



# **Rust China Conf 2020**

**Shenzhen, China**

[2020conf.rustcc.cn](http://2020conf.rustcc.cn)



# 使用 Rust 语言从零开始开发游戏引擎

主讲人：张润哲 (AKA LemonHX)



# 关于我

姓名：张润哲 Jack Zhang

网名：LemonHX

职业：在家听网课的学生

大学：Aberystwyth University (原威尔士国立大学)

邮箱：[Runzhe2001@hotmail.com](mailto:Runzhe2001@hotmail.com) (找实习中欢迎 Offer 砸我脸)

从 2017 年开始使用 Rust 是高级 Rust 黑：



# 游戏引擎架构

我知道看到这个标题会想到叶老师的那本书

整个游戏引擎可以大概简化为下面几大部分：

- 游戏循环：就一大号 `loop{}`
- 输入输出：鼠标键盘游戏手柄啥的
- 渲染引擎：负责把数据画屏幕上的
- 物理引擎：负责制造 `bug` 的
- 脚本引擎：负责写游戏逻辑的 俗称糊 \_\_\_\_




# 游戏循环和输入输出

都什么年代了还写同步代码？

Rust Async 都稳定了，还不快去写异步？

- 什么是 Winit
  - 什么是 EventLoop
  - 怎么在 EventLoop 里面处理 IO
  - 游戏手柄控制 Grils: <https://github.com/Arvamer/gilrs>



```

1 let event_loop = EventLoop::new();
2 let window = WindowBuilder::new().build(&event_loop).unwrap();
3
4 // 开始运行事件循环
5 event_loop.run(move |event, _, control_flow| {
6
7     // 更加适用于一直更新的游戏
8     *control_flow = ControlFlow::Poll;
9     // 更加适用于桌面GUI程序
10    // *control_flow = ControlFlow::Wait;
11
12    // 当有事件触发
13    match event {
14        // 请求关闭窗口
15        Event::WindowEvent {
16            event: WindowEvent::CloseRequested,
17            ..
18        } => {
19            *control_flow = ControlFlow::Exit
20        },
21        Event::MainEventsCleared => {
22            // 所有窗口事件处理完毕
23            window.request_redraw();
24        },
25        Event::RedrawRequested(_) => {
26            // 需要重新渲染时
27        },
28        _ => ()
29    }
30 });

```

```

1 // 包装在MainEventsCleared之后和Update逻辑之前
2 Event::MainEventsCleared => {
3     // 因为Gilrs使用了自己的事件循环
4     // 所以我们只能在Winit的事件循环里面再包装一个循环
5     if let Some(gilrs::Event { id, event, time }) = gilrs.next_event() {
6         gamepad_stat.update_events(&gilrs::Event { id, event, time });
7         // println!("{:?}", gamepad_stat);
8     }
9 }

```

```

1 #[derive(Debug)]
2 pub struct InputPC{
3     keys_down: HashSet<VirtualKeyCode>,
4     mouse_buttons_down: HashSet<MouseButton>,
5
6     // mouse position
7     mouse_position: Vec2,
8     // mouse delta
9     mouse_delta: Vec2,
10    // mouse wheel
11    mouse_wheel_delta: Vec2,
12 }
13
14 // currently only support one gamepad
15 use gilrs::Button;
16
17 #[derive(Debug)]
18 pub struct InputGamepad{
19     keys_down: HashSet<Button>,
20
21     left_trigger2_value: f32,
22     right_trigger2_value: f32,
23
24     left_axis_value: Vec2,
25     right_axis_value: Vec2,
26 }

```



# 渲染引擎

- 什么是 WGPU
- 支持啥平台？
- 为啥不用 OpenGL？
- 等下，这玩意儿还能异步？
- 还能多核渲染？



# WGPU 标准和 WGPU-RS

官网：

<https://www.w3.org/community/gpu/>

“The goal is to design a new Web API that exposes these modern technologies in a performant, powerful and safe manner. It should work with existing platform APIs such as Direct3D 12 from Microsoft, Metal from Apple, and Vulkan from the Khronos Group.”

