# Tidal Combinators: Visual Representation Ideas

Brainstorm session on how to visualize Tidal's Haskell combinators in the Algorave Pattern Visualizer.

## Philosophy

> "To simplify, add detail" - Edward Tufte

We can bring the full firepower of dataviz to bear: small multiples, brushing, transform/translate, projection, symbols, hover effects, and even inventing new visualizations.

---

## Time & Structure Combinators

### 1. `jux f` / `juxBy n f` — Juxtapose (stereo split)

*Applies function to one stereo channel, original to other*

**Visual ideas:**

- **Mirror/reflection**: Show the pattern and its transformed version side-by-side or as a reflection
- **Split ring**: Sunburst with left half original, right half transformed (like yin-yang)
- **Stereo field indicator**: L/R markers with connecting arc
- **Parallax depth**: Original in front plane, transformed slightly offset (3D effect)

---

### 2. `rev` — Reverse

*Plays pattern backwards*

**Visual ideas:**

- **Mirrored sunburst**: Flip the angular direction (counter-clockwise)
- **Reflection line**: Draw a vertical axis with mirrored pattern
- **Arrow indicators**: Counter-flow arrows around the ring
- **Color gradient reversal**: Hue shifts in opposite direction

---

### 3. `fast n` / `slow n` — Time scaling

*Speed up or slow down by factor n*

**Visual ideas:**

- **Ring thickness**: Fast = thinner rings (compressed), Slow = thicker rings (stretched)
- **Radial scaling**: Fast = smaller radius, Slow = larger radius
- **Density visualization**: Fast shows more repetitions in same angular space
- **Pulse rate indicator**: Small animation speed difference
- **Compression marks**: Like accordion pleats for fast, stretched marks for slow

---

### 4. `chop n` — Granular slicing

*Cuts each sample into n pieces*

**Visual ideas:**

- **Segmented arcs**: Each sound arc subdivided into n slices with subtle gaps
- **Hatching/striping**: Diagonal lines across the arc showing granularity
- **Pixelation effect**: More "digital" looking edges
- **Slice markers**: Small radial tick marks within each sound

---

### 5. `striate n` — Interleaved granular

*Like chop but interleaves the slices*

**Visual ideas:**

- **Woven pattern**: Alternating colored stripes showing interleave
- **Interlocking teeth**: Zipper-like visual between sounds
- **Phase offset markers**: Shows how slices are redistributed

---

### 6. `ply n` — Multiplication

*Repeats each event n times*

**Visual ideas:**

- **Echo rings**: Each sound has n concentric echo rings
- **Stutter marks**: Repeated tick marks
- **Multiplication badge**: Small "×n" indicator

---

## Layering & Stacking Combinators

### 7. `layer [f1, f2, ...]` — Parallel function application

*Applies multiple functions, stacks results*

**Visual ideas:**

- **Concentric rings**: Each function creates a new ring layer
- **Transparency/alpha**: Overlapping semi-transparent sunbursts
- **Small multiples**: Array of mini-sunbursts, one per function
- **Depth stacking**: Z-axis layering with shadow
- **Color blending**: Where layers overlap, colors blend

---

### 8. `stack [p1, p2, ...]` — Pattern stacking

*Plays patterns simultaneously*

**Visual ideas:**

- **Overlaid sunbursts**: Multiple patterns in same space, different opacity
- **Composite ring**: Merged into single ring with multi-colored segments
- **Vertical stack**: 3D extrusion showing depth
- **Interference pattern**: Moiré-like effect where patterns interact

---

### 9. `cat [p1, p2, ...]` — Concatenation

*Plays patterns one after another*

**Visual ideas:**

- **Sequential segments**: Like a pie chart where each pattern gets a wedge of the cycle
- **Timeline view**: Linear representation showing temporal sequence
- **Spiral**: Patterns continue around multiple rotations
- **Color-coded sections**: Clear boundaries between concatenated patterns

---

## Periodic & Conditional Combinators

### 10. `every n f` — Periodic transformation

*Applies f every nth cycle*

**Visual ideas:**

- **Highlighted cycle**: Every nth ring/rotation emphasized
- **Pulse animation**: Brightness pulse on the nth cycle
- **Small multiples timeline**: Show n cycles, highlight which ones transform
- **Orbital indicator**: Like moon phases showing which cycle is "active"

---

### 11. `every' n o f` — Offset periodic

*Like every but with offset o*

**Visual ideas:**

- **Phase-shifted highlight**: Same as every but rotated start position
- **Offset marker**: Arrow or indicator showing the phase shift

---

### 12. `chunk n f` / `chunk' n f` — Section-wise transform

*Applies f to 1/nth of the pattern at a time*

**Visual ideas:**

- **Highlighted sector**: One nth of the ring highlighted
- **Sweeping spotlight**: Animation showing which chunk is active

