An Overview to Game Development Using Rust

A Toxic Relationship With Rust

Marti

OmniMeet

November 25, 2025

Where am I?

Bevy Game Engine

Bevy is an open-source data-driven game engine built in Rust.

- It emphasizes simplicity, modularity, and performance.
- Bevy uses an Entity-Component-System (ECS) architecture.
- It provides a range of features including 2D/3D rendering, audio, input handling, and more.

Where to Learn More

- **Official Website**: https://bevy.org/
  - Main hub for Bevy news and documentation
- **Learning Resources**: https://bevy.org/learn/
  - Official learning guide and tutorials
- **Examples**: https://bevy.org/examples/
  - Interactive examples covering all features
- **GitHub Repository**: https://github.com/bevyengine/bevy
  - Source code and issue tracking
- **Bevy Cheatbook**: https://bevy-cheatbook.github.io/
  - Community-driven cookbook with snippets and dark knowledge

Where am I?

Advantages of Bevy

- **Rust Language**: Memory safety without garbage collection, zero-cost abstractions, and fearless concurrency.
- **ECS Architecture**: Promotes clean code organization, scalability, and high performance through data-oriented design.
- **Cross-Platform**: Deploy to Windows, macOS, Linux, Web (WASM), iOS, and Android from a single codebase.
- **Open Source**: MIT/Apache 2.0 licensed, actively maintained by a vibrant community.
- **Code-Driven**: Pure code workflow with no lock-in to proprietary editors (Official editor in development).
- **Modular Design**: Use only what you need - built as a collection of plugins you can mix and match.

Bevy vs Other Engines

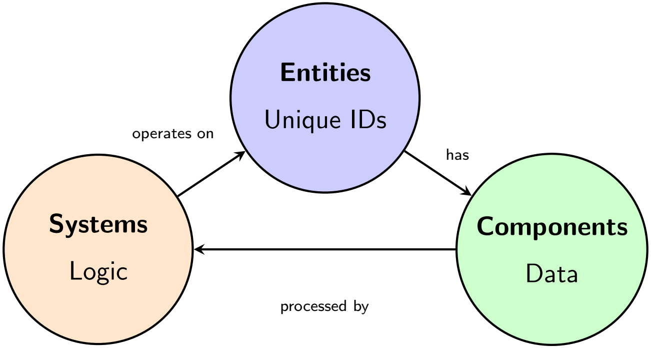
Godot Unity Unreal Engine

- **Lightweight**: Lightweight compared to larger engines.
- **Flexibility**: More control over low-level systems and architecture.
- **Paradigm**: ECS is still not really popular in general.

Where am I?

Outline

Entity-Component-System (ECS)



