太极图形课

第00讲 Introduction

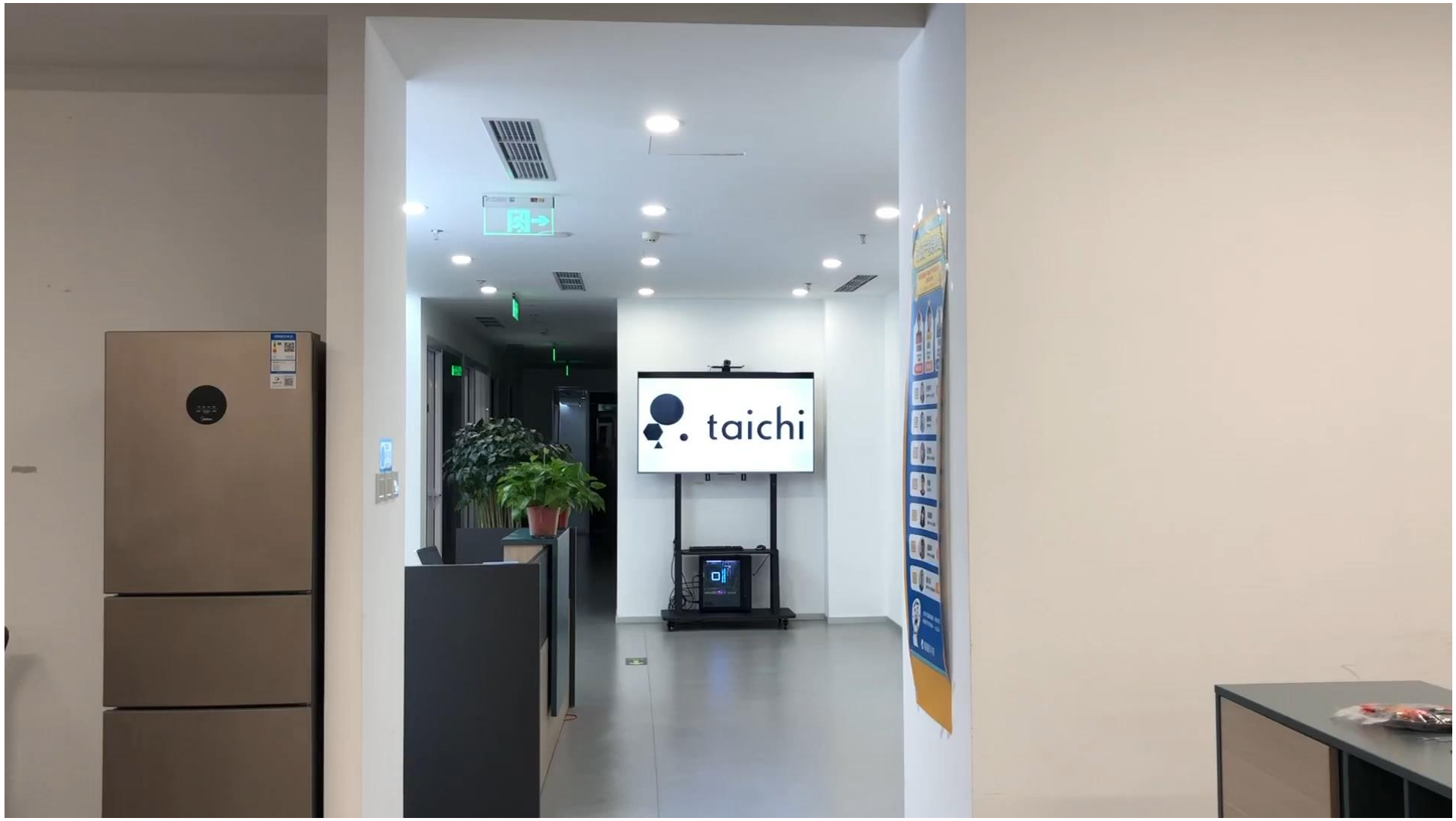
taichi

太极图形课

第00讲 Introduction

taichi

Welcome to the Taichi Graphics Course!



Instructor

- Tiantian Liu 刘天添
  - Senior Research Scientist @ Taichi Graphics
  - tiantianliu.cn
  - ZJU -> UPenn -> MSRA -> Taichi Graphics
  - Real-time Physics, High-performance Geometry Processing, Numerical Algorithms

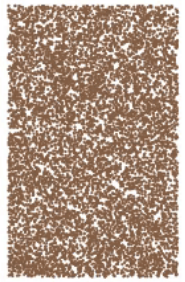
Lead Teaching Assistant

- Peng Yu 禹鹏
  - Ph.D. student @ BUAA
  - https://github.com/FantasyVR
  - Research Intern @ Taichi Graphics
  - Physically-based simulation, Virtual Surgery, Haptics

Teaching Assistants

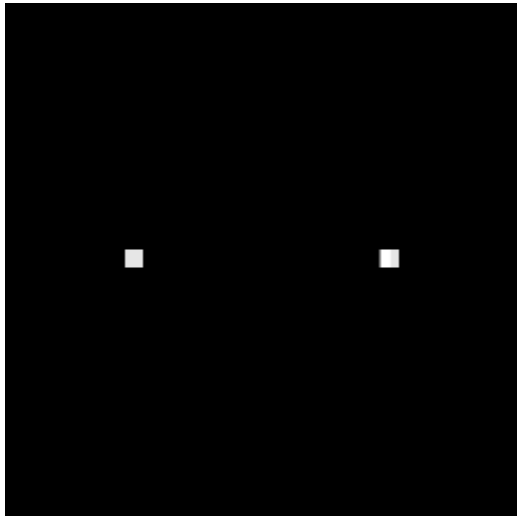
- Chang Yu 余畅 （https://github.com/g1n0st）
- Zhehao Li 李喆昊 （https://github.com/Ricahrd-Li）
- Ling Xie 谢泠 （https://github.com/Jack12xl）

TA Works



[Chang Yu 余畅]

[Zhehao Li 李喆昊]



[Ling Xie 谢泠]

GAMES 201

master · 1 branch · 10 tags · Go to file · Add file · Code

yuanming-hu Update README.md 0abe270 on Nov 23, 2020 20 commits

.gitignore	add mass_spring explicit	11 months ago
.style.yapf	.style.yapf	11 months ago
README.md	Update README.md	10 months ago
mass_spring_explicit.py	update	11 months ago

README.md

GAMES201线上课程：高级物理引擎实战指南2020

最新课程主页：https://yuanming.taichi.graphics/teaching/2020-games201/ (本页面不再更新)