- **Pie slice emphasis**: Distinct visual treatment for active chunk

---

## Rotation & Phase Combinators

### 13. `iter n` — Rotation

*Shifts pattern start point each cycle*

**Visual ideas:**

- **Rotating arrow**: Shows current rotation offset
- **Spiral unwind**: Pattern spirals outward showing progression
- **Clock hand**: Indicator that moves around each cycle
- **Multiple ghost positions**: Faint copies showing all rotation states

---

### 14. `spin n` — Multi-rotation

*Creates n copies at different rotations*

**Visual ideas:**

- **Kaleidoscope**: n-fold rotational symmetry
- **Propeller/fan blades**: Pattern repeated n times around center
- **Phase array**: Small multiples arranged radially

---

## Shuffle & Randomization Combinators

### 15. `shuffle n` / `scramble n` — Reordering

*Randomly reorders n subdivisions*

**Visual ideas:**

- **Jigsaw effect**: Pieces visually displaced/shuffled
- **Numbered segments**: Show original vs shuffled order
- **Connection lines**: Lines showing where segments moved from/to
- **Entropy indicator**: More "chaotic" visual treatment

---

### 16. `press` — Swing/shuffle feel

*Shifts every other event later*

**Visual ideas:**

- **Offset alternation**: Every other segment slightly rotated
- **Swing indicator**: Musical swing notation symbol
- **Alternating depths**: 3D push/pull effect

---

## Advanced Combinators

### 17. `bite n pat` — Pattern-controlled slicing

*Uses pat to select which of n slices to play*

**Visual ideas:**

- **Index overlay**: Small pattern controls which slices appear
- **Conditional highlighting**: Some slices solid, others ghosted
- **Puppet strings**: Lines from control pattern to selected slices

---

## Meta-Visualization Strategies

### Small Multiples

Show the pattern at different stages of transformation pipeline:

```
[original] → [after jux] → [after chop] → [final]
```

### Brushing & Linking

- Hover over a combinator in code → highlight affected parts in viz
- Select a sound in viz → highlight in code where it came from

### Transform Animation

- Animate the transformation: show pattern morphing from before→after
- Scrubber to see intermediate states

### Nesting Visualization

For deeply nested combinators like:

```
jux rev $ fast 4 $ chop 16 $ s "bd"
```

- **Onion layers**: Each combinator adds a ring/shell
- **Tree view**: AST-style breakdown alongside sunburst
- **Breadcrumb trail**: Shows transformation chain

### Diff View

Side-by-side: original pattern vs transformed, with visual diff highlighting what changed

---

---

## Key Insight: Structure vs Transformation

**Mini-notation = Nouns/Structure** — the "what"

- Sounds, sequences, parallels, choices
- Static tree structure
- Currently visualized as sunbursts

**Combinators = Verbs/Transformations** — the "how"

- jux, rev, chop, fast, slow, layer, every...
- Actions applied to nodes
- Transform the structure

### Affordances-First Thinking

Instead of asking "how do we visualize combinators?", ask: **"What actions should be possible on the visualization?"**

The mute button is already an affordance — click center to toggle. What if:

- **Click interior node** → menu of applicable combinators appears
- **Drag combinator tool** onto a node → applies transformation
- **Right-click** → "reverse this", "chop 4", "fast 2"
- **Keyboard shortcuts** while hovering → r=reverse, f=fast, s=slow
- **Gesture** → draw circle around node to add `spin`

### The Viz Becomes the Instrument

This shifts from "visualize the code" to **"the visualization IS the code"**:

1. Visual edits flow back to code (round-trip)
2. The sunburst is a live performance interface
3. Direct manipulation of musical structure
4. Combinators are tools/brushes, nodes are canvas

### Interior Nodes as Transformation Points

Combinators naturally apply to **interior nodes** (sequences, parallels, choices):

```
      [seq]  ← apply `fast 2` here
     / | \
   bd sn  hh
```

Becomes:

```
     [fast 2]
        |
      [seq]
     / | \
   bd sn  hh
```

The sunburst grows a new ring representing the combinator wrapper.

### Interaction Model Sketch

1. **Hover** over interior node → highlight, show tooltip with current transforms
2. **Click** → select node, show transform toolbar
3. **Apply transform** →
   - Update tree structure (add wrapper node)
   - Re-render sunburst (new ring/visual treatment)
   - Update code output (show new combinator)
4. **Undo/history** → step back through transformations

---

## Implementation Priority (TBD)

1. [ ] Start with `fast`/`slow` - already parsed, just need visual treatment
2. [ ] `jux` with mirror effect - high visual impact
3. [ ] `layer`/`stack` with transparency
4. [ ] `chop` with segmentation
5. [ ] `every` with cycle highlighting
6. [ ] Animation layer for temporal effects

---

## References

- Tidal documentation: https://tidalcycles.org/docs/
- Pondskater algorave code (screenshot in session)
- Patterning iPad app (radial drum sequencer inspiration)