— 说白了就是前端又想发明新轮子了 —

Rust 的实现：

<https://github.com/gfx-rs/wgpu>

支持的平台 

其实这家伙就是原来的 GFX 换皮



## Supported Platforms

API	Windows 7/10	Linux & Android	macOS & iOS
DX11	✓		
DX12	✓		
Vulkan	✓	✓	
Metal			✓
OpenGL		⚠	⚠





# 异步资源加载

```
let path = path.into();
let config: String = std::fs::read_to_string(path: &path.join("config.json")).ok()?;
let config: HashMap<String, String> = serde_json::from_str(config.as_str()).ok()?;
let name: &String = config.get("name").into()?;
let mut table: HashMap<ShaderStage, Arc<Shader>> = HashMap::new();
if let Some(vert_path: &String) = config.get("vertex").into(){
    let shader: Arc<Shader> = Shader::new(device.clone(), name: &(format!("{}", name, ".vert")),
    path: path.join(vert_path), kind: ShaderStage::Vertex).await?;
    table.insert(k: ShaderStage::Vertex, v: shader);
}
if let Some(frag_path: &String) = config.get("fragment").into(){
    let shader: Arc<Shader> = Shader::new(device.clone(), name: &(format!("{}", name, ".frag")),
    path: path.join(frag_path), kind: ShaderStage::Fragment).await?;
    table.insert(k: ShaderStage::Fragment, v: shader);
}
if let Some(comp_path: &String) = config.get("compute").into(){
    let shader: Arc<Shader> = Shader::new(device.clone(), name: &(format!("{}", name, ".comp")),
    path: path.join(comp_path), kind: ShaderStage::Compute).await?;
    table.insert(k: ShaderStage::Compute, v: shader);
}

self.shader_manager.insert(k: name.clone(), v: table);

println!("shader loaded {:?}", self.shader_manager);
```

- 这里我们看到，我们通过使用 Async 和 Await 去加载 Shader，我相信差不多大部分观众都对 Async 和 Await 的概念有所了解。
- 大家知道 Shader 编译是需要耗费时间的，所以这部分代码使用 Async 能够减少资源加载的延迟。

