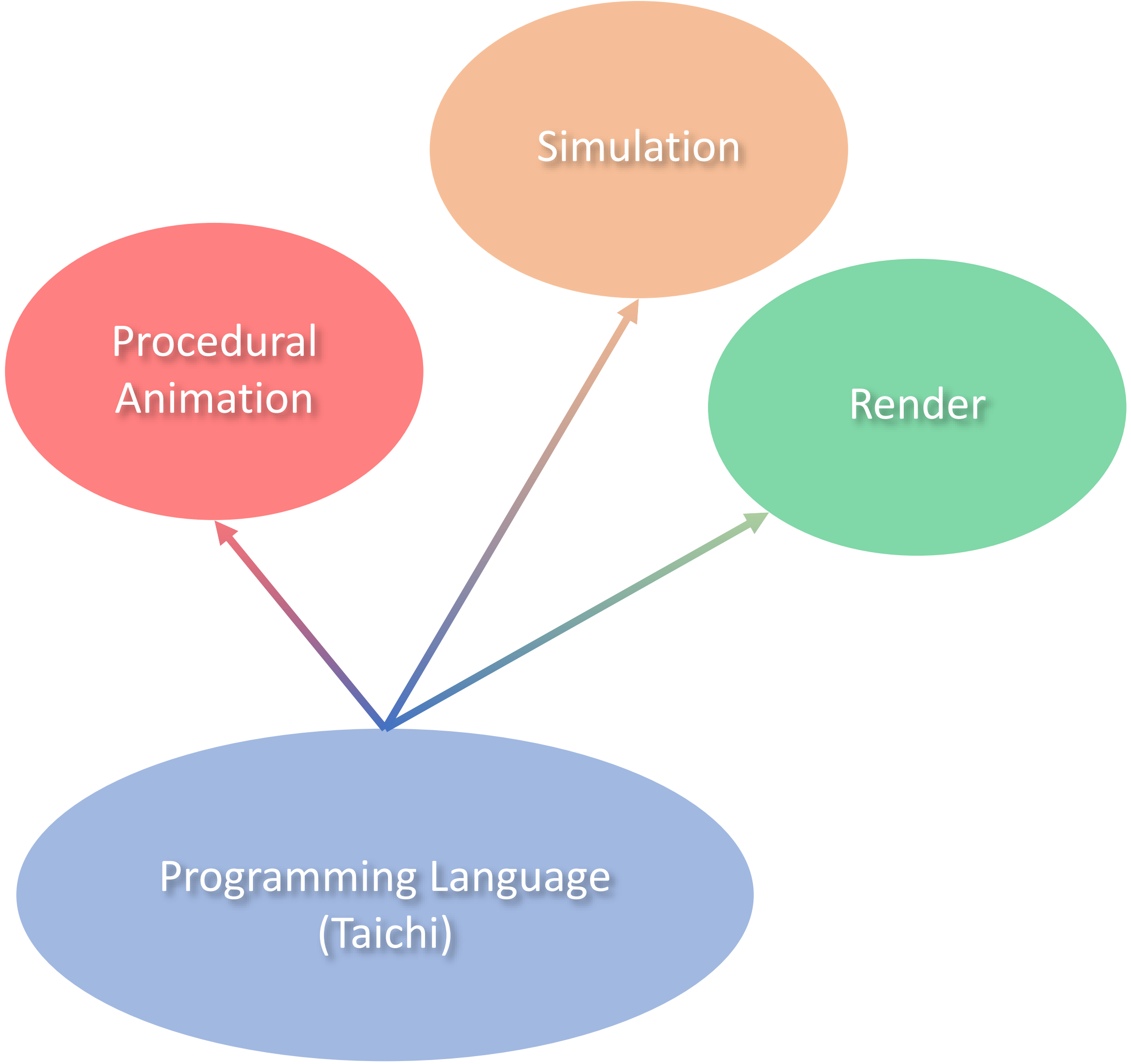
# 太极图形课

第05讲 Procedural Animation

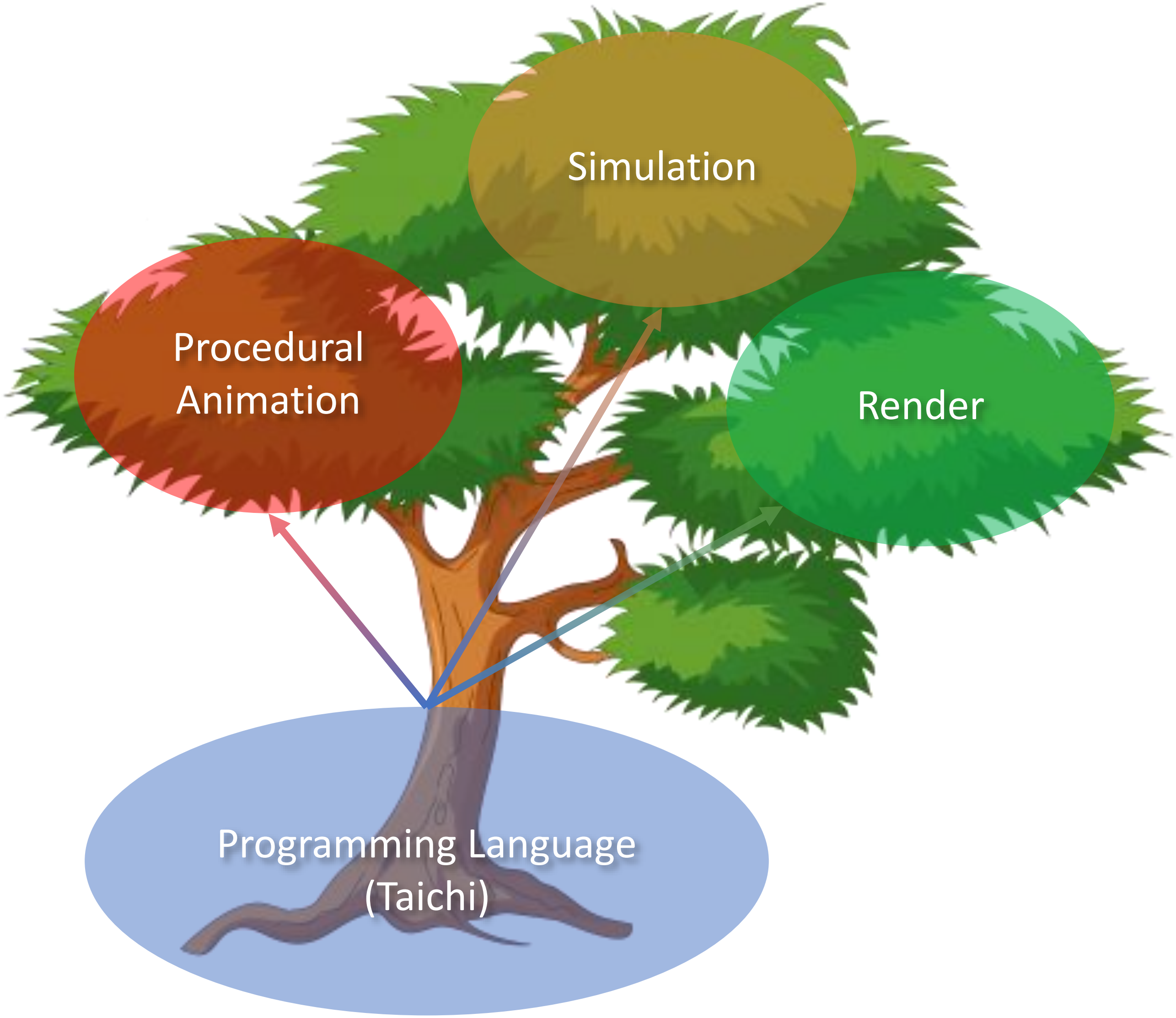
# 太极图形课

第05讲 Procedural Animation

# An overview of this Taichi Graphics course



# An overview of this Taichi Graphics course



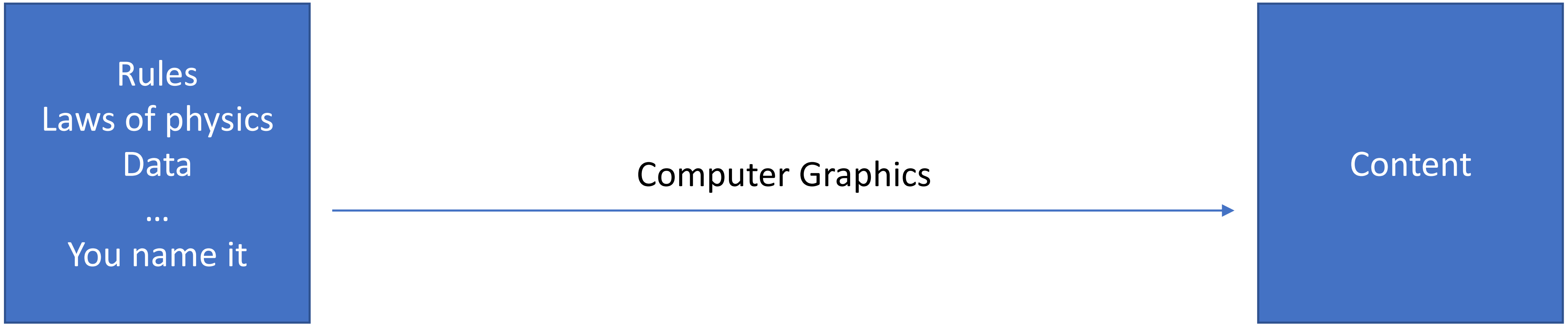
# Previously in this Taichi Graphics Course

- E1: Basics in Taichi:
  - data / computation / visualization
- E2: Improve the extensibility / maintainability of your code
  - Metaprogramming / Objective data-oriented programming
- E3: Performance is always the key
  - Advanced data layouts: dense and sparse
- E4: Miscs. and tips
  - Sparse linear algebra / debugging / performance profiling + tuning
- Using docs.taichi.graphics and api-docs.taichi.graphics as your manuals

# Taichi → Graphics

→

# The G in Graphics is for Generation

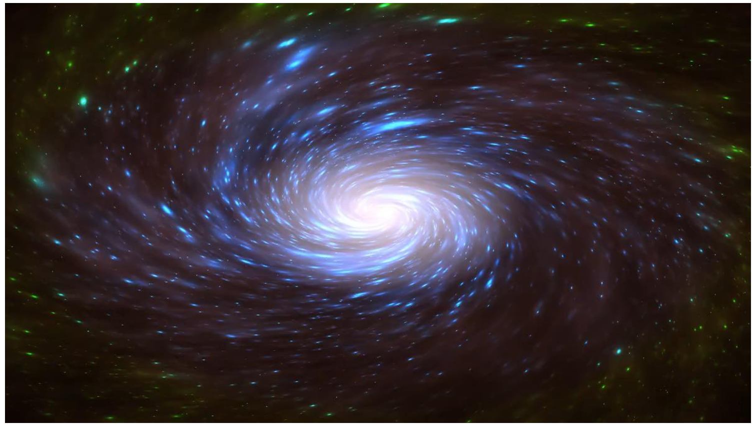


Rules
Laws of physics
Data
…
You name it

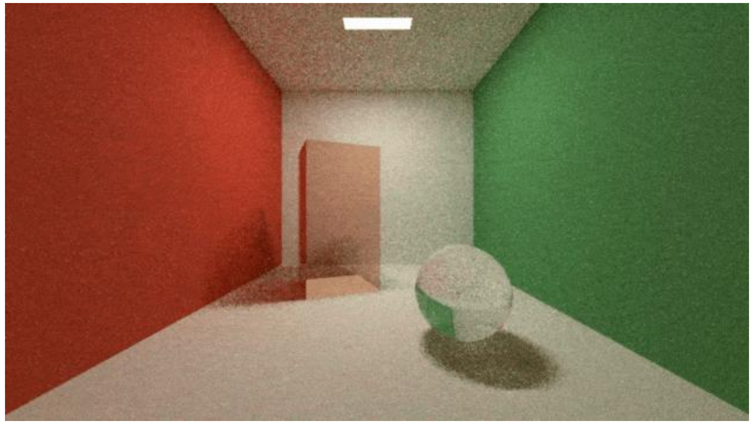
Computer Graphics

Content

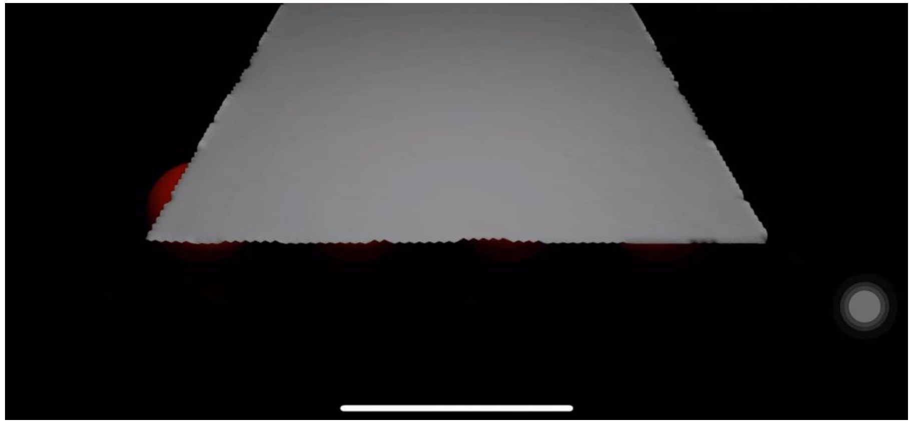
# Next part in this Taichi Graphics Course



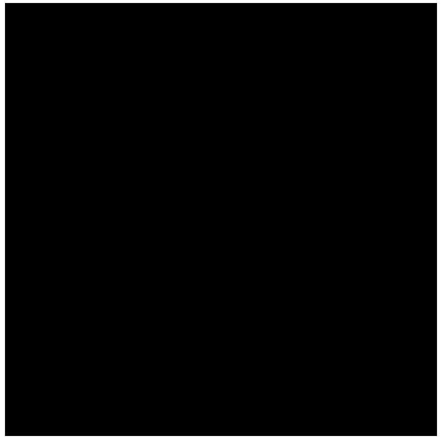
Procedural Animation



Render

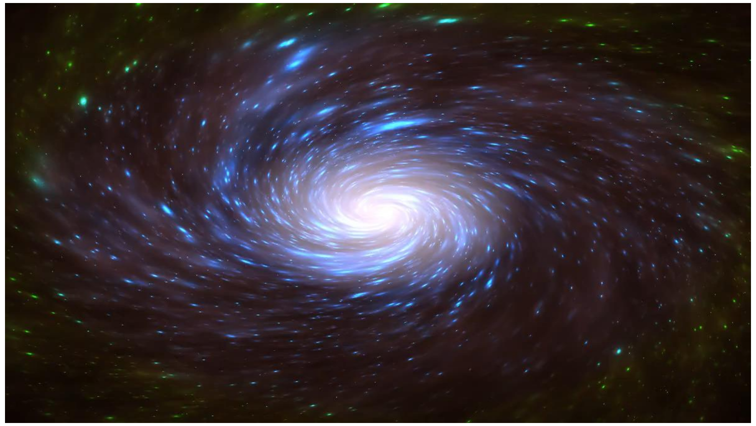


Deformable Simulation

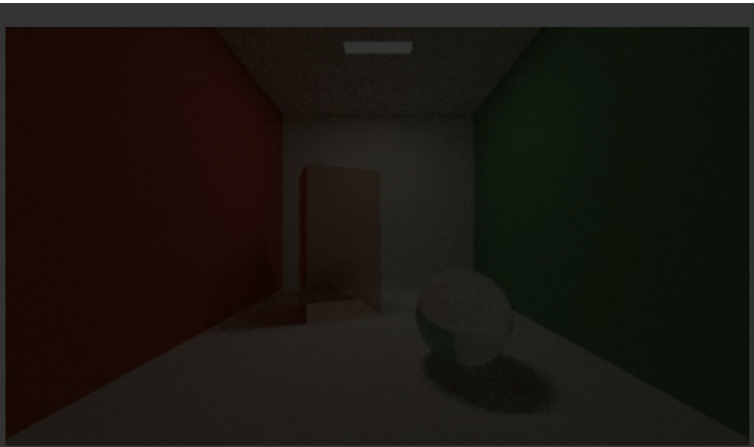


Fluid Simulation

# Today

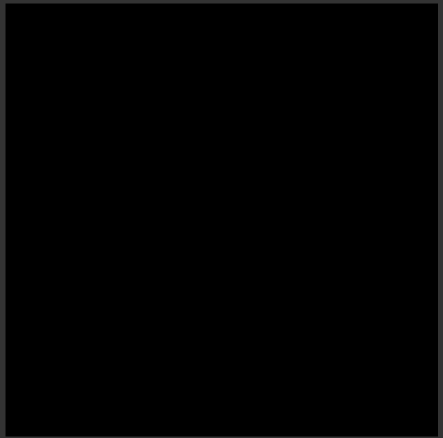


Procedural Animation



Render

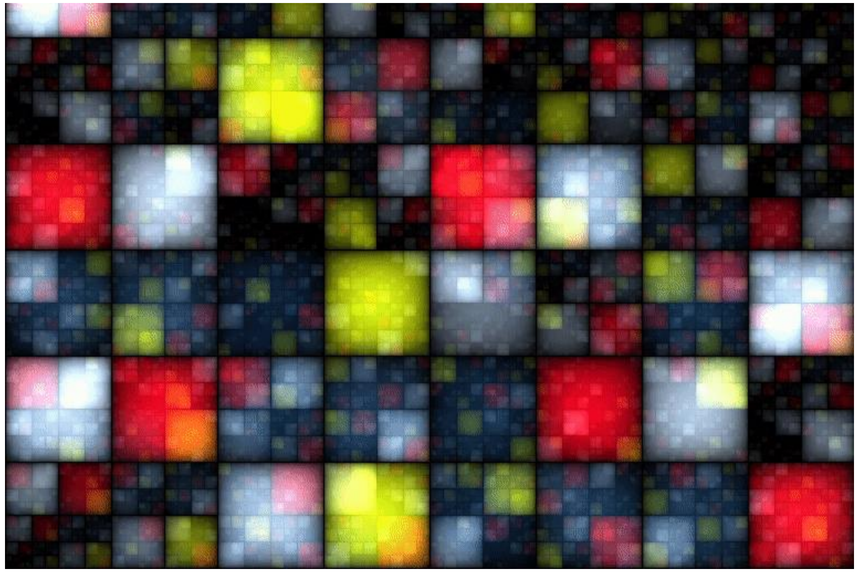
Deformable Simulation



Fluid Simulation

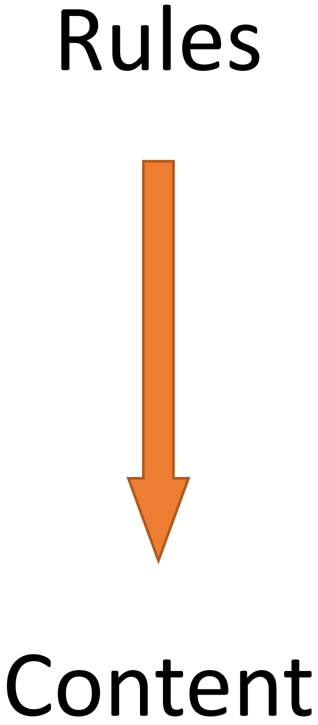
# Before we start today…

- Check https://github.com/taichiCourse01/--Shadertoys
- Will have a 15-mins quiz in the end
- Yuanming will take the quiz with you



