QRR Marine Simulation - NVIDIA Omniverse Demo

Quantum Relational Relativity mathematics for computational fluid dynamics

Relational Relativity LLC - Robin B. Macomber

Overview

This is a proof-of-concept demonstration showing Quantum Relational Relativity (QRR) mathematics applied to marine wave dynamics in NVIDIA Omniverse. The simulation demonstrates $O(N)$ computational complexity for fluid dynamics compared to traditional $O(N^2)$ particle-based CFD approaches.

What This Demo Shows

- **Linear wave propagation** using coherence field mathematics
- **Real-time buoyancy simulation** with relational dynamics
- **Efficient computation** on consumer-grade hardware (tested on GTX 1050 Ti)
- **AI pilot sensor framework** providing wave gradient and coherence data
- **Live parameter tuning** - modify physics and instantly see results

Key Innovation

Traditional CFD simulates millions of individual particles with $O(N^2)$ interaction complexity. QRR treats relationships as mathematical primitives with $O(N)$ relational transformations, resulting in significant performance improvements while maintaining physical accuracy.

Technical Details

QRR Mathematics Core

The simulation implements several QRR mathematical components:

- **CoherenceField**: Wave evolution using relational wave equations
- **RelationalEntity**: Entities defined by their relationships rather than intrinsic properties
- **QRRSystem**: Integrated system managing coherence, energy, and relational bandwidth
- **CoherenceDensityCalculator**: Coherence decay and density computations

Wave Dynamics

The demo uses linear wave patterns (ocean swells) that propagate independently of object positions:

```python
# Linear wave pattern - waves move along X axis
wave_phase = i * 0.5 - self.wave_speed * self.time
y = self.wave_amplitude * np.sin(wave_phase) * coherence_field[i, j]
```

Performance Characteristics

- **Grid**: 32x32 water surface (1,024 vertices)
- **Update rate**: ~15 FPS on GTX 1050 Ti (4GB VRAM)
- **Complexity**: O(N) for wave evolution
- **Memory**: ~1.3 GB process, ~440 MB GPU

Requirements

- **NVIDIA Omniverse Kit** (108.1.0 or later)
- **NVIDIA GPU** (GTX 1050 Ti or better)
- **Python 3.10+** (included with Omniverse)
- **NumPy** (included with Omniverse)

Quick Start

Installation

1. Install NVIDIA Omniverse
2. Clone this repository:

```bash
git clone https://github.com/Relational-Relativity-Corporation/qrr_omniverse_public.git
cd qrr_omniverse_public
```

Running the Simulation

1. Launch Omniverse Kit from your installation directory
2. Open the Script Editor (Window → Script Editor)
3. Load and run the simulation:

```python
exec(open('/path/to/marine_sim_linear_waves.py').read())
```

The simulation will auto-start after 2 seconds. You should see:

- Blue water surface with linear wave patterns
- Orange cube (boat) bobbing on the waves
- Console output showing FPS, coherence values, and AI pilot data

Interactive Controls

Once running, you can control the simulation from the Script Editor:

```python
# Pause animation
sim.pause()

# Resume animation
sim.play()

# Reset to beginning
sim.reset()

# Print current status
sim.print_status()

# Manual stepping (when paused)
for i in range(10):
    sim.step()
```

Customization

Tuning Wave Parameters

Edit these values in the `__init__` method of `QRRMarineSimulation`:

```python

```

```
self.grid_size = 32          # Water grid resolution (32x32)
self.grid_spacing = 2.0      # Distance between grid points
self.wave_speed = 1.5        # Wave propagation speed
self.wave_amplitude = 1.2    # Wave height (try 0.3 to 2.0)
self.dt = 0.016              # Time step (affects simulation speed)
```

Live Parameter Updates

You can modify parameters and reload without closing Omniverse:

1. Edit `wave_amplitude` or other parameters in the file
2. Save the file
3. Re-run the exec() command in Script Editor
4. New simulation starts with updated values

Example: Change `self.wave_amplitude = 1.2` to `self.wave_amplitude = 0.6` for calmer seas.