课件下载（讲义与代码）：https://forum.taichi.graphics/t/topic/272

课程回放地址：https://www.bilibili.com/video/BV1ZK411H7Hc

课程直播地址: http://webinar.games-cn.org （直播结束后Bilibili有回放，请点上一个链接）

建议前置课程： 高等数学、Python或任何一门程序设计语言

课程安排： 共10节课，每周一次。2020年6月1日开始，时间为北京时间晚上8：30-10：00。

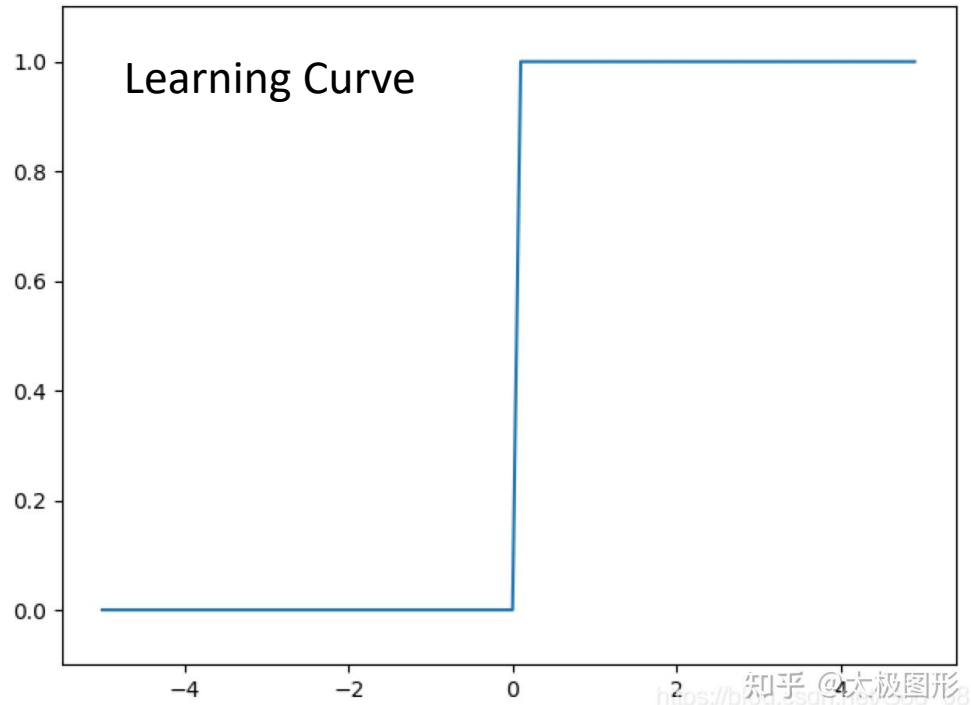
课程作业： 课程共有三个开放项目。自愿完成。开放项目可以1-3人组队。

开放项目以同学们自由发挥为主，同学们可以自己将代码分享到 https://forum.taichi.graphics/ 论坛进行讨论，互相切磋。每次我们会选择有代表性的几个项目会在下次上课展示、点评。课程最后会进行最终点评、线上颁奖。

About

Advanced Physics Engines 2020: A Hands-on Tutorial

Readme



The Taichi Graphics Course - S1

- A 0 to 1 course to learn graphics with Taichi.
- A Hands-on tutorial to code your first Taichi program.
- Cover basic ideas in rendering / simulation.

Syllabus

- 09/14: Intro <- We are here today ☺
- 09/22 (Wed.): Hello world: basis in Taichi
- 09/28: Metaprogramming and object-oriented programming
- 10/12: Advanced data types and data structures
- 10/19: Debugging and optimizing your Taichi code

Syllabus (cont’d)

- 10/26: Procedural Animation
- 11/02: Render 01: basis in ray tracing
- 11/09: Render 02: implement your first ray tracer

Syllabus (cont’d)

- 11/16: Deformable 01: Spatial and temporal discretization
- 11/23: Deformable 02: Implicit time integration
- 11/30: Fluid 01: Lagrangian view
- 12/07: Fluid 02: Eulerian view
- 12/14: (Secret) Guest speaker

Homework and Final Project

- Optional homework assignments after each course.
  - Code on your local Taichi build or at Taichi Zoo

- Mandatory final project **if** you want a certificate.
  - Deadline **Jan. 3rd 2022**
  - Submit using a **private** repo at GitHub/Gitee. Invite tgc01@taichi.graphics to your repo
  - Small-scaled teamwork (≤ 3) is welcome, manage your Git commits with care
  - Release your final project at will, **after** the grading period
  - Gifts and job/intern opportunity await

Where shall I raise questions?

- Live
  - @微信群
- Forum
  - https://forum.taichi.graphics/ - 太极图形课
- Office Hours
  - 19:00 – 20:00 Every **Thursday**
  - 10-20 mins for frequently asked questions
  - Online meeting for individual ones, @腾讯会议

Graphics

Me & Computer Graphics

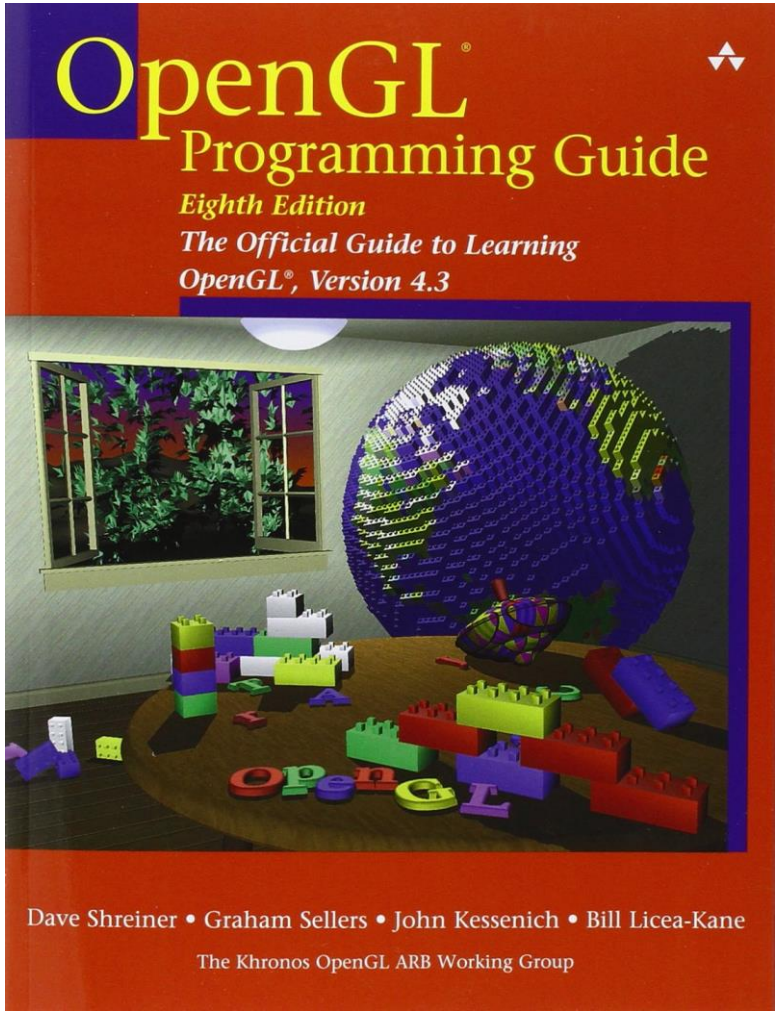
[大航海时代2, 1993]

[仙剑奇侠传, 1995]

