gdext: Using Rust for Games in Godot

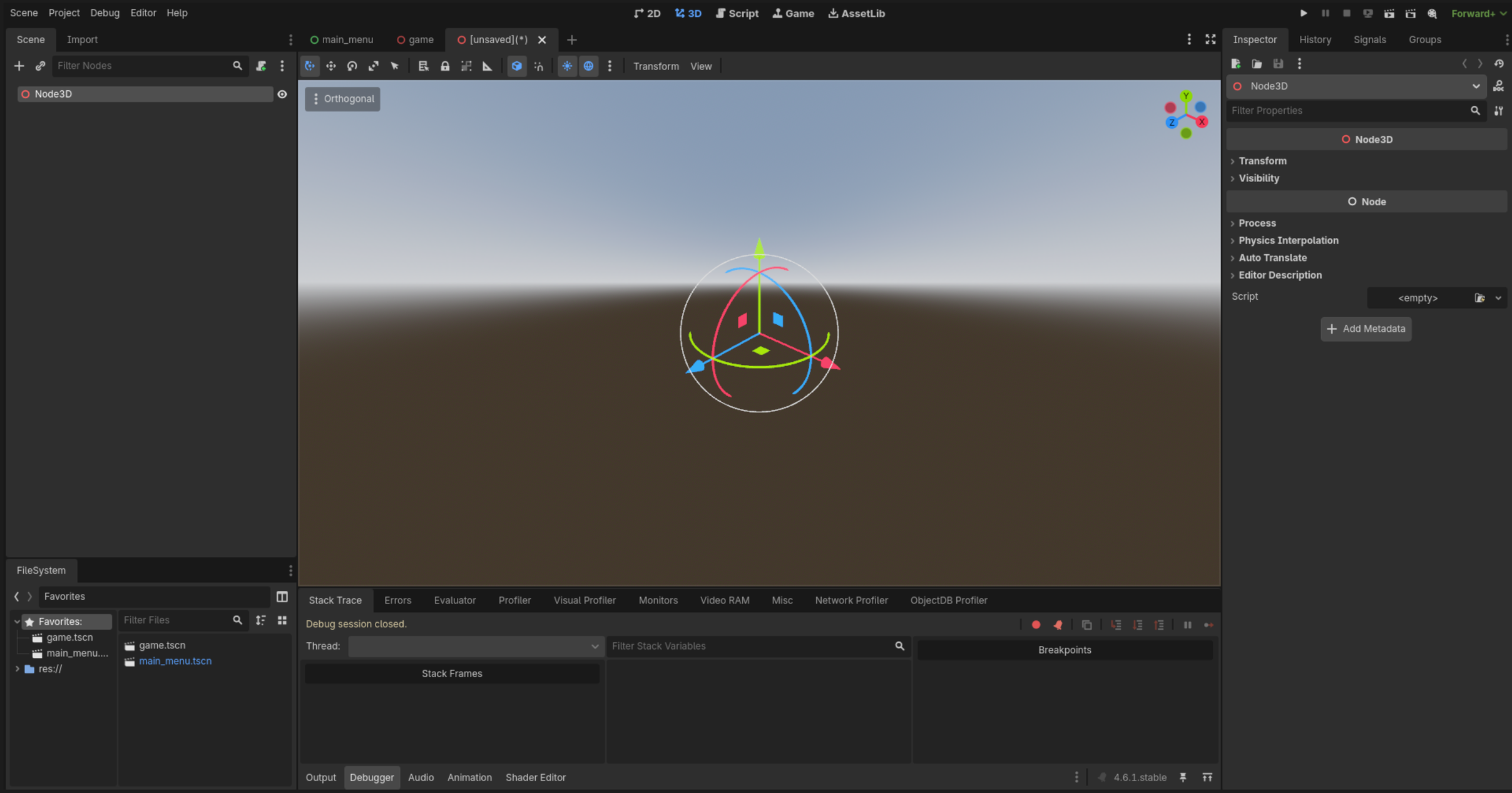
About Me

Choko

Study: Comp Sci @ UTokyo

Hobbies: Gamedev, Graphics Programming, VTubers

X / Github / Discord: @chokomancarr


Scene Project Debug Editor Help
2D 3D Script Game AssetLib
Forward+
Scene Import
main_menu game [unsaved](*)
Filter Nodes
Node3D
Transform View
Orthogonal
Inspector History Signals Groups
Node3D
Filter Properties
Node3D
Transform
Visibility
Node
Process
Physics Interpolation
Auto Translate
Editor Description
Script
<empty>
Add Metadata
FileSystem
Favorites
Favorites:
game.tscn
main_menu....
res://
Filter Files
game.tscn
main_menu.tscn
Stack Trace Errors Evaluator Profiler Visual Profiler Monitors Video RAM Misc Network Profiler ObjectDB Profiler
Debug session closed.
Thread:
Filter Stack Variables
Breakpoints
Stack Frames
Output Debugger Audio Animation Shader Editor
4.6.1.stable

What is Godot?