# Procedural Animation

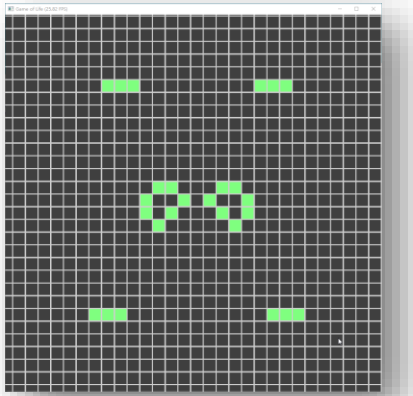
# Procedural modeling/animation



Rules

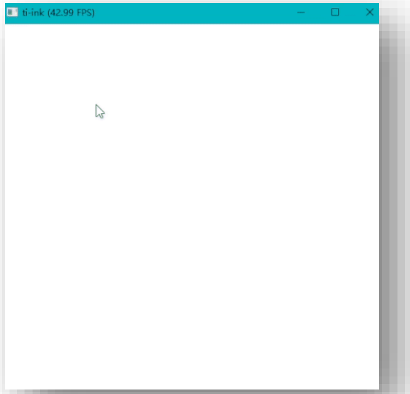
Content

# Submitted procedural animations

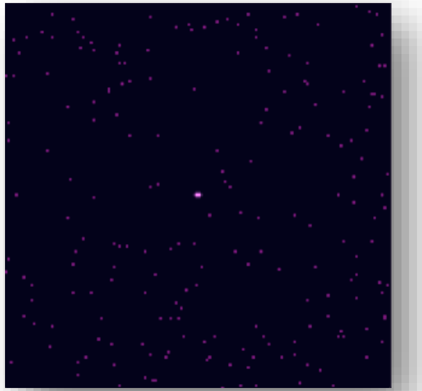


GoL @wuyingnan

Ant Colony
@theAfish

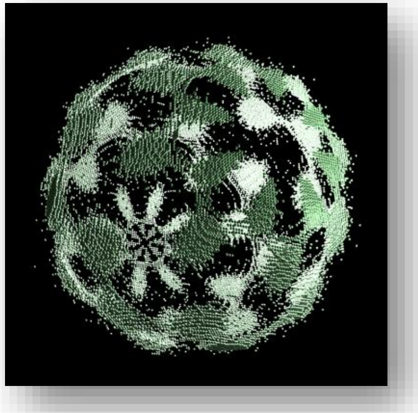


Moxi (墨戏) @Vineyo



DLA@theAfish

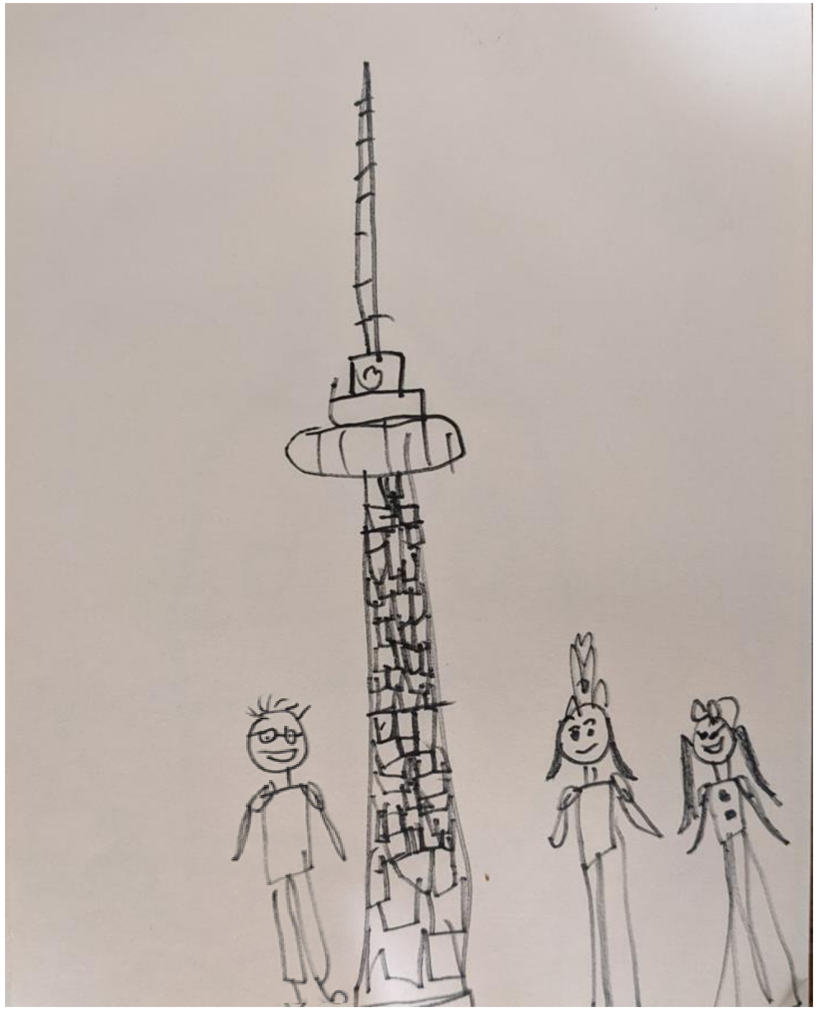
Flocking
@SIGUSR97



Mandelbulb
@rockeyshao

# TIL: A fancy galaxy ☺

# A simplest “procedural” animation: an analogy

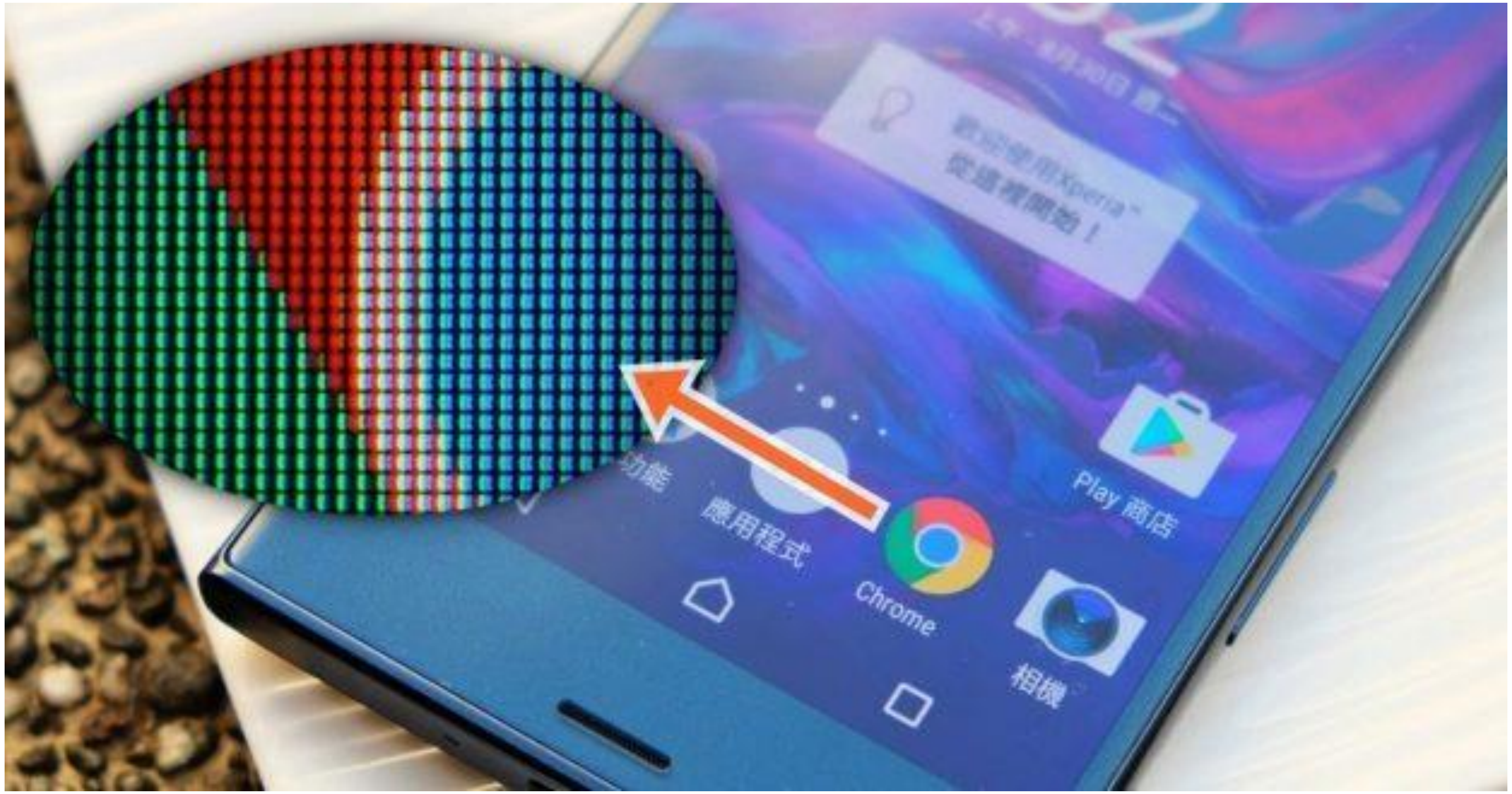


[Image courtesy of Elinor Liu]

# Procedural animation:

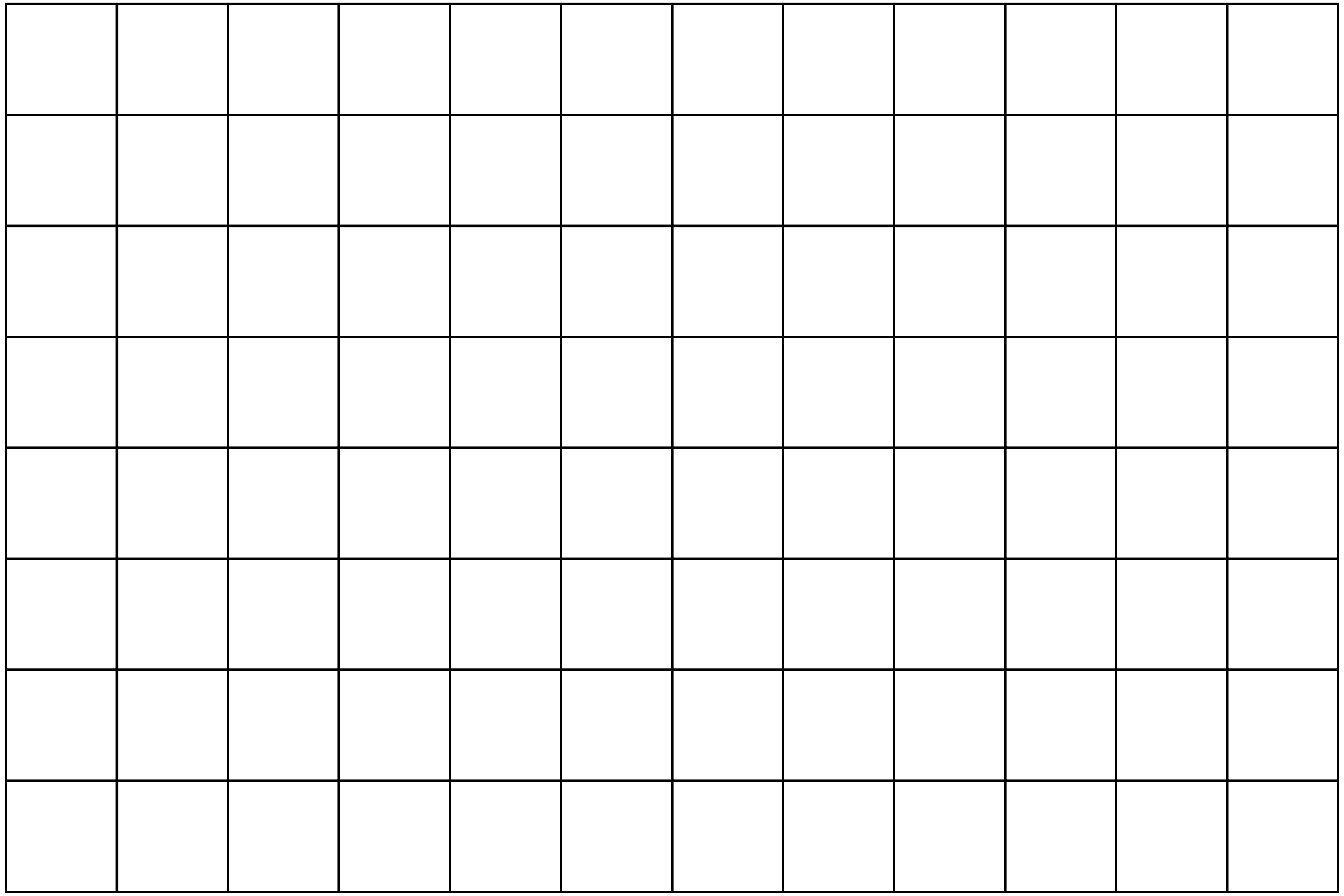
- Drawing / animating images on your screen
  - … with a few (or no) external assets
  - … based on your rules

# Your screen when zoomed in

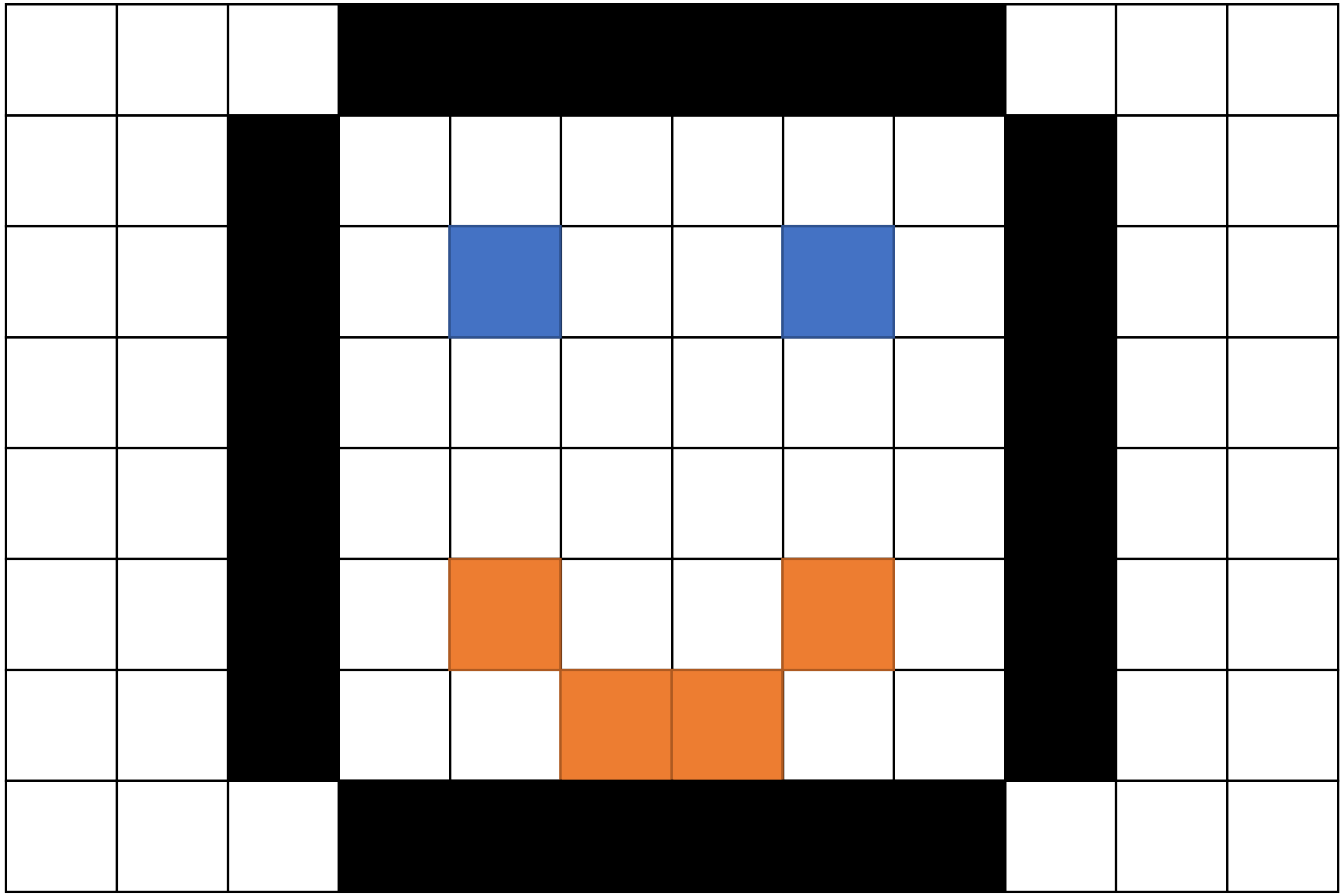


[Image courtesy of semiinsights.com]