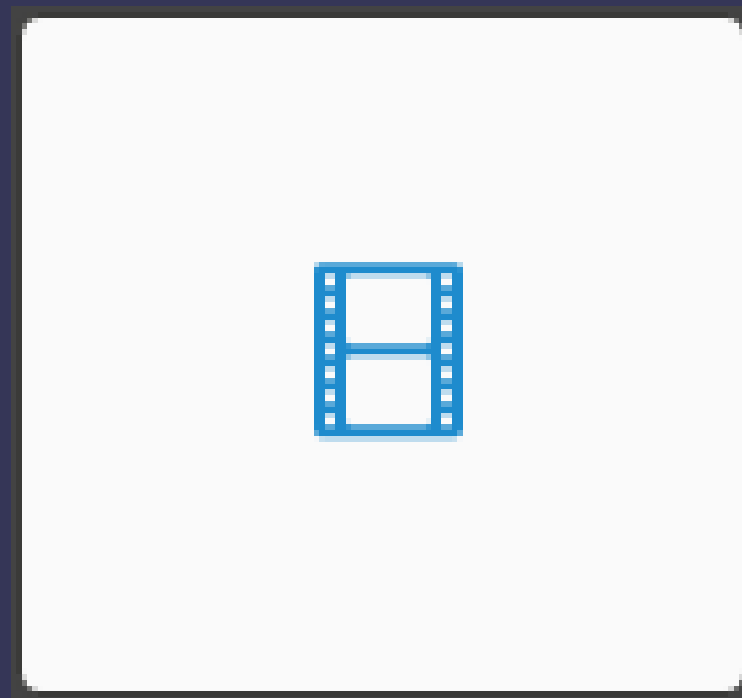
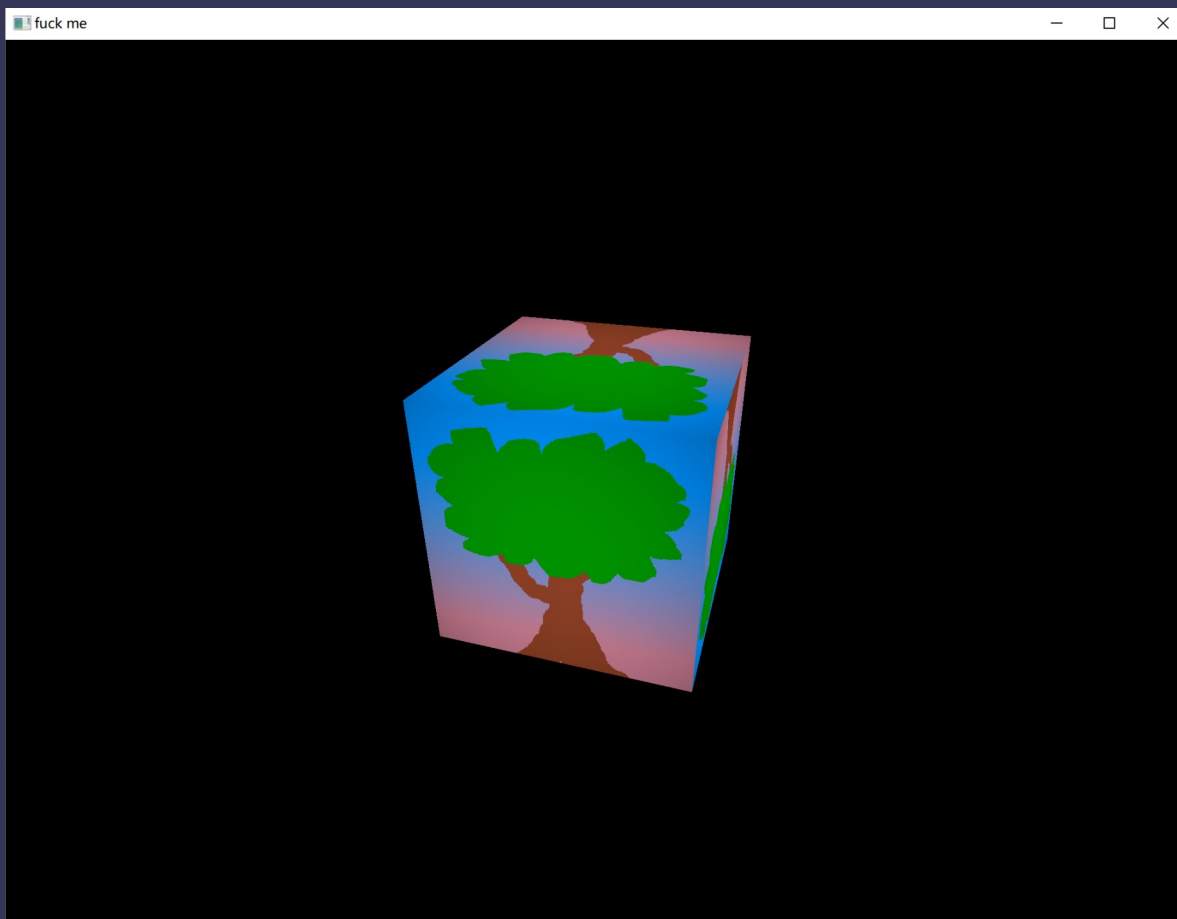


# 多核渲染

- 由于 WGPU-RS 的多核渲染相关的特性和跨平台跨平台之类的别的特性冲突，好像被干掉了（最近 API 改的有点频繁，也懒得读了），所以我们大致讲一下 Vulkan 里面的多核渲染吧，毕竟 WGPU 也是参考 Vulkan 等 API 实现的。
- 第一步，创建 RenderGraph.
- 第二步，画好 RenderGraph.
- 第三步，对 RenderGraph 的每一个节点收集他们的 CommandBuffer.
- 第四步，多核提交给 GPU.