- Open-source game engine
- Feature-rich (currently v4.6)
- Has a visual editor
- Has its own scripting language (gdscript) that is not Rust

gdscript

- Godot's scripting language
- Object-Oriented
- Hot-Reloadable
- Duck-Typed 🦆😱
- Dynamic-Typed 😱😱😱
- Everything Everywhere All At Once

```
var typed_var: int
var inferred_type := "String"

# Constants.
const ANSWER = 42
const THE_NAME = "Charly"

# Enums.
enum {UNIT_NEUTRAL, UNIT_ENEMY, UNIT_ALLY}
enum Named {THING_1, THING_2, ANOTHER_THING = -1}

# Built-in vector types.
var v2 = Vector2(1, 2)
var v3 = Vector3(1, 2, 3)


# Function, with a default value for the last parameter.
func some_function(param1, param2, param3 = 123):
    const local_const = 5

    if param1 < local_const:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("Fail!")

    for i in range(20):
        print(i)

    while param2 != 0:
        param2 -= 1

    match param3:
        3:
            print("param3 is 3!")
        _:
            print("param3 is not 3!")

    var local_var = param1 + 3
    return local_var
```

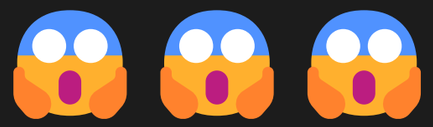
What’s Wrong With Ducks?

```
var wep = Inventory.weapons.get(1)
var mod = wep.modifier[1]
print(mod.name)
```

- This will compile no matter the type of Inventory.weapons
- I will not know if modifier.name is valid... until it runs (or crashes)!

The Cost of Abstractions

- Everything is an Object
- Everything has a HashMap lookup
- Everything is reference-counted*
  - Except when it doesn’t (Engine calls return a copy sometimes)
- Everything is mutable*
  - Except when it isn’t (Packed Arrays are CoW)



Memory usage of Godot's various base classes

Resources Tips & Tricks

thomastc 👑 Jul 2025

I was curious how many bytes e.g. a `Node` takes compared to a plain `Object`, so I modified the engine's code to print the `sizeof` various types. This is on x86_64 Arch Linux, using gcc 15.1.1, but I think the results would be fairly representative of other platforms as well.

...

Finally, the one that really matters, `target=template_release`:

Object: 320
RefCounted: 328
Resource: 496
Node: 968
Node2D: 1344
Node3D: 1232
Control: 2432

As you can see, Node takes about twice as much memory as Resource, and three times as much as RefCounted. RefCounted seems to be a better default choice than Object, since the difference is very small, also in terms of performance.

Why Rust for Games?

- Thread-Safety
- Runtime Stability
- Bottleneck Speedup
- Static-Typing

Why Not Bevy / Fyrox …

- Rust engines are good for prototyping, for now
- Bigger games require a feature rich engine with mature editor
- Iteration speed is a must (no waiting 5 minutes to change color)
- Unity / Unreal is too bloated, Godot is a good compromise (and is open source!)

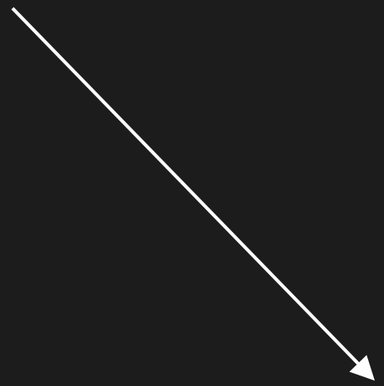
Godot Plugins: GDExtension

GDExtension is a Godot-specific technology that lets the engine interact with native shared libraries at runtime. You can use it to run native code without compiling it with the engine.

There are three primary methods with which this is achieved:

- `gdextension_interface.h` : A set of C functions that Godot and a GDExtension can use to communicate.
- `extension_api.json` : A list of C functions that are exposed from Godot APIs (Core Features).
- *.gdextension: A file format read by Godot to load a GDExtension.

Most people create GDExtensions with some existing language binding, such as godot-cpp (for C++), or one of the community-made ones.



There are no plans to support additional languages officially. That said, the comm
bindings for other languages (see below).