# A simplest procedural animation



# A simplest procedural animation



# Procedural animations: a step-by-step example

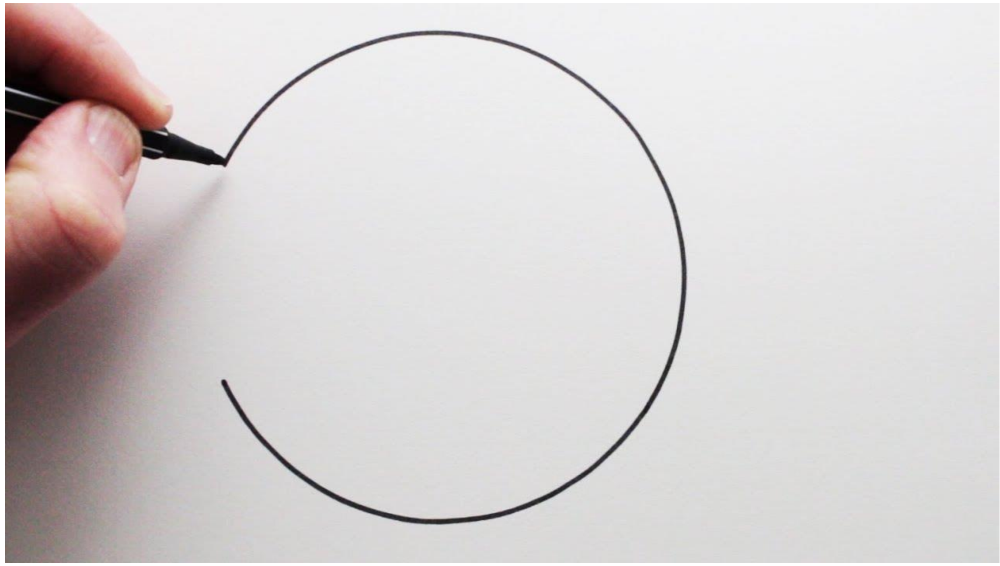
- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)

# Procedural animations: a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)

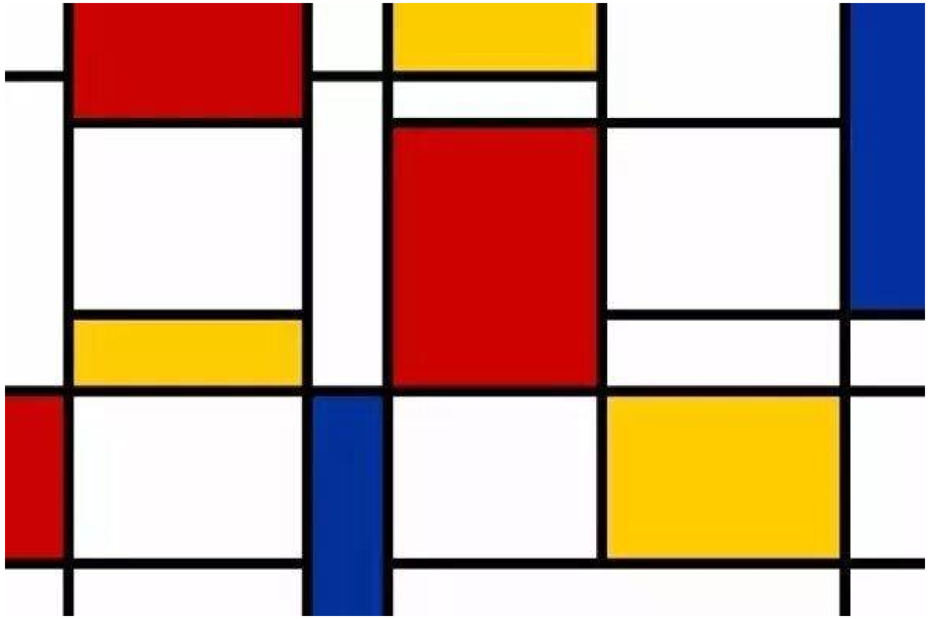
# Procedural animations: a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)



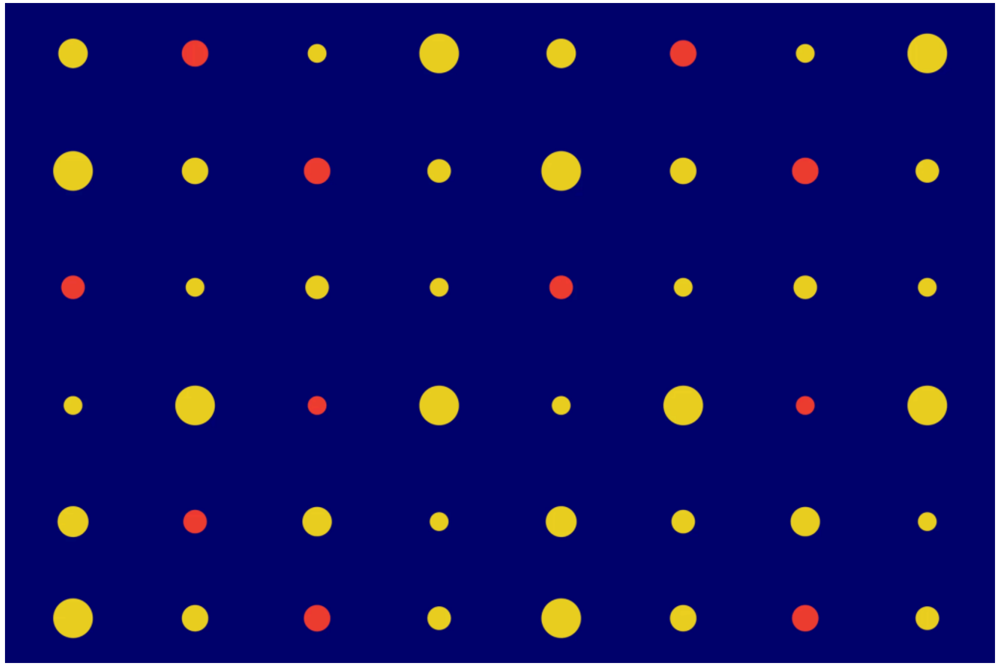
# Procedural animations: a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)



# Procedural animations: a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)



# Procedural animations: a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)

# Setup your canvas in Taichi

```
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```

# Setup your canvas in Taichi

```python
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```

# Setup your canvas in Taichi

```
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```

# Setup your canvas in Taichi

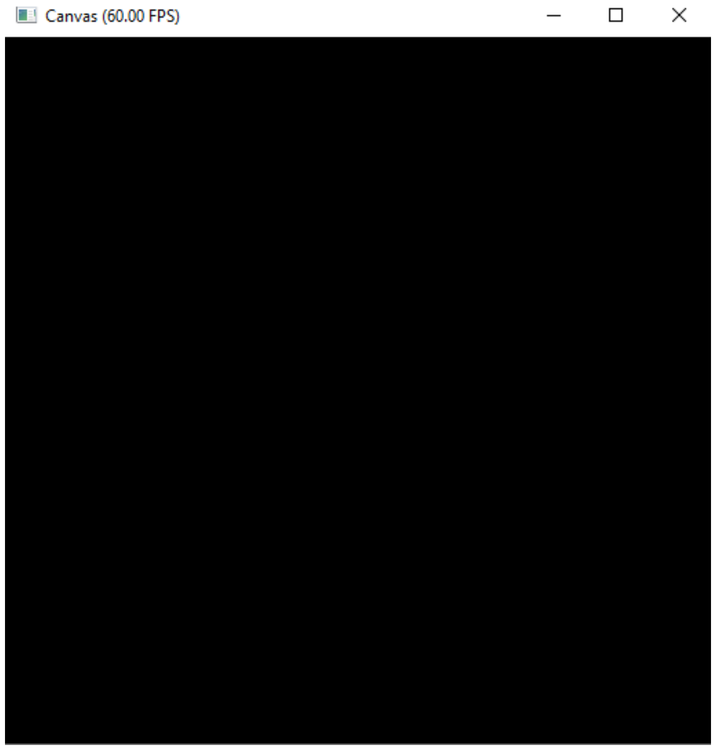
```
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```

# Setup your canvas in Taichi

```python
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```

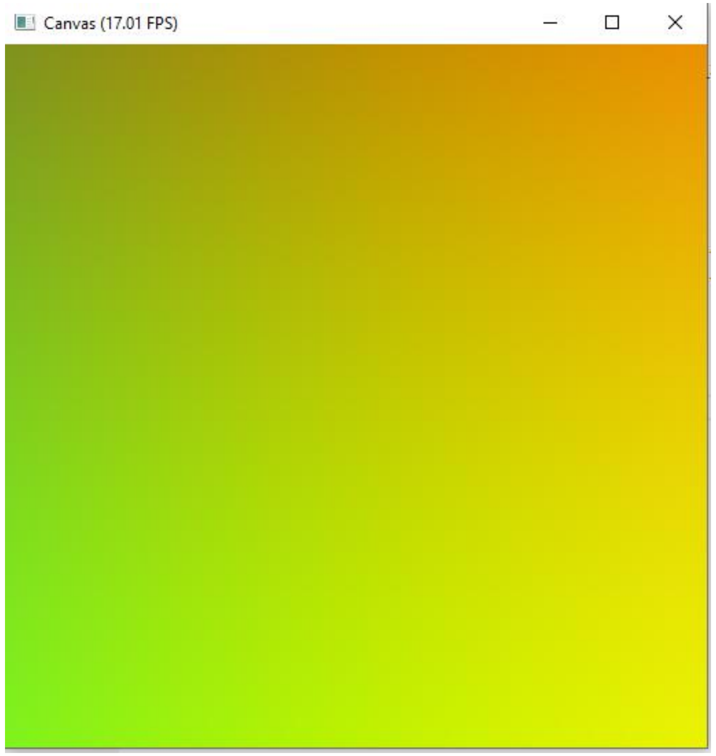
# Setup your canvas in Taichi

```python
import taichi as ti
ti.init(arch = ti.cuda)

res_x = 512
res_y = 512
pixels = ti.Vector.field(3, ti.f32, shape=(res_x, res_y))

@ti.kernel
def render():
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pixels[i,j] = color

gui = ti.GUI("Canvas", res=(res_x, res_y))

for i in range(100000):
    render()
    gui.set_image(pixels)
    gui.show()
```



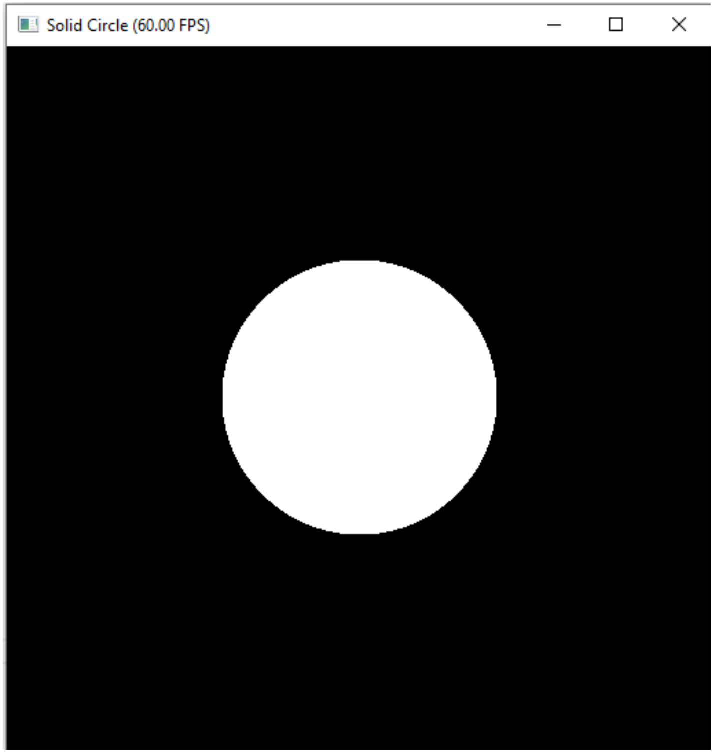
# Put some colors on your canvas

```
@ti.kernel
def render(t:ti.f32):
    # draw something on your canvas
    for i,j in pixels:
        r = 0.5 * ti.sin(float(i) / res_x) + 0.5
        g = 0.5 * ti.sin(float(j) / res_y + 2) + 0.5
        b = 0.5 * ti.sin(float(i) / res_x + 4) + 0.5
        color = ti.Vector([r, g, b])
        pixels[i, j] = color
```



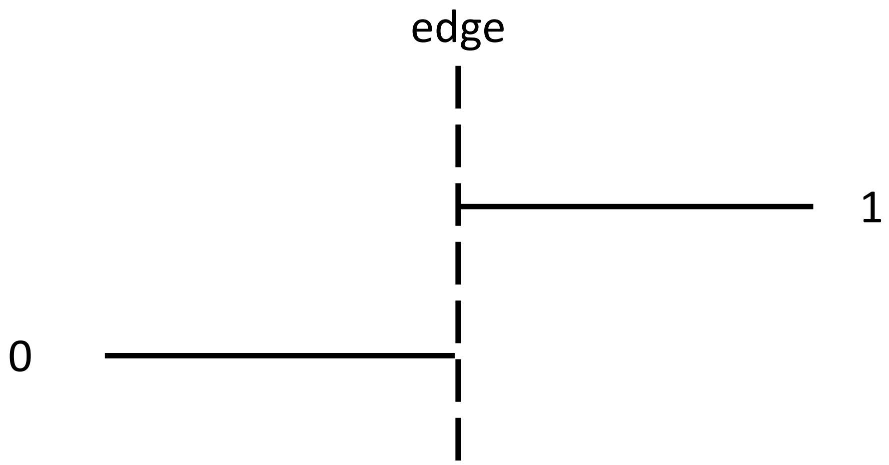
# Draw a basic unit

```
@ti.kernel
def render(t:ti.f32):
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black
        pos = ti.Vector([i//scatter, j//scatter])
        center = ti.Vector([res_x//2, res_y//2])
        r1 = 100.0
        r = (pos - center).norm()

        if r < r1:
            color = ti.Vector([1.0, 1.0, 1.0])

        pixels[i, j] = color
```



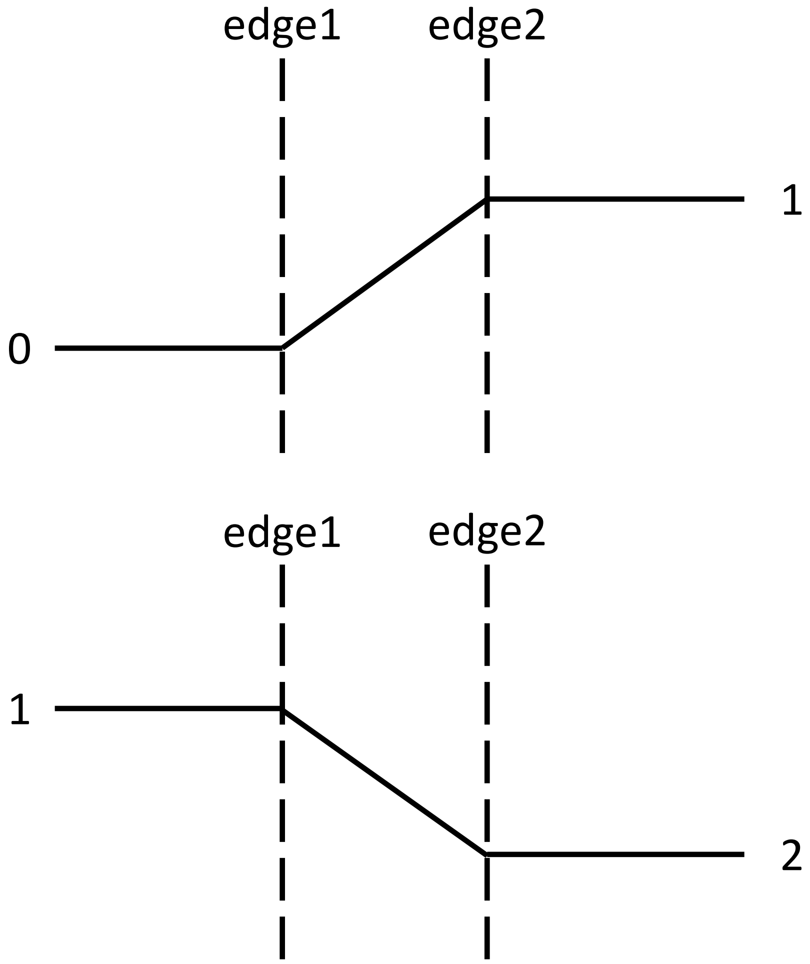
# Some helper functions: step