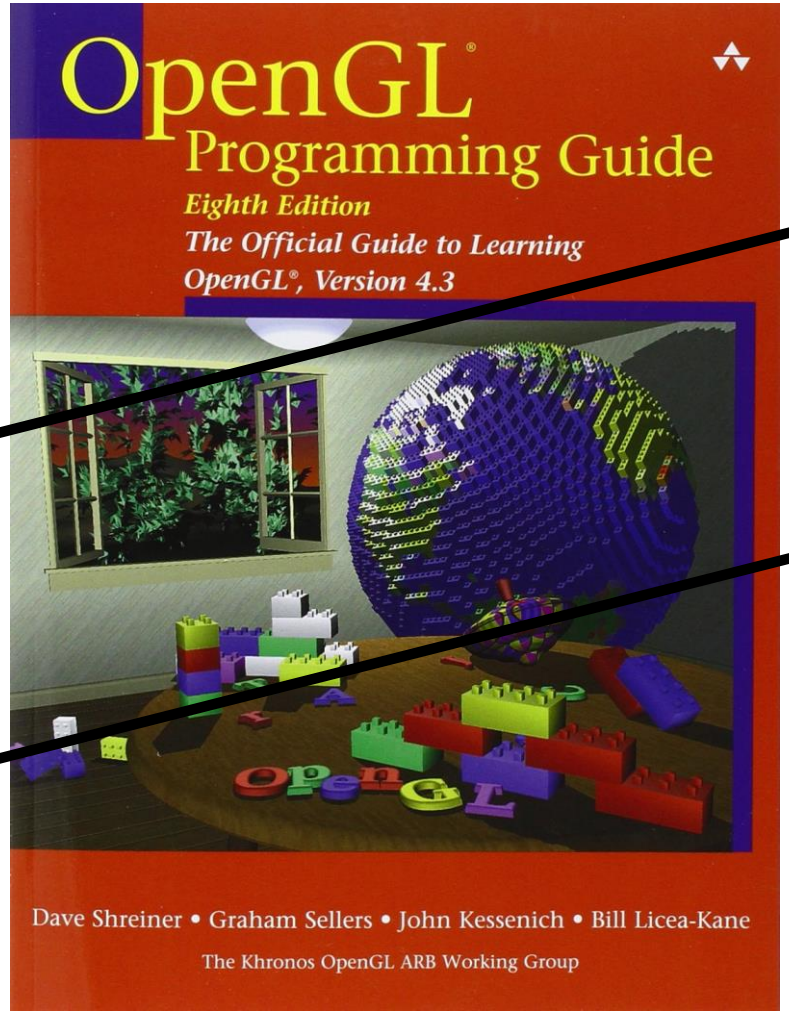
Me & Computer Graphics

[Final Fantasy VIII, 1999]

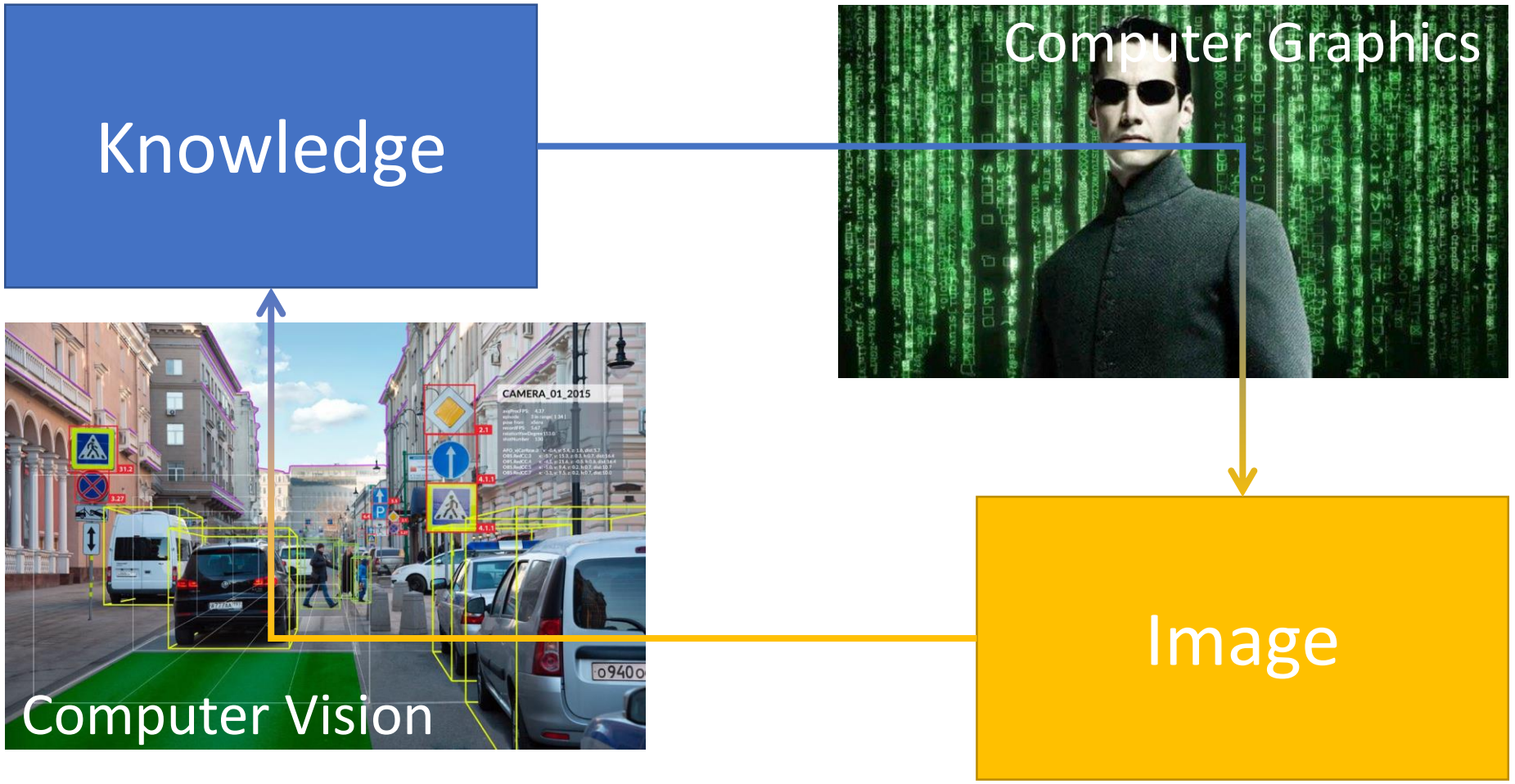
Computer Graphics (When I was a student)



~~Computer Graphics (When I was a student)~~



The Knowledge-Image Cycle



The G in Graphics is for Generation!

The G in Graphics is for Generation!

Rules

Content

The G in Graphics is for Generation!

Laws of physics

Content

[pbrt.org]

The G in Graphics is for Generation!

Laws of physics

Content

[Scalable Galerkin Multigrid]

The G in Graphics is for Generation!

Data

Content

[CycleGAN]

What can we generate?

We generate **movies**

[Frozen, 2013]

We generate **movies**

[哪吒之魔童降世，2019]

