🎮 Game Engines

⚛️ Newtonian Physics in Godot

Dr Bryan Duggan

TU Dublin

📧 bryan.duggan@tudublin.org

🐦 @skooter500

🌐 http://bryanduggan.org

📋 What we will cover

- 📐 Coordinate Geometry
- 📊 Trigonometry
- ➡️ Vectors & Simple Movement
- 🔄 Transforms
- ⚡ Physics in Godot

🔬 Newtonian Physics - Fundamentals

⏱️ Time

Scalar measured in **seconds (s)**

📏 Distance

Scalar measured in **Meters (m)**

⚖️ Mass

- Measure of the amount of matter
- Scalar measured in **Kg**
- Unit of matter

📍 Position & Velocity

Position Vector

- Vector relative to origin
- Centre of mass
- Balance point

Velocity Vector

Formula: `v = Δx / Δt`

- Has magnitude (speed)
- Has direction
- Measured in **m/s**

⚡ Acceleration

📈 Rate of Change in Velocity

- Measured in **m/s²** or m/s/s
- **Formula:** `a = Δv / Δt`
- Change in velocity per time interval

```
Velocity increases over time
    ↑
  6 |         •
  4 |      •
  2 |   •
  0 |•_____________→
    0   1   2   3   Time (s)
```

🚗 Example

Car accelerating at 2 m/s²

🧮 Velocity, Acceleration & Time

Linear Equation: `y = mx + c`

For constant acceleration:

$v = at + u$

- **v** = final velocity
- **a** = acceleration
- **t** = time
- **u** = initial velocity (at t = 0)

💡 Example

Car starting at **3 m/s**, accelerating at **2 m/s²** for **3 seconds**:

`v = 2 × 3 + 3 = 9 m/s` ✅

📐 Distance Calculations

Area Under the Curve

Area A = $\Delta t \times u$

Area B = $.5 \times (v-u) \times \Delta t$

🎯 Final Formula

💪 Force

What is Force?

- Can alter speed or direction
- **NOT** motion itself
- Only required for **change**
- No force needed to keep moving

Multiple Forces

⚙️ Calculating Force & Acceleration

$F = ma$

- **F** = Force (Newtons)
- **m** = mass (kg)
- **a** = acceleration (m/s^2)

🚤 Example

Boat: **2000 kg**

Acceleration: **1.5 m/s^2**

```
F = 2000 × 1.5
```

= 3000 N 💪

📚 Equations Summary

🎯 Key Formulas

- $F = Ma$
- $a = \Delta v/\Delta t$
- $v = \Delta x/\Delta t$

📊 Movement

- $\Delta x = v\ \Delta t$
- $\Delta x = u\ \Delta t + \frac{1}{2} a\ \Delta t^2$

💻 Example 1: Moving to Destination

```
var dest = Vector3(20, 5, 20)
var to_dest = dest - global_position

if to_dest.length() > 0.1:
    to_dest = to_dest.normalized()
    var speed = 5.0
    global_position += \
        to_dest * speed * delta
    look_at(dest, Vector3.UP)
```

🎮 Better Godot Way

```
# Simple & clean!
look_at(dest, Vector3.UP)
translate(
    Vector3.FORWARD *
    speed * delta
)
```

⚡ Example 2: Physics Integration

```
var acceleration = force / mass
velocity += acceleration * delta
global_position += velocity * delta
force = Vector3.ZERO

if velocity.length() > 0.01:
    look_at(global_position + velocity, Vector3.UP)

velocity *= 0.99  # damping - friction effect
```

✨ **This implements Newton's laws directly in your game!**

🎯 Seek Behavior

Steering toward target

```
var desired_velocity = \
    target_pos - global_position

desired_velocity = \
    desired_velocity.normalized()

desired_velocity *= max_speed

var steering_force = \
    desired_velocity - velocity

return steering_force
```