# 上次构建成功.....

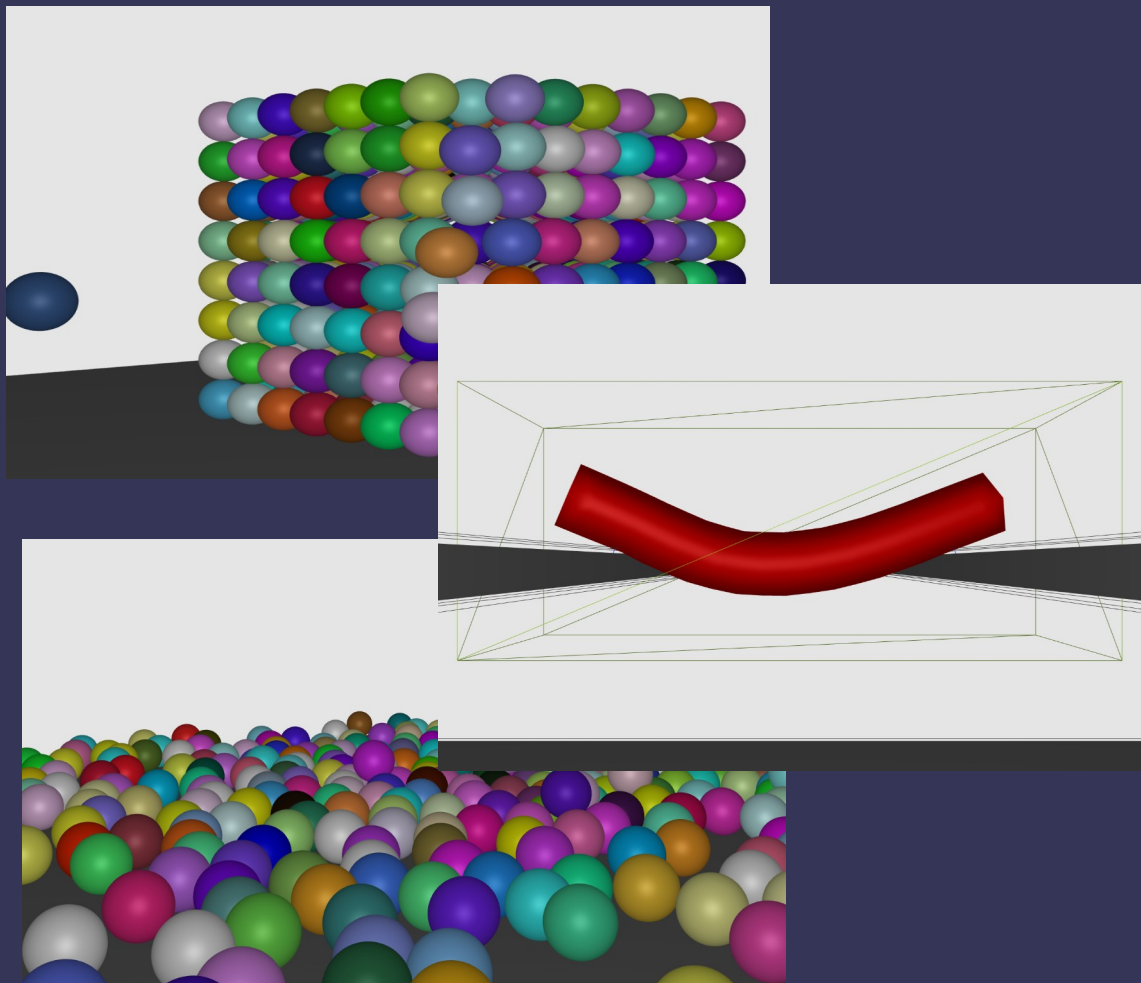




由于时间有限，  
以下部分还没有实现，  
但是相关的生态我还算比较了解...



# 物理引擎



我预期采用的物理引擎是 nphysics, 这应该是 Rust 社区里完成度最高的物理引擎, 当然... 肯定没办法跟 Physx 比, 和 Bullet 还是有的一拼的.

<https://nphysics.org/> 网页没 CDN 请自备工具.

html 4.09 KB 16.65 KB 4820 ms

同时支持各种常见的物理如关节, 刚体, 软体, 布娃娃等.

这个物理引擎可以编译到 WASM 从而完整的运行在 WEB 上, 跨平台属性拉满, 也比较和我的游戏引擎契合



# (物理相关) 声音引擎

- 讲真的，我一看 Rust 游戏制作相关的生态...还算可以，至少比两年前好多了，等到开始研制游戏引擎时，我发现... Rust 居然没有一个靠谱的声音引擎.....
- 基于 OpenAL 的绑定 ALTO <https://github.com/jpernst/alto> 弃坑两年
- Rust 用 Roda 自行封装 Ambisonic <https://github.com/mbillingr/ambisonic> 还算能动...
- FMOD 绑定 <https://github.com/GuillaumeGomez/rust-fmod> 弃坑三年 ..... 真的不会被递律师函吗？



# 脚本引擎

- 现在，游戏引擎大多采用一种语言书写，这样能保证... FFI 不会特别难做，当然为了动态性和性能的取舍，大部分游戏引擎都加入了脚本层或者脚本语言层。
- 比如 Unity 使用 C# 作为脚本语言，但是编译死了... 最后<sup>手游厂商</sup>都使用 LUA 之类的做内嵌的脚本语言。
- UE4 有蓝图和自己拿近乎于魔幻且不可调试的 C++ 作为脚本层 UEC++ 有动态分发，反射等拿宏和模板一起包装的魔幻特性，提前完成 C++30.



