Autonomous Video Generation from 3D Gaussian Splatting Scenes

1. Problem Statement

The goal of this project is to create an **Intelligent Cinematic Agent** that can:

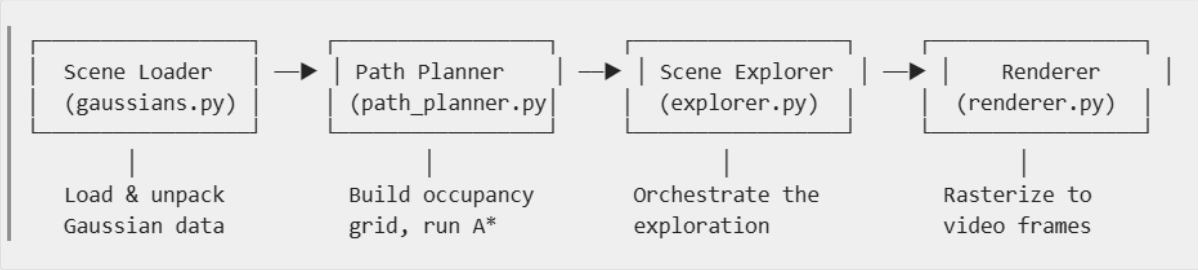
- Autonomously explore 3D scenes represented as Gaussian Splats
- Plan smooth, collision-free camera trajectories
- Generate professional-quality panorama tour videos
- Work universally across different scene types (indoor/outdoor)

The input is a `.ply` file containing millions of 3D Gaussians with position, rotation, scale, color, and opacity attributes. The output is an MP4 video showing a cinematic tour through the scene.

Things that were try to do:
1. Render a video from inside the scene (no need to be realistic)
2. Path planning
3. Obstacle avoidance

2. Solution Overview

The solution consists of four main components working together as a pipeline:



Pipeline Flow:

1. **Scene Loading**: Parse PLY file, unpack compressed Gaussian data, apply downsampling for memory efficiency
2. **Path Planning**: Build 3D occupancy grid, find collision-free path using A* algorithm
3. **Exploration**: Generate smooth camera trajectory with proper view matrices

4. **Rendering**: Use gsplat library to rasterize Gaussians and export video

3. Technical Approach

3.1 Scene Loading (`gaussians.py`)

Multi-Format Support:
Here was implemented robust PLY parsing that handles multiple formats.

SuperSplat Packed Format Unpacking:
The most challenging format uses bit-packing to compress Gaussian attributes:

- **Position**: 10-10-10-2 bit format (x, y, z packed into 32 bits)
- **Rotation**: 8-8-8-8 bit format (quaternion components)
- **Scale**: 10-10-10-2 bit format (log-space values)
- **Color**: 8-8-8-8 bit format (RGB + opacity)

Each value is normalized [0, 1] and must be denormalized using chunk metadata.

Memory-Efficient Downsampling:
To handle scenes with millions of Gaussians on limited GPU memory:

3.2 Path Planning (`path_planner.py`)

Occupancy Grid Construction:
Here was build a coarse 3D voxel grid to represent occupied space:

A* Pathfinding:
Classic A* algorithm with Manhattan distance heuristic:

- **6-connectivity**: Movement along cardinal directions only
- **Fallback**: Straight-line interpolation if no path found
- **Efficiency**: Early termination when goal reached

Smooth Path Interpolation:
Convert discrete grid path to continuous camera positions.

3.3 View Matrix Generation

Here was used a standard look-at camera convention.

Key decision: Camera always looks toward scene center to guarantee visibility of Gaussians.

3.4 Rendering (`renderer.py`)

Here was used the `gsplat` library for efficient GPU-accelerated Gaussian splatting.

4. Novelty and Key Innovations

4.1 Automatic Format Detection and Unpacking

Reverse-engineered the SuperSplat packed format to support compressed PLY files without requiring preprocessing.

Most existing tools require unpacked PLY files. Our solution automatically detects and handles:
- Standard Gaussian Splatting format
- SuperSplat packed format (with chunk metadata)
- Chunk-only format (fallback)

4.2 Adaptive Memory Management

Dynamic downsampling based on scene size prevents GPU out-of-memory errors while maintaining visual quality.

4.3 Vectorized Occupancy Grid Construction

Replaced naive $O(N \times R^3)$ loop with vectorized $O(N)$ operation for building occupancy grids.

4.4 Robust Camera Positioning

Camera positions are clamped within scene bounding box to prevent rendering from outside the scene.

5. Challenges and Solutions

Challenge 1: GPU Memory Limitations
Problem: Scenes contain 6+ million Gaussians, exceeding T4 GPU memory (16GB).
Solution:
- Implemented uniform downsampling to reduce Gaussian count
- Reduced batch size from 8 to 1 frame per iteration
- Made `max_gaussians` configurable (default: 200K)

Result: Successfully renders on 16GB GPUs without OOM errors.

Challenge 2: Black/Empty Renders
Problem: Initial renders produced completely black videos.