3D Physics in Godot

Topics Covered

- Physics recap
- Equations of motion (movement & rotation)
- Physics Engines
- RigidBody3D
- CollisionShape3D
- Joints
- Raycasting
- Practical examples

Torque

The measure of force applied to produce rotational motion

- `torque = position × force`
- Torque = position (relative to centre of gravity) crossed with force
- Torque is a vector
- Magnitude = amount of torque
- Direction = axis of rotation

Angular Velocity & Acceleration

Angular Velocity

- Rate at which a body rotates
- Example: Earth rotates at 15 degrees per hour
- Given as a vector

Angular Acceleration

- Rate of change of angular velocity over time
- Measured in radians per second^2 (rad/s^2)
- Denoted by Greek letter α
- Also a vector

Inertia

The property of a body which resists change in its motion

Inertial Tensor

- A matrix describing how mass is distributed around a shape
- Different for different geometric primitives
- You can approximate complex shapes with simple ones
- Will anyone notice the approximation? Usually not!

Equations of Motion for Rotation

- Torque = `position × force`
- Angular acceleration = `torque * (1 / inertial_tensor)`
- Angular velocity = `angular_velocity + angular_acceleration * time`
- Orientation = `orientation + (time/2) * w * orientation`
    - Where w = pure quaternion of the angular velocity

Simple, isn't it! 😅

Two Ways of Adding Force

1. **Add force at centre of mass**
   - Doesn't generate any torque
   - Just updates force accumulator
   - Movement only
2. **Add force at a point on the object**
   - May generate torque
   - Updates force accumulator
   - AND updates torque accumulator
   - Movement and rotation

⚙️ Physics Engines

🎯 What They Simulate

🔷 **Rigid body dynamics**
🎯 **Collision detection**
🧣 **Soft body dynamics**
💧 **Fluid dynamics**
⚡ **Real-time** (speed over accuracy)

🏆 Major Players

- **Jolt Physics** ⭐ (Godot 4.x!)
- **Bullet** (Godot 3.x)
- **PhysX** (NVIDIA/Unreal)
- **Havok** (commercial)
- **Box2D** (2D physics)

🎯 RigidBody3D Properties

🔒 Constant

- ⚖️ Mass
- 📐 Inertial tensor

🔄 Changes Over Time

- 📍 Position vector
- ➡️ Linear velocity
- 🔄 Orientation (quaternion)
- 🌀 Angular velocity

🧮 Calculated Each Frame

- 💪 Force accumulator
- 🔄 Torque accumulator

💻 RigidBody3D in Godot

📊 Key Properties

```
mass
linear_velocity
angular_velocity
inertia
gravity_scale
```

⚡ Key Methods

```
apply_force(force, pos)
apply_central_force(force)
apply_torque(torque)
apply_impulse(impulse, pos)
apply_central_impulse(impulse)
```

🎭 RigidBody3D Types

⚡ Dynamic

```
↓ Falls
```

- Positive mass
- Affected by forces
- Full simulation

🧱 Static

```
═══════
Ground
```

- Zero mass

🔲 CollisionShape3D

Works with RigidBody3D

✅ Collision detection
✅ Trigger areas
✅ Raycasting
✅ Can differ from visual mesh

Lower resolution = better performance! 🚀

Common Shapes

🔴 **SphereShape3D**
📦 **BoxShape3D**
💊 **CapsuleShape3D**
🌭 **CylinderShape3D**

📡 Collision Signals

```
# Connect these signals for collision detection

func _on_body_entered(body):
    print("💥 Started touching: ", body.name)
    # Apply damage, play sound, etc.

func _on_body_exited(body):
    print("👋 Stopped touching: ", body.name)
    # Clean up, stop effects, etc.
```

🎯 Area3D for Triggers

```
func _on_area_entered(area):
    print("🚪 Entered trigger zone!")
    # Open door, spawn enemies, save checkpoint
```