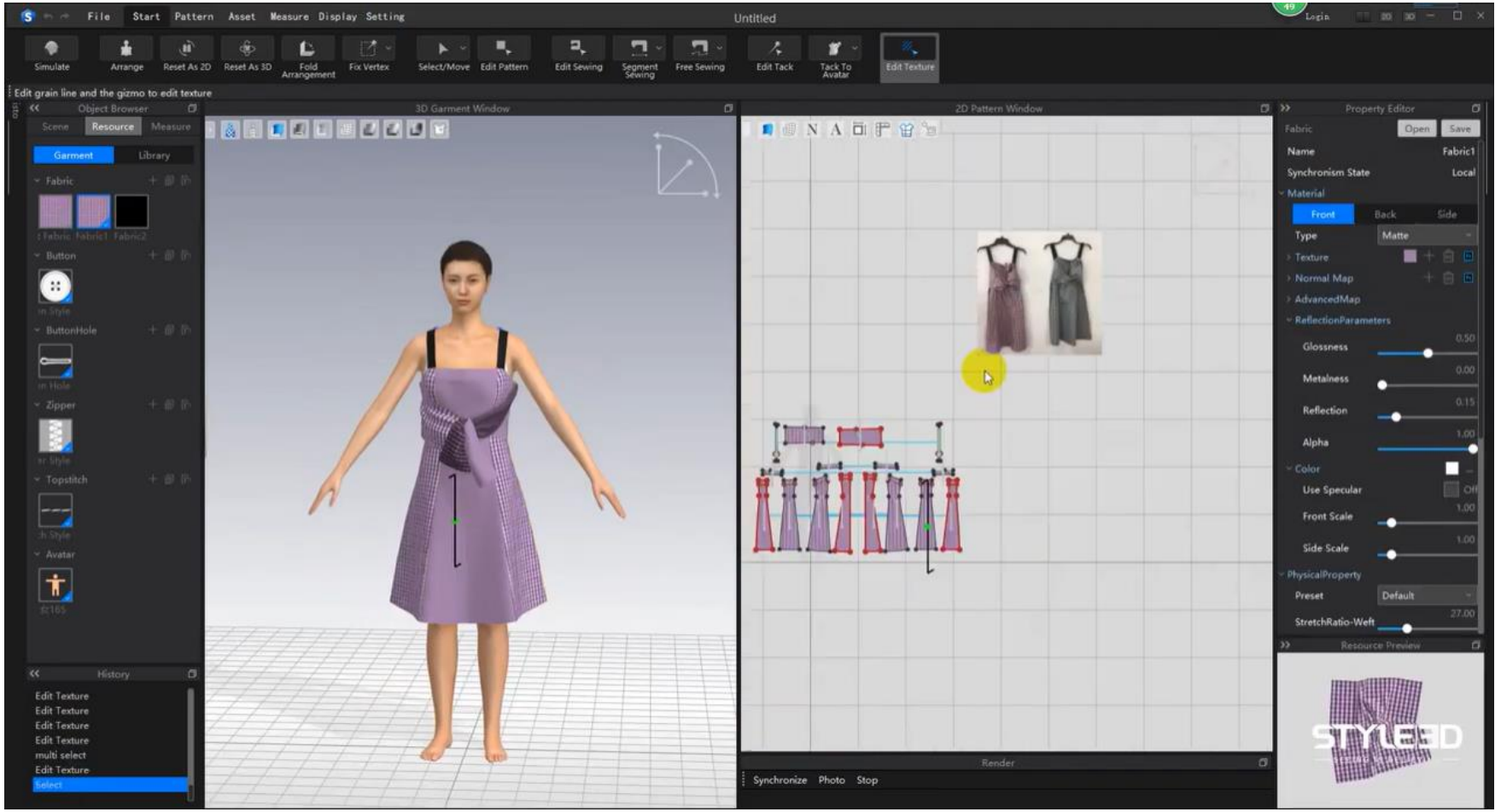
[白蛇：缘起, 2019]

[青蛇：劫起, 2021]

We generate **games**

[黑神话：悟空, 2021]

We generate **design tools**



[Style3D 凌迪科技]

We generate the **Reality** in the virtual space

[Microsoft HoloLens 2]

Maybe ultimately…

[Sword Art Online, 2012]

… but apparently we are not there yet. Why?

- Lack of virtual contents
  - lack of handy tools

… but apparently we are not there yet. Why?

- Lack of virtual contents
  - lack of handy tools

- Hard to code an efficient graphics program
  - Hard to reuse someone else's code
  - Performance highly depends on coding skills

The **Taichi** Programming Language

What do we want from a programming language for computer graphics?

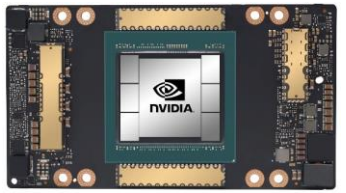
- Productivity
- Portability
- Performance

What do we want from a programming language for computer graphics?

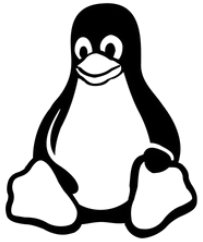
- Productivity
- Portability
- Performance

What do we want from a programming language for computer graphics?

- Productivity
- Portability
- Performance

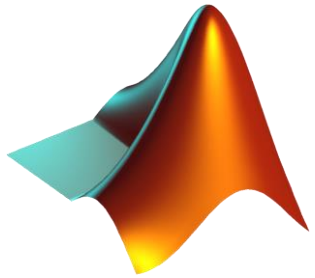


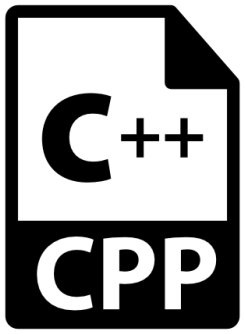


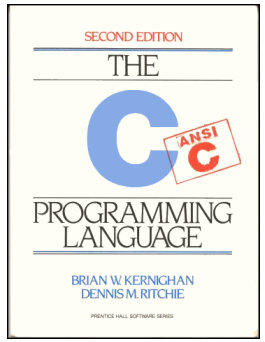


What do we want from a programming language for computer graphics?