The bindings below are developed and maintained by the community:

- D
- Go
- Java/Kotlin
- Nim
- Rust
- Swift
- Odin

Rust in Godot

The **godot-rust** library provides Rust language bindings for the Godot game engine.

Rust is an alternative to GDScript, with different trade-offs for users. While GDScript enables fast prototyping and short feedback cycles, games of larger scale may benefit from a stronger type system, a rich library ecosystem and native performance.

Cargo.toml

```
[dependencies]
godot = { version = "0.4.5", features = [
    "serde",
    "register-docs",
    "experimental-threads",
] }
```

lib.rs

```rust
use godot::prelude::*;

struct MyExtension;

#[gdextension]
unsafe impl ExtensionLibrary for MyExtension {}
```

mylib.gdextension

```
[configuration]
entry_symbol = "gdext_rust_init"
compatibility_minimum = 4.1
reloadable = true

[libraries]
windows.debug.x86_64 =   "res://rust/target/debug/my_extension.dll"
```

gdext macros

```rust
#[GodotClass]
struct MyClass {
    #[var]
    visible_in_godot: u32,

    pub visible_in_rust: HashMap<usize, f32>,
}

#[godot_api]
impl MyClass {
    #[func]
    fn callable_from_godot(&mut self) -> bool {}

    #[func]
    fn callable_from_godot_static() -> u32 {}

    pub fn callable_from_rust(&self, foo: Option<i8>) {}
}
```

```gdscript
func _ready():
    var x = MyClass.new()
    var y = x.visible_in_godot
    var z = x.callable_from_godot()
    var w = MyClass.callable_from_godot_static()
```

Should You Use Rust in Godot?

- Small games → no
- Medium games → if you need Extra Features 🦀
- Big games → yes! 🦀 🦀 🦀

Extra Features?

- Let’s say we want this function in gdscript:

```
pub const fn from_bits(v: u32) -> f32
```

Raw transmutation from `u32`.

e.g.: `f32::from_bits(0x45d42800)` → `6789.0f`

… it does not exist :/

… Now It Does! :D

```rust
use godot::prelude::*;

/// its insane how godot doesnt have this function
#[derive(GodotClass)]
#[class(no_init)]
9 implementations
struct BitsEncoder {}

#[godot_api]
impl BitsEncoder {
    #[func]
    fn i2f(val: u32) -> f32 {
        f32::from_bits(val)
    }
    #[func]
    fn si2f(val: i32) -> f32 {
        f32::from_bits(val.cast_unsigned())
    }
    #[func]
    fn f2i(val: f32) -> u32 {
        val.to_bits()
    }

```

```
aterial.set_shader_parameter("active", true)

 max_num1 * max_num1:
id()
g.set_pixel(num % max_num1, num / max_num1, Color(BitsEncoder.i2f((r << 24)
```

<Native> class BitsEncoder extends RefCounted

its insane how godot doesnt have this function

What About Bigger Games?


Work in progress examples © hololndie

Every entity has its own logic

Every moving item has belt logic

The Usual Godot Way™ of

```
class_name WorkerEntity extends Block

var working := false
var time := 0

func tick():
    if working:
        time += 1
```

… doesn’t scale when there’s Millions of Entities!

