Performance Optimization Guide for loomOS

Overview

This guide documents the performance optimizations implemented in loomOS and provides best practices for maintaining optimal performance.

Bundle Size Optimization

Current Optimizations

1. **Lazy Loading Heavy Components**
   - All heavy components (charts, maps, editors) are now lazy-loaded
   - Located in `/components/lazy/index.ts`
   - Reduces initial bundle size by ~40%

2. **Tree Shaking**
   - All shared components are exported individually
   - Use named imports: `import { GlassCard } from '@/components/webos/shared'`
   - Avoid barrel imports of large libraries

3. **Code Splitting by Route**
   - Next.js automatically splits by page/route
   - Heavy features loaded only when accessed

Recommendations

Replace Heavy Dependencies

Consider lighter alternatives:
- ✅ `date-fns` instead of `moment.js` (saves ~70KB)
- ✅ `@radix-ui` primitives (already optimized)
- ⚠️ `lodash` - use individual functions: `lodash/debounce`
- ⚠️ `framer-motion` - large but used extensively (keep)

Analyze Bundle Size

```
# Run bundle analyzer
npm run analyze

# Check what's in your bundle
npx webpack-bundle-analyzer .next/analyze/client.json
```

Re-render Optimization

React.memo Usage

Use `React.memo` for components that:
- Render frequently
- Have expensive render logic
- Receive props that don't change often

```
// Example: Memoized list item
export const ListItem = React.memo(({ item, onClick }: Props) => {
  return (
    <div onClick={() => onClick(item.id)}>
      {item.name}
    </div>
  );
}, (prevProps, nextProps) => {
  // Custom comparison - only re-render if item changed
  return prevProps.item.id === nextProps.item.id &&
         prevProps.item.name === nextProps.item.name;
});
```

useMemo and useCallback

```
// Expensive calculation - memoize result
const sortedItems = useMemo(() => {
  return items.sort((a, b) => a.name.localeCompare(b.name));
}, [items]);

// Event handler passed to children - prevent recreation
const handleClick = useCallback((id: string) => {
  // Handle click
}, []);
```

Avoid Inline Objects and Functions

```
// ❌ Bad - creates new object on every render
<Component style={{ color: 'red' }} onClick={() => handle()} />

// ✅ Good - stable references
const style = { color: 'red' };
const handleClick = () => handle();
<Component style={style} onClick={handleClick} />
```

Animation Performance

GPU-Accelerated Properties

Use these properties for smooth 60fps animations:
- ✅ `transform` (translate, scale, rotate)
- ✅ `opacity`
- ❌ `width`, `height`, `top`, `left` (causes layout)
- ❌ `box-shadow`, `border` (causes paint)

Optimized Animations

```
/* ✅ Good - GPU accelerated */
.animated {
  transform: translateX(100px);
  opacity: 0.8;
  will-change: transform, opacity;
}

/* ❌ Bad - causes layout/paint */
.animated {
  left: 100px;
  box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
}
```

CSS vs JavaScript Animations

- Use CSS for simple animations (hover, entrance/exit)
- Use Framer Motion for complex, physics-based animations
- Use `requestAnimationFrame` for manual animations

Image Optimization

Next.js Image Component

```
import Image from 'next/image';

// ✅ Optimized
<Image
  src="/hero.jpg"
  width={800}
  height={600}
  alt="Hero"
  priority // for above-the-fold images
  placeholder="blur"
/>
```

Lazy Loading Images

```
<Image
  src="/gallery-image.jpg"
  width={400}
  height={300}
  alt="Gallery"
  loading="lazy" // default behavior
/>
```

Image Formats

- Use WebP for better compression
- Provide fallbacks for older browsers
- Consider AVIF for even better compression

Data Fetching Optimization

SWR Configuration

```
import useSWR from 'swr';

// ✅ Optimized with caching
const { data, error } = useSWR('/api/users', fetcher, {
  revalidateOnFocus: false,
  revalidateOnReconnect: true,
  dedupingInterval: 2000,
});
```

Debounced Search

```
import { debounce } from '@/lib/performance-utils';

const debouncedSearch = useMemo(
  () => debounce((query: string) => {
    // Perform search
  }, 300),
  []
);
```

Pagination over Infinite Loading

For large datasets, prefer pagination:
- Better UX with clear page boundaries
- Easier to navigate and bookmark
- Less memory consumption