- Productivity
- Portability
- Performance



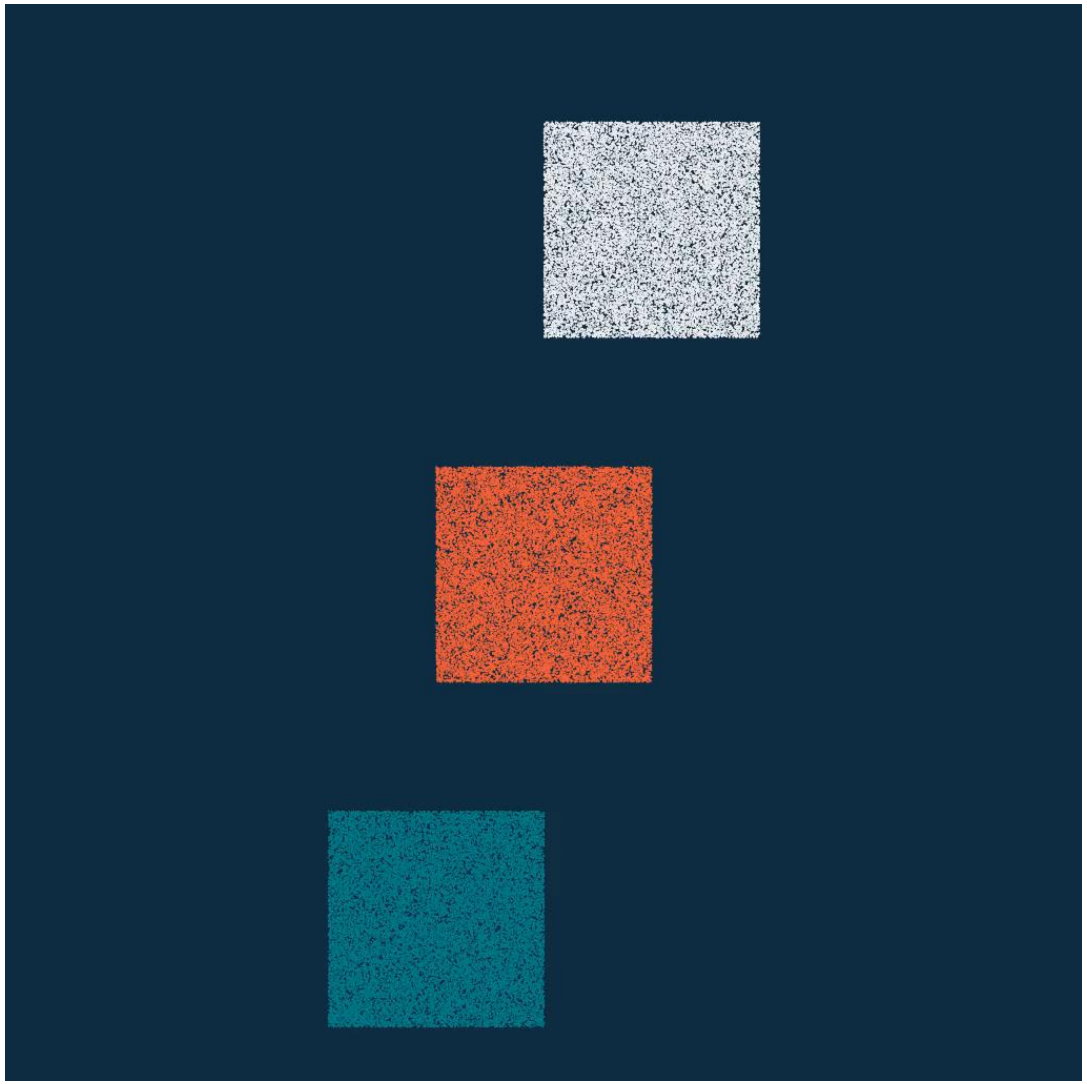




Taichi – A DSL for Computer Graphics

- Productivity
  - Friendly learning curve
  - Shorter code, higher perf.
- Portability
  - Multi-backend support
- Performance
  - Optimized for bandwidth, locality and load balancing

Taichi – A DSL for Computer Graphics



99行代码的《冰雪奇缘》

胡渊鸣
知乎十年新知答主

已关注

编辑推荐

知乎日报、fifizoo、东半球吃货等 26,141 人赞同了该文章

——闲聊物理引擎，可微编程，SIGGRAPH生产力难题，与Taichi编程语言

2021年4月29日更新：Taichi社区的 VictoriaCity 同学关于相关话题的精美视频介绍：

99 行代码写出来的特效，《冰雪奇缘》都用过？揭秘物理引擎中的算法和生产力工...
www.bilibili.com/video/BV19K4y1P7fb?sh...

目录

~~“7行《冰雪奇缘》”~~

```
#include <your_awesome_mpm_lib>

int main(int argc, char **argv)
{
    AWESOME_LIB::run_mpm(argc, argv);
    return 0;
}
```

“99行《冰雪奇缘》”

```python
import numpy as np

import taichi as ti

ti.init(arch=ti.gpu)  # Try to run on GPU
quality = 1  # Use a larger value for higher-res simulations
n_particles, n_grid = 9000 * quality**2, 128 * quality
dx, inv_dx = 1 / n_grid, float(n_grid)
dt = 1e-4 / quality
p_vol, p_rho = (dx * 0.5)**2, 1
p_mass = p_vol * p_rho
E, nu = 0.1e4, 0.2  # Young's modulus and Poisson's ratio
mu_0, lambda_0 = E / (2 * (1 + nu)), E * nu / (
    (1 + nu) * (1 - 2 * nu))  # Lame parameters
x = ti.Vector.field(2, dtype=float, shape=n_particles)  # position
v = ti.Vector.field(2, dtype=float, shape=n_particles)  # velocity
C = ti.Matrix.field(2, 2, dtype=float,
                    shape=n_particles)  # affine velocity field
F = ti.Matrix.field(2, 2, dtype=float,
                    shape=n_particles)  # deformation gradient
material = ti.field(dtype=int, shape=n_particles)  # material id
Jp = ti.field(dtype=float, shape=n_particles)  # plastic deformation
grid_v = ti.Vector.field(2, dtype=float,
                         shape=(n_grid, n_grid))  # grid node momentum/velocity
grid_m = ti.field(dtype=float, shape=(n_grid, n_grid))  # grid node mass
```

Data

Computation

```python
@ti.kernel
def substep():
    for i, j in grid_m:
        grid_v[i, j] = [0, 0]
        grid_m[i, j] = 0
    for p in x:  # Particle state update and scatter to grid (P2G)
        base = (x[p] * inv_dx - 0.5).cast(int)
        fx = x[p] * inv_dx - base.cast(float)
        # Quadratic kernels  [http://mpm.graphics   Eqn. 123, with x=fx, fx-1,fx-2]
        w = [0.5 * (1.5 - fx)**2, 0.75 - (fx - 1)**2, 0.5 * (fx - 0.5)**2]
        F[p] = (ti.Matrix.identity(float, 2) +
                dt * C[p]) @ F[p]  # deformation gradient update
        h = ti.exp(
            10 *
            (1.0 -
             Jp[p]))  # Hardening coefficient: snow gets harder when compressed
        if material[p] == 1:  # jelly, make it softer
            h = 0.3
        mu, la = mu_0 * h, lambda_0 * h
        if material[p] == 0:  # liquid
            mu = 0.0
        U, sig, V = ti.svd(F[p])
        J = 1.0
        for d in ti.static(range(2)):
            new_sig = sig[d, d]
            if material[p] == 2:  # Snow
                new_sig = min(max(sig[d, d], 1 - 2.5e-2),
                              1 + 4.5e-3)  # Plasticity
            Jp[p] *= sig[d, d] / new_sig
            sig[d, d] = new_sig
            J *= new_sig
        if material[
                p] == 0:  # Reset deformation gradient to avoid numerical instability
            F[p] = ti.Matrix.identity(float, 2) * ti.sqrt(J)
        elif material[p] == 2:
            F[p] = U @ sig @ V.transpose(
            )  # Reconstruct elastic deformation gradient after plasticity
        stress = 2 * mu * (F[p] - U @ V.transpose()) @ F[p].transpose(
        ) + ti.Matrix.identity(float, 2) * la * J * (J - 1)
        stress = (-dt * p_vol * 4 * inv_dx * inv_dx) * stress
        affine = stress + p_mass * C[p]
        for i, j in ti.static(ti.ndrange(
                3, 3)):  # Loop over 3x3 grid node neighborhood
            offset = ti.Vector([i, j])
            dpos = (offset.cast(float) - fx) * dx
            weight = w[i][0] * w[j][1]
            grid_v[base + offset] += weight * (p_mass * v[p] + affine @ dpos)
            grid_m[base + offset] += weight * p_mass
    for i, j in grid_m:
        if grid_m[i, j] > 0:  # No need for epsilon here
            grid_v[i,
                   j] = (1 / grid_m[i, j]) * grid_v[i,
                                                    j]  # Momentum to velocity
            grid_v[i, j][1] -= dt * 50  # gravity
            if i < 3 and grid_v[i, j][0] < 0:
                grid_v[i, j][0] = 0  # Boundary conditions
            if i > n_grid - 3 and grid_v[i, j][0] > 0: grid_v[i, j][0] = 0
            if j < 3 and grid_v[i, j][1] < 0: grid_v[i, j][1] = 0
            if j > n_grid - 3 and grid_v[i, j][1] > 0: grid_v[i, j][1] = 0
    for p in x:  # grid to particle (G2P)
        base = (x[p] * inv_dx - 0.5).cast(int)
        fx = x[p] * inv_dx - base.cast(float)
        w = [0.5 * (1.5 - fx)**2, 0.75 - (fx - 1.0)**2, 0.5 * (fx - 0.5)**2]
        new_v = ti.Vector.zero(float, 2)
        new_C = ti.Matrix.zero(float, 2, 2)
        for i, j in ti.static(ti.ndrange(
                3, 3)):  # loop over 3x3 grid node neighborhood
            dpos = ti.Vector([i, j]).cast(float) - fx
            g_v = grid_v[base + ti.Vector([i, j])]
            weight = w[i][0] * w[j][1]
            new_v += weight * g_v
            new_C += 4 * inv_dx * weight * g_v.outer_product(dpos)
        v[p], C[p] = new_v, new_C
        x[p] += dt * v[p]  # advection
```