💻 Creating RigidBodies in Code

```
func create_brick(pos: Vector3,
                  scale_vec: Vector3 = Vector3.ONE):
    # Create RigidBody as root
    var rigid_body = RigidBody3D.new()
    rigid_body.position = pos
    rigid_body.mass = 1.0
    add_child(rigid_body)

    # Add visual mesh as child
    var mesh_instance = MeshInstance3D.new()
    mesh_instance.mesh = BoxMesh.new()
    mesh_instance.scale = scale_vec
    rigid_body.add_child(mesh_instance)

    # Add collision shape as child
    var collision = CollisionShape3D.new()
    collision.shape = BoxShape3D.new()
    rigid_body.add_child(collision)

    return rigid_body
```

Creating Cylinders (Wheels)

```
func create_wheel(
    pos: Vector3,
    diameter: float,
    width: float
):
    # RigidBody as root
    var rigid_body = RigidBody3D.new()
    rigid_body.position = pos
    add_child(rigid_body)

    # Mesh as child
    var mesh_instance = MeshInstance3D.new()
    var cylinder_mesh = CylinderMesh.new()
    cylinder_mesh.height = width
    cylinder_mesh.top_radius = diameter / 2
    cylinder_mesh.bottom_radius = diameter / 2
    mesh_instance.mesh = cylinder_mesh
    mesh_instance.rotation_degrees = \
        Vector3(0, 0, 90)
    rigid_body.add_child(mesh_instance)

    return rigid_body
```

🔗 Joints in Godot

Nodes that constrain RigidBody3D movement relative to each other

📌 **PinJoint3D**

Point-to-point connection

🚪 **HingeJoint3D**

Rotation around axis

↔️ **SliderJoint3D**

Movement along axis

🎯 **Generic6DOF**

Configure everything!

All joints have `node_a`, `node_b`, and `breaking` threshold

HingeJoint3D - Doors & Wheels

```
var hinge = HingeJoint3D.new()
add_child(hinge)

hinge.node_a = wheel.get_path()
hinge.node_b = chassis.get_path()

# Optional: add motor
hinge.set_flag(
    HingeJoint3D.FLAG_ENABLE_MOTOR,
    true
)
hinge.set_param(
    HingeJoint3D.PARAM_MOTOR_TARGET_VELOCITY,
    10.0
)
```

🚗 Perfect For:

```
    Wheel          Door
```

↔️ SliderJoint3D - Linear Motion

```
var slider = SliderJoint3D.new()
add_child(slider)

slider.node_a = piston.get_path()
slider.node_b = cylinder.get_path()

# Set movement limits
slider.set_param(
    SliderJoint3D.PARAM_LINEAR_LIMIT_UPPER,
    5.0
)
slider.set_param(
    SliderJoint3D.PARAM_LINEAR_LIMIT_LOWER,
    -5.0
)
```

🔧 Piston Example

📌 PinJoint3D - Chains & Ropes

```
var pin = PinJoint3D.new()
add_child(pin)

pin.node_a = object_a.get_path()
pin.node_b = object_b.get_path()

# Add damping for stability
pin.set_param(
    PinJoint3D.PARAM_BIAS,
    0.3
)
pin.set_param(
    PinJoint3D.PARAM_DAMPING,
    1.0
)
```

⛓️ Use Cases

🎯 Generic6DOFJoint3D

Most Flexible Joint - Configure Everything!

Six Degrees of Freedom:

- 3 Linear (X, Y, Z movement)
- 3 Angular (X, Y, Z rotation)

Each axis can be:

- 🔒 Locked
- 🔓 Free
- ⚙️ Limited
- 🔄 Motorized
- 🌊 Spring-damped

Use when other joints don't fit your needs! 🛠️

📡 Raycasting

Laser-precise collision detection! 🎯

📡 Raycasting in Godot

What is it?

