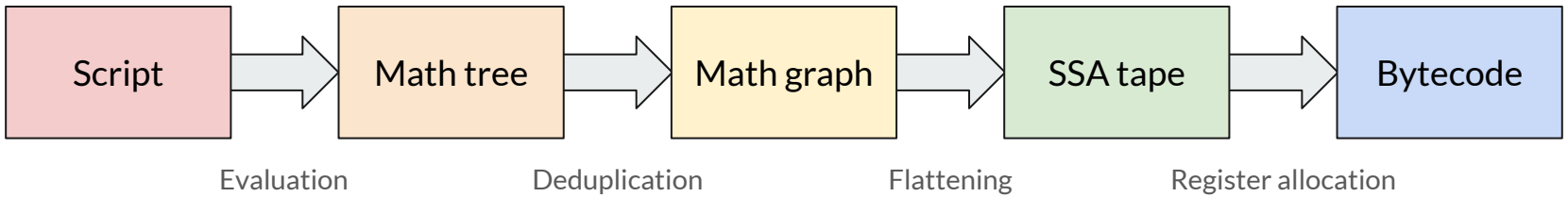
Size	`libfive`	MPR	Fidget (VM)	Fidget (JIT)
1024^3	66.8 ms	22.6 ms	61.7 ms	23.6 ms
1536^3	127 ms	39.3 ms	112 ms	45.4 ms
2048^3	211 ms	60.6 ms	184 ms	77.4 ms

Cross-platform support

Platform	JIT support	CI	Support
`aarch64-apple-darwin`	Yes	✅ Tested	⭐ Tier 0
`x86_64-unknown-linux-gnu`	Yes	✅ Tested	🥇 Tier 1
`x86_64-pc-windows-msvc`	Yes	✅ Tested	🥈 Tier 2
`aarch64-unknown-linux-gnu`	Yes	⚠️ Checked	🥇 Tier 1
`aarch64-pc-windows-msvc`	Yes	⚠️ Checked	🥉 Tier 3
`wasm32-unknown-unknown`	No	⚠️ Checked	🥇 Tier 1

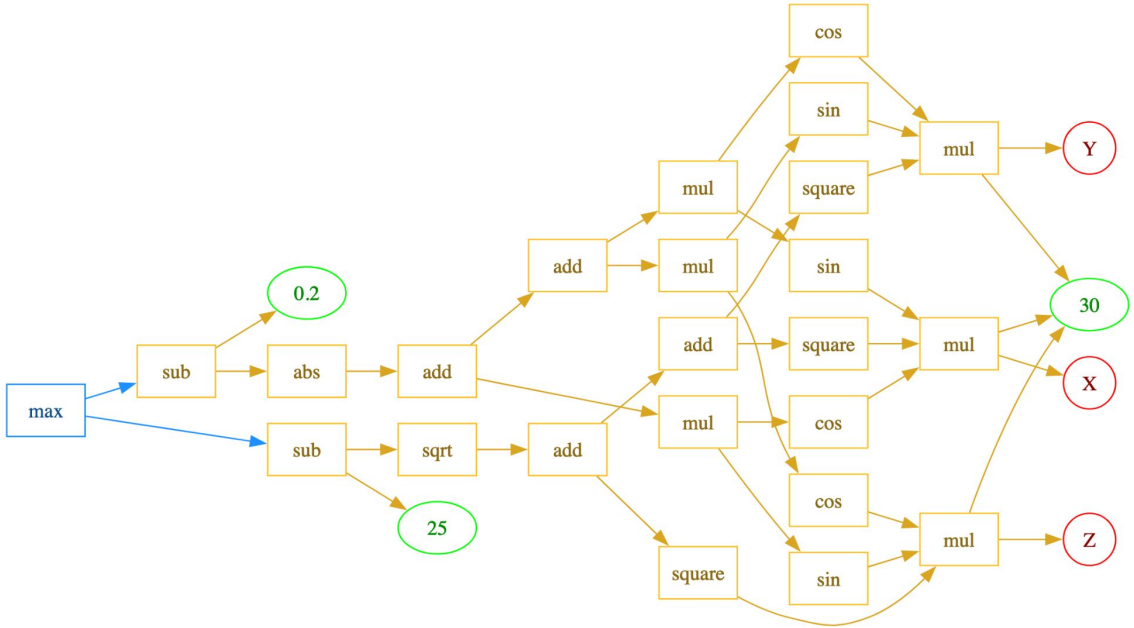
Frontend
Building math expressions

Front-end


Script
Math tree
Math graph
SSA tape
Bytecode
Evaluation
Deduplication
Flattening
Register allocation

Front-end: Script

Front-end: Tree & Graph



Front-end: SSA Tape

```
Input(7, 0)
MulRegImm(6, 7, 30.0)
SquareReg(26, 6)
Input(16, 2)
MulRegImm(15, 16, 30.0)
SquareReg(25, 15)
AddRegReg(24, 26, 25)
Input(10, 1)
MulRegImm(9, 10, 30.0)
SquareReg(23, 9)
AddRegReg(22, 24, 23)
SqrtReg(21, 22)
SubRegImm(20, 21, 25.0)
SinReg(19, 6)
```

```
(continued)
CosReg(18, 15)
MulRegReg(17, 19, 18)
SinReg(14, 15)
CosReg(13, 9)
MulRegReg(12, 14, 13)
AddRegReg(11, 17, 12)
SinReg(8, 9)
CosReg(5, 6)
MulRegReg(4, 8, 5)
AddRegReg(3, 11, 4)
AbsReg(2, 3)
SubRegImm(1, 2, 0.2)
MaxRegReg(0, 20, 1)
```