```
@ti.func
def step(edge, v):
    ret = 0.0
    if (v < edge): ret = 0.0
    else: ret = 1.0
    return ret
```



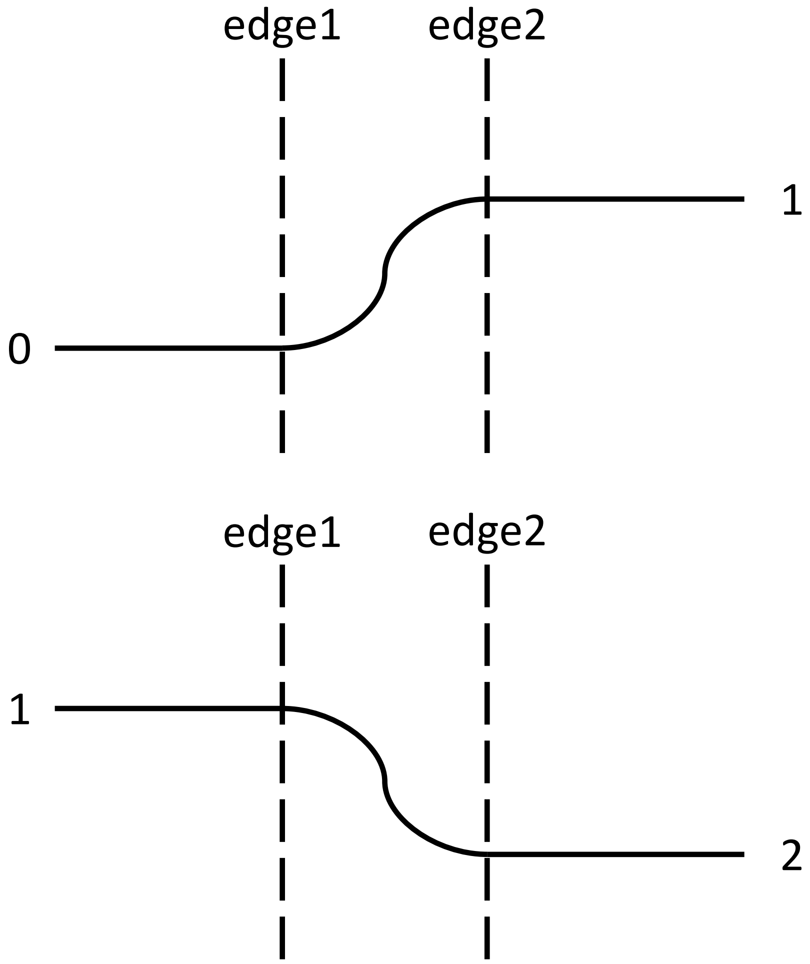
# Some helper functions: linearstep

```python
@ti.func
def linearstep(edge1, edge2, v):
    assert(edge1 != edge2)
    t = (v-edge1) / float(edge2-edge1)
    t = clamp(t, 0.0, 1.0)

    return t
```



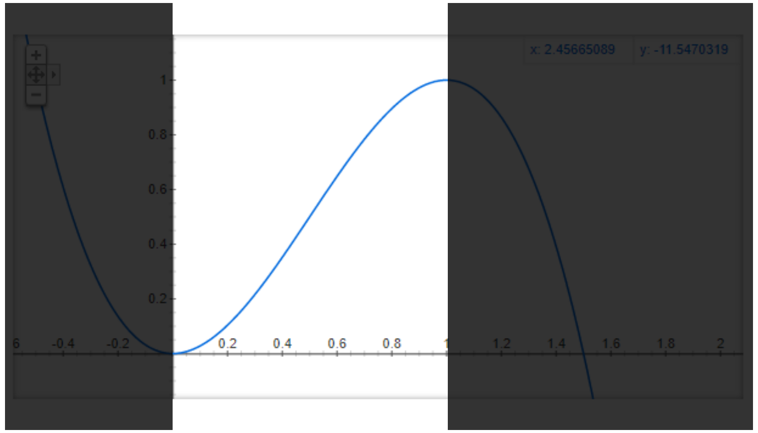
# Some helper functions: linearstep

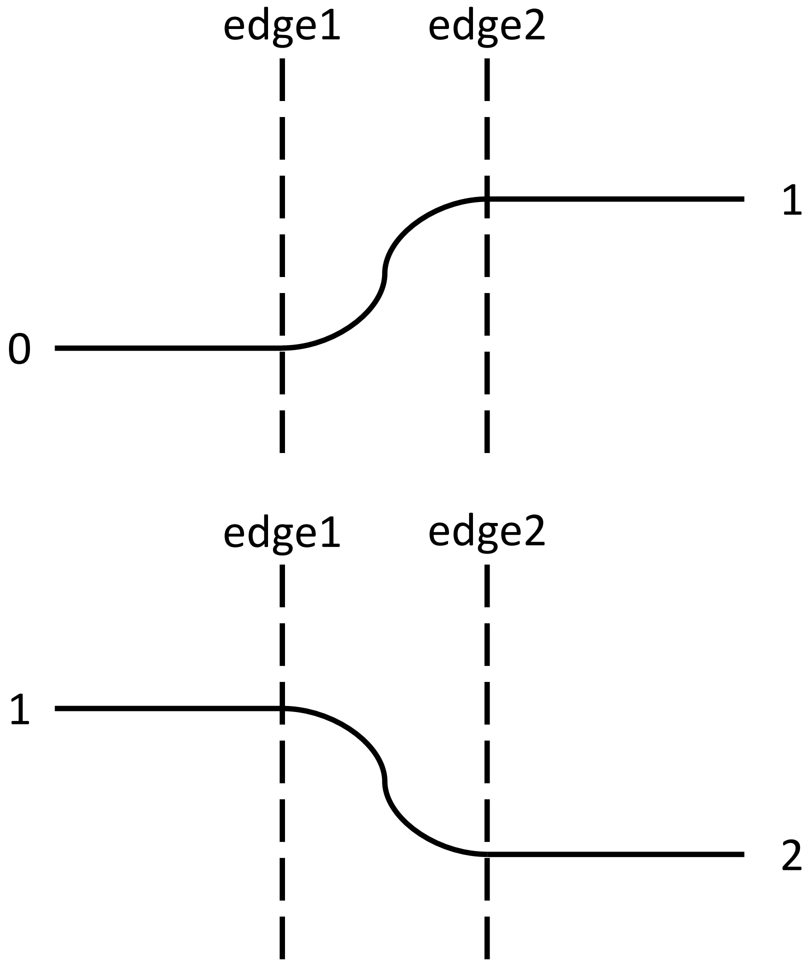
```
@ti.func
def smoothstep(edge1, edge2, v):
    assert(edge1 != edge2)
    t = (v-edge1) / float(edge2-edge1)
    t = clamp(t, 0.0, 1.0)

    return (3-2 * t) * t**2
```



# Some helper functions: linearstep

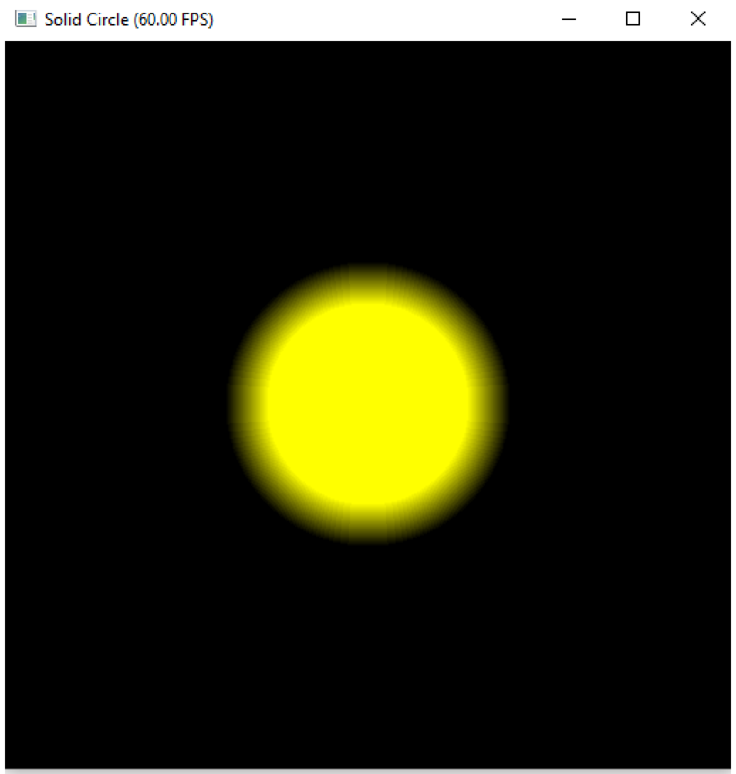
```
@ti.func
def smoothstep(edge1, edge2, v):
    assert(edge1 != edge2)
    t = (v-edge1) / float(edge2-edge1)
    t = clamp(t, 0.0, 1.0)

    return (3-2 * t) * t**2
```





# Draw a basic unit

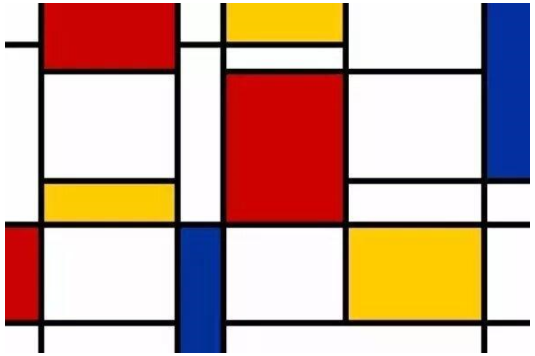
```
@ ti.func
def circle(pos, center, radius, blur):
    r = (pos - center).norm()
    t = 0.0
    if blur > 1.0: blur = 1.0
    if blur <= 0.0:
        t = 1.0-hsf.step(1.0, r/radius)
    else:
        t = hsf.smoothstep(1.0, 1.0-blur, r/radius)
    return t

@ti.kernel
def render(t:ti.f32):
    for i,j in pixels:
        ...

        c = circle(pos, center, r1, 0.1)

        color = ti.Vector([1.0, 1.0, 1.0]) * c
        pixels[i, j] = color
```



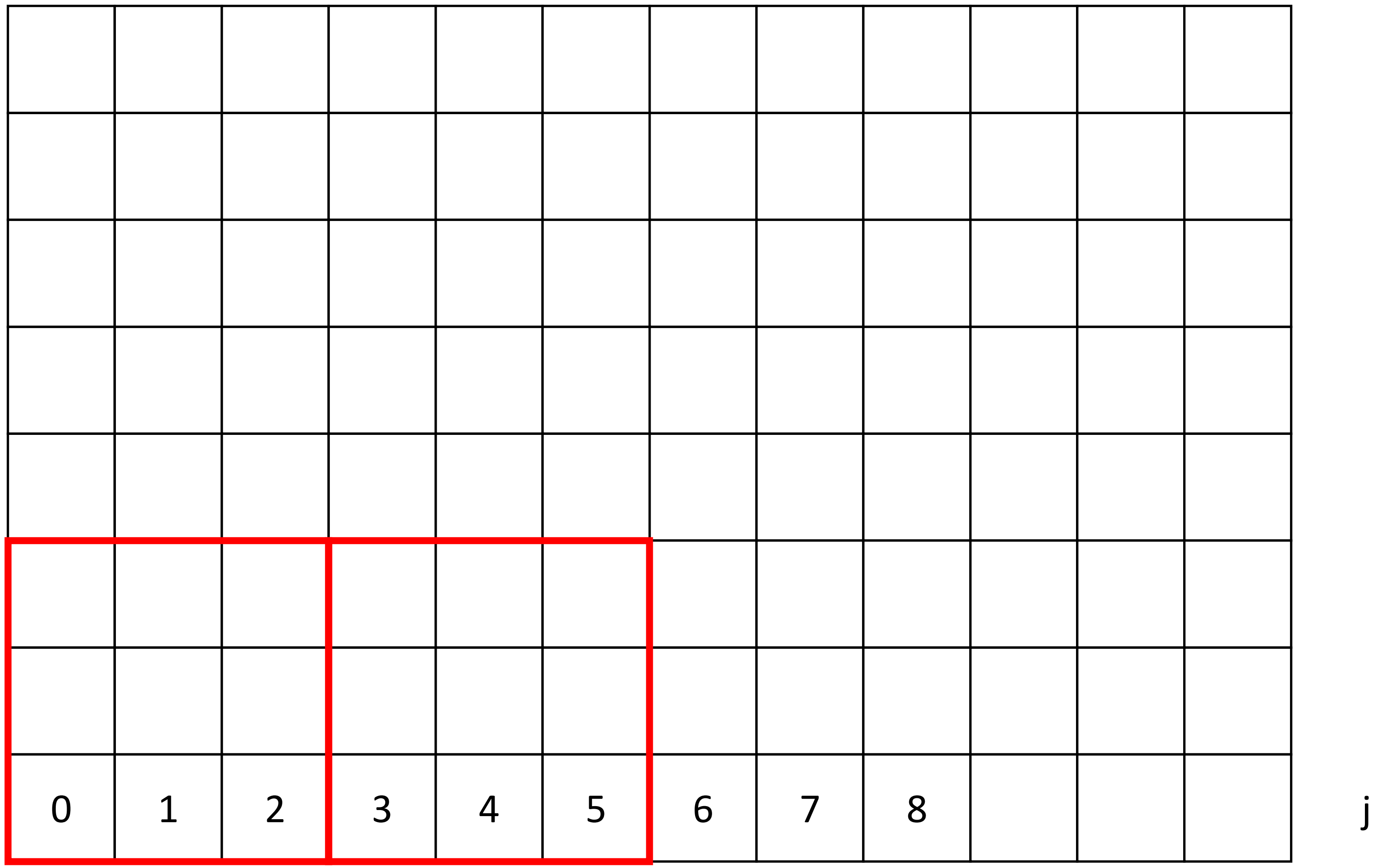
# Repeat the basic units: tiles

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | |

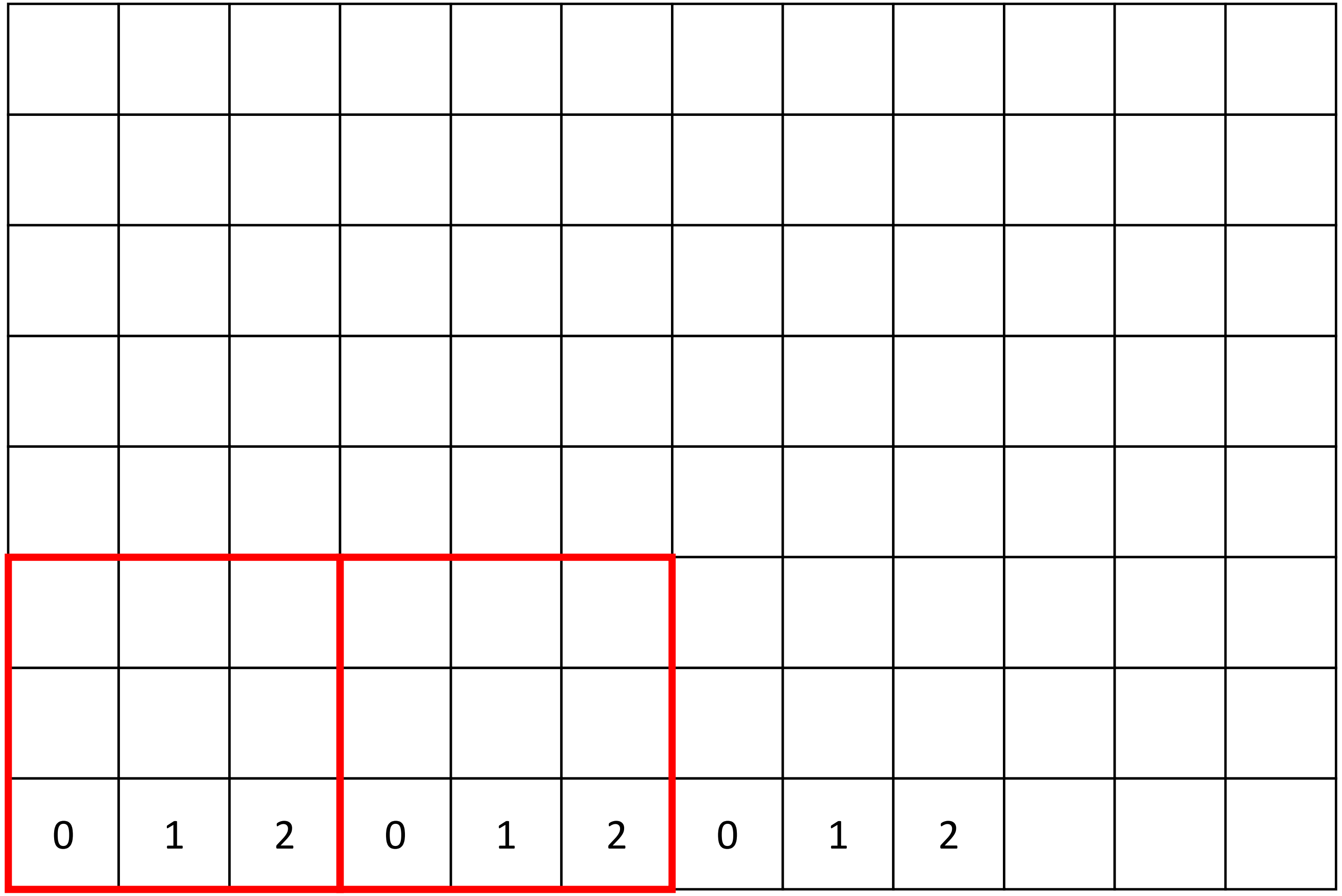
j



# Repeat the basic units: tiles



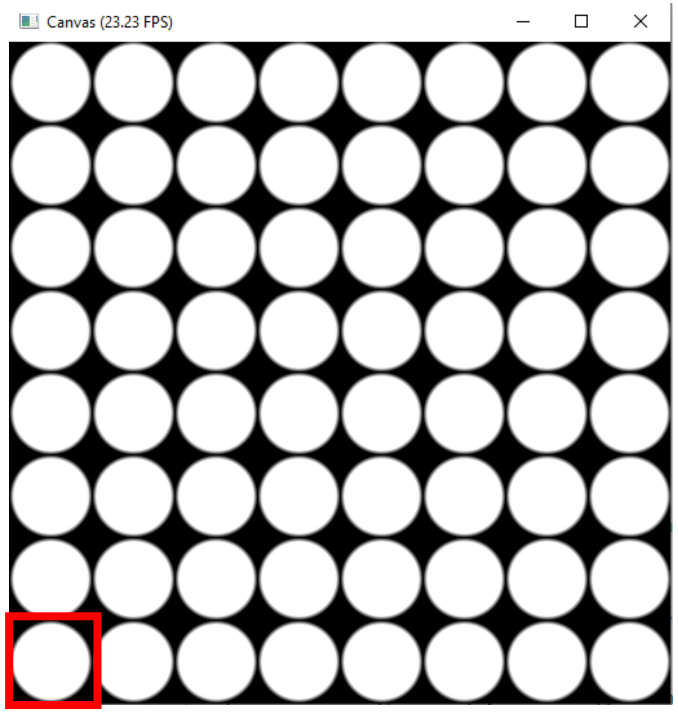
# Repeat the basic units: tiles



j mod 3

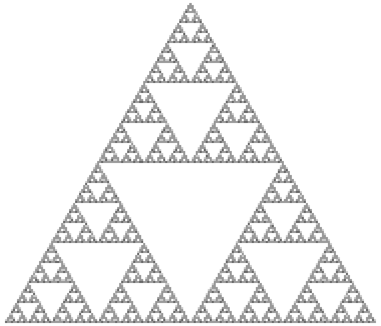
# Repeat the basic units: tiles

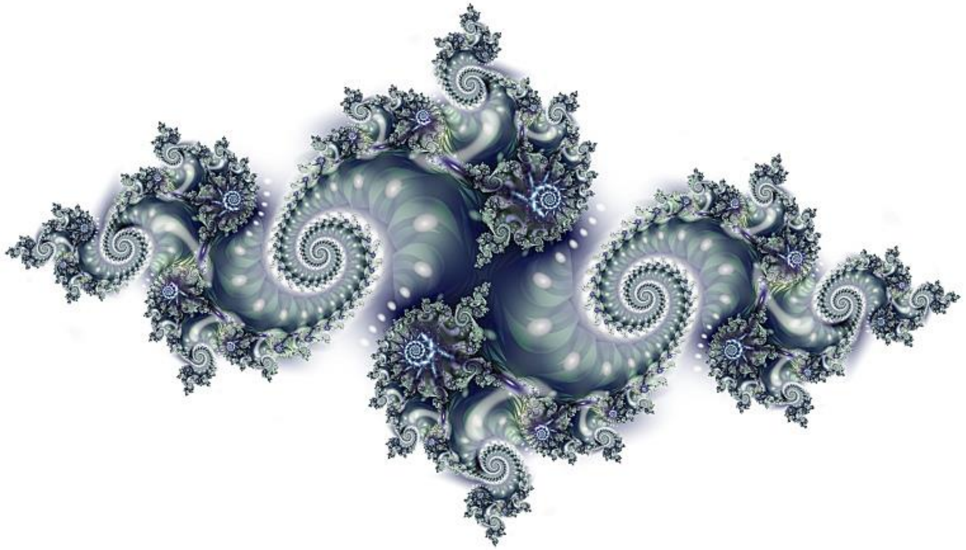
```
@ti.kernel
def render(t:ti.f32):
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to black