A **ray** from origin to target

Returns collision information:

- 📍 Hit position
- 🎯 Hit object
- 📐 Surface normal
- 📏 Distance

Very efficient! Hundreds per frame 🚀

📊 Visualization

```
Camera 📷
  |
  |  Ray
```

💻 Raycasting Code

```
var space_state = get_world_3d().direct_space_state
var query = PhysicsRayQueryParameters3D.create(origin, target)
var result = space_state.intersect_ray(query)

if result:
    print("🎯 Hit: ", result.collider.name)
    print("📍 Position: ", result.position)
    print("📐 Normal: ", result.normal)

    # Spawn effect at hit point
    spawn_explosion(result.position)
```

💡 **Use cases:** Line of sight, shooting, object placement, mouse picking!

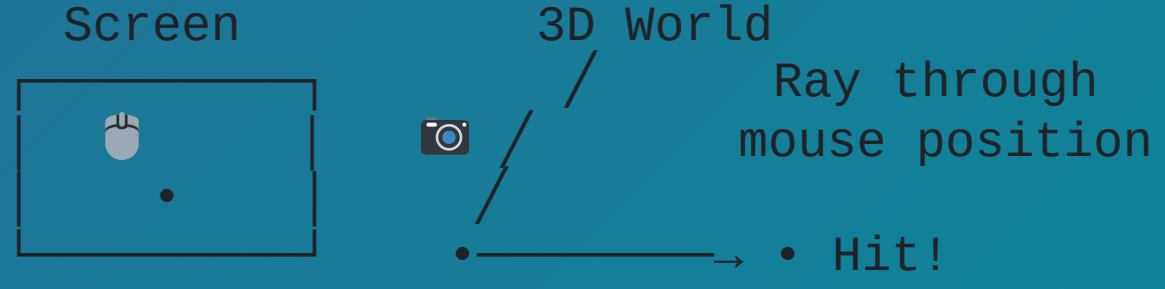
Raycasting from Camera

```
var camera = get_viewport().get_camera_3d()
var mouse_pos = get_viewport().get_mouse_position()

# Get ray from camera through mouse position
var from = camera.project_ray_origin(mouse_pos)
var to = from + camera.project_ray_normal(mouse_pos) * 1000

var space_state = get_world_3d().direct_space_state
var query = PhysicsRayQueryParameters3D.create(from, to)
var result = space_state.intersect_ray(query)

if result:
    spawn_object(result.position)  # Click to place!
```



🔫 Gravity Gun

Half-Life 2's iconic physics manipulator!

🎮 Gravity Gun Concept

💡 The Idea

Players directly manipulate physics objects

Use them as:

- 🎯 Projectiles
- 🛡️ Shields
- 🧩 Puzzle solutions
- ⚙️ Tools

🎬 Famous Examples

- **Half-Life 2** (original!)
- Portal

🔫 Gravity Gun - Algorithm

1 Check for Input (mouse click)

2 If Nothing Picked Up:

- Raycast from camera → If hit valid object → pick it up!

3 If Something Held:

- Calculate hold position (in front of camera)
- Calculate velocity toward hold position
- Apply velocity to object's RigidBody3D

4 On Release:

- Drop the object (set `picked_up = null`)

📝 Gravity Gun - Pseudocode

```
if mouse_clicked:
    if picked_up == null:
        ┌──────────────────────────────┐
        │ Raycast from camera          │
        │ if hit_object and not world: │
        │     picked_up = hit_object   │
        └──────────────────────────────┘
    else:
        ┌──────────────────────────────────────┐
        │ hold_pos = camera + forward * dist   │
        │ direction = hold_pos - picked_pos    │
        │ velocity = direction * power         │
        │ velocity = clamp(velocity, max_vel)  │
        │ picked_up.linear_velocity = velocity │
        └──────────────────────────────────────┘
else:
    picked_up = null  # Drop it!
```

💻 Gravity Gun - Setup

```
extends Node3D

var picked_up = null
var hold_distance = 5.0
var power_factor = 10.0
var max_velocity = 20.0

func _process(delta):
    if Input.is_action_pressed("shoot"):
        if picked_up == null:
            raycast_and_pickup()
        else:
            hold_object()
    else:
        picked_up = null  # Release
```