App Structure

Godot

- Rendering
- IO
- Light-Weight Code

Rust

- World Simulation
- Performance-Critical Code

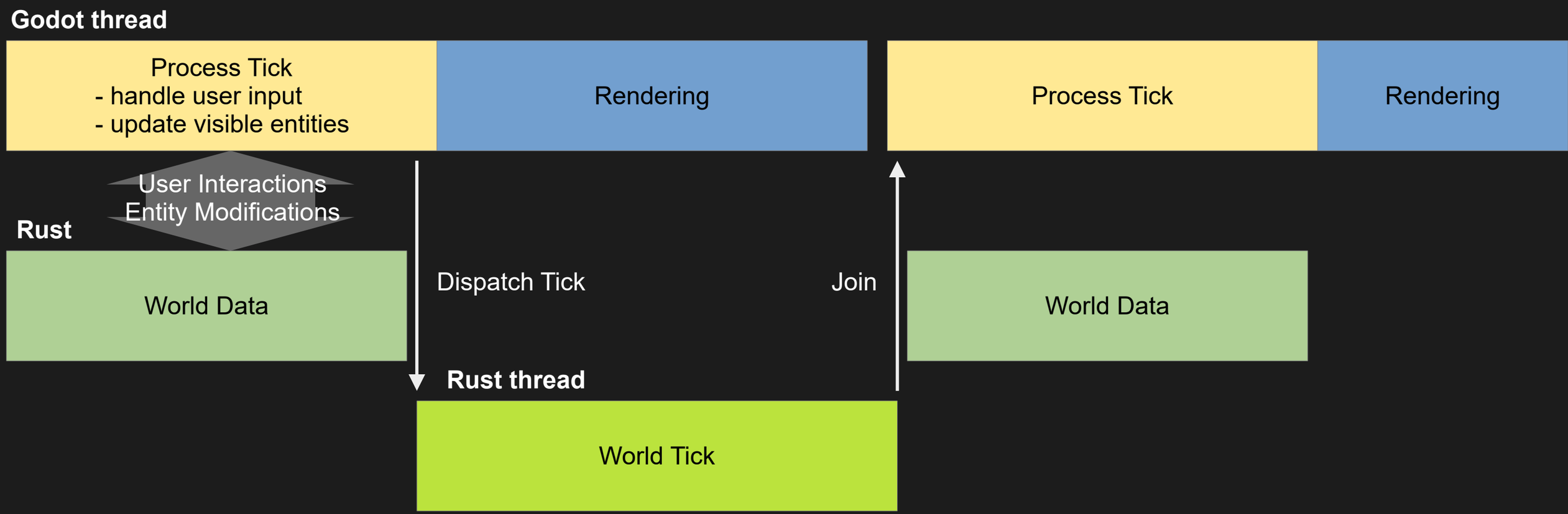
Server Backend.rs

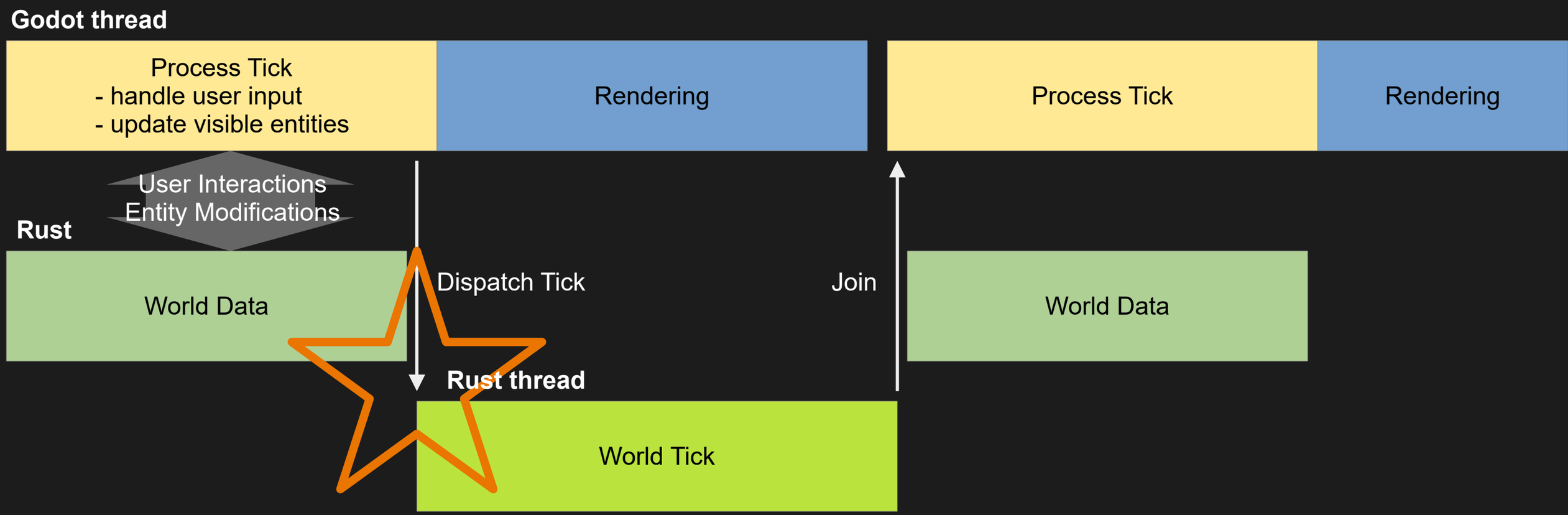
- Belt Solver (DAG)
- Entity Logic
- Entity Connections
- Custom Addons for Instanced Rendering

language	files	code	comment	blank	total
GDScript	259	16,066	396	3,751	20,213
Rust	151	15,237	1,615	2,219	19,071

Godot is single threaded

*technically there is a multithreaded version, but its “buggy and should be avoided in production”


Godot thread
Process Tick
- handle user input
- update visible entities
Rendering
Process Tick
Rendering
User Interactions
Entity Modifications
Rust
World Data
Dispatch Tick
Join
World Data
Rust thread
World Tick



It Just Works™ !

