When Taichi meets Julia

A short introduction to Taichi.jl

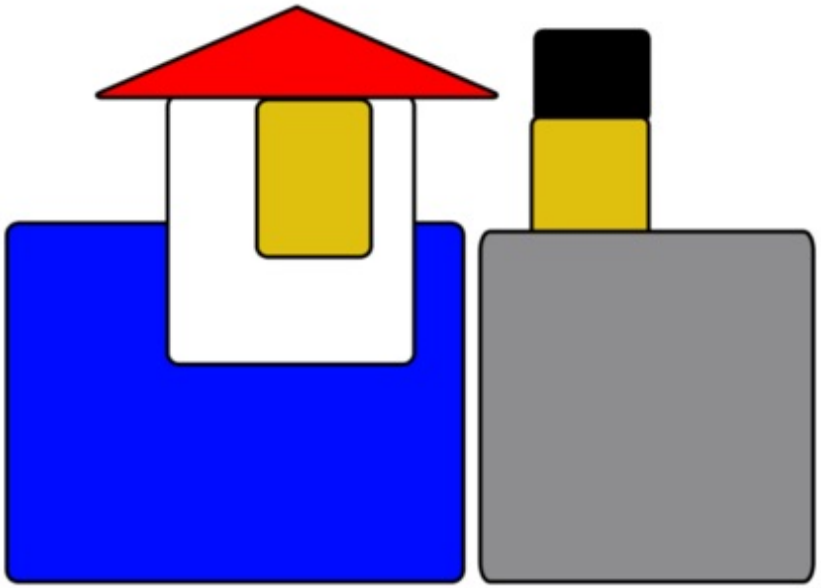
Gabriel Wu (@lucifer1004)

September 15, 2022

Taichi Lang julia

Why Taichi.jl?

- I love Julia...
- I love Taichi...
- Why not both?



First attempt I

Using `PythonCall.jl`, most methods of `Taichi` can be directly called:

```
ti = pyimport("taichi")
ti.init(arch=ti.gpu)
n = 640
pixels = ti.Vector.field(3, dtype=pytype(1.0), shape=(n * 2, n))
```

First attempt II

However, the following would fail:

```
complex_sqr = ti.func(z -> ti.Vector([z[1]^2 - z[2]^2, z[1] * z[0] * 2]))
```

Because when `PythonCall.jl` passes a Julia function to Python, it is more like a pointer and Python will not know much about inside the function. But `Taichi` needs even the source code of the function to generate and transform the AST.

Second attempt

We could write the function in Python, and the following worked!

```
ti_str = """
@ti.kernel
def paint(t: float):
    for (i, j) in pixels:
        c = ti.Vector([(1 + ti.sin(t)) * 0.285, (1 + ti.cos(t)) * 0.1])
        z = ti.Vector([i / n - 1, j / n - 0.5]) * 2
        rgb = ti.Vector([0, 1, 1])
        iterations = 0
        while z.norm() < 20 and iterations < 50:
            z = ti.Vector([z[0]**2 + z[1]**2, z[0] * z[1] * 2]) + c
            iterations += 1
        pixels[i, j] = (1 - iterations * 0.02) * rgb
"""

    namespace = pydict(["ti" => ti, "n" => n, "pixels" => pixels])
    write("__tmp.py", ti_str)
    code = pycompile(ti_str; filename="__tmp.py", mode="exec")
    pyexec(code, namespace)
    paint = namespace.get("paint")
```

Third attempt I

Not elegant, though. Can't we just write the function in Julia? Here is another Julia package by me, `Jl2Py.jl`:

```julia
function f(x, y)
    x + y
end
```

```python
def f(x, y):
    return x + y
```

Third attempt II

Success!