        tile_size = 64

        center = ti.Vector([tile_size//2, tile_size//2])
        radius = tile_size//2

        pos = ti.Vector([hsf.mod(i, tile_size), hsf.mod(j, tile_size)])
# scale i, j to [0, tile_size-1]

        c = circle(pos, center, radius, 0.1)

        color += ti.Vector([1.0, 1.0, 1.0])*c

        pixels[i,j] = color
```



# Repeat the basic units: fractals

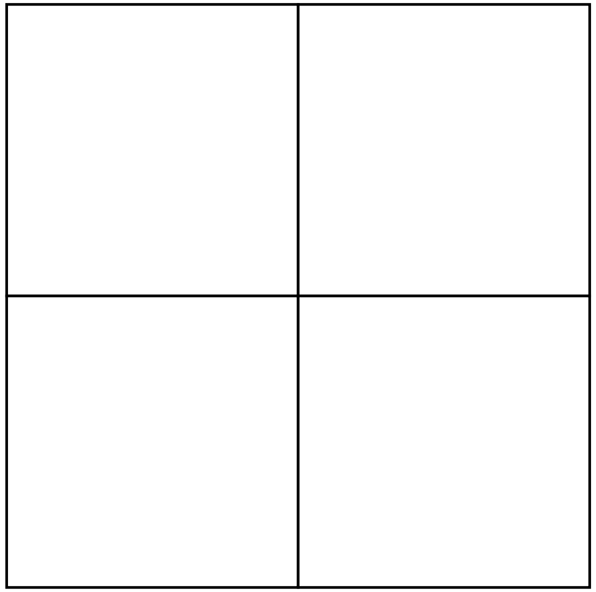
- *In mathematics, a fractal is a **subset** of Euclidean space … Fractals **appear the same** at different scales … Fractals often exhibit **similar patterns** at increasingly smaller scales …*





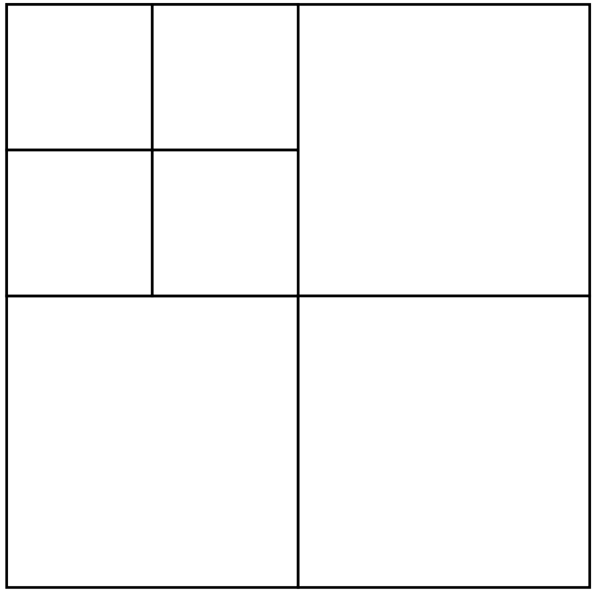
# Repeat the basic units: fractals

- *In mathematics, a fractal is a* ***subset*** *of Euclidean space … Fractals* ***appear the same*** *at different scales …  Fractals often exhibit* ***similar patterns*** *at increasingly smaller scales …*



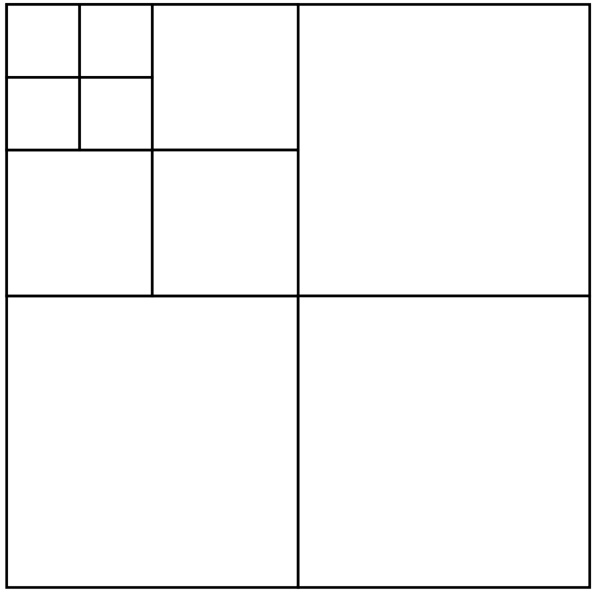
# Repeat the basic units: fractals

- *In mathematics, a fractal is a* ***subset*** *of Euclidean space … Fractals* ***appear the same*** *at different scales …  Fractals often exhibit* ***similar patterns*** *at increasingly smaller scales …*



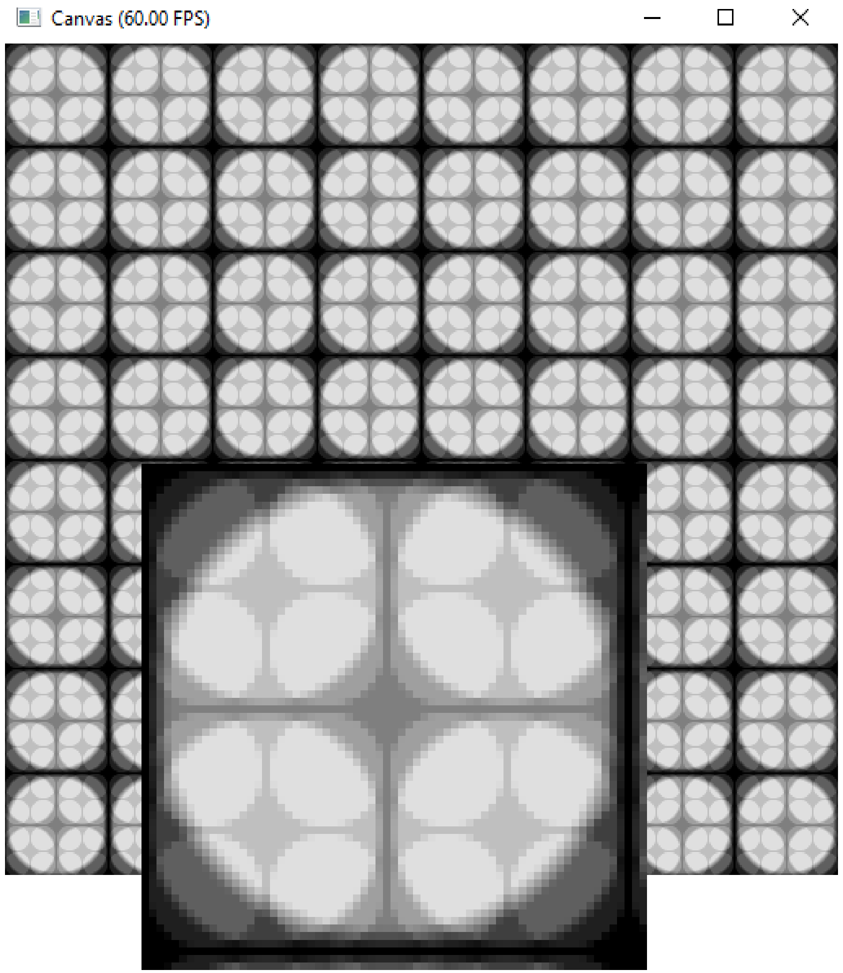
# Repeat the basic units: fractals

- *In mathematics, a fractal is a **subset** of Euclidean space … Fractals **appear the same** at different scales … Fractals often exhibit **similar patterns** at increasingly smaller scales …*



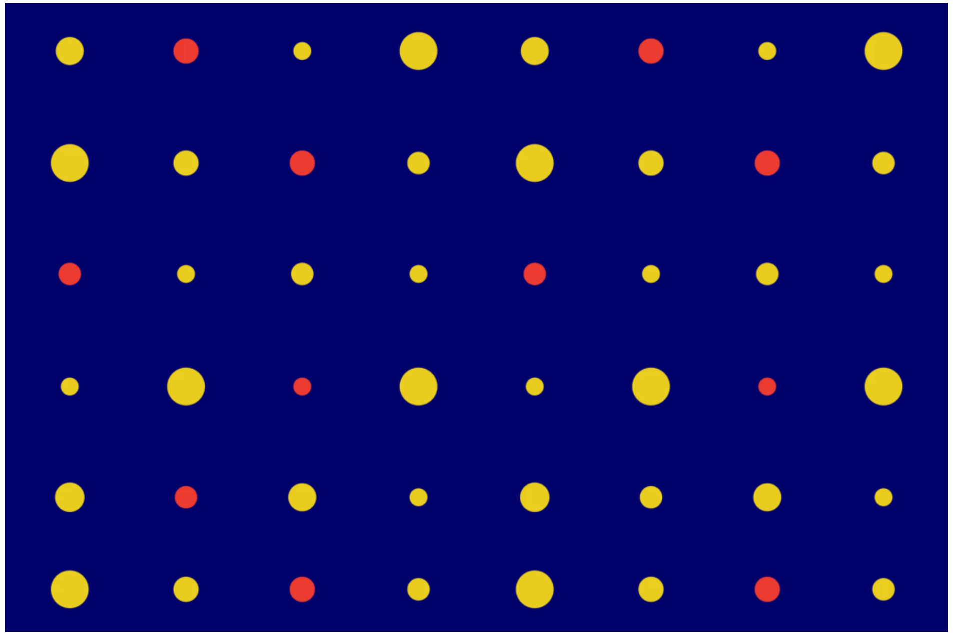
# Repeat the basic units: fractals

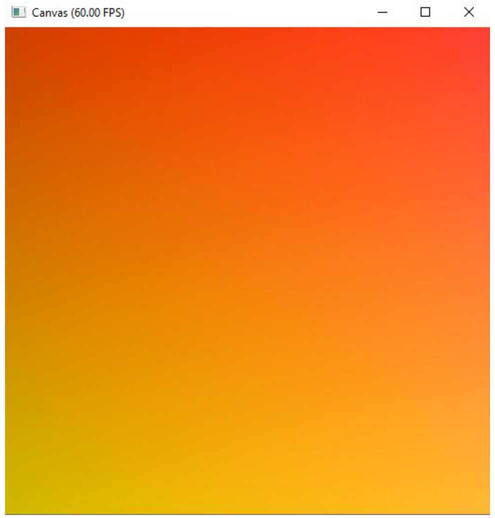
```
@ti.kernel
def render(t:ti.f32):
    # draw something on your canvas
    for i,j in pixels:
        color = ti.Vector([0.0, 0.0, 0.0]) # init your canvas to
black

        tile_size = 16

        for k in range(3):
            center = ti.Vector([tile_size//2, tile_size//2])
            radius = tile_size//2

            pos = ti.Vector([hsf.mod(i, tile_size), hsf.mod(j,
tile_size)]) # scale i, j to [0, tile_size-1]

            c = circle(pos, center, radius, 0.1)

            color += ti.Vector([1.0, 1.0, 1.0])*c

            color /= 2
            tile_size *= 2

        pixels[i,j] = color
```

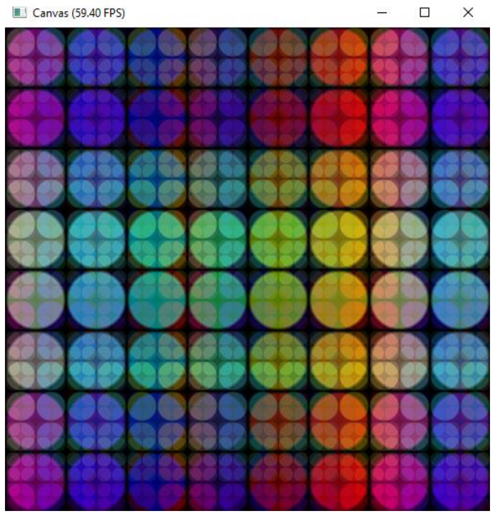


# Animate your pictures