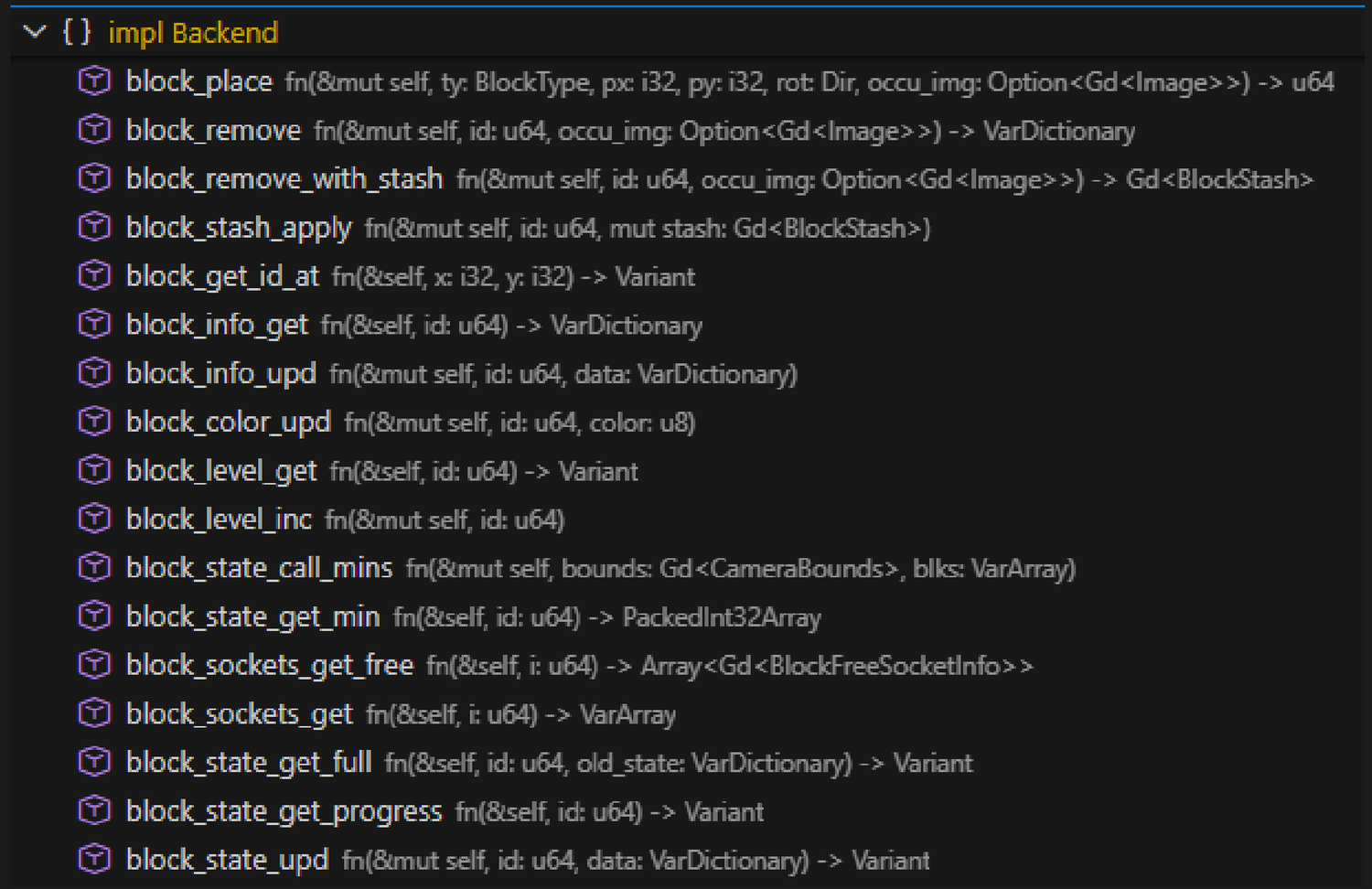
Rust's concurrency safety guarantees that World Tick will never corrupt the world / Godot data!

Change The World?

It is very clear which server call modifies the world

via observing (&mut self)

- Debugging is a lot easier!