Front-end: Bytecode

```
Input(3, 0)
MulRegImm(3, 3, 30.0)
SquareReg(0, 3)
Input(4, 2)
MulRegImm(4, 4, 30.0)
SquareReg(2, 4)
AddRegReg(0, 0, 2)
Input(2, 1)
MulRegImm(2, 2, 30.0)
SquareReg(1, 2)
AddRegReg(0, 0, 1)
SqrtReg(0, 0)
SubRegImm(0, 0, 25.0)
SinReg(1, 3)
```

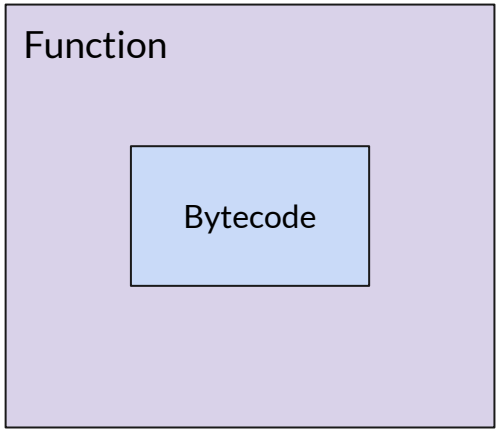
```
(continued)
CosReg(5, 4)
MulRegReg(1, 1, 5)
SinReg(4, 4)
CosReg(5, 2)
MulRegReg(4, 4, 5)
AddRegReg(1, 1, 4)
SinReg(2, 2)
CosReg(3, 3)
MulRegReg(2, 2, 3)
AddRegReg(1, 1, 2)
AbsReg(1, 1)
SubRegImm(1, 1, 0.2)
MaxRegReg(0, 0, 1)
```

Backend

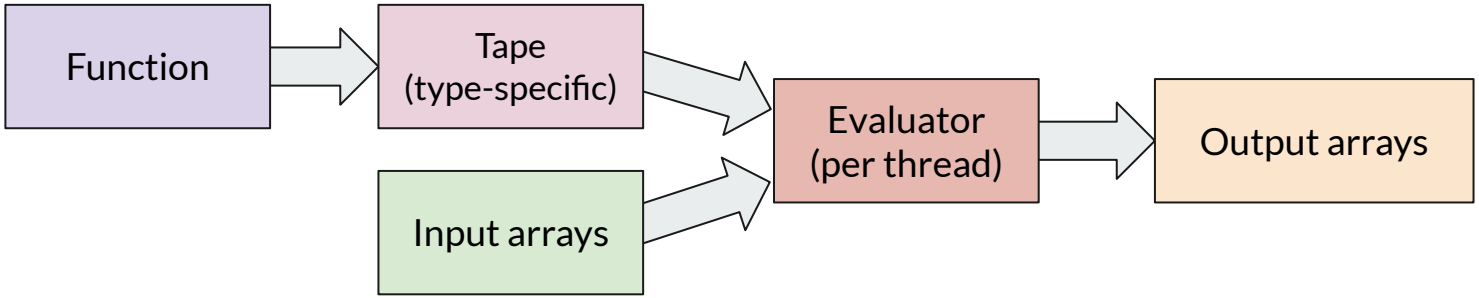
Fast, flexible evaluation

Backend: Traits

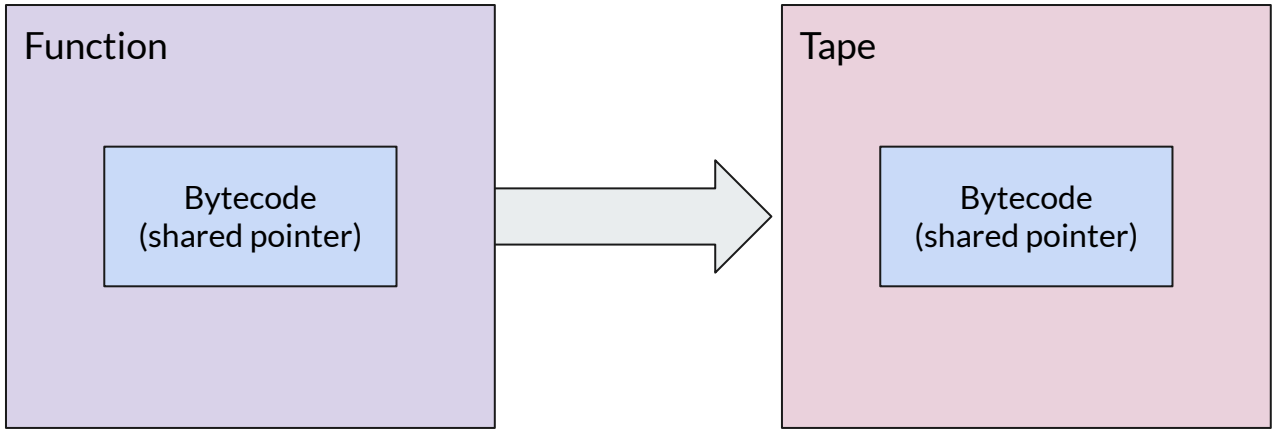
```
pub trait Function: Send + Sync + Clone {
    type Trace: Clone + Eq + Send + Trace;
    type Storage: Default + Send;
    type Workspace: Default + Send;
    type TapeStorage: Default + Send;
    type PointEval: TracingEvaluator<Data = f32, Trace = Self::Trace, TapeStorage = Self::TapeStor
    type IntervalEval: TracingEvaluator<Data = Interval, Trace = Self::Trace, TapeStorage = Self::
    type FloatSliceEval: BulkEvaluator<Data = f32, TapeStorage = Self::TapeStorage> + Send + Sync;
    type GradSliceEval: BulkEvaluator<Data = Grad, TapeStorage = Self::TapeStorage> + Send + Sync;

    // Required methods
    fn point_tape(
        &self,
        storage: Self::TapeStorage
    ) -> <Self::PointEval as TracingEvaluator>::Tape;

    fn interval_tape(
        &self,
        storage: Self::TapeStorage
    ) -> <Self::IntervalEval as TracingEvaluator>::Tape;

    fn float_slice_tape(
        &self,
        storage: Self::TapeStorage
    ) -> <Self::FloatSliceEval as BulkEvaluator>::Tape;

    fn grad_slice_tape(
        &self,
        storage: Self::TapeStorage
    ) -> <Self::GradSliceEval as BulkEvaluator>::Tape;

    fn simplify(
        &self,
        trace: &Self::Trace,
        storage: Self::Storage,
        workspace: &mut Self::Workspace
    ) -> Result<Self, Error>
       where Self: Sized;
```