Code Structure

Main Components

QRR Mathematics (lines 20-180)

- `QRRConstants`: Physical constants and thresholds
- `MathPrimitives`: Core mathematical operations
- `CoherenceField`: Wave field evolution with gradient/laplacian computations
- `RelationalEntity`: Objects defined by relationships
- `QRRSystem`: Integrated coherence and energy management

Marine Simulation (lines 182-620)

- `QRRMarineSimulation`: Main simulation class
  - Scene creation (water, boat, camera, lighting)
  - Wave dynamics using coherence fields
  - Boat physics responding to water surface
  - AI pilot sensor data generation

Key Methods:

- `update_wave_dynamics()`: Evolves coherence field and updates water mesh
- `update_boat_physics()`: Computes buoyancy from wave field
- `ai_pilot_decision()`: Generates sensor data for autonomous navigation

Mathematical Foundation

Coherence Field Evolution

The water surface evolves using a discrete Laplacian operator on the coherence field:

```python
laplacian = (
    field[i-1,j] + field[i+1,j] +
    field[i,j-1] + field[i,j+1] -
    4 * field[i,j]
)
velocity_field += dt * wave_speed² * laplacian
field_state += dt * velocity_field
```

This represents the wave equation in relational space, where coherence propagates through relationships rather than through a physical medium.

Buoyancy Computation

The boat samples the local coherence field to determine water height:

```python
water_height = wave_amplitude * sin(wave_phase) * coherence[grid_x, grid_z]
target_height = water_height + boat_draft
boat_position.y += (target_height - boat_position.y) * damping_factor
```

AI Pilot Sensor Data

The simulation generates sensor data suitable for training autonomous navigation systems:

```python
```

```
sensor_data = {
    "wave_gradient": [grad_x, grad_z],      # Direction of steepest wave slope
    "boat_orientation": [x, y, z],          # Current position
    "velocity": [vx, vy, vz],               # Motion vector
    "coherence_local": float                # Local field coherence (0-1)
}
```

This data could be used to train AI systems for:

- Collision avoidance in rough seas
- Optimal path planning through wave fields
- Auto-docking in dynamic conditions
- Energy-efficient navigation

Applications

This demonstration of QRR mathematics has potential applications in:

- **Autonomous marine vehicles** - Real-time navigation and control
- **Naval architecture** - Wave interaction modeling and hull optimization
- **Oceanographic simulation** - Large-scale fluid dynamics with reduced computational cost
- **Game development** - Realistic water physics with minimal performance impact
- **VFX and animation** - Efficient fluid simulation for visual effects

---

Performance Notes

Tested Configuration

- **GPU**: NVIDIA GeForce GTX 1050 Ti (4GB VRAM)
- **Grid**: 32x32 vertices (1,024 total)
- **FPS**: ~14-15 fps
- **Memory**: 1.3 GB RAM, 441 MB VRAM

Scalability

The O(N) complexity means performance scales linearly with grid size:

Traditional $O(N^2)$ CFD would see quadratic degradation.

Limitations

This is a **proof-of-concept demonstration**, not production-ready code:

- Simplified wave physics (no wave breaking, spray, foam)
- Basic buoyancy model (no hull shape consideration)
- Single boat entity (no multi-object interactions)
- Fixed environmental conditions (no wind, currents)
- Geometric primitive boat (cube) rather than realistic mesh

The focus is on demonstrating QRR mathematical principles and computational efficiency.

Technical Background

What is QRR?

Quantum Relational Relativity is a mathematical framework that treats relationships as fundamental rather than emergent properties. Instead of modeling systems as collections of objects with properties, QRR models systems as networks of relationships.

Traditional CFD vs QRR

Traditional CFD:

```
Objects → Properties → Interactions → Emergent Relationships
Complexity: O(N²) for particle interactions
```

QRR Approach:

```
Relationships → Coherence Fields → Object Behavior
Complexity: O(N) for field evolution
```

This inversion of the traditional approach yields significant computational advantages while maintaining physical fidelity.

License

Contact

Relational Relativity LLC

- Website: relationalrelativity.dev
- GitHub: @Relational-Relativity-Corporation

For technical discussions, demonstrations, or partnership inquiries, please visit our website or open an issue on this repository.

Demonstrating quantum relational mathematics for real-world computational efficiency