Entities

Unique identifiers representing objects in the game world

Components

Data containers that hold attributes of entities

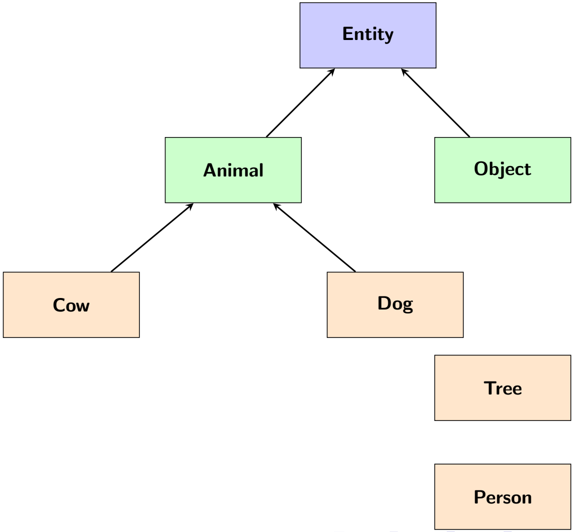
Systems

Logic that operates on entities with specific components

Outline

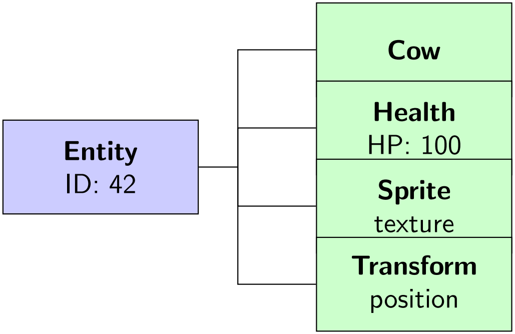
Traditional OOP Approach

Shitty Inheritance Hierarchy



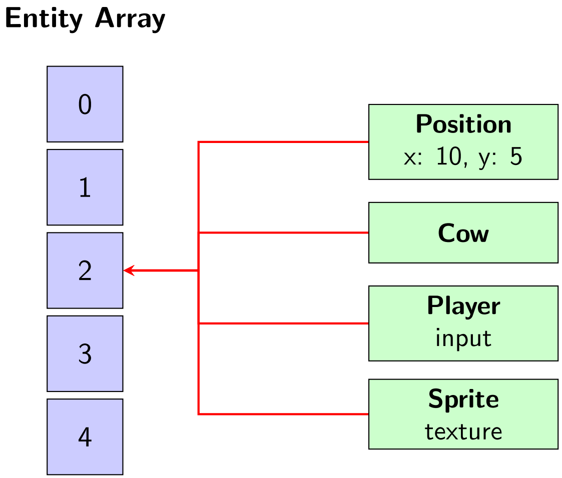
ECS Approach

Nice GIGACHAT and Clean Composition



Outline

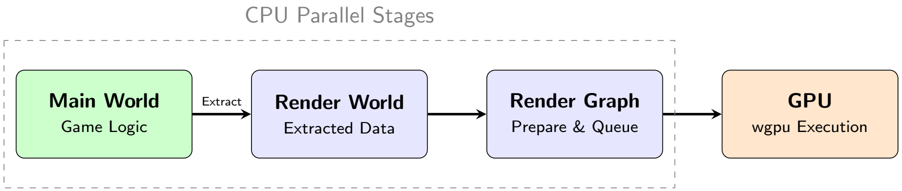
Entities in Bevy



Outline

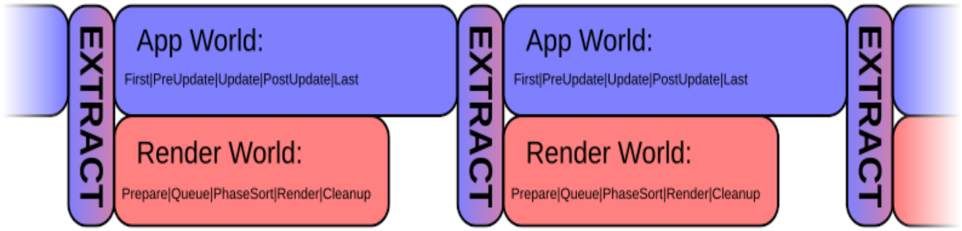
Rendering in Bevy

For more info check https://bevy-cheatbook.github.io/gpu/intro.html

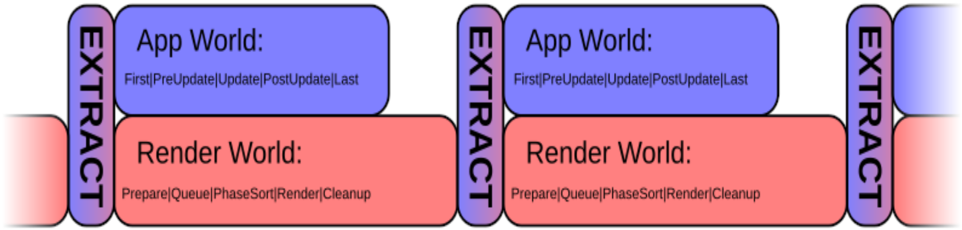


- **Pipelined**: Rendering logic runs in parallel with the next frame's game logic.
- **wgpu**: Modern backend supporting Vulkan, Metal, DX12, and WebGPU.
- **Render Graph**: Modular and customizable rendering passes.

App-Bound Scenario (app takes longer than render):


EXTRACT
App World:
First|PreUpdate|Update|PostUpdate|Last
Render World:
Prepare|Queue|PhaseSort|Render|Cleanup
EXTRACT
App World:
First|PreUpdate|Update|PostUpdate|Last
Render World:
Prepare|Queue|PhaseSort|Render|Cleanup
EXTRACT

Render-Bound Scenario (render takes longer than app):


EXTRACT
App World:
First|PreUpdate|Update|PostUpdate|Last
Render World:
Prepare|Queue|PhaseSort|Render|Cleanup
EXTRACT
App World:
First|PreUpdate|Update|PostUpdate|Last
Render World:
Prepare|Queue|PhaseSort|Render|Cleanup
EXTRACT

Outline

Resources vs Components

Components

Data attached to entities

- Attached to specific entities
- Multiple instances exist
- Defines object properties
- Examples: Position, Health, Sprite

Resources

Global unique data

- Accessible by all systems
- Only one instance exists
- Defines world state
- Examples: Time, Score, AssetServer

Outline

Creating a Plugin

```rust
use bevy::prelude::*;

#[derive(Resource, Default)]
struct Score(u32);

pub struct GamePlugin;

impl Plugin for GamePlugin {
    fn build(&self, app: &mut App) {
        app.init_resource::<Score>()
            .add_systems(Startup, setup)
            .add_systems(Update, update_score);
    }
}

fn setup() { println!("Game Started"); }
fn update_score(mut score: ResMut<Score>) { score.0 += 1; }
```

Bevy System Parameters

```rust
use bevy::prelude::*;
fn my_system(
    mut query: Query<&mut Transform>, // Access Components
    time: Res<Time>,                  // Access Resource
) {
    for mut transform in query.iter_mut() {
        transform.translation.x += time.delta_seconds() * 100.0;
    }
}
```

Outline

Creating Entities in Bevy

```rust
use bevy::prelude::*;
fn spawn_entity(mut commands: Commands) {
    commands.spawn((
        Transform::default(),
        Sprite::default(),
        Health(100),
    ));
}
```

Querying Entities in Bevy with some logic

```
fn query_enemies(query: Query<Entity, (With<Health>, Without<Cow>)>
    for enemy in query.iter() {
        println!("Found enemy entity: {:?}", enemy);
    }
}
```

Where am I?

Building a Simple Game with Bevy

Gragusi Survivor

Where am I?

Thank You!

Q&A

Queso y Ambutido