Gaussian Splat Viewer - Project Planning Document

Project Overview

Objective: Create a web-based application that allows users to upload and visualize Gaussian splat files (.ply and .splat formats) with real-time 3D rendering capabilities.

Target Users: 3D artists, researchers, developers working with Gaussian splatting, and enthusiasts exploring 3D reconstruction techniques.

Core Value Proposition: Instant, browser-based visualization of Gaussian splat data without requiring specialized software installation.

Technical Architecture

Frontend Stack

- **Framework:** React with TypeScript for robust component architecture
- **3D Rendering:** Three.js for WebGL-based 3D graphics
- **File Processing:** Custom parsers for .ply and .splat formats
- **UI Framework:** Modern CSS with Tailwind for responsive design
- **State Management:** React hooks for application state

Backend Considerations

- **Phase 1:** Client-side only (no backend required)
- **Future phases:** Optional cloud storage and processing capabilities

Browser Requirements

- Modern browsers with WebGL 2.0 support
- File API support for local file uploads
- Adequate GPU memory for large splat datasets

Project Phases

Phase 1: Foundation & Core Parsing (Weeks 1-2)

Deliverables:

- Project setup with React + TypeScript + Three.js

- File upload interface with drag-and-drop support
- Basic .ply file parser implementation
- Simple point cloud visualization
- Error handling for unsupported files

Technical Tasks:

- Set up development environment and build pipeline
- Implement PLY format parser (handle vertices, normals, colors)
- Create basic Three.js scene with camera controls
- Build file validation and error messaging system
- Implement basic loading states and progress indicators

Success Criteria:

- Users can upload .ply files
- Basic point visualization appears in 3D space
- Camera controls (orbit, zoom, pan) function properly

Phase 2: Gaussian Splat Rendering (Weeks 3-4)

Deliverables:

- .splat file format support
- Gaussian splat shader implementation
- Proper splatting visualization with ellipsoids
- Performance optimization for medium-sized datasets

Technical Tasks:

- Research and implement .splat format parser
- Develop custom WebGL shaders for Gaussian splat rendering
- Implement ellipsoid geometry generation from splat parameters
- Add opacity and color blending for realistic visualization
- Optimize rendering pipeline for smooth frame rates

Success Criteria:

- Both .ply and .splat files render correctly
- Gaussian splats display as proper ellipsoids, not just points
- Maintains 30+ FPS for datasets under 100K splats

Phase 3: Enhanced Visualization & Controls (Weeks 5-6)

Deliverables:

- Advanced camera controls and presets
- Visualization settings panel (point size, opacity, color modes)
- Scene statistics and information display

- Screenshot/export functionality

Technical Tasks:

- Implement orbital camera with smooth transitions
- Create settings panel for visual customization
- Add scene information display (splat count, bounds, etc.)
- Implement canvas-to-image export functionality
- Add keyboard shortcuts for common actions

Success Criteria:

- Users can easily navigate and customize the visualization
- Settings persist during session
- Export functionality works reliably

Phase 4: Performance & Large Dataset Support (Weeks 7-8)

Deliverables:

- Level-of-detail (LOD) system for large datasets
- Memory management and streaming capabilities
- Progressive loading with visual feedback
- Performance monitoring and optimization

Technical Tasks:

- Implement frustum culling for off-screen splats
- Create LOD system based on distance and splat size
- Add memory usage monitoring and cleanup
- Implement progressive loading for large files
- Optimize shader performance and reduce overdraw

Success Criteria:

- Handles datasets with 500K+ splats smoothly
- Memory usage remains reasonable (< 2GB for large files)
- Loading times are acceptable with clear progress indication

Phase 5: User Experience & Polish (Weeks 9-10)

Deliverables:

- Responsive design for various screen sizes
- Comprehensive error handling and user feedback
- Help documentation and usage examples
- Performance analytics and user guidance

Technical Tasks:

- Implement responsive layout for mobile and tablet
- Create comprehensive error messages and recovery options
- Add interactive tutorial or help system
- Implement usage analytics (file sizes, performance metrics)
- Polish UI/UX based on testing feedback

Success Criteria:

- Application works well on desktop, tablet, and mobile
- Users can easily understand how to use the application
- Error states are handled gracefully

Phase 6: Advanced Features (Weeks 11-12)

Deliverables:

- Multiple file support (compare/overlay splats)
- Animation timeline for temporal datasets
- Advanced rendering modes (wireframe, normals, etc.)
- Sharing capabilities

Technical Tasks:

- Implement multi-file loading and scene management
- Create timeline controls for animated sequences
- Add alternative rendering modes and debug views
- Implement URL-based sharing of view states
- Add measurement tools and scene analysis features

Success Criteria:

- Users can load and compare multiple splat files
- Animation playback is smooth and controllable
- Sharing functionality enables collaboration

File Format Specifications

.ply Format Support

- **Vertices:** Position (x, y, z), Color (r, g, b), Normal (nx, ny, nz)
- **Headers:** Property parsing and validation
- **Encoding:** Both ASCII and binary formats
- **Extensions:** Support for custom properties related to Gaussian splatting

.splat Format Support

- **Structure:** Position, scale, rotation, opacity, spherical harmonics
- **Compression:** Handle various compression schemes

- **Metadata:** Extract and display relevant file information

Performance Targets

Rendering Performance

- **Small datasets (< 10K splats):** 60+ FPS
- **Medium datasets (10K-100K splats):** 30+ FPS
- **Large datasets (100K-1M splats):** 15+ FPS with LOD

Loading Performance

- **File parsing:** < 5 seconds for 100MB files
- **Initial render:** < 10 seconds for complex scenes
- **Memory usage:** < 2GB total browser memory

Risk Assessment & Mitigation

Technical Risks

1. **Browser memory limitations**
    - *Mitigation:* Implement streaming and LOD systems
    - *Fallback:* Provide file size recommendations
2. **WebGL compatibility issues**
    - *Mitigation:* Feature detection and graceful degradation
    - *Fallback:* Simple point cloud rendering for unsupported features
3. **File format variations**
    - *Mitigation:* Robust parsing with format validation
    - *Fallback:* Clear error messages with format requirements

User Experience Risks

1. **Complex interface overwhelming users**
    - *Mitigation:* Progressive disclosure and guided tutorials
    - *Fallback:* Simple mode with basic controls only
2. **Poor performance on low-end devices**
    - *Mitigation:* Automatic quality adjustment based on device capabilities
    - *Fallback:* Performance warnings and manual quality controls

Success Metrics

Technical Metrics

- File parsing success rate: > 95%
- Rendering performance: Meets targets above
- Memory efficiency: < 10MB per 1K splats
- Cross-browser compatibility: Chrome, Firefox, Safari, Edge

User Experience Metrics

- Time to first render: < 30 seconds for typical files
- User task completion: Upload and view in < 5 clicks
- Error recovery: Clear error messages with actionable solutions

Future Enhancements (Post-Launch)

Advanced Features

- Cloud storage integration
- Collaborative viewing sessions
- VR/AR support for immersive viewing
- AI-powered scene analysis and optimization
- Integration with popular 3D software pipelines

Performance Improvements

- WebGPU support for better performance
- Web Workers for background processing
- Advanced compression algorithms
- Server-side preprocessing options

Development Resources

Required Skills

- **Frontend:** React, TypeScript, Three.js, WebGL
- **3D Graphics:** Shader programming, 3D mathematics, rendering pipelines
- **File Processing:** Binary format parsing, data structure optimization

Development Environment

- **IDE:** VS Code with TypeScript and React extensions
- **Testing:** Jest for unit tests, Cypress for integration tests
- **Deployment:** Vercel/Netlify for static hosting
- **Version Control:** Git with feature branch workflow

External Dependencies

- Three.js for 3D rendering
- File processing libraries for binary parsing
- UI component libraries for interface elements

Conclusion

This project will deliver a powerful, accessible tool for viewing Gaussian splat data directly in web browsers. The phased approach ensures steady progress while allowing for iteration and improvement based on user feedback. The focus on performance and user experience will make this tool valuable for both technical users and newcomers to Gaussian splatting technology.