Visualization

```python
group_size = n_particles // 3

@ti.kernel
def initialize():
    for i in range(n_particles):
        x[i] = [
            ti.random() * 0.2 + 0.3 + 0.10 * (i // group_size),
            ti.random() * 0.2 + 0.05 + 0.32 * (i // group_size)
        ]
        material[i] = i // group_size  # 0: fluid 1: jelly 2: snow
        v[i] = ti.Matrix([0, 0])
        F[i] = ti.Matrix([[1, 0], [0, 1]])
        Jp[i] = 1

initialize()
gui = ti.GUI("Taichi MLS-MPM-99", res=512, background_color=0x112F41)
while not gui.get_event(ti.GUI.ESCAPE, ti.GUI.EXIT):
    for s in range(int(2e-3 // dt)):
        substep()
    gui.circles(x.to_numpy(),
                radius=1.5,
                palette=[0x068587, 0xED553B, 0xEEEEF0],
                palette_indices=material)
    gui.show()  # Change to gui.show(f'{frame:06d}.png') to write images to disk
```

Taichi – A DSL for Computer Graphics

- Productivity
  - Friendly learning curve
  - Shorter code, higher perf.
- Portability
  - Multi-backend support
- Performance
  - Optimized for bandwidth, locality and load balancing
- Spatially sparse computation / Differentiable programming / Metaprogramming etc.

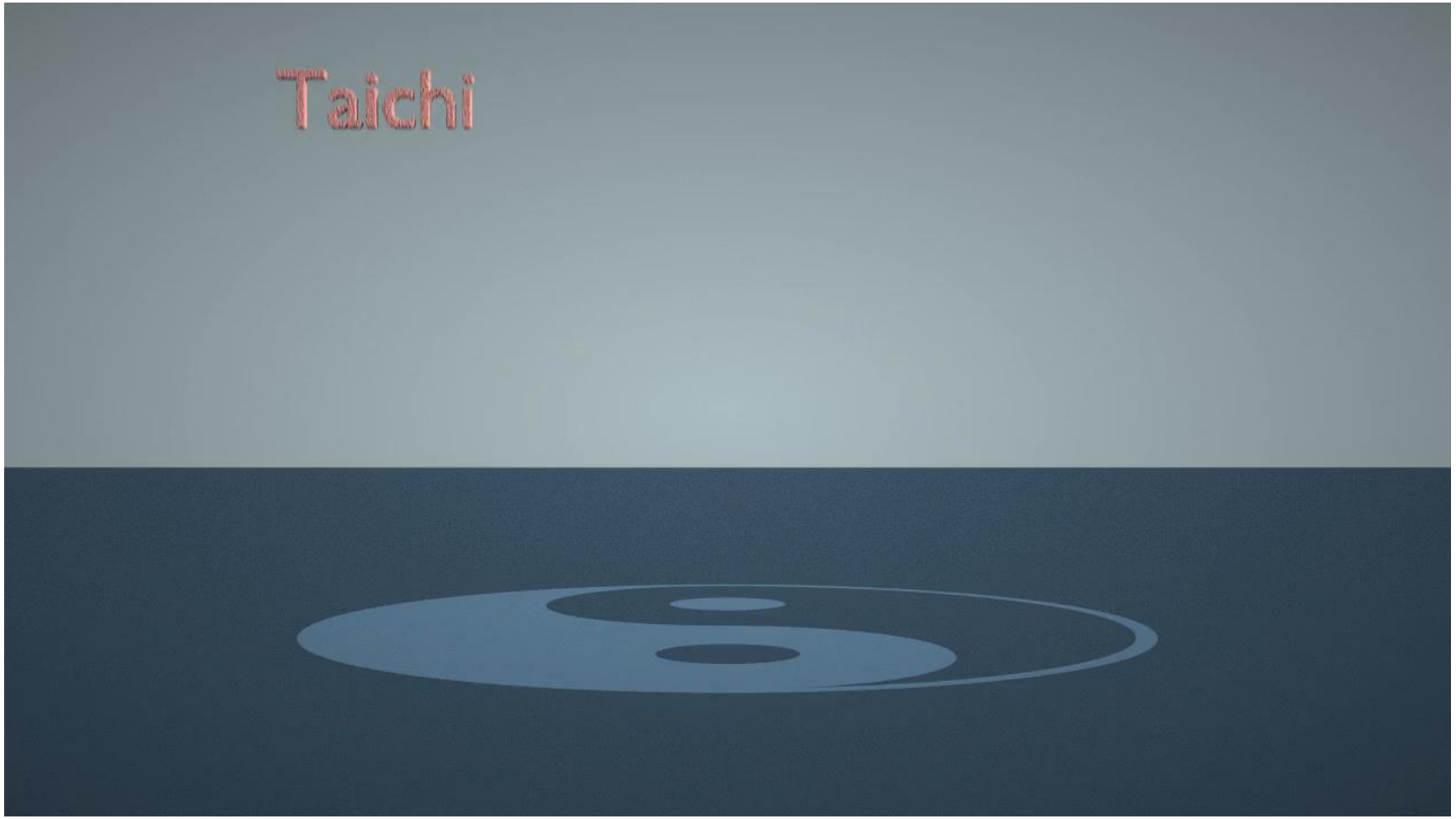
Taichi – A DSL for Computer Graphics

- Productivity
  - Friendly learning curve
  - Shorter code, higher perf.
- Portability
  - Multi-backend support
- Performance
  - Optimized for bandwidth, locality and load balancing
- Spatially sparse computation / Differentiable programming / Metaprogramming etc.
- Open Source

