Arbor Roadmap

Arbor is a renderer-agnostic widget library for Lean 4 that emits abstract render commands instead of directly rendering to a specific backend. This document outlines proposed improvements, new features, and cleanup tasks.

Feature Proposals

[Priority: High] Animation System

Description: Add support for animated values and transitions between widget states.

Rationale: Modern UI frameworks require animation support for polished user experiences. Arbor's render-command architecture is well-suited for this since animations would simply produce interpolated render commands over time.

Proposed API:

```
structure AnimatedValue (  : Type) where
  current :
  target :
  duration : Float
  elapsed : Float
  easing : EasingFunction

inductive RenderCommand where
  | ... -- existing commands
  | animate (id : WidgetId) (property : AnimatableProperty) (value : AnimatedValue Float)
```

Affected Files: - `Arbor/Core/Animation.lean` (new file) - `Arbor/Widget/Core.lean` - add animated widget variants - `Arbor/Render/Command.lean` - animation-related commands

Estimated Effort: Large

Dependencies: None

[Priority: High] Focus Management System

Description: Implement a focus system for keyboard navigation between widgets.

Rationale: Currently there is no focus tracking. Keyboard events go to no specific widget by default. For accessibility and usability, widgets need focus state management with tab navigation.

Proposed API:

```
structure FocusState where
  focused : Option WidgetId
  tabOrder : Array WidgetId
  focusVisible : Bool -- for keyboard-triggered focus styling

def Widget.focusable : Widget -> Bool
def FocusManager.focusNext : FocusState -> Widget -> FocusState
def FocusManager.focusPrev : FocusState -> Widget -> FocusState
```

Affected Files: - `Arbor/App/Focus.lean` (new file) - `Arbor/Event/Types.lean` - add focus-related events - `Arbor/App/UI.lean` - integrate focus into event dispatch

Estimated Effort: Medium

Dependencies: None

[Priority: High] Image Render Command

Description: Add render commands for displaying images and textures.

Rationale: The current render command set lacks image drawing capabilities. While backends would implement the actual image loading, Arbor should define the abstract command.

Proposed API:

```
structure ImageId where
  id : Nat
  path : Option String  -- for debug/identification

inductive RenderCommand where
  | ... -- existing commands
  | drawImage (imageId : ImageId) (rect : Rect) (tint : Option Color := none)
  | drawImageSliced (imageId : ImageId) (rect : Rect) (slices : NineSlice)
```

Affected Files: - `Arbor/Core/Types.lean` - add `ImageId` type - `Arbor/Render/Command.lean` - add image commands - `Arbor/Widget/Core.lean` - add image widget variant

Estimated Effort: Small

Dependencies: None

[Priority: Medium] Shadow and Blur Effects

Description: Add render commands for drop shadows and blur effects.

Rationale: Shadows are essential for modern UI design to convey depth and hierarchy. The render command abstraction makes this straightforward to add.

Proposed API:

```
structure Shadow where
  offsetX : Float
  offsetY : Float
  blur : Float
  spread : Float
  color : Color
  inset : Bool := false

inductive RenderCommand where
  | ... -- existing commands
  | withShadow (shadow : Shadow) (cmds : Array RenderCommand)
  | blur (rect : Rect) (radius : Float)
```

Affected Files: - `Arbor/Core/Types.lean` - add `Shadow` type - `Arbor/Render/Command.lean` - add shadow/blur commands - `Arbor/Widget/Core.lean` - add `BoxStyle.shadow` field

Estimated Effort: Small

Dependencies: None

[Priority: Medium] Gradient Support

Description: Add render commands for linear and radial gradients.

Rationale: Tincture already provides gradient types. Arbor should support rendering gradients for backgrounds and fills.

Proposed API:

```
inductive GradientType where
  | linear (angle : Float)
  | radial (centerX centerY : Float)

structure Gradient where
  type : GradientType
  stops : Array (Float × Color)

inductive RenderCommand where
  | ... -- existing commands
  | fillGradient (rect : Rect) (gradient : Gradient)
```

Affected Files: - `Arbor/Render/Command.lean` - add gradient commands - `Arbor/Widget/Core.lean` - add `BoxStyle.backgroundGradient`

Estimated Effort: Small

Dependencies: Tincture gradient types

[Priority: Medium] Accessibility Annotations

Description: Add accessibility metadata to widgets for screen readers and assistive technologies.