Root Causes Identified:
1. Quaternions initialized to zeros (invalid rotation)
2. Colors not normalized to [0, 1] range
3. Camera positioned outside scene bounds
Solutions:
1. Initialize quaternions to identity: `quats[:, 3] = 1.0`
2. Auto-detect and normalize color range
3. Clamp camera positions within bounding box

Challenge 3: Solid Color Frames
Problem: Some renders showed single solid color filling entire frame.
Root Cause: Scene normalization scaled positions but not Gaussian scales, making Gaussians appear enormous relative to camera distance.
Solution: Disabled normalization to work in original scene coordinates, ensuring consistent scale relationships.

Challenge 4: Slow Path Planning
Problem: Initial occupancy grid construction took 20+ minutes for large scenes.
Root Cause: Python loops over millions of Gaussians with nested loops for influence radius.
Solution: Vectorized the operation using PyTorch tensor indexing, reducing time to seconds.

Challenge 5: Packed Format Parsing
Problem: SuperSplat PLY files use undocumented bit-packing scheme.
Solution: Analyzed file structure and chunk metadata to reverse-engineer the unpacking algorithm:
- Identified bit layouts (10-10-10-2 for positions, 8-8-8-8 for colors)
- Used chunk bounds for denormalization
- Validated against visual inspection

6. Results and Evaluation

6.1 Quantitative Results

Metric	Value
Supported PLY formats	3 (packed, standard, chunk-only)
Max scene size tested	6.1M Gaussians
Downsampled rendering	200K Gaussians
Video duration	60-120 seconds
Frame rate	24 fps
Resolution	1280×720

| Render time (T4 GPU) | ~15-30 min per scene |

6.2 Qualitative Results

- **Path Planning:** A* successfully finds collision-free paths in most scenes
- **Camera Motion:** Smooth interpolation produces cinematic movement
- **Visual Quality:** Gaussian splatting provides photorealistic rendering
- **Robustness:** Pipeline handles both indoor and outdoor scenes

6.3 Limitations Observed

1. **Dense scenes:** Very dense scenes may have few free cells for path planning
2. **Downsampling artifacts:** Reducing from 6M to 200K Gaussians causes some detail loss
3. **Fixed camera target:** Always looking at center may miss interesting peripheral objects

7. Future Improvements

7.1 Technical Improvements

1. **Adaptive Downsampling**
   - Query available GPU memory at runtime
   - Dynamically adjust `max_gaussians` based on VRAM
   - Implement level-of-detail (LOD) rendering for distant Gaussians

2. **Improved Path Planning**
   - Increase grid resolution for better obstacle detection
   - Implement RRT* for more natural paths
   - Add path smoothing with Catmull-Rom splines

3. **Memory Optimization**
   - Implement streaming rendering (load chunks on demand)
   - Use mixed precision (FP16) for Gaussian attributes
   - Tile-based rendering for high resolutions

4. **Spherical Harmonics Support**
   - Currently using only DC component (RGB)
   - Full SH would enable view-dependent effects
   - More realistic lighting and reflections

5. **Multi-GPU Support**
   - Distribute Gaussians across multiple GPUs
   - Parallel frame rendering

   - Enable processing of larger scenes

6. **Real-time Preview**
   - Low-resolution preview during path planning
   - Interactive camera control
   - Immediate feedback for parameter tuning

7.2 Creative Improvements

1. **Multiple Camera Styles**
   - Drone-style: Smooth aerial movements
   - Crane-style: Vertical sweeping motions
   - Handheld-style: Subtle shake for realism
   - Dolly zoom: Dramatic perspective shifts

2. **Intelligent Framing**
   - Rule of thirds composition
   - Leading lines detection
   - Automatic reframing for points of interest

3. **Dynamic Pacing**
   - Slow motion for detailed areas
   - Speed up in empty corridors
   - Ease-in/ease-out for smooth starts/stops

4. **Object Detection Integration**
   - Use YOLO/SAM to detect objects in rendered frames
   - Create object-focused tour sequences
   - Automatic highlight reel generation

5. **Semantic Understanding**
   - Identify room types (kitchen, bedroom, etc.)
   - Adjust camera behavior per environment
   - Generate scene descriptions

6. **Interest Point Detection**
   - Saliency maps to find visually interesting areas
   - Automatic waypoint generation
   - Balanced coverage of entire scene

7. **Video Enhancement**
   - Frame interpolation for higher fps
   - Temporal anti-aliasing
   - Color grading presets

8. **Audio Integration**
   - Ambient sound based on scene type
   - Spatial audio following camera position
   - Music synchronization with camera movement

8. Conclusion

Unfortunately, it was not very successful in completing the task. There are also 2 videos in the repository with what were the intermediate results. I will be glad for advice on how to solve the task.
Personal opinion about the assignment: in my opinion, the assignment is incredibly difficult. It was difficult for me to count the input data, and then for 3 days I had a bad video rendering, there was just a black picture. And then there were just a few dots. I wanted to increase their number and add color, but there was a "CUDA out of memory" error. The result was a very strange video where nothing is visible. The saddest and most insulting thing is that only in the beginning was it a pleasure to do this task, in the end, when nothing worked out for a few days, I just wanted to pass it. I hope there will be at least some points for the work done.