Force Configuration Refactor - Progress Report

Overview

We're replacing the mutable force handle approach with an immutable configuration system to solve the force parameter reset problem.

Problem Statement

When transitioning between scenes in CodeExplorerV2, force parameters defined in `Forces.purs` are not being correctly reset. For example:

- `chargeTree` should have strength `-290` but shows `-100`
- `collide2` should have radius `+19` but shows `10`
- `links` should have distance `40` but shows `50`

Root cause: d3-force uses mutable JavaScript objects (force handles) that persist across scene transitions. The ForceControlPanel and scene transitions share references to these mutable objects, leading to unpredictable state.

Solution

Separate force **configuration** (immutable PureScript data) from force **execution** (ephemeral d3 handles).

Key Insight

> d3 force handles should be ephemeral runtime artifacts created from immutable configurations, not persistent objects stored in PureScript records.

Implementation Progress

✅ Phase 1: Core Types & Infrastructure (COMPLETED)

1. Core Configuration Module

File: `src/lib/PSD3/Config/Force.purs`

- ☑ `ForceConfig` type - pure configuration data (no JavaScript references)
- ☑ `ForceParams` sum type - type-specific parameters for each force type
- ☑ `AttrValue` type - static values or functions
- ☑ `ForceFilter` type - filter forces to specific nodes
- ☑ Smart constructors: `manyBodyForce`, `centerForce`, `collideForce`, etc.
- ☑ Parameter update helpers: `withStrength`, `withRadius`, etc.
- ☑ Show instances for debugging

2. Scene Configuration Module

File: `src/lib/PSD3/Config/Scene.purs`

- ☑ `SceneConfig` type - complete scene specification
- ☑ `SimulationParams` type - alpha, decay rates, etc.
- ☑ Default parameter sets: `defaultSimParams`, `fastSimParams`, `slowSimParams`
- ☑ Scene builders: `scene`, `sceneWithParams`
- ☑ Scene composition: `addForce`, `removeForce`, `replaceForce`, `mergeScenes`
- ☑ Query functions: `getForce`, `hasForce`, `forceNames`

3. Application Logic Module

File: `src/lib/PSD3/Config/Apply.purs`

- ☑ `applySceneConfig` - main entry point for scene transitions
- ☑ `createForceHandle` - create fresh d3 handle from config
- ☑ `applyForceParams` - apply parameters to handle (with filtering)
- ☑ `removeAllForces` - clear simulation
- ☑ Helper functions: `addForceToSimulation`, `updateForceParams`

4. CodeExplorerV2 Force Configurations

File: `src/website/Component/CodeExplorerV2/ForceConfigs.purs`

- ☑ All force configs matching existing `Forces.purs`:
  - `charge`, `charge2`, `chargeTree`, `chargePack`
  - `collision`, `collide2`, `collidePack`
  - `center`, `centerStrong`
  - `links`
  - `packageOrbit`, `moduleOrbit`
  - `clusterX`, `clusterY`
- ☑ Force filters: `packagesOnly`, `modulesOnly`, `treeParentsOnly`
- ☑ Helper functions: `gridPointX`, `collideRadius`, etc.

5. Example Scene Configuration

File: `src/website/Component/CodeExplorerV2/Scenes/ForceGraphV2.purs`

- ☑ `sceneConfig` - ForceGraph scene using new system
- ☑ Documents expected parameter values for verification

6. POC Component

File: `src/website/Component/ForceConfigPOC.purs`

- ☑ Simple test component with 2 scenes
- ☑ Scene transition buttons
- ☑ Parameter display
- ☐ **TODO**: Wire up to actual simulation (currently just logs)

🔄 Phase 2: Integration & Testing (IN PROGRESS)

Next Steps

1. **Test the new system**:
    - [ ] Create a test page that uses `ForceGraphV2.sceneConfig`
    - [ ] Apply scene config to simulation via `applySceneConfig`
    - [ ] Verify console logs show correct parameter values
    - [ ] Verify force parameters reset correctly on scene transition
    - [ ] Compare behavior with old system
2. **Integrate into Orchestration**:
    - [ ] Add function to convert `PSD3.Config.Scene.SceneConfig` to simulation state
    - [ ] Update `goToForceGraph` to optionally use new system
    - [ ] Test alongside existing implementation
3. **Update ForceControlPanel**:
    - [ ] Modify to work with `ForceConfig` instead of d3 handles
    - [ ] Store current scene config in state
    - [ ] When parameter changes, update config and re-apply
    - [ ] Add "Reset to Scene Defaults" button