Rationale: Widget semantics (role, label, state) should be captured for accessibility purposes, even if backends implement the actual accessibility APIs.

Proposed API:

```
inductive AccessibilityRole where
  | button | link | heading | list | listItem | textInput | ...

structure AccessibilityProps where
  role : Option AccessibilityRole
  label : Option String
  description : Option String
  hidden : Bool := false
  live : Option LiveRegion := none
```

Affected Files: - `Arbor/Core/Accessibility.lean` (new file) - `Arbor/Widget/Core.lean` - add accessibility props to widgets

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Theming System

Description: Add a theming layer for consistent styling across widgets.

Rationale: Currently styles are set per-widget. A theming system would provide default styles, color tokens, and typography presets that widgets inherit.

Proposed API:

```
structure Theme where
  colors : ThemeColors
  typography : ThemeTypography
  spacing : ThemeSpacing
  borders : ThemeBorders

structure ThemeColors where
  primary : Color
  secondary : Color
  background : Color
  surface : Color
  error : Color
  onPrimary : Color
  ...
```

Affected Files: - `Arbor/Theme.lean` (new file) - `Arbor/Widget/DSL.lean` - themed widget builders

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Path Render Command

Description: Add support for arbitrary vector paths (bezier curves, arcs).

Rationale: The current command set only supports rectangles and convex polygons. Vector paths would enable more complex shapes.

Proposed API:

```
inductive PathSegment where
  | moveTo (x y : Float)
  | lineTo (x y : Float)
  | quadTo (cx cy x y : Float)
  | cubicTo (c1x c1y c2x c2y x y : Float)
  | arcTo (rx ry rotation largeArc sweep x y : Float)
  | close

inductive RenderCommand where
  | fillPath (segments : Array PathSegment) (color : Color)
  | strokePath (segments : Array PathSegment) (color : Color) (lineWidth : Float)
```

Affected Files: - `Arbor/Core/Path.lean` (new file) - `Arbor/Render/Command.lean` - add path commands

Estimated Effort: Medium

Dependencies: None

[Priority: Low] Z-Index Support

Description: Allow widgets to specify explicit z-ordering independent of tree order.

Rationale: Currently z-order is determined by tree traversal order. Explicit z-index would allow popups, tooltips, and overlays to render on top.

Affected Files: - `Arbor/Widget/Core.lean` - add `zIndex` field - `Arbor/Render/Collect.lean` - sort commands by z-index

Estimated Effort: Small

Dependencies: None

[Priority: Low] Lazy Widget Loading

Description: Support for lazy/virtualized lists that only instantiate visible items.

Rationale: Large lists would benefit from virtualization to avoid creating thousands of widget nodes.

Affected Files: - `Arbor/Widget/Virtualized.lean` (new file) - `Arbor/Widget/Measure.lean` - virtual measurement

Estimated Effort: Large

Dependencies: Scroll container improvements

[Priority: Low] Drag and Drop Support

Description: Add drag-and-drop event handling and visual feedback.

Rationale: Common UI pattern that would extend the existing event system.

Affected Files: - `Arbor/Event/DragDrop.lean` (new file) - `Arbor/Event/Types.lean` - drag events

Estimated Effort: Medium

Dependencies: Pointer capture (already implemented)

Code Improvements

[Priority: High] Eliminate Partial Functions

Current State: Multiple functions are marked `partial` due to recursive widget tree traversal: - `Widget.widgetCount` (Core.lean:206) - `Widget.allIds` (Core.lean:210) - `measureWidget` (Measure.lean:40) - `intrinsicSize` (Measure.lean:179) - `collectWidget` (Collect.lean:90) - `collectDebugBorders` (Collect.lean:161) - `hitTest` (HitTest.lean:41) - `hitTestAll` (HitTest.lean:107) - `pathToWidget` (HitTest.lean:162) - `collectWidgetInfo` (Renderer.lean:402)

Proposed Change: Use well-founded recursion with fuel parameter or widget depth bound to prove termination.

Benefits: Total functions are safer and enable more Lean optimizations.

Affected Files: All files listed above

Estimated Effort: Medium

[Priority: High] Type-Safe Widget ID Generation

Current State: Widget IDs are just `Nat` values generated by a `StateM` counter. There is no type-level guarantee that IDs are unique or that referenced IDs exist.