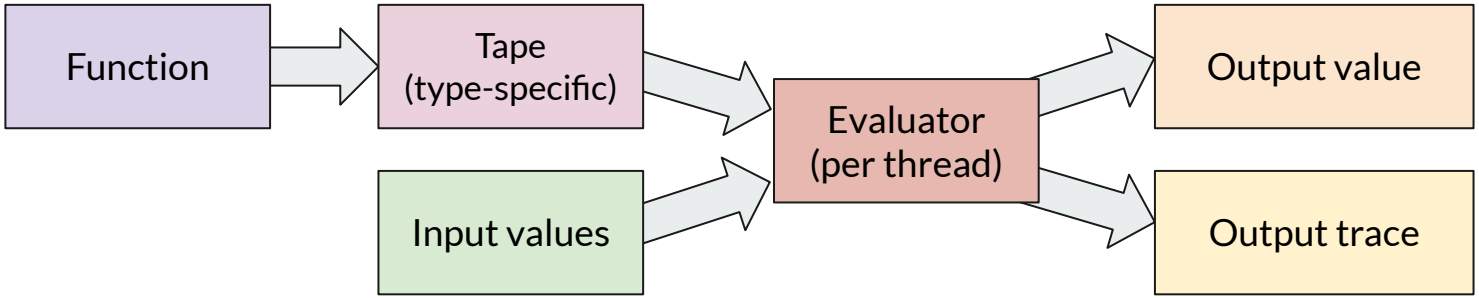
Backend: Bulk evaluation



Interpreter Functions and Tapes

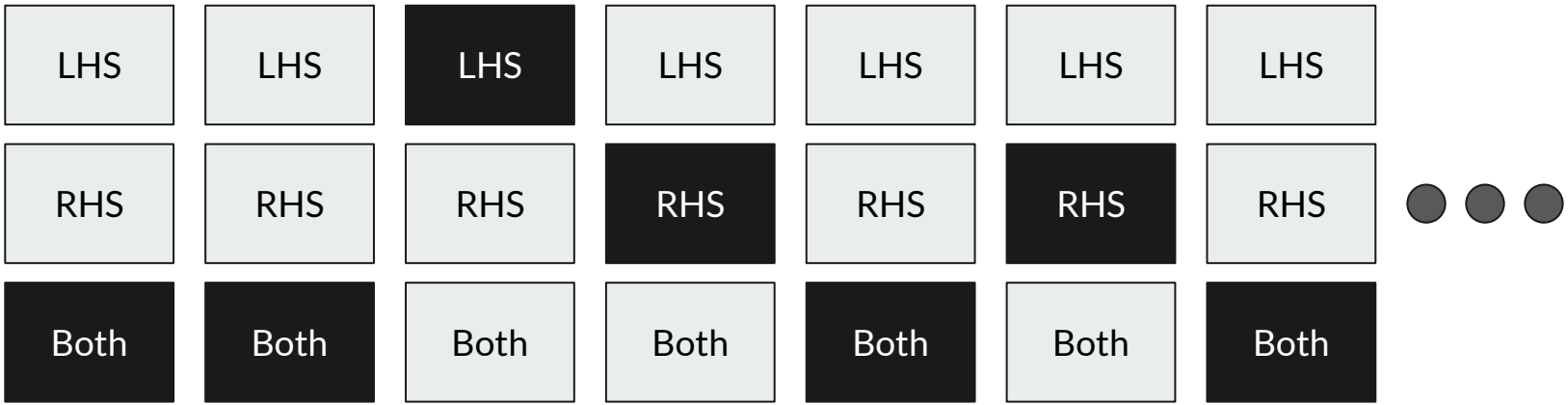


Backend: Tracing evaluation

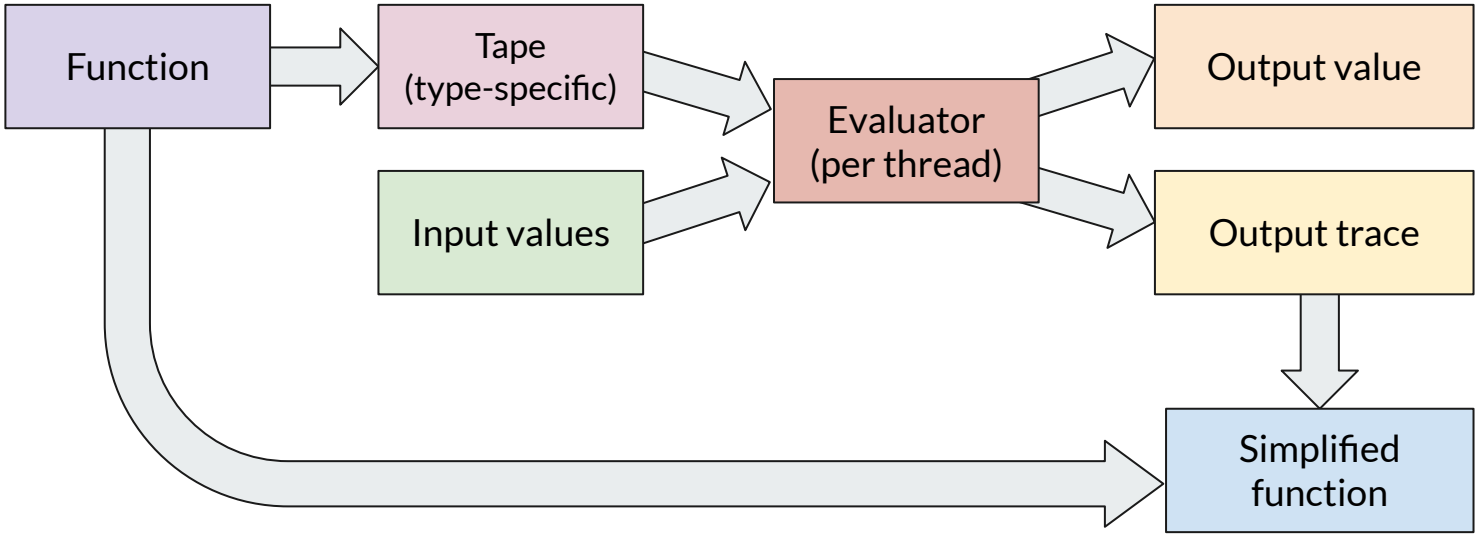


Backend: What is a trace?

One **Choice** per `min` / `max` clause in the tape



Backend: Function simplification



Why JIT compilation?

- Performance!
- Interpreter overhead is significant
  - Main opcode switch is an unpredictable branch
  - Values are loaded and stored from arrays
- With a JIT…
  - No dispatch loop
  - Values can be stored in machine registers
  - Only spilled to the stack as needed

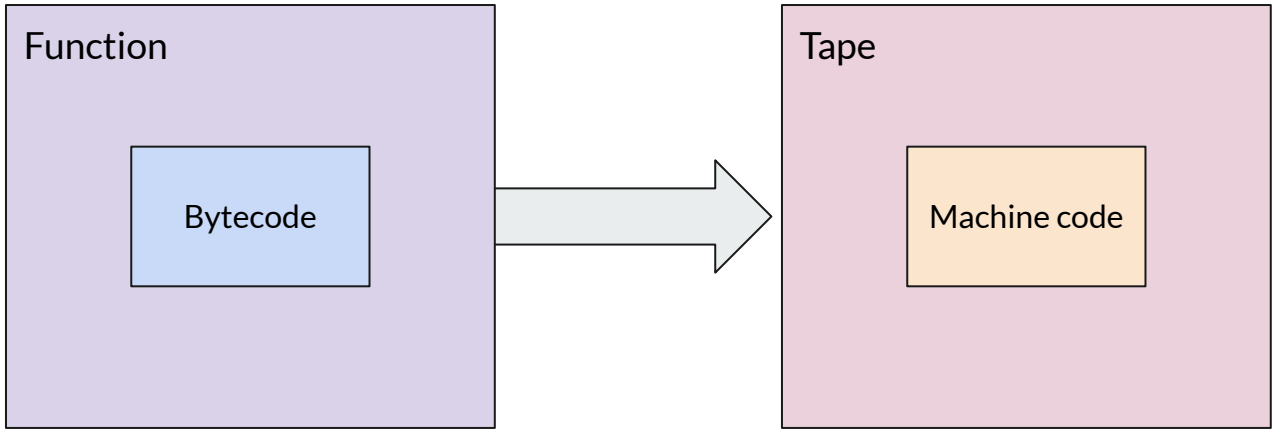
```
for op in tape.iter_asm() {
    match op {
        RegOp::Input(out, i) => {
            v[out] = vars[i as usize];
        }
        RegOp::NegReg(out, arg) => {
            v[out] = -v[arg];
        }
        RegOp::AbsReg(out, arg) => {
            v[out] = v[arg].abs();
        }
        RegOp::RecipReg(out, arg) => {
            v[out] = 1.0 / v[arg];
        }
        RegOp::SqrtReg(out, arg) => {
            v[out] = v[arg].sqrt();
        }
```

How JIT compilation?

- Start from the same bytecode
  - Specialized to *N* registers, instead of 256
  - Bytecode registers correspond to machine registers
- Lots and lots of hand-written assembly:

```
fn build_recip(&mut self, out_reg: u8, lhs_reg: u8) {
    dynasm!(self.0.ops
        ; fmov s7, 1.0
        ; dup v7.s4, v7.s[0]
        ; fdiv V(reg(out_reg)).s4, v7.s4, V(reg(lhs_reg)).s4
    )
}
```