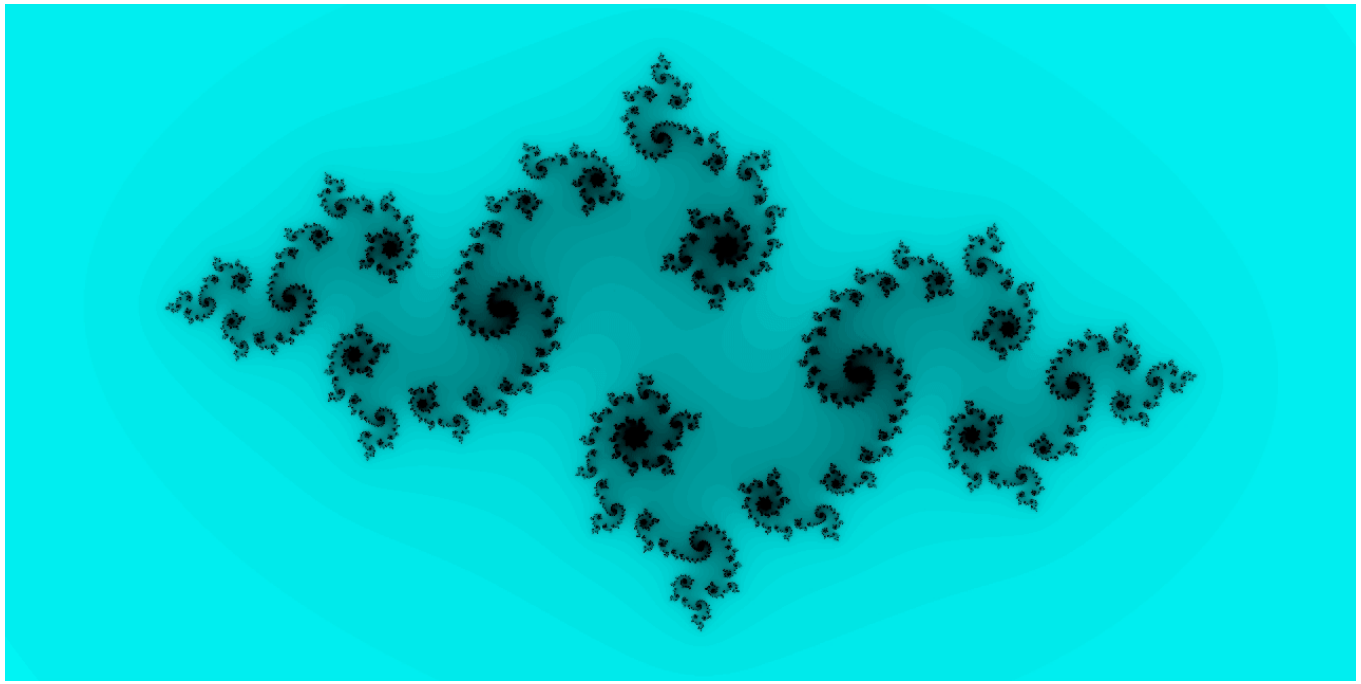
```
paint = @ti_kernel (t::Float64) -> for (i, j) in pixels
    c = ti.Vector([-0.8, ti.cos(t) * 0.2])
    z = ti.Vector([i / n - 1, j / n - 0.5]) * 2
    rgb = ti.Vector([0, 1, 1])
    iterations = 0
    while z.norm() < 20 && iterations < 50
        z = ti.Vector([z[0]^2 - z[1]^2, z[0] * z[1] * 2]) + c
        iterations += 1
        pixels[i, j] = (1 - iterations * 0.02) * rgb
    end
end
```

Third attempt III

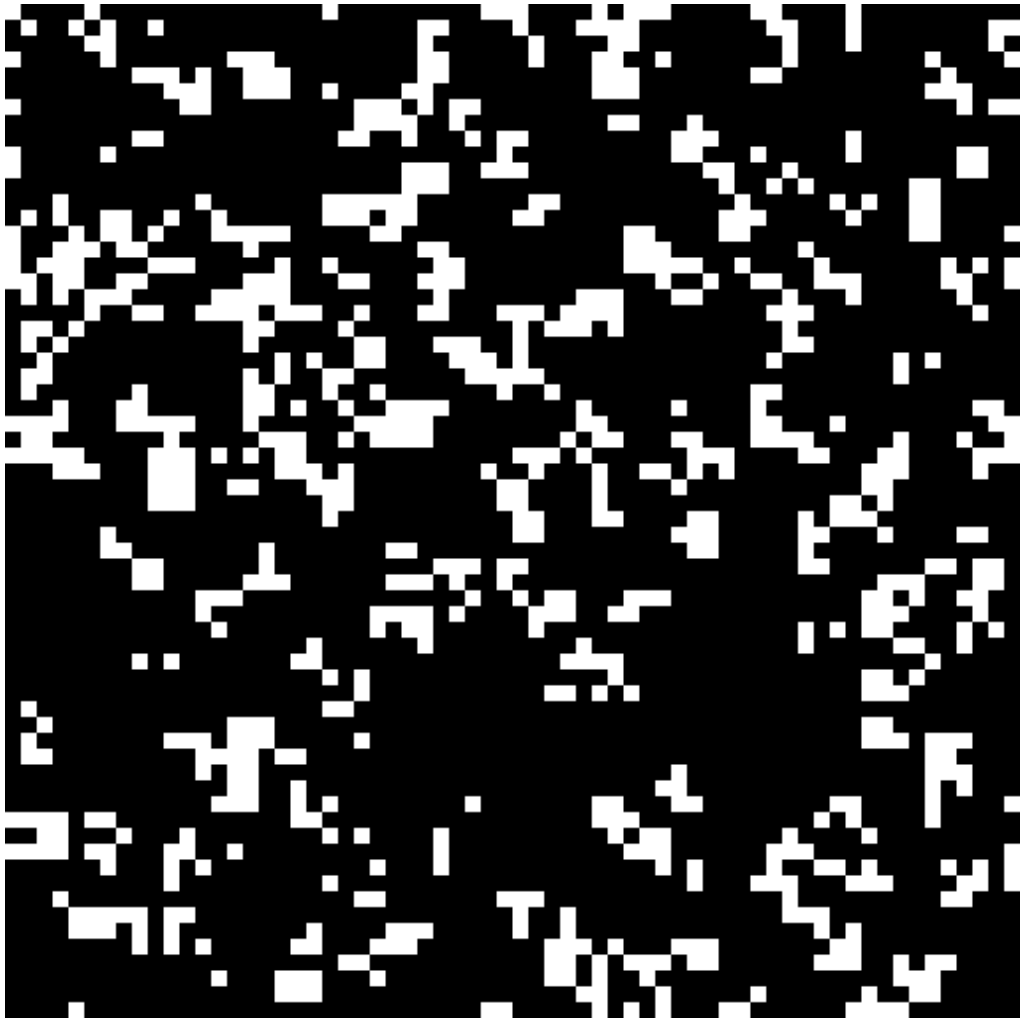
And I have published it as a Julia package: Taichi.jl

```
||==========||                ||==========||                ||==========||
||  Julia   ||    Taichi.jl   ||  Python  ||      Taichi    ||  Taichi  ||
||          ||                ||          || (via PythonCall)||          ||
||  kernel  ||  ===========>  ||  kernel  ||  ===========>  ||  kernel  ||
||==========||                ||==========||                ||==========||
```