Proposed Change: Consider using a phantom-typed ID system or indexed family to provide stronger guarantees:

```
structure WidgetId (tree : WidgetTree) where
  id : Nat
  valid : id < tree.size
```

Benefits: Eliminates a class of runtime errors where invalid widget IDs are referenced.

Affected Files: - `Arbor/Widget/Core.lean` - `Arbor/Widget/DSL.lean` - `Arbor/App/UI.lean`

Estimated Effort: Large

[Priority: Medium] Consolidate Point/Rect Type Conversions

Current State: There are repeated conversions between `Arbor.Rect` and `Trellis.LayoutRect`:

```
-- In Collect.lean
let r : Rect :=  rect.x, rect.y ,  rect.width, rect.height
-- In HitTest.lean
layout.borderRect.contains adjX adjY
```

Proposed Change: Add explicit coercion instances or helper functions to reduce boilerplate.

Benefits: Cleaner code, reduced duplication.

Affected Files: - `Arbor/Core/Types.lean` - add `Coe` instances - All files that convert between types

Estimated Effort: Small

[Priority: Medium] Extract Common Widget Traversal Pattern

Current State: Multiple functions implement the same tree traversal pattern with slight variations (collect, hit test, measure, etc.).

Proposed Change: Create a generic `foldWidget` or `traverseWidget` combinator:

```
def Widget.fold {M : Type -> Type} [Monad M] {  : Type}
    (f :   -> Widget -> M  ) (init :  ) (w : Widget) : M
```

Benefits: Reduces code duplication, makes traversal patterns consistent.

Affected Files: - `Arbor/Widget/Core.lean` - add fold/traverse - Refactor dependent files

Estimated Effort: Medium

[Priority: Medium] Improve TextMeasurer Typeclass Design

Current State: The `TextMeasurer` typeclass uses a monad parameter `M`:

```
class TextMeasurer (M : Type -> Type) where
  measureText : String -> FontId -> M TextMetrics
```

This requires carrying the monad type through many function signatures.

Proposed Change: Consider a simpler interface or effect system integration:

```
structure TextMeasurerHandle where
  measureText : String -> FontId -> IO TextMetrics
```

Benefits: Simpler API, easier backend integration.

Affected Files: - `Arbor/Core/TextMeasurer.lean` - `Arbor/Widget/Measure.lean` - `Arbor/Widget/TextLayout.lean`

Estimated Effort: Medium

[Priority: Medium] Add Render Command Batching

Current State: Render commands are collected into a flat array. Backends may benefit from batched commands (e.g., batch all fills of the same color).

Proposed Change: Add optional command batching/optimization pass:

```
def optimizeCommands (cmds : RenderCommands) : RenderCommands :=
  cmds
  |> batchSimilarFills
  |> eliminateNoOps
  |> mergeClipRegions
```

Benefits: Potential rendering performance improvements.

Affected Files: - `Arbor/Render/Optimize.lean` (new file)

Estimated Effort: Medium

[Priority: Low] Use Batteries/Mathlib Collections

Current State: Uses standard `Array` and manual iteration patterns.

Proposed Change: Could leverage `batteries` for enhanced collection operations if needed.

Benefits: More idiomatic Lean, tested implementations.

Affected Files: Various

Estimated Effort: Small

Code Cleanup

[Priority: High] Add Documentation for Public API

Issue: While there are doc comments on major types, many public functions lack documentation.

Location: - `Arbor/Widget/DSL.lean` - DSL functions have minimal docs - `Arbor/App/UI.lean` - `EventResult`, `Handler` types need examples - `Arbor/Event/Types.lean` - Event variants could use more context

Action Required: 1. Add module-level documentation to each file 2. Document all public functions with usage examples 3. Add cross-references between related concepts

Estimated Effort: Medium

[Priority: Medium] Standardize Error Handling

Issue: Functions use a mix of `Option`, silent failures, and panicking patterns.