```
@ti.kernel
def render(t:ti.f32):
    # draw something on your canvas
    for i,j in pixels:
        r = 0.5 * ti.sin(t+float(i) / res_x) + 0.5
        g = 0.5 * ti.sin(t+float(j) / res_y + 2) + 0.5
        b = 0.5 * ti.sin(t+float(i) / res_x + 4) + 0.5
        color = ti.Vector([r, g, b])
        pixels[i, j] = color
```





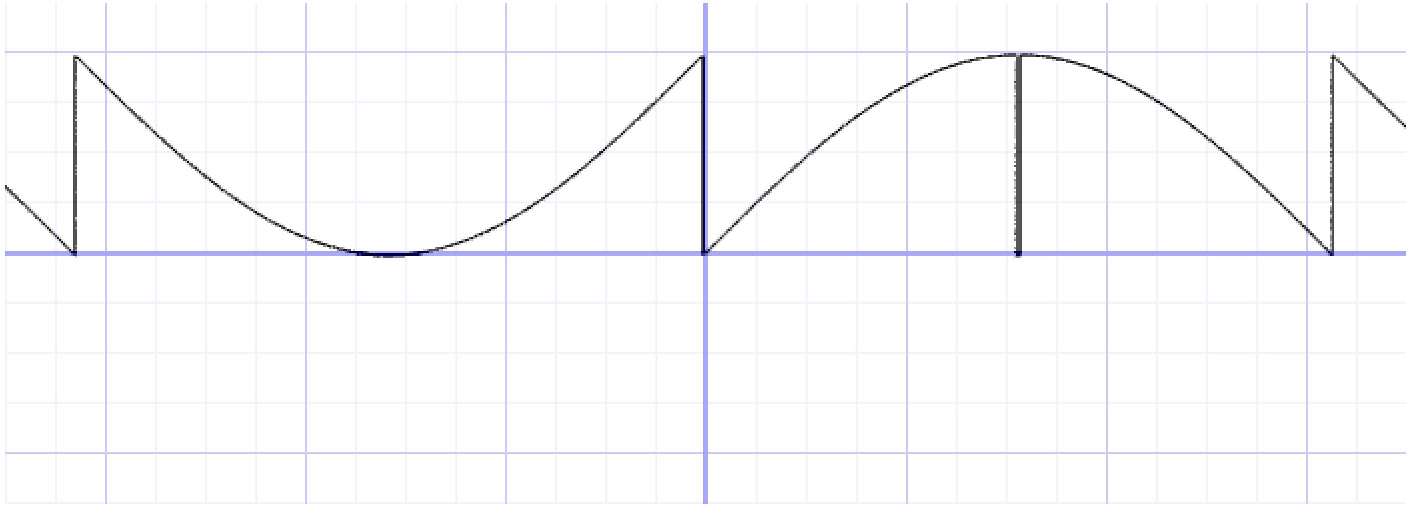


# Introduce some randomness (Chaos!)

- $y = rand(x)$ or y = ti.random() ?

# Introduce some randomness (Chaos!)

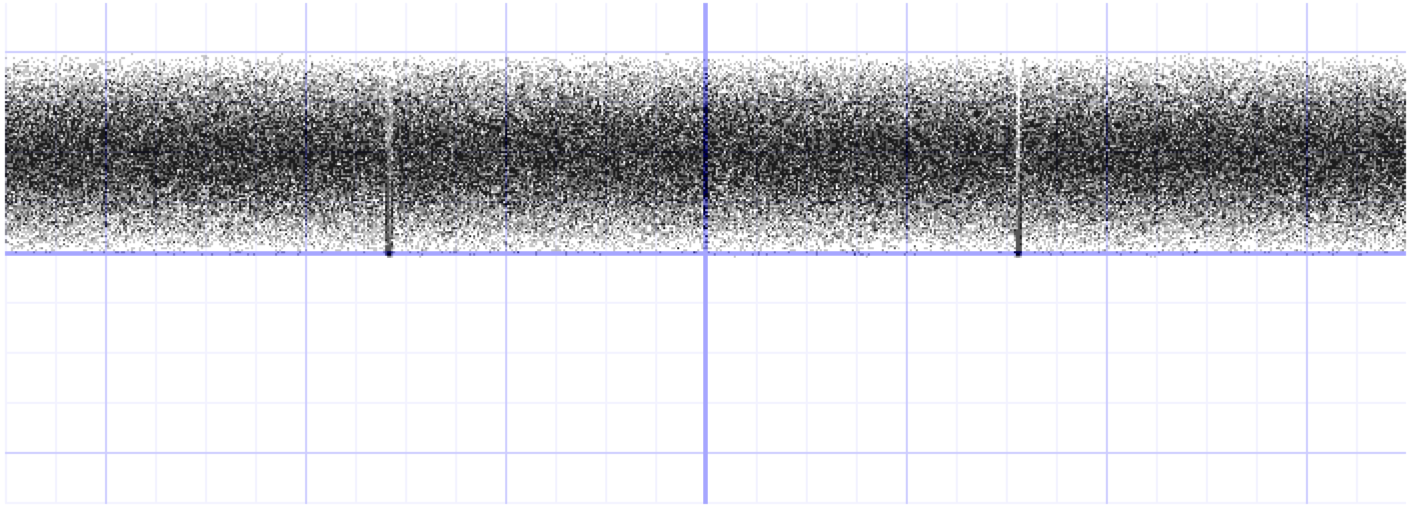
- $y = fract(sin(x) * 1.0)$



[Image courtesy to ***The Book of Shaders***]

# Introduce some randomness (Chaos!)

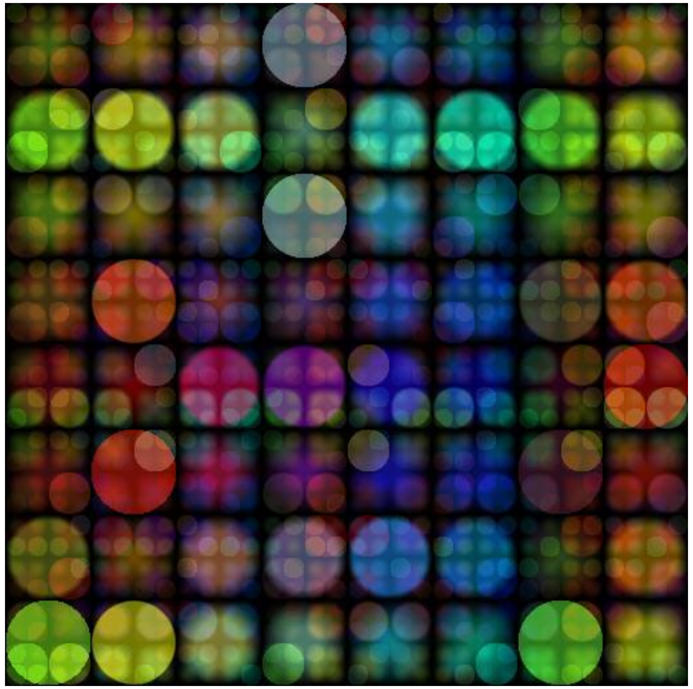
- $y = fract(sin(x) * 100000.0)$



[Image courtesy to ***The Book of Shaders***]

# Introduce some randomness (Chaos!)