Case Studies

Case Study 1: Processing Entities

Entities

For each tick, call a specialized step() for each entity

Case Study 1: Processing Entities

```cpp
#include <vector>
#include <memory>

class Entity {
public:
    virtual void step();
};

class EntityState {
    std::vector<std::unique_ptr<Entity>> entities;

public:
    void step() {
        for (auto& e: entities) {
            e->step();
        }
    }
};


//specific implementations for each entity...

class Miner : Entity {
    void step() override {
        // ...
    }
};
```

- A list of different entities, each having a step() function…
- … it might be tempting to write it like this…
- … but it’s very cache unfriendly, and introduces an indirection! (very slow)

Using unions instead

```cpp
#include <vector>

enum class EntityType { M, D, B };

class Miner {
public:
    void step() {
        // ...
    }
};

union EntityUnion {
    Miner miner;
};

struct EntityItem {
    EntityType ty;
    EntityUnion eu;
};

class EntityState {
    std::vector<EntityItem> entities;

public:
    void step() {
        for (auto& e: entities) {
            switch (e.ty) {
                case EntityType::B: {
                    e.eu.miner.step();
                    break;
                }
            }

        }
    }
};
```

- Keep all the state objects contiguously with a union
- Step through them by checking the type

Using unions instead

```cpp
#include <vector>

enum class EntityType { M, D, B };

class Miner {
public:
    void step() {
        // ...
    }
};

union EntityUnion {
    Miner miner;
};

struct EntityItem {
    EntityType ty;
    EntityUnion eu;
};

class EntityState {
    std::vector<EntityItem> entities;

public:
    void step() {
        for (auto& e: entities) {
            switch (e.ty) {
                case EntityType::B: {
                    e.eu.miner.step();
                    break;
                }
            }

        }
    }
};
```

- Keep all the state objects contiguously with a union
- Step through them by checking the type

We checked the wrong type when accessing the union!

Aw Dang It! *Program explodes under Undefined Behavior*

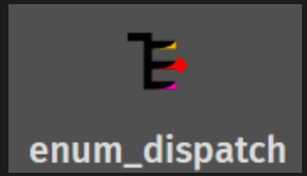
Doing it safely in Rust

```rust
enum EntityItem {
    M(Miner),
    D(Driller),
    B(Burner),
}

struct EntityState {
    entities: Vec<EntityItem>;
}

impl EntityState {
    fn step(&mut self) {
        for e in &mut self.entities {
            match e {
                M(miner) => miner.step(),
                D(driller) => driller.step(),
                // ...
            }
        }
    }
}
```

- The object is contained in the type, so it is impossible to get it wrong!
- The data is still stored contiguously, so looping is fast.
- Compile to tagged unions, so Zero Cost in addition to safety!

Static Dispatching

```rust
#[enum_dispatch]
enum EntityItem {
    M(Miner),
    D(Driller),
    B(Burner),
}

#[enum_dispatch(EntityItem)]
trait EntityCanStep {
    fn step();
}

struct EntityState {
    entities: Vec<EntityItem>;
}

impl EntityState {
    fn step(&mut self) {
        for e in &mut self.entities {
            e.step();
        }
    }
}
```



Case Study 2: Minimizing Godot Work

Godot thread

Process Tick
- handle user input
- update visible entities

User Interactions
Entity Modifications

Rust

World Data

“Should Godot change the playing animation?”

Case Study 2: Minimizing Godot Work

- We do not want to update our rendering data every frame
- We poll the server: “which visible entities have changed?”

```rust
impl BlockStateInner for BlockStateBurner {
    state_inner_common! {}

    fn step(&mut self, args: BlockStepArgs) -> NextStepTick {
        self.modifier.step(args)
    }

    state_hasmin! {self, _info, procs,
        Actv: self.modifier.time_spent > 0,
        Fuel: (self.modifier.powers[0] * 3).div_ceil(FUEL_LENGTH),
        Type: self.modifier.recipe.as_ref().map_or(0, |r| r.input as i32),
        Outd: procs[0].is_some_and(|v| v.num == 99)
    }
```

Hash an [i32; N] array to check if state changed

```
macro_rules! state_hasmin {
    ($sel:ident, $info:ident, $proc:ident, $k1:ident: $v1:expr $(, $k2:ident: $v2:expr)*) => {
        fn get_state_min(&$sel, $info: &BlockInfo, $proc: &BlockProdBuf, last_hash: u64, mins: &mut [i32]) -> Option<u64> {
            use std::hash::Hasher;
            let mut hasher = std::hash::DefaultHasher::new();

            let tmp = ($v1) as i32;
            tmp.hash(&mut hasher);
            mins[crate::StateMinItemTy::$k1 as usize] = tmp;
            $(
                let tmp = ($v2) as i32;
                tmp.hash(&mut hasher);
                mins[crate::StateMinItemTy::$k2 as usize] = tmp;
            )*

            let hash = hasher.finish();
            (hash != last_hash).then_some(hash)
        }
    };
}
```