Julia Set

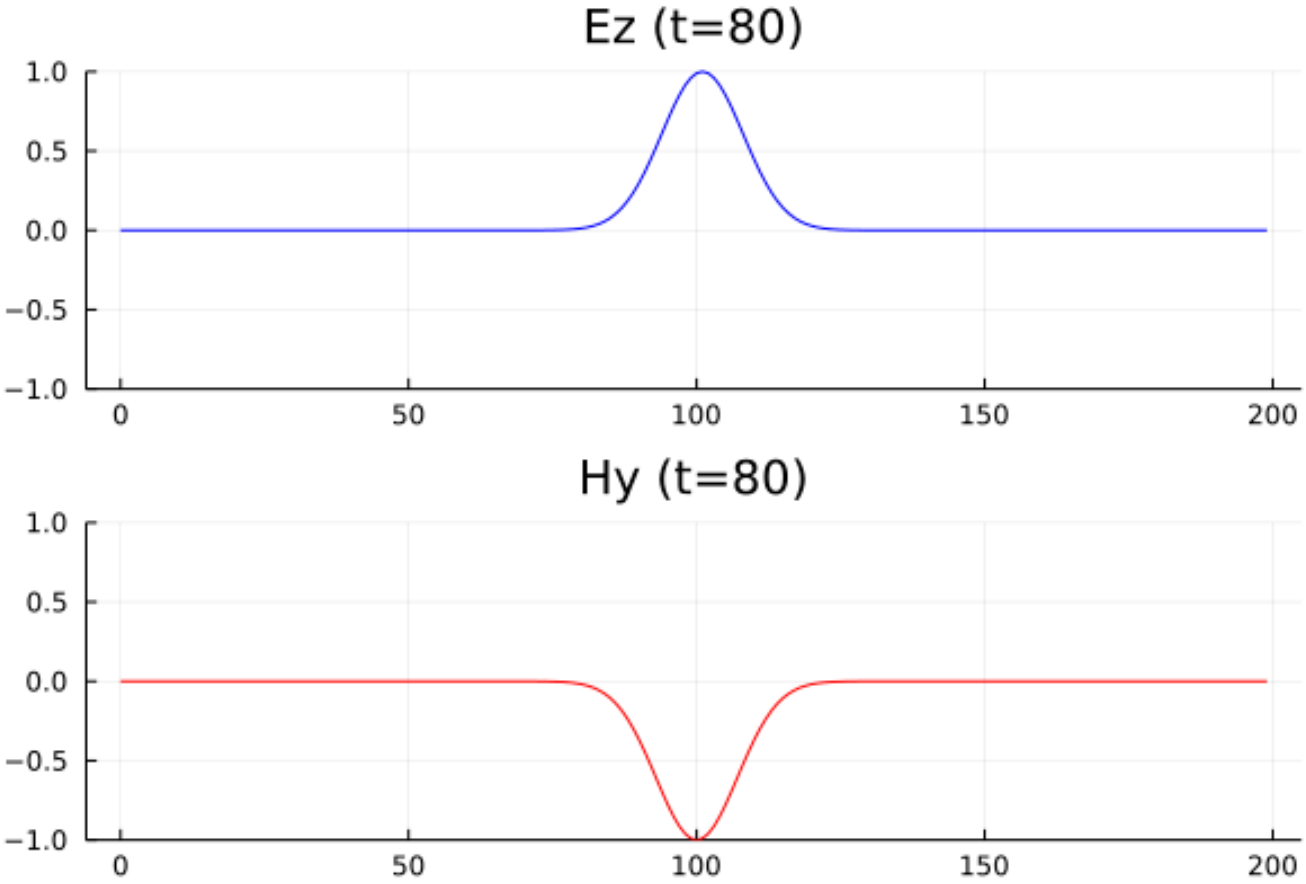


Game of Life



uFDTD

This example lies in lucifer1004/uFDTD-Taichi



The Future

- Bypass Python:
  Transpile directly from Julia AST to Taichi AST
- Memory share:
  Share device memory between Julia and Taichi

Thanks!