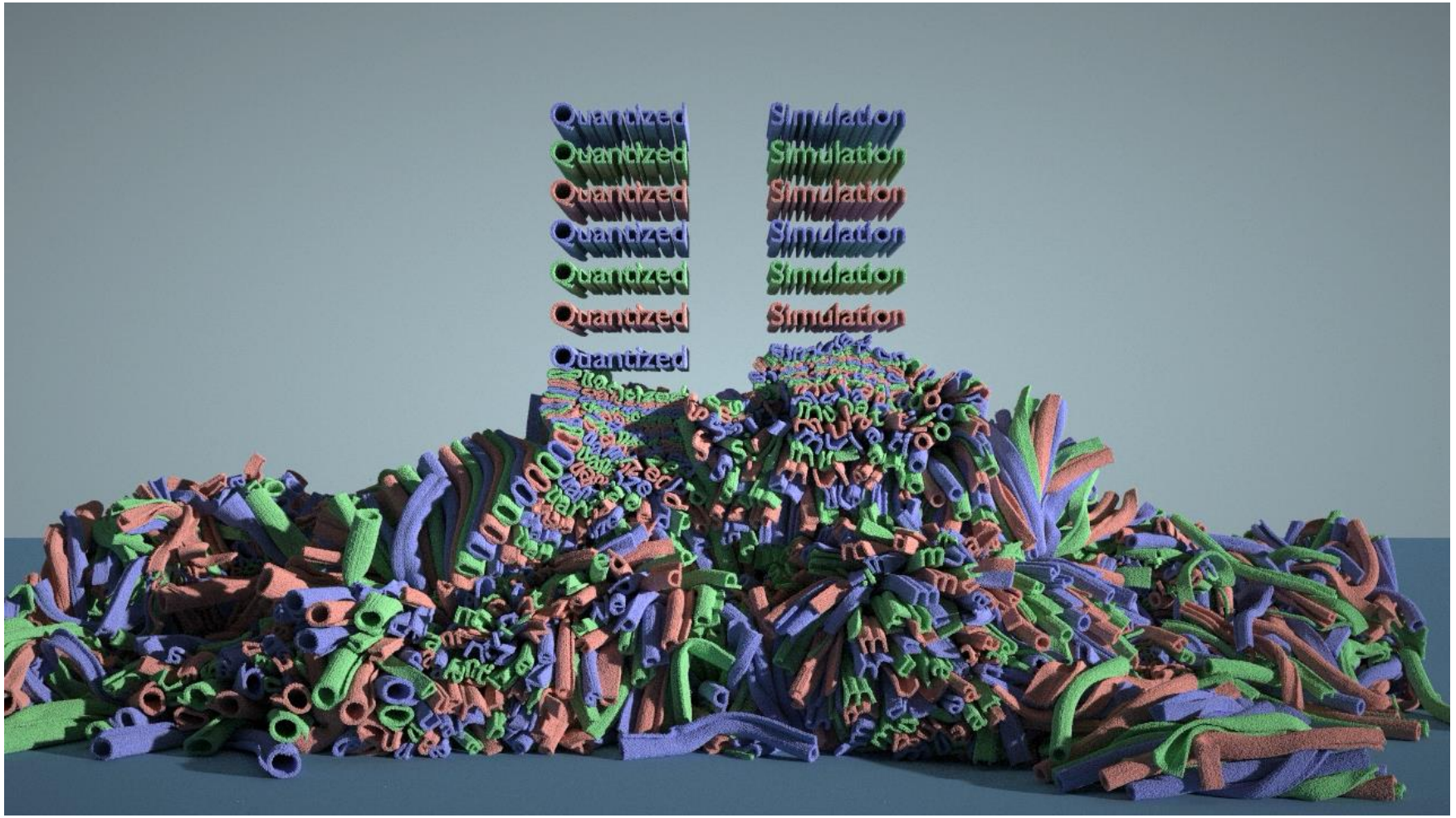
Taichi – A DSL for Computer Graphics

- Productivity
- Portability
- Performance
- Spatially sparse computation / Differentiable programming / Metaprogramming etc.
- Open Source

What you can do using Taichi



What you can do using Taichi



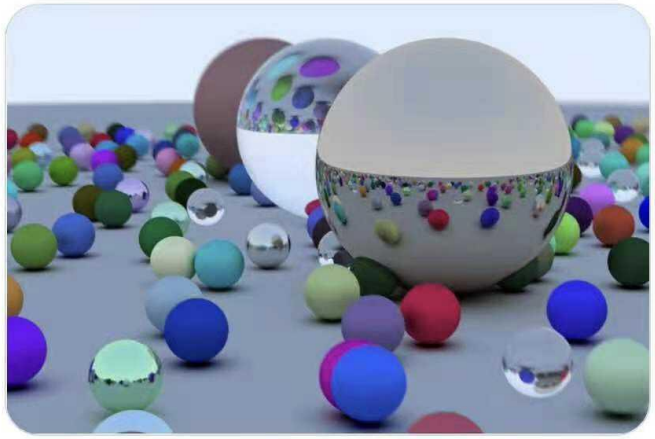
What you can do using Taichi

Peter Shirley 🔮🧢 Retweeted

Brian @bhsavery · 4h

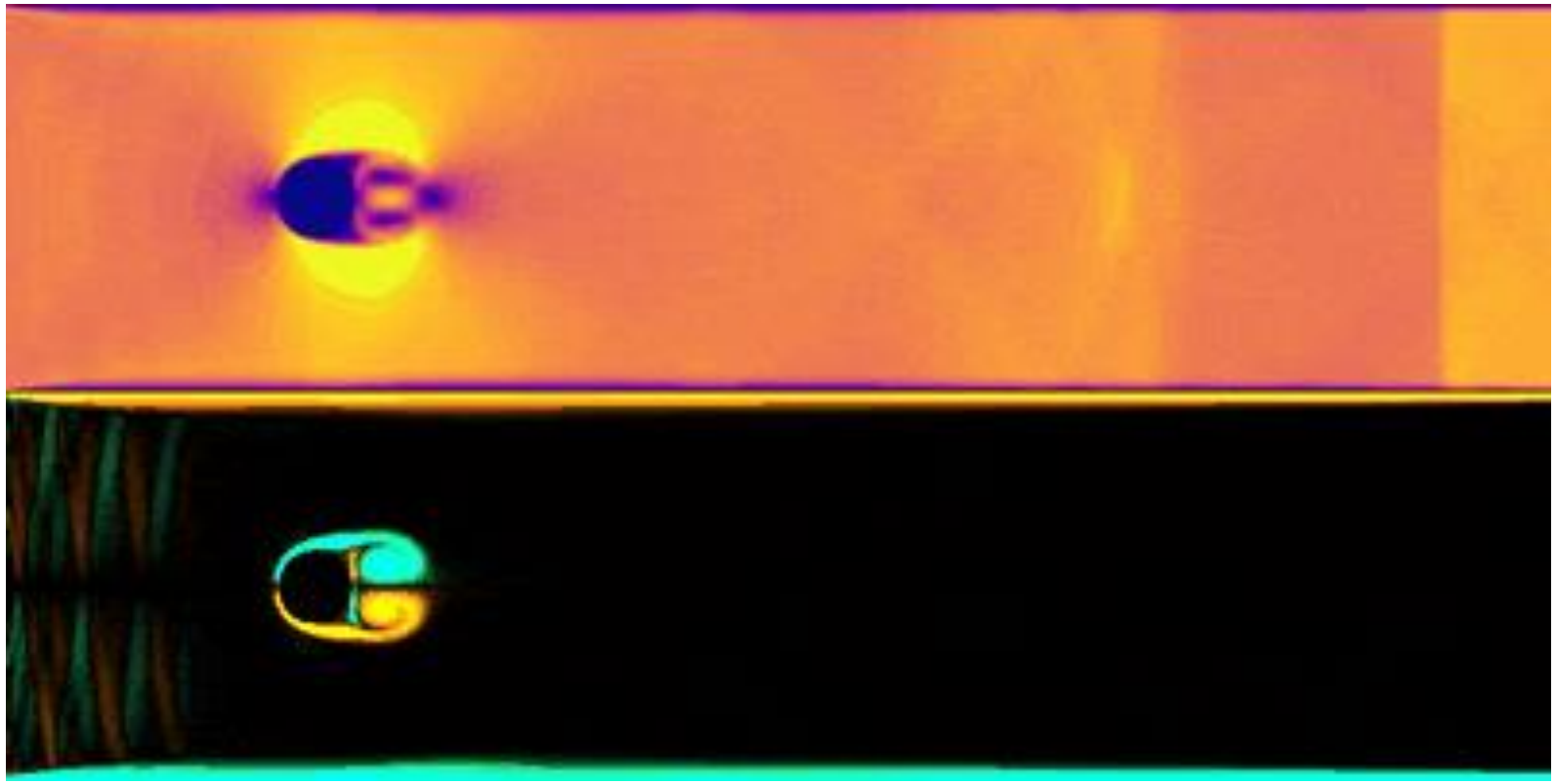
Implemented @Peter_shirley's "Ray Tracing in One Weekend" in pure #python that can execute on the GPU (via Metal, Vulkan, CUDA) using a package called "Taichi".
github.com/bsavery/ray-tr...

Was surprisingly easy and runs fast, these higher level languages for GPU could be the future!