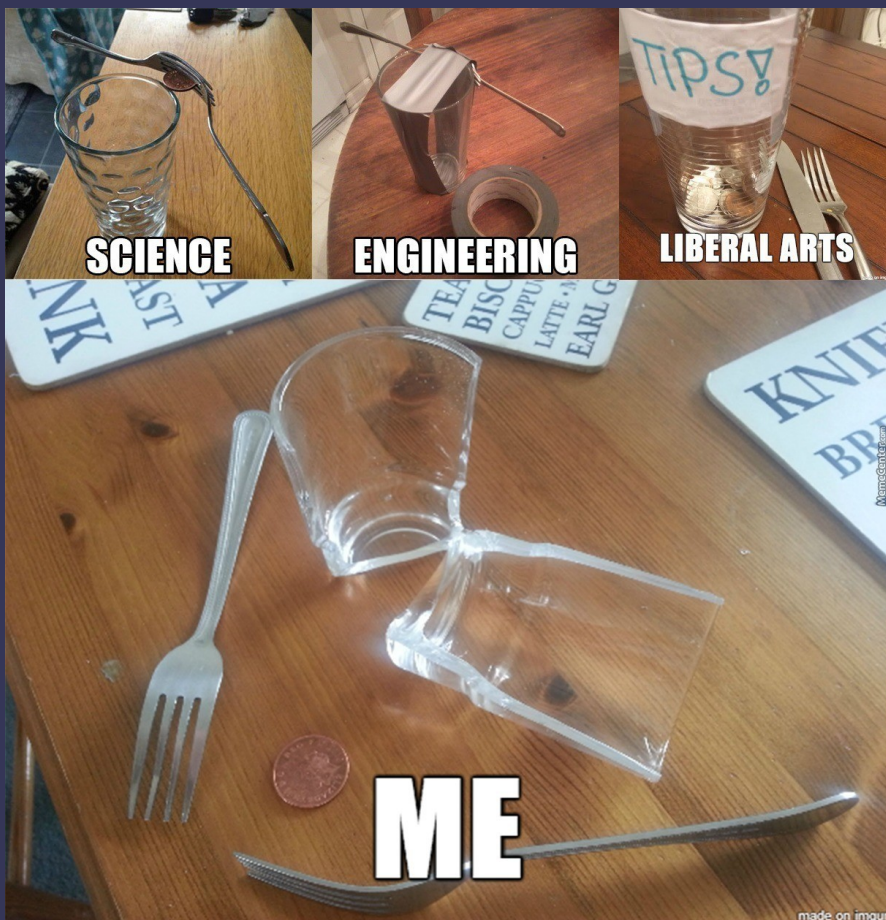
# 脚本语言

- Rust 制作的脚本语言：
  - Dyon
  - Gluon 类型系统很赞
  - Rhai
  - RustPython 隔壁 PythonConf 在拿 Go 调用这个
  - 锤哥的 PR47 开发中遇到坑子
  - ~~WASMER~~ 这东西算脚本？
- 其他脚本语言的 Rust 绑定
  - Rlua 用过一次，体验还行
  - Pyo3 完成度较高
  - Mini-v8 见过一眼...
  - ~~Jni-rs Java 是脚本~~ ✓





# 最后...



- 制作一款游戏引擎绝非是一件容易的事情，但是游戏引擎相关的生态基本能放映这个语言的活跃度，我很希望看到 Rust 语言达到今天这个高度。

- 相关网站：

- <https://arewegameyet.rs/>
- <https://vulkan-tutorial.com/>
- <https://sotrh.github.io/learn-wgpu/>



# 谢谢大家

感谢所有对我提出指导意见的人和我的群友们。

张润哲 (AKA LemonHX)