Location: - `HitTest.lean` - returns `Option` (good) - `Collect.lean:91` - `let some computed := layouts.get w.id | return` (early return) - `Renderer.lean:142` - `points[0]!` uses `!` indexing

Action Required: 1. Replace `!` indexing with safe alternatives where possible 2. Document expected failure modes 3. Consider a unified error type for widget operations

Estimated Effort: Small

[Priority: Medium] Extract Text Rendering Utilities

Issue: `Arbor/Text/Renderer.lean` is 670 lines with multiple responsibilities: rendering, debugging, hierarchy display, and widget info collection.

Location: `Arbor/Text/Renderer.lean`

Action Required: 1. Extract `WidgetInfo` and collection logic to `Arbor/Debug/WidgetInfo.lean` 2. Keep core rendering in `Renderer.lean` 3. Move hierarchy/structure modes to `Arbor/Debug/Modes.lean`

Estimated Effort: Medium

[Priority: Medium] Consistent Naming Conventions

Issue: Some inconsistencies in naming: - `text'` uses prime suffix (Lean convention for variants) - `namedText` uses prefix (clearer intent) - `hitTestId` vs `hitTestPath` vs `hitTest` (good, but could add `hitTestWidget`)

Location: `Arbor/Widget/DSL.lean`, `Arbor/Event/HitTest.lean`

Action Required: 1. Decide on consistent naming pattern 2. Add aliases for discoverability if needed

Estimated Effort: Small

[Priority: Low] Remove Unused Imports

Issue: Some files may have unused imports accumulated during development.

Location: All `.lean` files

Action Required: Audit imports and remove unused ones.

Estimated Effort: Small

[Priority: Low] Add Type Annotations to Complex Expressions

Issue: Some complex expressions rely on type inference, making code harder to read.

Location: - `Arbor/Widget/Measure.lean` - various tuple constructions - `Arbor/Text/Renderer.lean` - fold operations

Action Required: Add explicit type annotations for clarity.

Estimated Effort: Small

Testing Improvements

[Priority: High] Add Property-Based Tests

Issue: Current tests are example-based. Property-based testing would catch edge cases.

Location: `ArborTests/Main.lean`

Action Required: 1. Add Plausible dependency (already available in tincture) 2. Test properties like: - `measureWidget` is idempotent - `collectCommands` produces valid command sequences (balanced push/pop) - Hit testing respects widget bounds

Estimated Effort: Medium

[Priority: Medium] Add Snapshot Tests

Issue: ASCII rendering output is tested by checking substrings. Full output comparison would catch regressions.

Location: `ArborTests/Main.lean`, `ArborTests/AsciiRendererTests.lean`

Action Required: 1. Add expected output files 2. Compare rendered output against snapshots 3. Provide snapshot update mechanism

Estimated Effort: Small

[Priority: Medium] Test Event Dispatch

Issue: Event dispatch logic in `Arbor/App/UI.lean` has no dedicated tests.

Location: New test file needed

Action Required: 1. Create `ArborTests/EventTests.lean` 2. Test bubbling, capture, stop propagation 3. Test handler registration and lookup

Estimated Effort: Medium

API Ergonomics

[Priority: Medium] Add Widget Modification Helpers

Issue: Widgets are immutable, but there is no convenient way to modify nested widget properties.

Location: `Arbor/Widget/Core.lean`

Action Required: Add lens-like helpers or use Collimator optics:

```
def Widget.withStyle (f : BoxStyle -> BoxStyle) : Widget -> Widget
def Widget.mapChildren (f : Widget -> Widget) : Widget -> Widget
```

Estimated Effort: Small (Medium if using Collimator)

Dependencies: Optional: Collimator optics library

[Priority: Low] Builder Do-Notation Sugar

Issue: Building complex widget trees requires explicit `#[...]` arrays.

Location: `Arbor/Widget/DSL.lean`

Action Required: Explore do-notation builder pattern:

```
-- Current
row {} #[box a, box b, box c]
-- Potential
row {} do
  box a
  box b
  box c
```

Estimated Effort: Medium

Summary

Category	High	Medium	Low
Features	3	5	3
Improvements	2	5	1
Cleanup	1	4	3
Testing	1	2	0
Ergonomics	0	1	1
Total	**7**	**17**	**8**

Recommended Starting Points

1. **Eliminate partial functions** - improves code quality with low risk
2. **Add image render command** - small feature with high value
3. **Focus management** - essential for keyboard accessibility
4. **Property-based tests** - catches bugs early
5. **Documentation** - helps users adopt the library