📋 Phase 3: Full Migration (NOT STARTED)

1. **Convert all scenes**:
    - [ ] `Orbit.purs` → `OrbitV2.purs`
    - [ ] `Tree.purs` → `TreeV2.purs`
    - [ ] `TreeReveal.purs` → `TreeRevealV2.purs`
    - [ ] `BubblePack.purs` → `BubblePackV2.purs`
2. **Update Orchestration**:
    - [ ] Replace `setForces` calls with `applySceneConfig`
    - [ ] Remove dependency on `Forces.purs`
    - [ ] Test all scene transitions
3. **Deprecate old system**:
    - [ ] Mark `Force` type as deprecated
    - [ ] Mark `Forces.purs` as deprecated
    - [ ] Update documentation

🔮 Future Enhancements

1. **Configuration diffing**:
    - Only recreate handles for changed forces
    - Optimize scene transitions
2. **Preset system**:

- Save/load scene configurations
- User-defined force presets
- Export/import configurations as JSON

3. **Validation**:
   - Compile-time checks for parameter types
   - Runtime validation of parameter ranges
   - Warning for ineffective force combinations

4. **Separate package**:
   - Extract to `purescript-d3-force-config`
   - Independent of visualization code
   - Usable with other PureScript projects

Files Created

Library Code

1. `src/lib/PSD3/Config/Force.purs` (239 lines)
2. `src/lib/PSD3/Config/Scene.purs` (142 lines)
3. `src/lib/PSD3/Config/Apply.purs` (176 lines)

Application Code

4. `src/website/Component/CodeExplorerV2/ForceConfigs.purs` (211 lines)
5. `src/website/Component/CodeExplorerV2/Scenes/ForceGraphV2.purs` (49 lines)
6. `src/website/Component/ForceConfigPOC.purs` (143 lines)
7. `src/website/Component/ForceConfigPOC.js` (12 lines)

Total: ~972 lines of new code

Key Design Decisions

1. Immutability First

Force configurations are pure data. Creating a "new" configuration is just creating a new record. No hidden mutation.

2. Type Safety

The `ForceParams` sum type ensures you can only set parameters that exist for a force type. Type errors at compile time, not runtime.

3. Explicit Filters

Filters are first-class, named, and composable. No magic string matching.

4. Separate Concerns

- Configuration (PureScript): What forces exist, what their parameters are
- Execution (JavaScript): The physics calculations
- Application (PureScript): When to apply which configurations

5. Backward Compatibility

New system coexists with old. Can migrate incrementally. Can compare behavior side-by-side.

Comparison: Old vs New

Aspect	Old (Mutable Handles)	New (Immutable Config)
Force definition	`createForce` with d3 handle	Pure `ForceConfig` record
Parameter updates	Mutate handle in place	Create new config
Scene transitions	`refreshForce`, hope it works	`applySceneConfig`, guaranteed fresh
State tracking	Force records with mutable `force_`	Pure configs, ephemeral handles
Debugging	Inspect opaque JS objects	Inspect PureScript data
Testing	Requires simulation instance	Test configs in isolation
Type safety	Runtime string-based params	Compile-time param validation
Multiple forces	Name conflicts possible	Arbitrary names, multiple of same type

Benefits

1. **Predictability**: Force configs are immutable, scene transitions are deterministic
2. **Debuggability**: Can inspect configs at any time, no hidden mutation
3. **Composability**: Configs are data, can merge, filter, transform them
4. **Type Safety**: Invalid parameters caught at compile time
5. **Testability**: Can test configuration logic without simulation
6. **Flexibility**: Can have multiple forces of the same type with different names

What Stays in JavaScript FFI

Performance-critical parts remain in JavaScript:

- Verlet integration loop
- Force calculations (Barnes-Hut, collision detection)
- Position/velocity updates
- Quadtree operations

Next Session Plan

1. Create a test harness that applies `ForceGraphV2.sceneConfig`
2. Verify parameters are set correctly (console logs)
3. Test scene transitions (Tree → ForceGraph → Tree)
4. Verify parameters reset correctly

5. If successful, integrate into main CodeExplorerV2
6. Update ForceControlPanel to use new system
7. Migrate remaining scenes

Success Criteria

✅ The new system will be successful if:

- Switching to ForceGraph scene sets: chargeTree=-290, collide2=+19, links=40
- Switching away and back resets parameters correctly
- Console logs confirm correct values being applied
- No performance regression
- Code is more maintainable and debuggable

Status: Phase 1 complete, ready for Phase 2 testing **Next**: Create test harness and verify parameter reset works correctly