```
blur =hsf.fract(ti.sin(float(0.1*t+i//tile_size*5+j//tile_size*3)))
c = circle(pos, center, radius, blur)
```

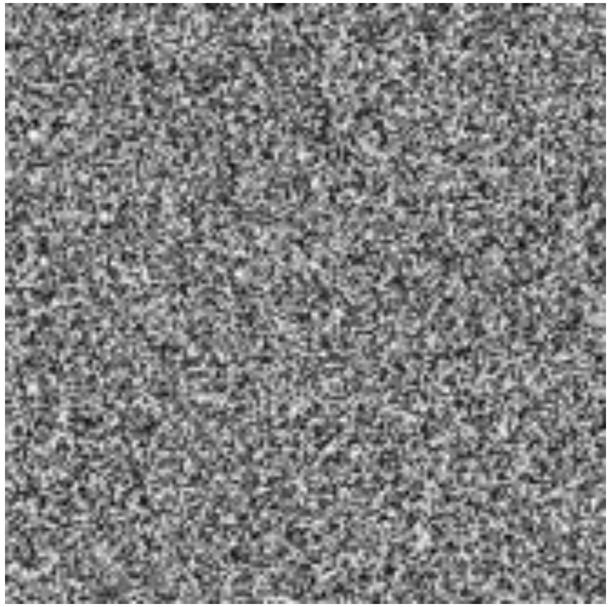


# Balance between: the randomness and the smoothness

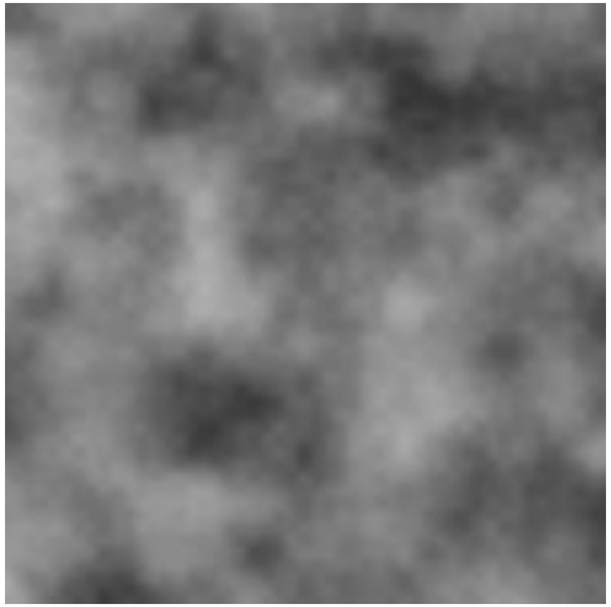
- Perlin noise:
  - https://en.wikipedia.org/wiki/Perlin_noise

# Balance between:
# the randomness and the smoothness

- Perlin noise:
  - https://en.wikipedia.org/wiki/Perlin_noise



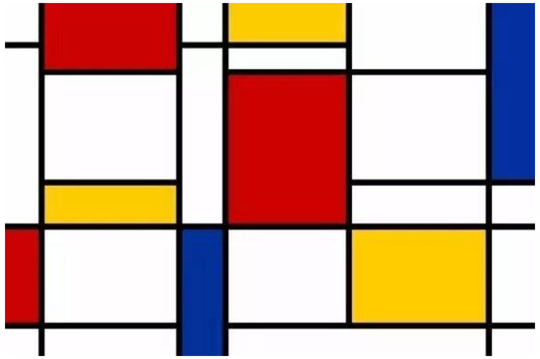
White noise

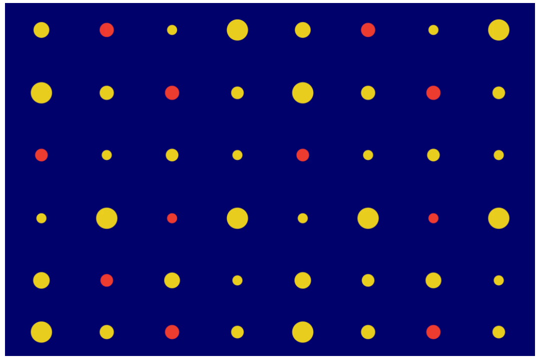


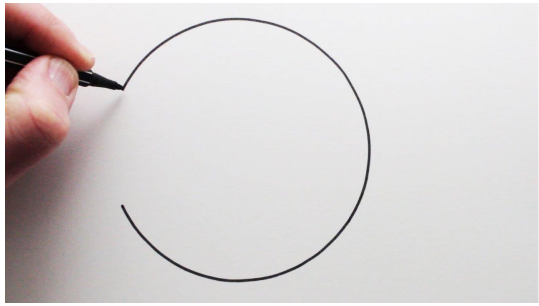
Perlin noise

# Procedural animations:
# a step-by-step example

- Steps:
  1. Setup your canvas
  2. Put some colors on your canvas
  3. Draw a basic unit
  4. Repeat the basic units: tiles and fractals
  5. Animate your pictures
  6. Introduce some randomness (Chaos!)

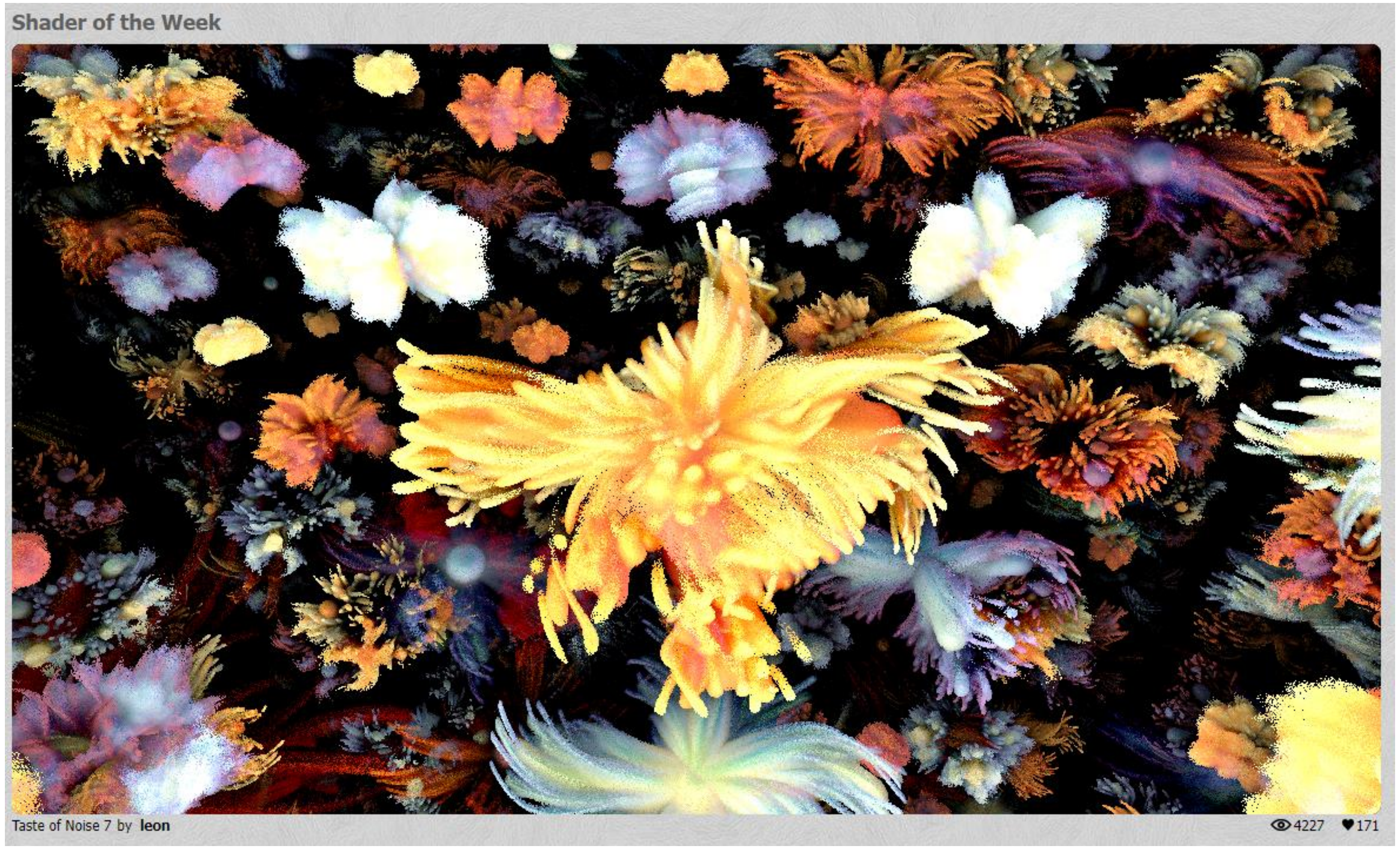






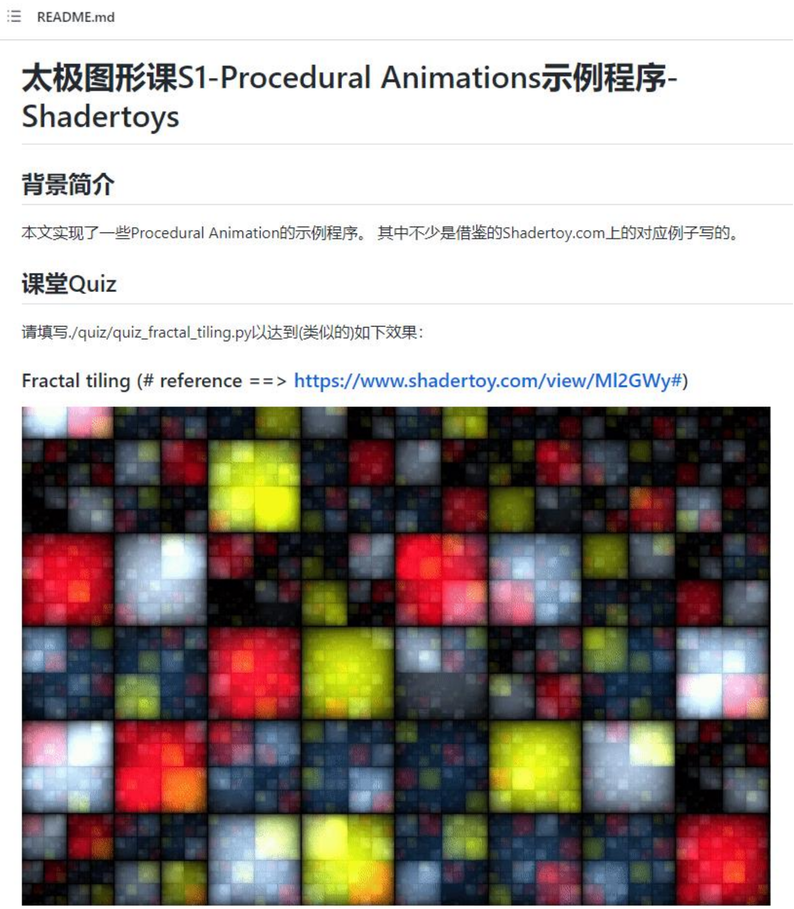
# A great website for procedural animations

- https://www.shadertoy.com/



# Check our compiled examples

- https://github.com/taichiCourse01/--Shadertoys



README.md

## 太极图形课S1-Procedural Animations示例程序-Shadertoys

### 背景简介

本文实现了一些Procedural Animation的示例程序。 其中不少是借鉴的Shadertoy.com上的对应例子写的。

### 课堂Quiz

请填写./quiz/quiz_fractal_tiling.py以达到(类似的)如下效果：

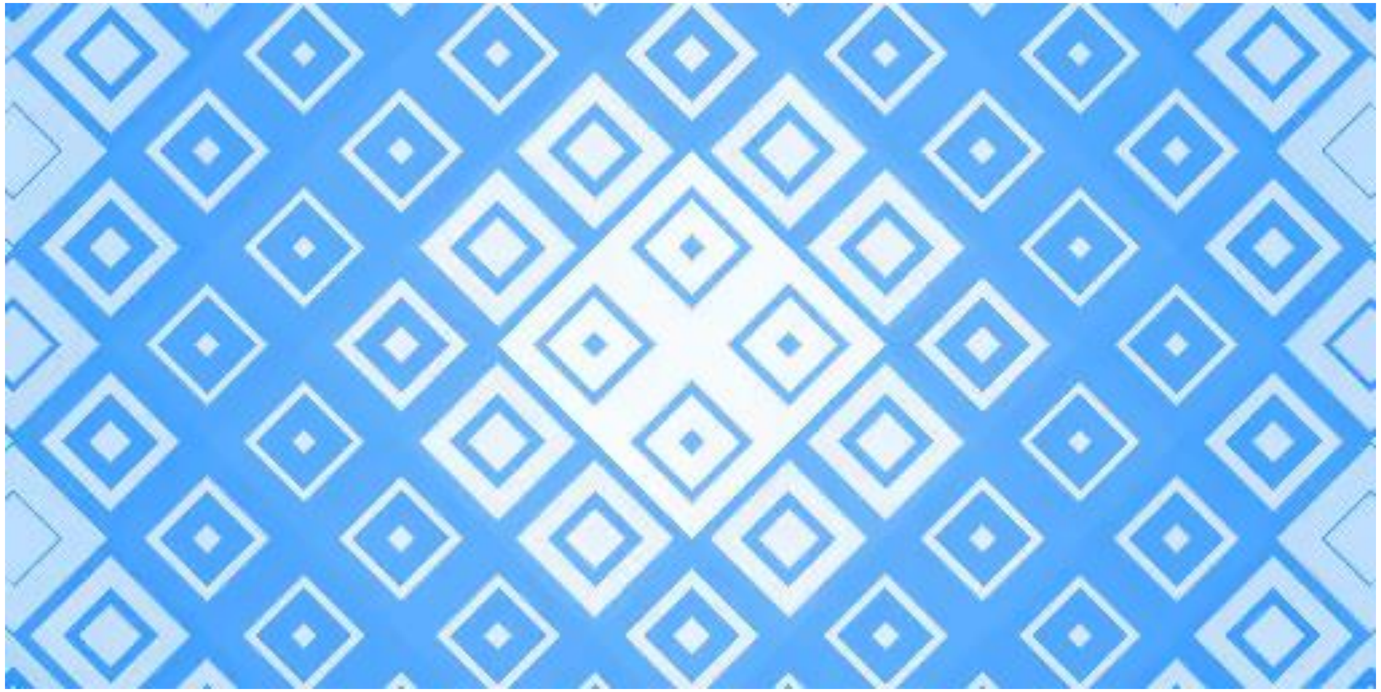
Fractal tiling (# reference ==> https://www.shadertoy.com/view/Ml2GWy#)

# More examples: geometries

- https://www.shadertoy.com/view/MdXSzS

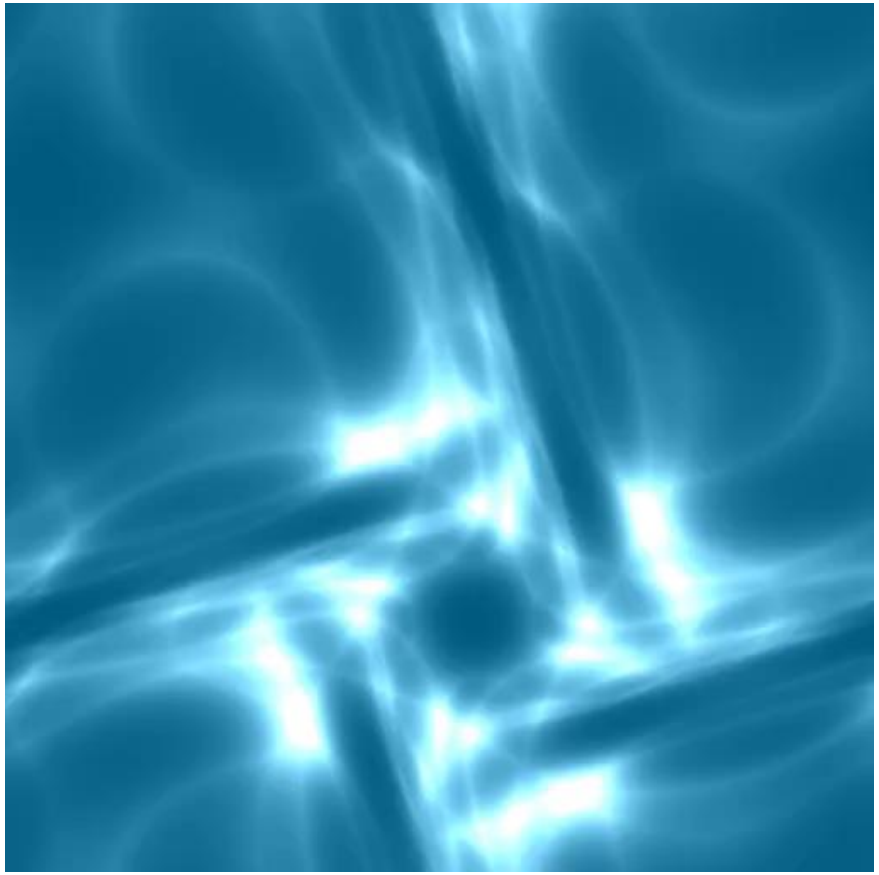
# More examples: geometries

- https://www.shadertoy.com/view/XsBfRW



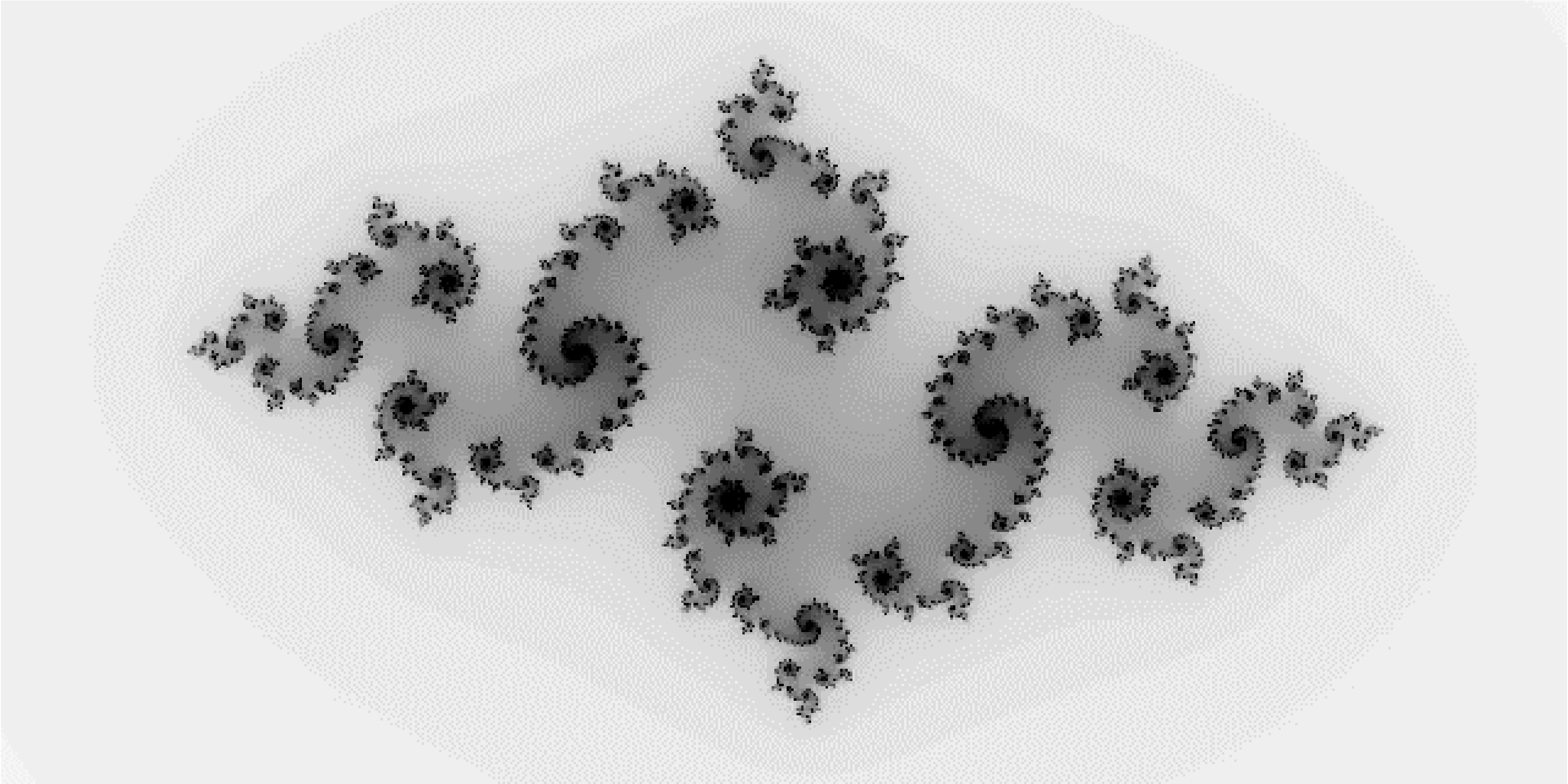
# More examples: random textures

- https://www.shadertoy.com/view/MdlXz8



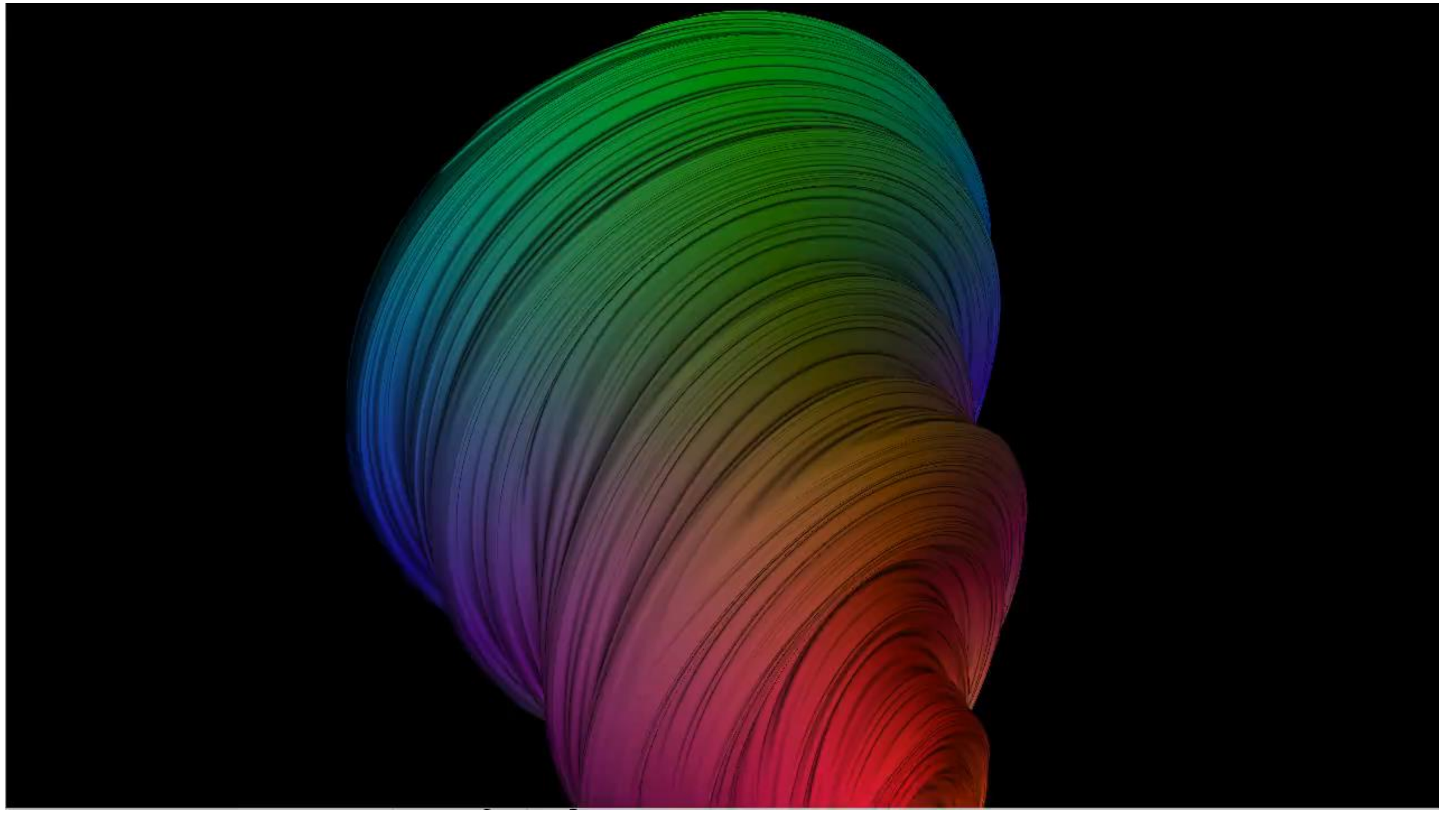
# More examples: fractals

- A 2D Julia-set



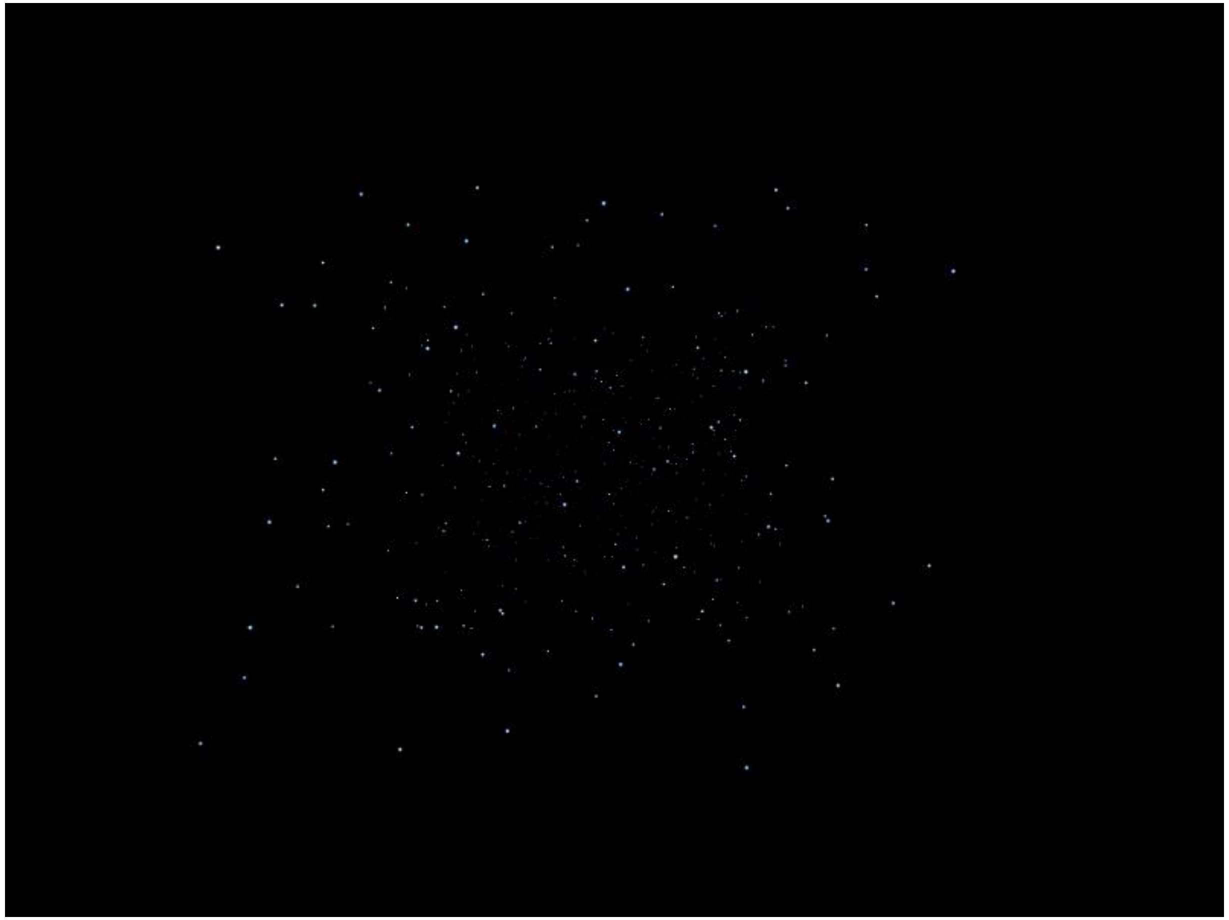
# More examples: fractals

- 3D slides of a 4D Julia-set (by Dunfan Lu @AmesingFlank)



# More examples: a 3D space walkthrough

- Interstellar (by Andrew Sun @victoriacity)

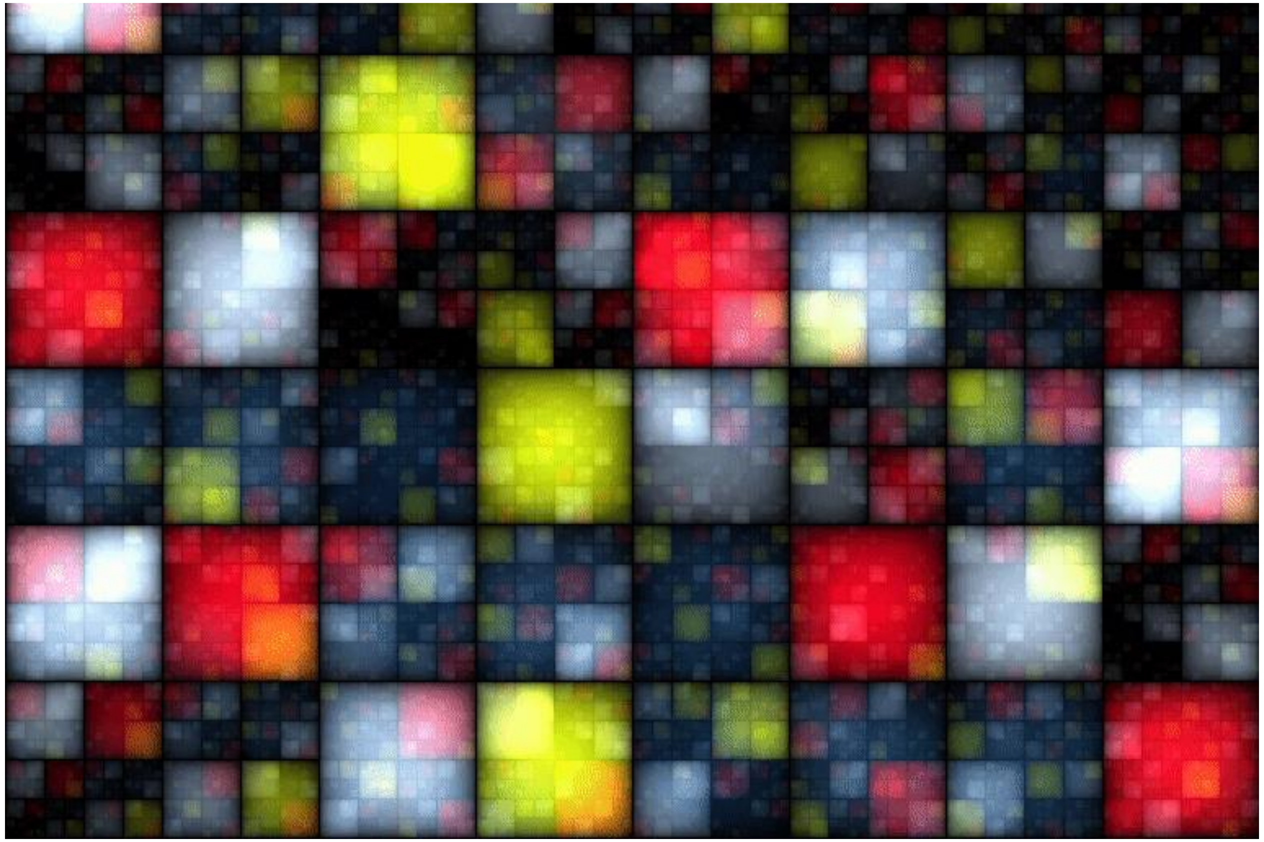


# Quiz

# Quiz:

## Generate the following procedural animations in 15 mins

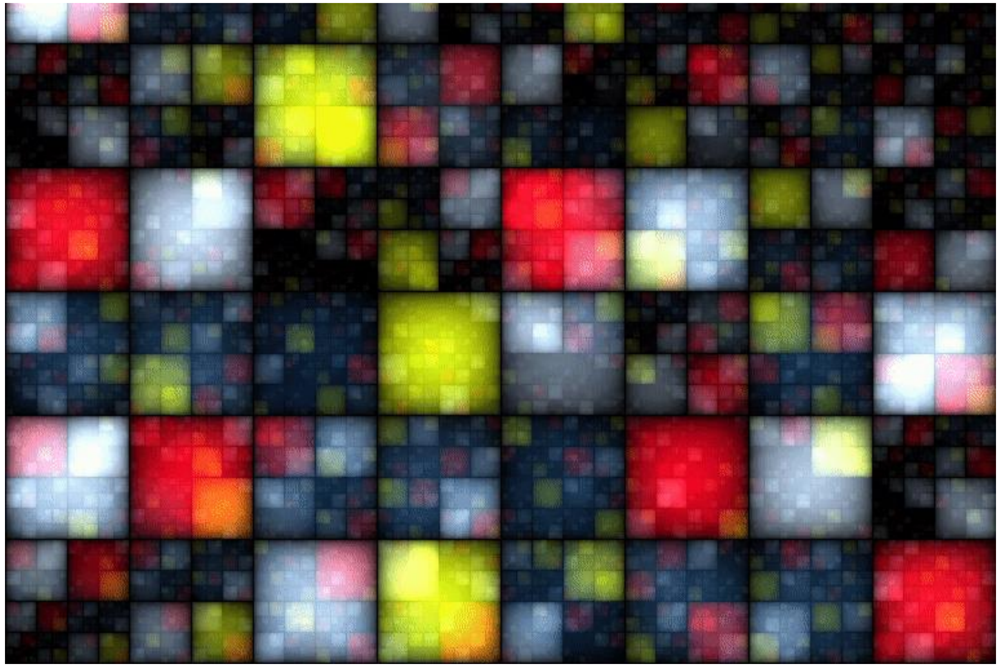
- https://github.com/taichiCourse01/--Shadertoys
  - Check for the quiz folder
  - Fill quiz_fractal_tiling.py



# Remark

- Procedural animation is a lot of fun!
- Your best friends in Procedural animations
  - Tiles, fractals and noises

- References:
  - The book of shaders: [link]



# Homework

# Homework today

- Check shadertoy.com for inspirations
  - Put the webpage references if you did your homework based on any shadertoy examples
  - Try to figure out how your shadertoy works

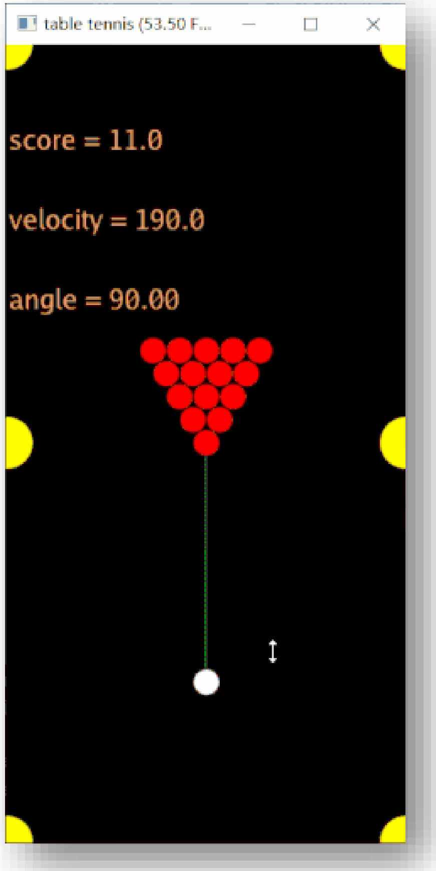
- Check our compiled examples in Taichi, and get some handy helper functions
  - https://github.com/taichiCourse01/--Shadertoys

- Go code your favorite procedural animation!

# Share your homework

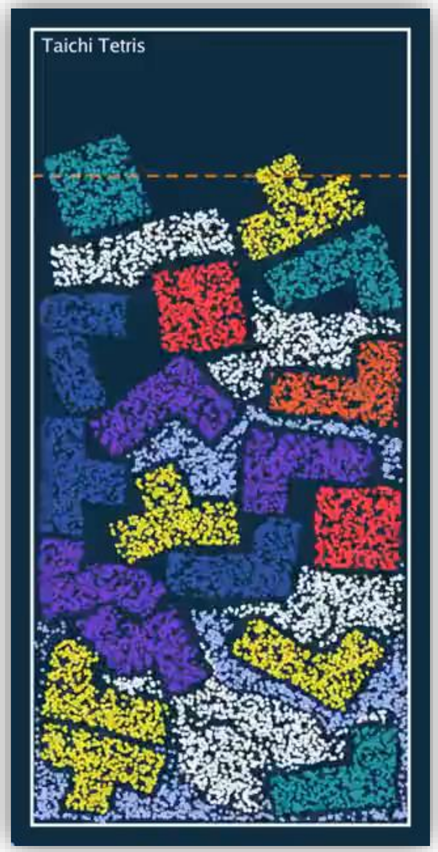
- Could be ANYTHING you programmed using Taichi
- Help us find your homework by using Template
- Share it with your classmates at forum.taichi.graphics
  - 太极图形课作业区: https://forum.taichi.graphics/c/homework/14
  - Share your Taichi zoo link or your github link
  - Compile a .gif animation at your will

# Excellent homework assignments

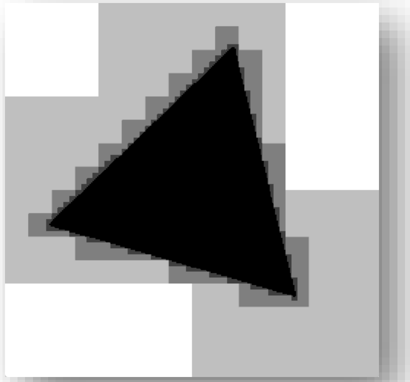