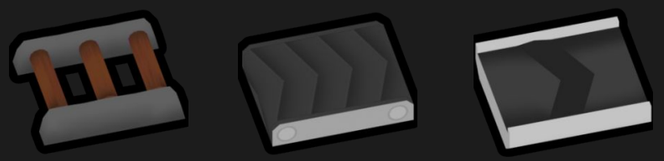
Minimizing Godot Work

3. Pass the array to Godot to update rendering if Hash is different

```
func bk_on_upd_state_mins(inst: BlockInstInfo, v: PackedInt32Array):
    play_anim(inst.uid, "burning" if v[SM.Actv] else "idle")
    bytes2sbuf(inst.uid, v[SM.Type], 0, inst.level, 1)
```

Case Study 3: Rendering Thousands of Dynamic Entities

We want to draw every belt on the screen in a single GPU instruction



Case Study 3: Rendering Thousands of Dynamic Entities

The Naive Way:

- Spawn 1 object for each entity
- Game stutters every time the player moves
- Game crashes when loading

The Vulkan / OpenGL Way:

- Use instancing

glDrawArraysInstanced behaves identically to glDrawArrays except that *instancecount* instances of the range of elements are executed and the value of the internal counter *instanceID* advances for each iteration. *instanceID* is an internal 32-bit integer counter that may be read by a vertex shader as `gl_InstanceID`.

glDrawArraysInstanced has the same effect as:

```
if ( mode or count is invalid )
    generate appropriate error
else {
    for (int i = 0; i < instancecount ; i++) {
        instanceID = i;
        glDrawArrays(mode, first, count);
    }
    instanceID = 0;
}
```

Instancing

Mesh Data

+

Instance Matrices Array

1	0	0	0	1	0	0	2	1	0	0	3	1	0	0	2
0	1	0	0	0	1	0	0	0	1	0	1	0	1	0	3
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

→ Flattened as [float], then reintepreted as [u8]

=

MultiMesh

Inherits: Resource < RefCounted < Object

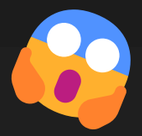
Provides high-performance drawing of a mesh multiple times using GPU instancing.

```
PackedFloat32Array buffer = PackedFloat32Array()
```

- `void set_buffer(value: PackedFloat32Array)`
- `PackedFloat32Array get_buffer()`

There is currently no description for this property. Please help us by contributing one!

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.



```
for i in range($School.multimesh.instance_count):
    var position = Transform3D()
    position = position.translated(Vector3(randf() * 100 - 50, randf() * 50 - 25, randf() * 50 - 25))
    $School.multimesh.set_instance_transform(i, position)
```

https://docs.godotengine.org/en/stable/tutorials/performance/vertex_animation/animating_thousands_of_fish.html

What if I want to use the nice Rust iterators?

1. Create a wrapper for MultiMesh...

```rust
#[derive(GodotClass)]
#[class(base=MultiMesh)]
16 implementations
pub struct MultiMeshIndirect {
    rid: Rid,
    cmd_rid: Rid,
    buf_rid: Rid,
    capacity: i32,
    base: Base<MultiMesh>,
}
```

2. Create a wrapper for add_entity() and remove_entity()

```rust
#[derive(Default, Debug)]
3 implementations
pub struct MMBufferInnerInd {
    pub num: usize,
    num_visible: usize,

    lookup: HashMap<u32, usize>,
    pub buffer: Vec<(u32, [f32; 16])>,
    pub buffer_dirty: bool,

    pub visible_bufs: HashSet<usize>,
    pub array: PackedByteArray,

    pub mm_targets: ArrayVec<Gd<MultiMeshIndirect>, CAP: 2>,
}
```