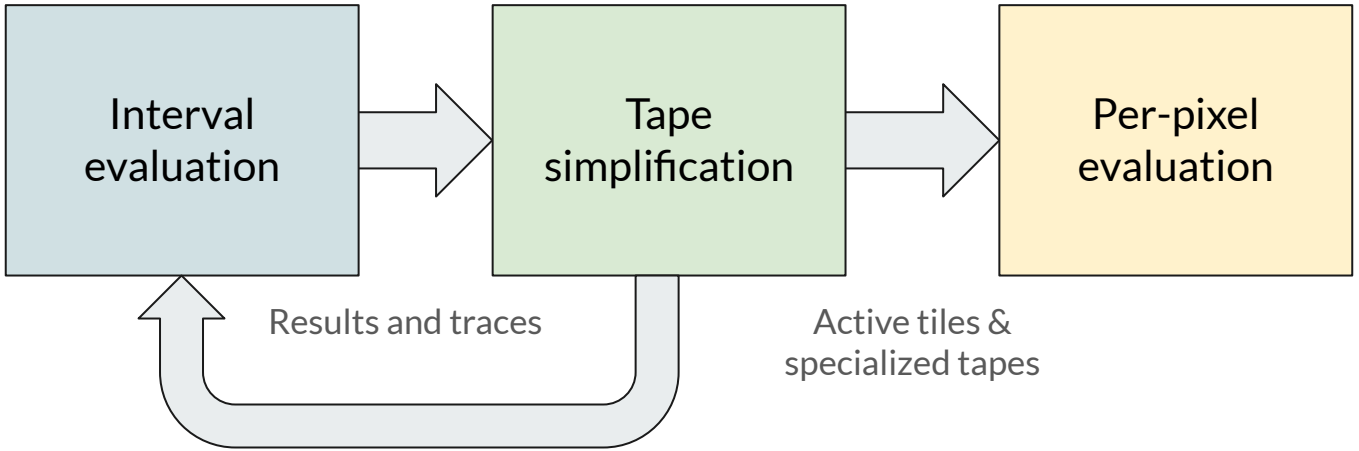
JIT Functions and Tapes



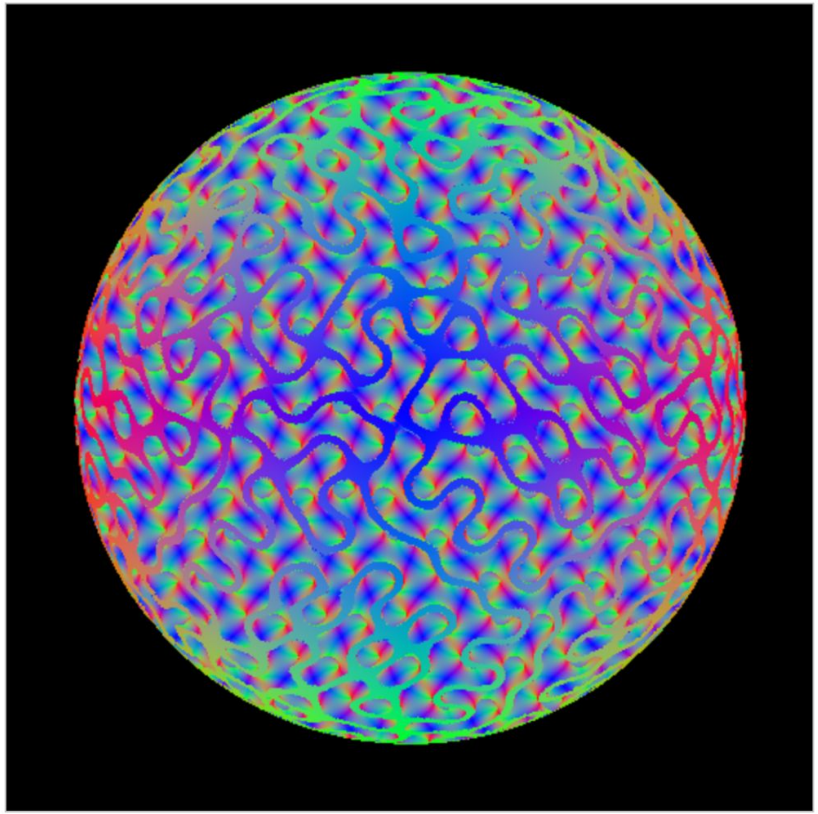
Algorithms

What can we build on this?

2D and 3D rendering



```
1  // Scaling
2  let scale = 30;
3  let x = x * scale;
4  let y = y * scale;
5  let z = z * scale;
6
7  // Rotation
8  let z_ = z;
9  let x_ = x;
10 let a = 0.3;
11 let z = cos(a) * z_ - sin(a) * x_;
12 let x = sin(a) * z_ + cos(a) * x_;
13
14 let gyroid = sin(x)*cos(y) + sin(y)*cos(z) + sin(z)*cos(x);
15 let fill = abs(gyroid) - 0.5;
16
17 let sphere = sqrt(square(x) + square(y) + square(z)) - 25;
18 max(sphere, fill)
```



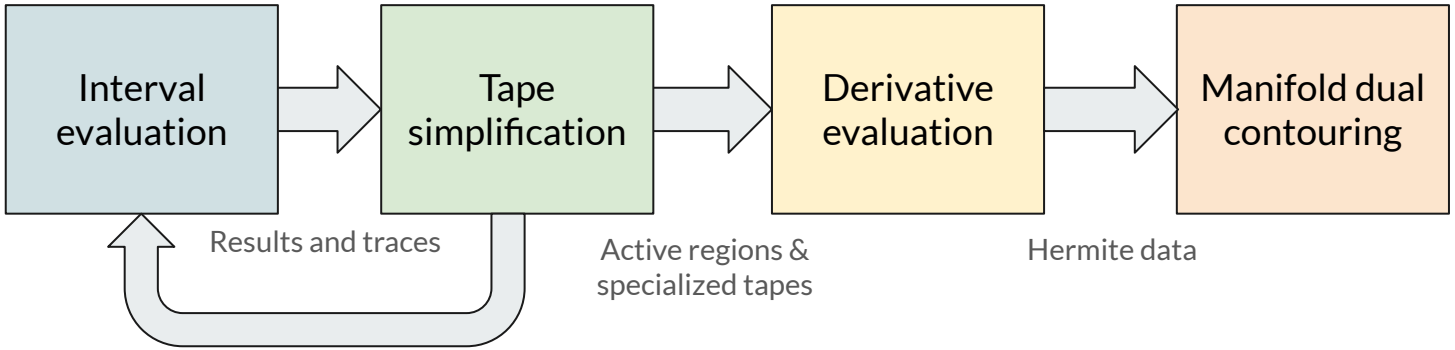
Ok(..)