🎯 Raycast & Pickup

```
func raycast_and_pickup():
    var camera = get_viewport().get_camera_3d()
    var from = camera.global_position
    var to = from + camera.global_transform.basis.z * -100

    var space_state = get_world_3d().direct_space_state
    var query = PhysicsRayQueryParameters3D.create(from, to)
    var result = space_state.intersect_ray(query)

    if result:
        var hit_object = result.collider
        # Only pick up RigidBody3D objects
        if hit_object is RigidBody3D and hit_object.name != "Ground":
            picked_up = hit_object
            print("📦 Picked up: ", hit_object.name)
```

Hold Object

```
func hold_object():
    var camera = get_viewport().get_camera_3d()

    # Calculate position in front of camera
    var hold_pos = camera.global_position + \
                   camera.global_transform.basis.z * -hold_distance

    # Calculate velocity to move object to hold position
    var to_hold_pos = hold_pos - picked_up.global_position
    to_hold_pos *= power_factor

    # Clamp to maximum velocity
    if to_hold_pos.length() > max_velocity:
        to_hold_pos = to_hold_pos.normalized() * max_velocity

    # Apply the velocity
    picked_up.linear_velocity = to_hold_pos
```

🔧 Physics Integration in Code

```
# Newtonian Physics (Linear Motion)
var acceleration = force / mass
velocity += acceleration * delta
position += velocity * delta
velocity *= 0.99  # damping/friction

# Hamiltonian Physics (Rotational Motion)
var angular_acceleration = torque / inertia
angular_velocity += angular_acceleration * delta
# Godot handles quaternion rotation automatically!

# Reset force accumulators
force = Vector3.ZERO
torque = Vector3.ZERO
```

⚡ **Godot's physics engine does this for you automatically!**

🎓 Advanced Integration

⚡ Explicit Euler (Current)

What we've been using:

```
Simple & fast
One step per frame
```

Good for most games! ✅

🎯 Runge-Kutta (RK4)

More accurate:

```
4 samples per timestep
t = 0, 0.25, 0.5, 0.75
Final value at t = 1
```

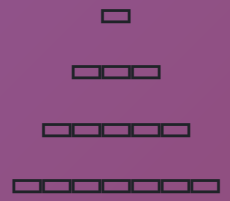
Better stability 🎯

🎓 Lab Exercises

Time to practice! 💪

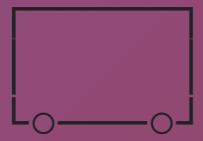
Lab Exercises

1 Tower Builder



- Stack blocks
- Destroy mechanic
- Make it fun!

2 Simple Car



📚 Recap - What We Learned

🎯 Core Concepts

- ✅ Newtonian physics
- ✅ Forces & acceleration
- ✅ Velocity & position
- ✅ Torque & rotation
- ✅ Angular motion

🔧 Godot Tools

- ✅ RigidBody3D
- ✅ CollisionShape3D
- ✅ Joints (all types!)
- ✅ Raycasting
- ✅ Physics integration

🎯 Quick Reference

📊 Key Formulas

```
F = ma
v = v₀ + at
x = x₀ + v₀t + ½at²
τ = r × F
```

🔧 Joint Types

- 📌 PinJoint3D → chains
- 🚪 HingeJoint3D → doors/wheels
- ↔️ SliderJoint3D → pistons
- 🎯 Generic6DOF → custom

🌐 Resources & Links

📖 Official Docs

🔗 Godot Physics

docs.godotengine.org/physics

🔗 RigidBody3D

docs.godotengine.org/rigidbody3d

🔗 Joints

docs.godotengine.org/joints

🎬 Learning Resources

🎓 GDQuest

gdquest.com

🎓 Godot Tutorials

godottutorials.com

🎉 Questions?

Thanks for your attention!

Now go build something awesome! 🚀