3. Use the nice Rust iterators to compose matrices!

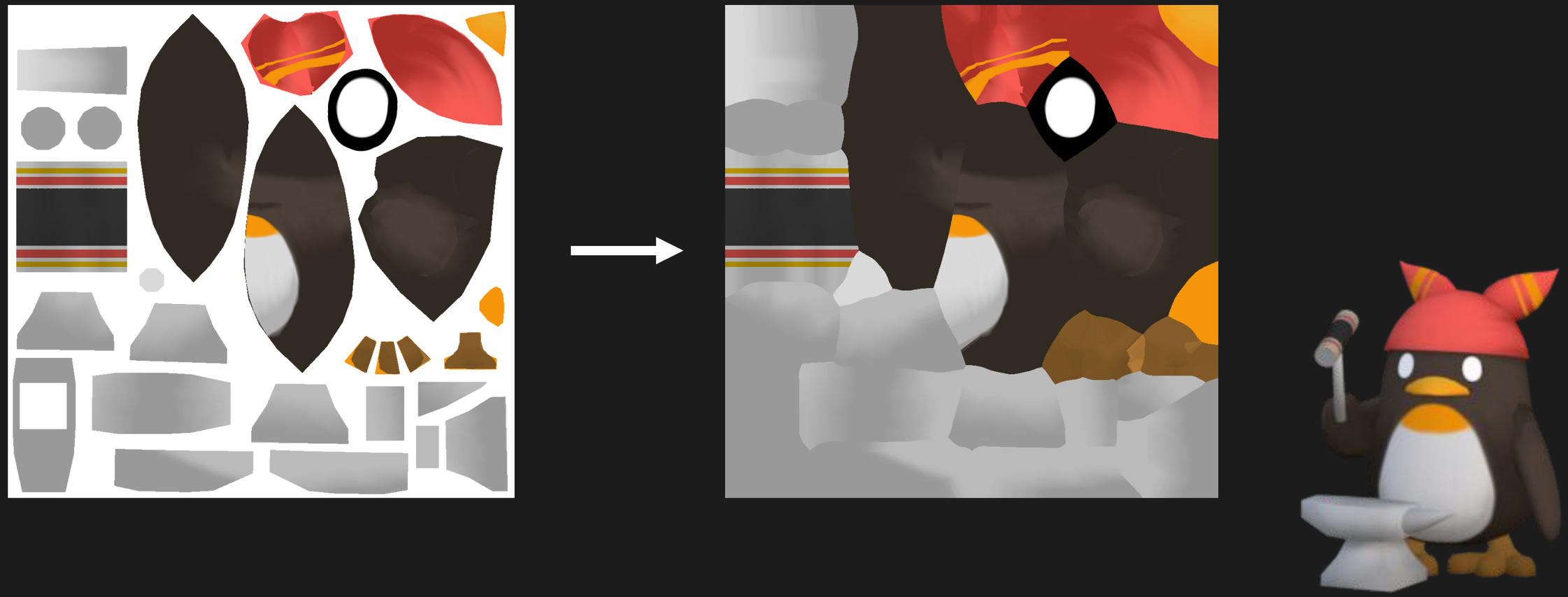
```rust
pub fn apply_buffer(&mut self) -> bool {
    if self.buffer_dirty {
        self.buffer_dirty = false;

        self.array.clear();
        self.array.extend(iter: self.visible_bufs.iter().flat_map(|v: &usize| self.buffer[*v].1).flat_map(|f: f32| f.to_ne_bytes()));
```

“Matrices, flattened as [float], then reintepreted as [u8]”
The Rust Way 🦀

* if the matrix is stored contiguously, the double flat_map is optimized into a single slice memcpy! Zero Cost!

Case Study 4: Texture Dilation

- Extending the edges of textures to avoid bleeding in mipmaps



Special algorithm to dilate tiled texture

- I do NOT want to write this in Godot

Pass the image to Rust, and operate on the inner slice!

```rust
#[derive(GodotClass)]
#[class(no_init)]
9 implementations
struct DilateEntitiesTex {}

#[godot_api]
impl DilateEntitiesTex {
    #[func]
    fn exec(img: Gd<Image>) -> PackedByteArray {
        let data: PackedArray<u8> = img.get_data();
        let data: &[u8] = data.as_slice();
        let sz: usize = img.get_width() as usize;
        let (data: &[[u8; 4]], []) = data.as_chunks::<4>() else {
            panic!("expected 4 bytes slicable!");
        };
        assert_eq!(data.len(), sz * sz, "expected square image with 4 bytes per pixel!");

        let mut res: Vec<[u8; 4]> = vec![[0; 4]; data.len()];

        let w: usize = sz / 8;

        for i: usize in 0..8 {
            for j: usize in 0..8 {
                godot_print!(" .. dilating {i}, {j}...");
                let x0: usize = i * w + j * w * sz;
                Self::do_one(w, sz, src: &data[x0..], dst: &mut res[x0..]);
            }
        }

        res.into_flattened().into()
    }

    fn do_one(w: usize, sz: usize, src: &[[u8; 4]], dst: &mut [[u8; 4]]) {
```

Image buffer is just a 2-dim array of RGBA bytes

The actual algorithm works on pure Rust slices!

Summary

Rust in Godot is good for big projects…

- When you need extra language features
- When you need Fearless Concurrency
- When you need Zero Cost Abstractions
- When you need strong typing

Cons

- Learning Curve
- FFI boundary overhead (when dereferencing Gd<> or calling a function)
- Slower Iteration Speed

Thank You!

Questions?

Try the free demo of my factory game! :D

Chattini Jetpack Project

https://store.steampowered.com/app/4434670