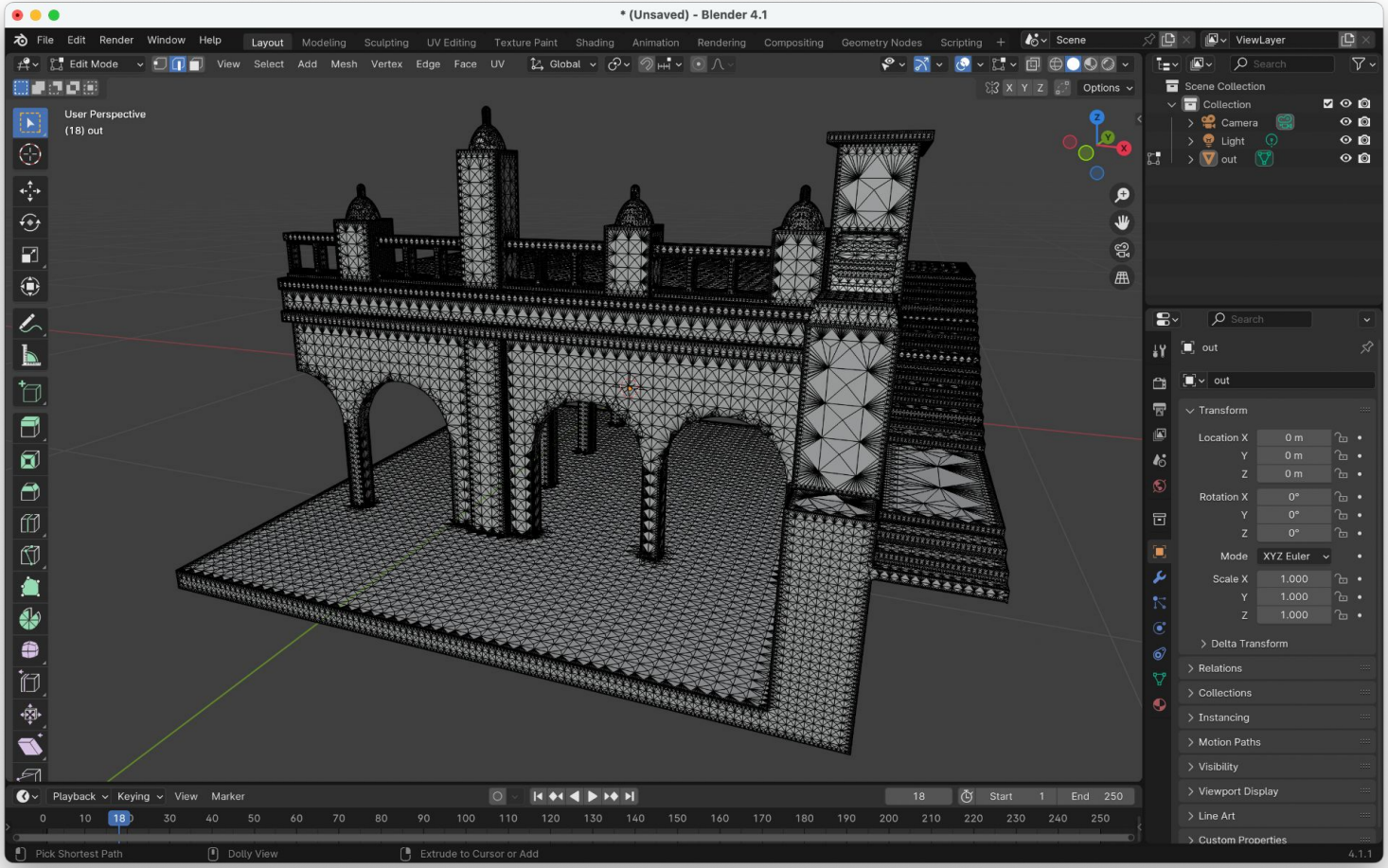
3D (normals)