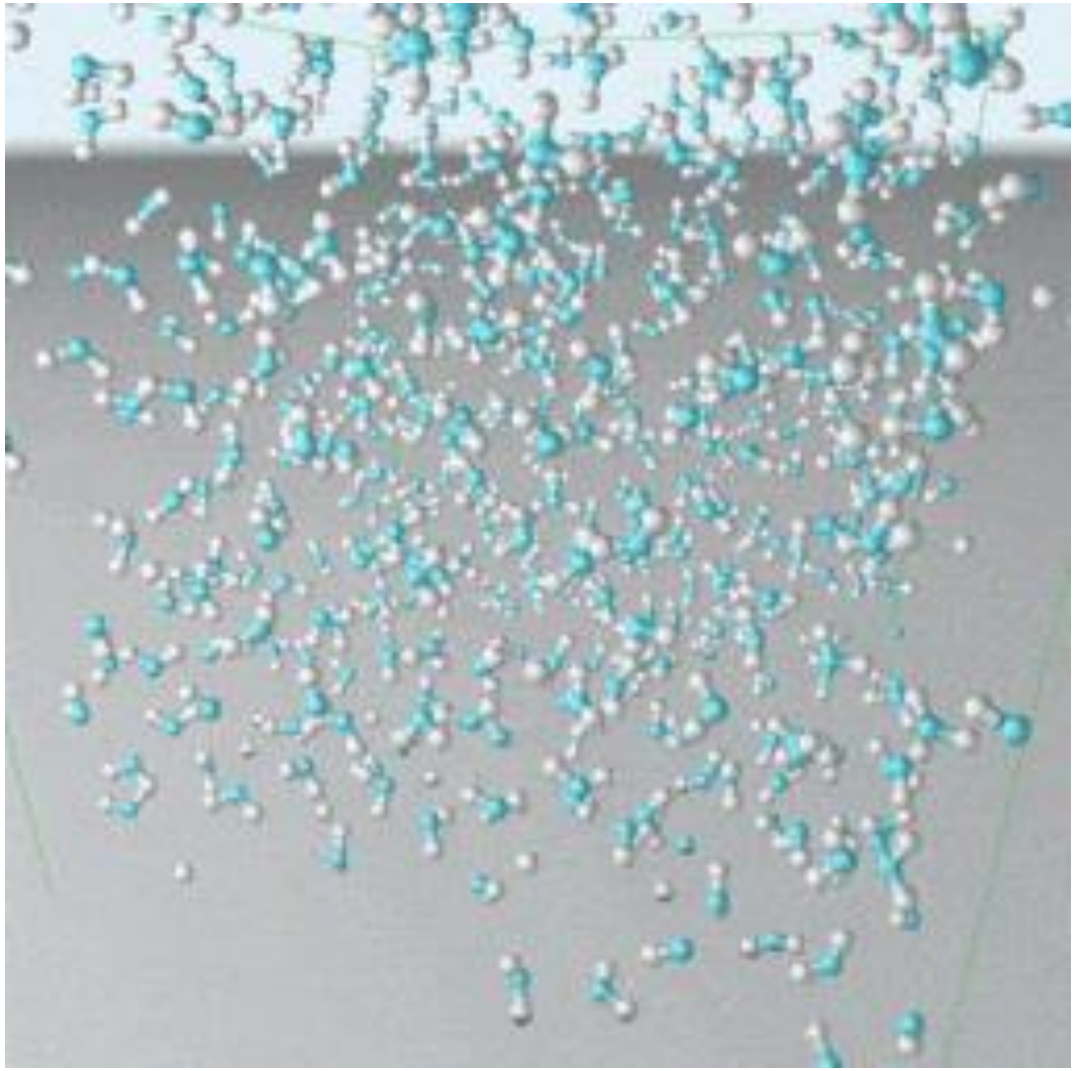


2 9 54

What you can do using Taichi



What you can do using Taichi



Taichi Installation

Taichi Installation

- Python3 -m **pip install taichi**

- Support Windows/Mac/Linux
  - For Windows users, you may need to install **Microsoft Visual C++ Redistributable** if you haven't done so.
  - For MacOS users, you may need to update your OS to **10.15(Catalina)+**
  - For Ubuntu 19.04+ users, you may need to install **libtinfo5**
  - You may need to build from source otherwise... (Not recommended for first-time users)

- Currently, Taichi only supports Python 3.6/3.7/3.8/3.9 (64-bit).

Taichi Zoo

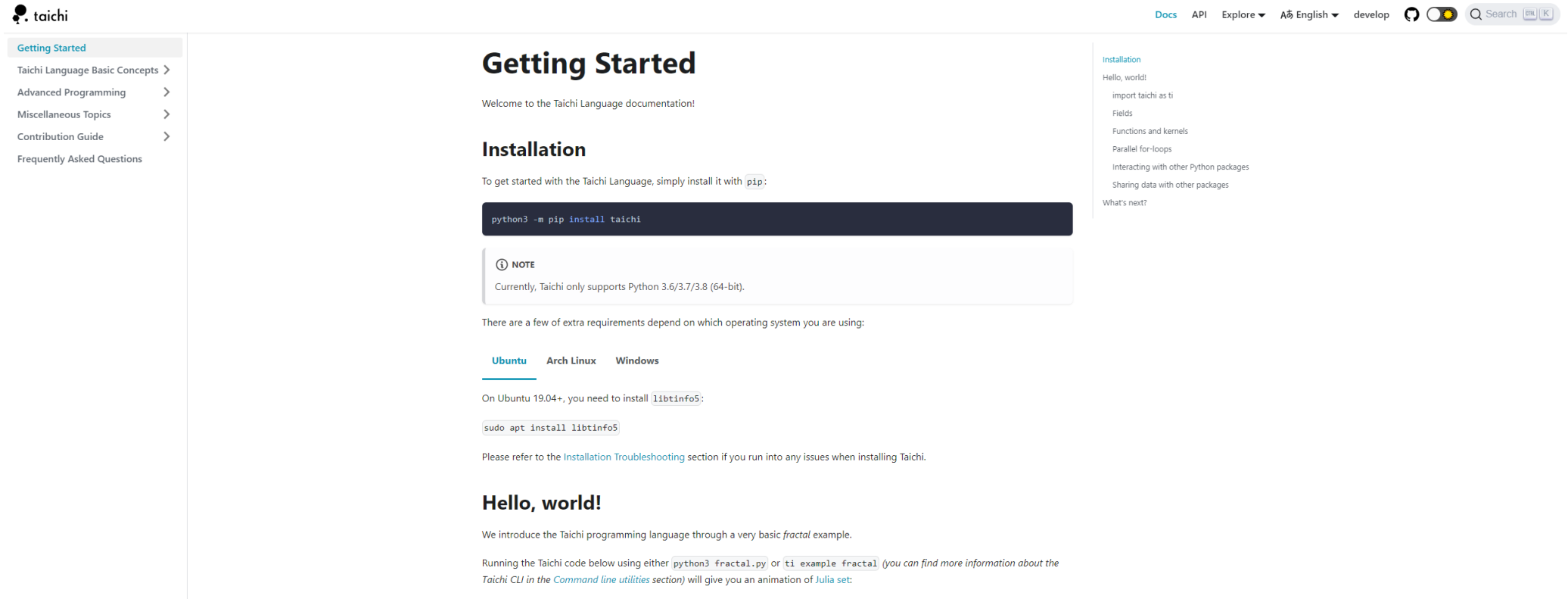
- https://zoo.taichi.graphics/

Since Zoo is still in BETA…

- Raise your issues at https://github.com/taichi-dev/taichi-zoo-issue-tracker

Taichi Documentation

- https://docs.taichi.graphics/docs/



Questions?