```
score = 11.0

velocity = 190.0

angle = 90.00
```

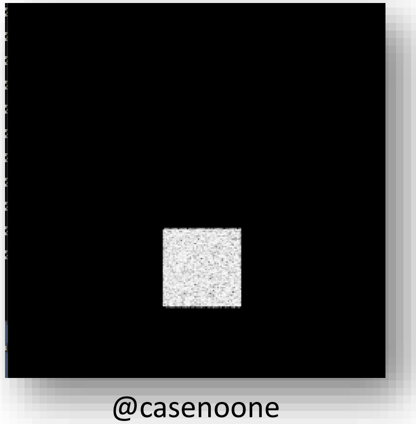
@Pierce-qiang



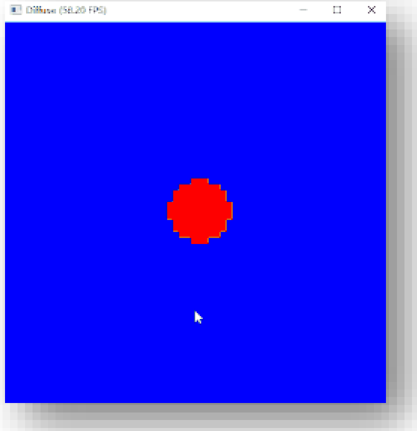
@yuanming-hu
@k-ye



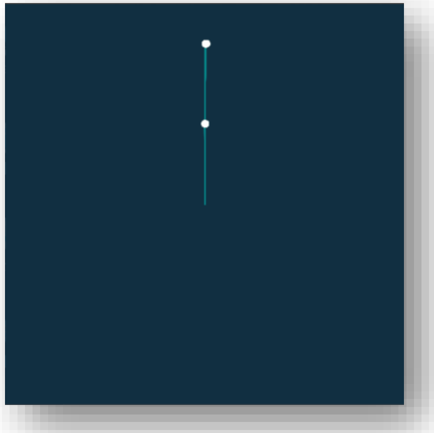
@cflw



@casenoone



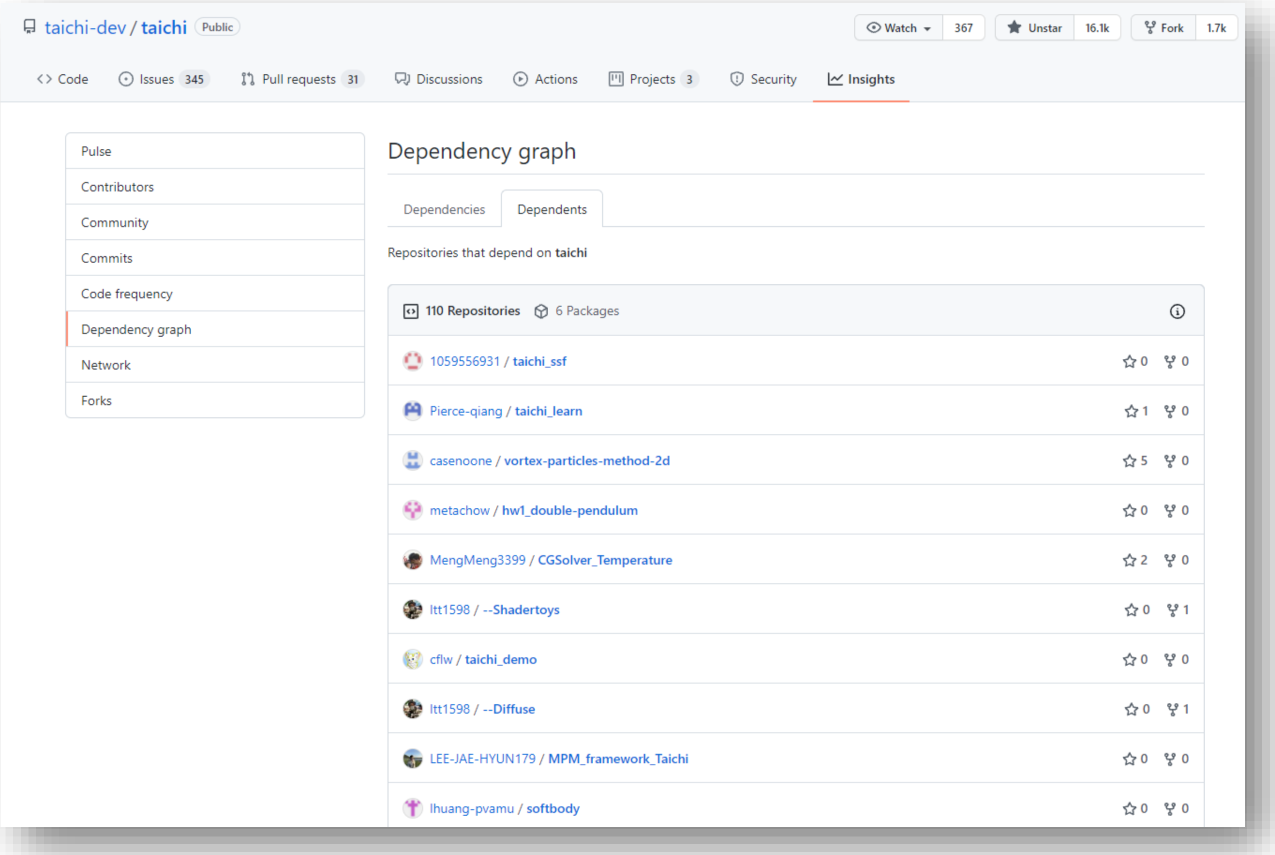
@MengMeng3399



@metachow

# Gifts for the gifted

- Check your Github issues ☺
- Our Next check date will be: Nov. 9th 2021



taichi-dev / taichi Public

Watch 367 | Unstar 16.1k | Fork 1.7k

Code | Issues 345 | Pull requests 31 | Discussions | Actions | Projects 3 | Security | Insights

- Pulse
- Contributors
- Community
- Commits
- Code frequency
- Dependency graph
- Network
- Forks

## Dependency graph

Dependencies | Dependents

Repositories that depend on taichi

110 Repositories | 6 Packages

| Repository | Stars | Forks |
|---|---|---|
| 1059556931 / taichi_ssf | 0 | 0 |
| Pierce-qiang / taichi_learn | 1 | 0 |
| casenoone / vortex-particles-method-2d | 5 | 0 |
| metachow / hw1_double-pendulum | 0 | 0 |
| MengMeng3399 / CGSolver_Temperature | 2 | 0 |
| ltt1598 / --Shadertoys | 0 | 1 |
| cflw / taichi_demo | 0 | 0 |
| ltt1598 / --Diffuse | 0 | 1 |
| LEE-JAE-HYUN179 / MPM_framework_Taichi | 0 | 0 |
| lhuang-pvamu / softbody | 0 | 0 |

# A reminder for the final project

- Deadline **Jan. 3rd 2022**
- Submit using a ~~**private**~~ repo at GitHub/Gitee. Invite tgc01@taichi.graphics to your repo
- Use the Taichi Template to create your repo
- Small-scaled teamwork ($\leq 3$) is welcome, manage your Git commits with care
- Gifts and job/intern opportunity await

# Questions?

本次答疑：10/28

下次直播：11/02

直播回放：Bilibili 搜索「太极图形」

主页&课件：https://github.com/taichiCourse01