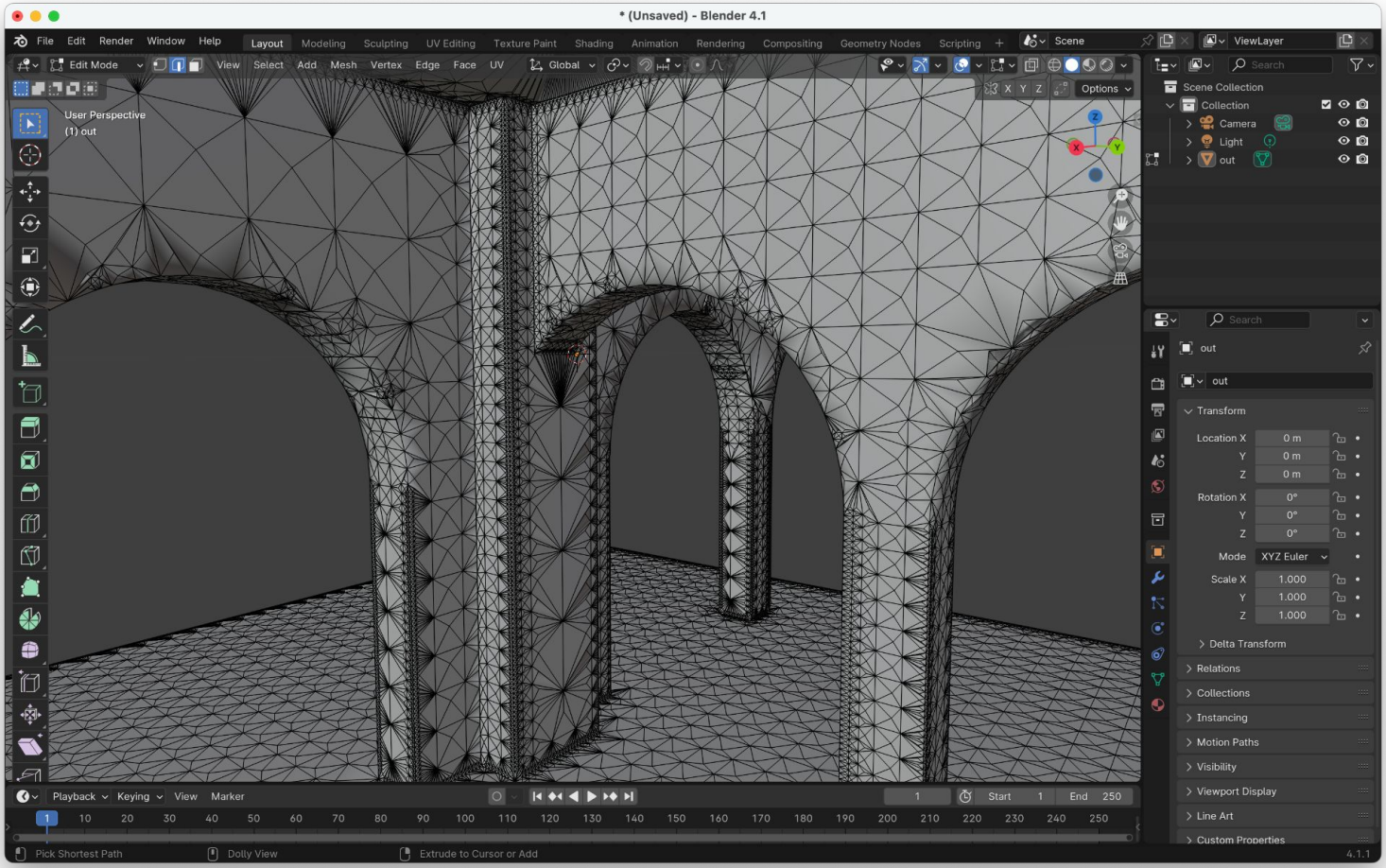
Rendered in 127 ms

mattkeeter.com/projects/fidget/demo

Meshing

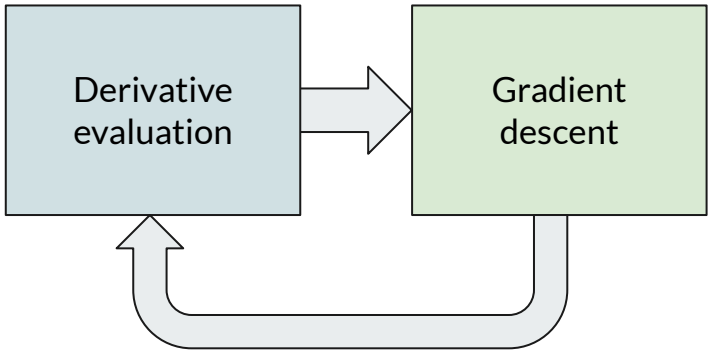

Interval evaluation
Tape simplification
Derivative evaluation
Manifold dual contouring
Results and traces
Active regions & specialized tapes
Hermite data

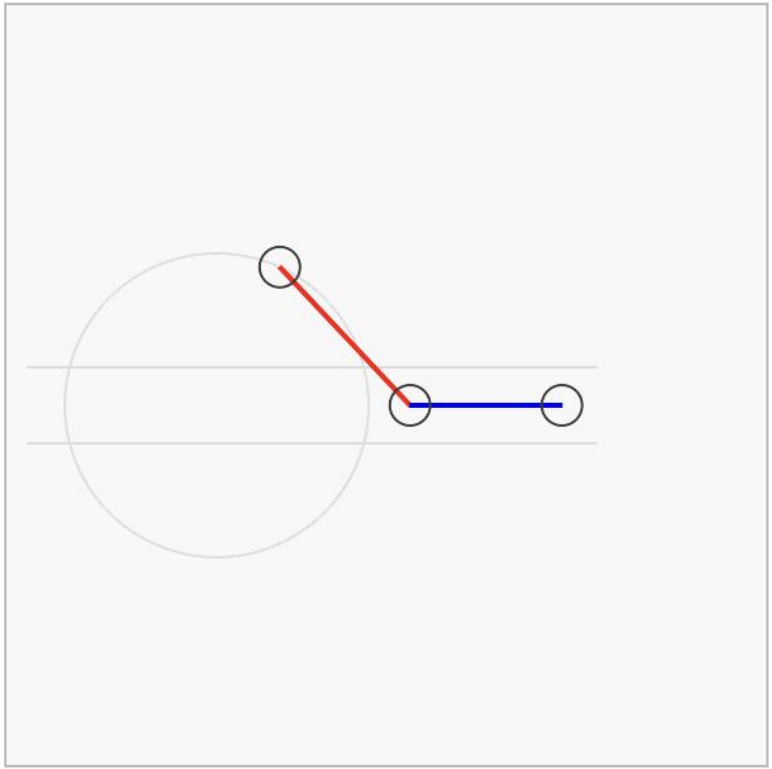

* (Unsaved) - Blender 4.1
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Scene
ViewLayer
Edit Mode View Select Add Mesh Vertex Edge Face UV
Global
Options
User Perspective
(18) out
Scene Collection
Collection
Camera
Light
out
Search
out
Transform
Location X 0 m
Y 0 m
Z 0 m
Rotation X 0°
Y 0°
Z 0°
Mode XYZ Euler
Scale X 1.000
Y 1.000
Z 1.000
Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display
Line Art
Custom Properties
Playback Keying View Marker
18
Start 1
End 250
0 10 18 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250
Pick Shortest Path
Dolly View
Extrude to Cursor or Add
4.1.1


* (Unsaved) - Blender 4.1
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Scene
ViewLayer
Edit Mode View Select Add Mesh Vertex Edge Face UV
Global
Options
User Perspective
(1) out
Scene Collection
Collection
Camera
Light
out
Search
out
Transform
Location X 0 m
Y 0 m
Z 0 m
Rotation X 0°
Y 0°
Z 0°
Mode XYZ Euler
Scale X 1.000
Y 1.000
Z 1.000
Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display
Line Art
Custom Properties
Playback Keying View Marker
Start 1 End 250
Pick Shortest Path
Dolly View
Extrude to Cursor or Add
4.1.1