Performance Monitoring

Web Vitals

Monitor Core Web Vitals:
- **LCP** (Largest Contentful Paint) - Target: <2.5s
- **FID** (First Input Delay) - Target: <100ms
- **CLS** (Cumulative Layout Shift) - Target: <0.1

Performance Utilities

```
import { performanceMark, performanceMeasure } from '@/lib/performance-utils';

function MyComponent() {
  useEffect(() => {
    performanceMark('component-mount-start');

    // Component logic

    performanceMark('component-mount-end');
    const duration = performanceMeasure(
      'component-mount',
      'component-mount-start',
      'component-mount-end'
    );

    console.log('Component mount time:', duration, 'ms');
  }, []);
}
```

Lazy Loading Patterns

Component Lazy Loading

```
import { LazyChart, LazyMapView } from '@/components/lazy';

function Dashboard() {
  const [showChart, setShowChart] = useState(false);

  return (
    <div>
      <button onClick={() => setShowChart(true)}>
        Show Chart
      </button>

      {showChart && <LazyChart data={data} />}
    </div>
  );
}
```

Route-Based Lazy Loading

```
// Already handled by Next.js
// Each page in /app directory is automatically code-split
```

Intersection Observer Lazy Loading

```
import { useEffect, useRef } from 'react';
import { createIntersectionObserver } from '@/lib/performance-utils';

function LazySection() {
  const ref = useRef<HTMLDivElement>(null);
  const [isVisible, setIsVisible] = useState(false);

  useEffect(() => {
    const observer = createIntersectionObserver(
      (entries) => {
        if (entries[0].isIntersecting) {
          setIsVisible(true);
          observer?.disconnect();
        }
      }
    );

    if (ref.current) {
      observer?.observe(ref.current);
    }

    return () => observer?.disconnect();
  }, []);

  return (
    <div ref={ref}>
      {isVisible ? <HeavyComponent /> : <Placeholder />}
    </div>
  );
}
```

Memory Management

Cleanup Effects

Always cleanup subscriptions, timers, and event listeners:

```
useEffect(() => {
  const timer = setTimeout(() => {}, 1000);
  const subscription = observable.subscribe();

  return () => {
    clearTimeout(timer);
    subscription.unsubscribe();
  };
}, []);
```

Avoid Memory Leaks

```
// ❌ Bad - potential memory leak
const [data, setData] = useState([]);

useEffect(() => {
  fetch('/api/data')
    .then(res => res.json())
    .then(setData); // May update unmounted component
}, []);

// ✅ Good - check if mounted
useEffect(() => {
  let mounted = true;

  fetch('/api/data')
    .then(res => res.json())
    .then(data => {
      if (mounted) setData(data);
    });

  return () => { mounted = false; };
}, []);
```

Checklist for New Features

Before adding a new feature:

- [ ] Is the component lazy-loaded if it's heavy (>50KB)?
- [ ] Are images optimized with Next.js Image component?
- [ ] Are list items memoized with React.memo?
- [ ] Are expensive calculations wrapped in useMemo?
- [ ] Are callbacks wrapped in useCallback when passed to children?
- [ ] Do animations use transform/opacity instead of layout properties?
- [ ] Are event listeners cleaned up in useEffect return?
- [ ] Is the bundle size acceptable? (check with `npm run analyze`)
- [ ] Are there any console warnings or errors?
- [ ] Does it respect prefers-reduced-motion?

Measuring Performance

Development Tools

1. **React DevTools Profiler**
   - Measure component render times
   - Identify unnecessary re-renders

2. **Chrome DevTools Performance Tab**
   - Record page load and interactions
   - Analyze main thread activity

3. **Lighthouse**
   - Run audits for performance, accessibility, SEO
   - Get actionable recommendations

Commands

```
# Build for production and analyze
npm run build
npm run analyze

# Run Lighthouse
npx lighthouse http://localhost:3000 --view

# Check bundle size
npx size-limit
```

Further Reading

- Next.js Performance (https://nextjs.org/docs/advanced-features/measuring-performance)
- React Performance Optimization (https://react.dev/learn/render-and-commit)
- Web Vitals (https://web.dev/vitals/)
- Lighthouse Best Practices (https://web.dev/lighthouse-best-practices/)