Constraint solving

- Sum-of-squares minimization
- Each constraint is one expression
- As many input variables as you want
  - Not limited to x, y, z
- Levenberg-Marquardt algorithm
- Not a serious implementation
  - This is just a demo!





mattkeeter.com/projects/fidget/constraints

$$a_x^2 + a_y^2 = 1$$

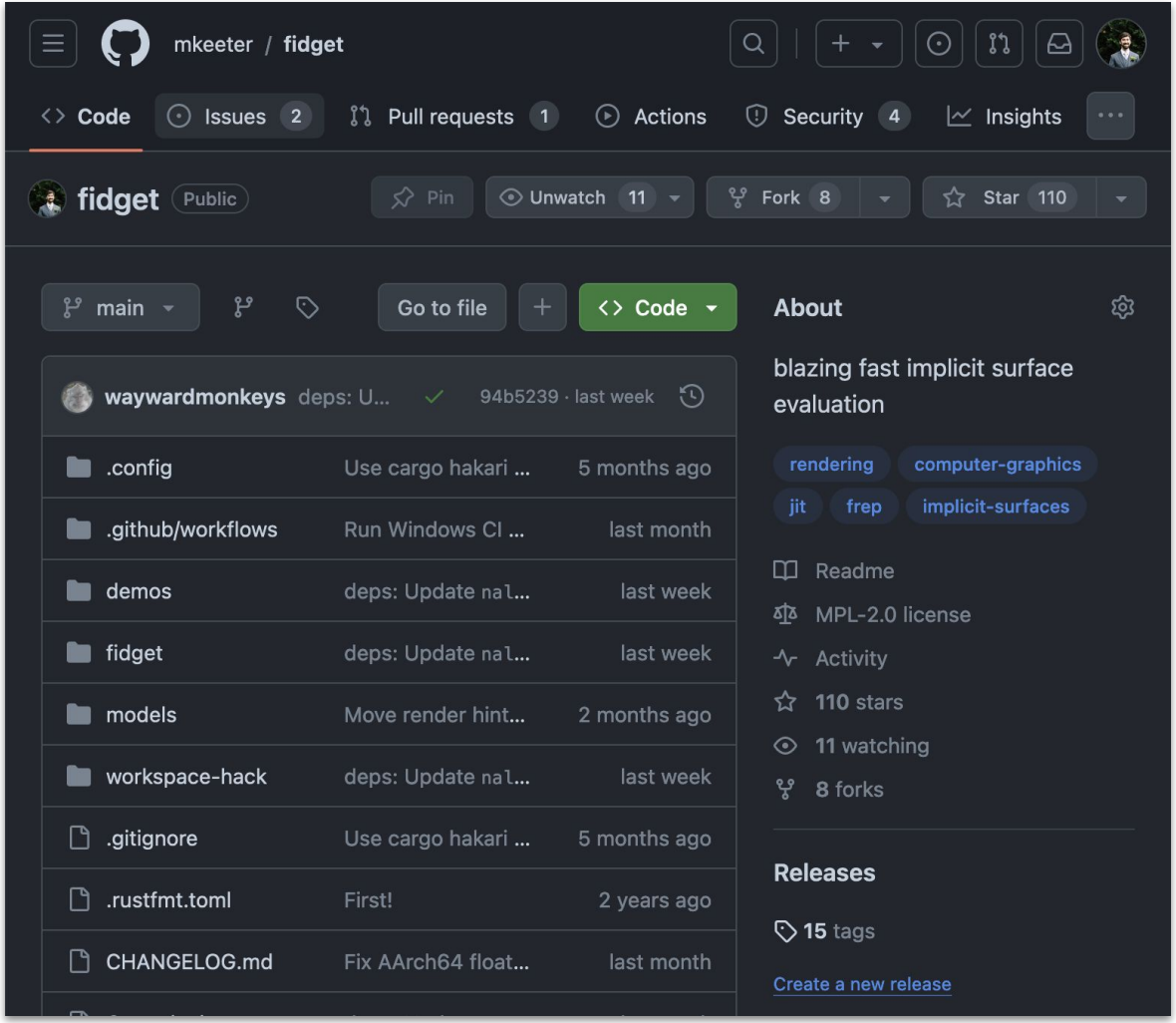
$$(a_x - b_x)^2 + (a_y - b_y)^2 = 2$$

$$b_y = c_y = 0$$

$$c_x - b_x = 1$$

Future possibilities

- GPU-based rendering
  - Probably as a separate crate, because it doesn't fit our traits
- Oracles for black-box objects
  - Meshes, NURBS, etc
  - Tricky to integrate with JIT
- Better meshing?


mkeeter / fidget
Code
Issues 2
Pull requests 1
Actions
Security 4
Insights
fidget
Public
Pin
Unwatch 11
Fork 8
Star 110
main
Go to file
Code
waywardmonkeys deps: U... 94b5239 · last week
.config Use cargo hakari ... 5 months ago
.github/workflows Run Windows CI ... last month
demos deps: Update nal... last week
fidget deps: Update nal... last week
models Move render hint... 2 months ago
workspace-hack deps: Update nal... last week
.gitignore Use cargo hakari ... 5 months ago
.rustfmt.toml First! 2 years ago
CHANGELOG.md Fix AArch64 float... last month
About
blazing fast implicit surface evaluation
rendering
computer-graphics
jit
frep
implicit-surfaces
Readme
MPL-2.0 license
Activity
110 stars
11 watching
8 forks
Releases
15 tags
Create a new release