Technical Report: Cinematic Gaussian Splatting Tour

1. Problem Solution

The goal of this project was to generate a cinematic fly-through video of a 3D scene represented by 3D Gaussian Splatting, with the additional capability of detecting objects within the scene.

Our solution is composed of a pipeline with the following stages:

1. **Scene Loading & Normalization**:
    - We load the Gaussian Splatting scene (from a `.ply` file) using a custom loader.
    - To ensure consistent camera movement regardless of the scene's scale or origin, we calculate the scene's bounding box and centroid. The scene is then normalized (centered and scaled) so that the camera path parameters work universally across different scenes.
2. **Path Generation (The "Explorer")**:
    - Instead of simple circular orbits, we implemented a **Lissajous-based path generator**. This creates a complex, organic 3D curve that explores the volume of the scene.
    - The path is generated by combining sine waves of different frequencies on the X, Y, and Z axes.
    - We identify the **principal axis** (the longest dimension of the scene) and align the primary movement along it to maximize coverage.
3. **Path Smoothing & Orientation**:
    - The coarse waypoints from the generator are interpolated using **Cubic Splines** to create a smooth, continuous path for the video frames.
    - Camera orientation is handled by a "look-ahead" mechanism. The camera target is calculated by sampling the path function slightly ahead in time, ensuring the camera naturally looks in the direction of travel.
4. **Rendering**:
    - We utilize the `gsplat` library for high-performance rasterization of the Gaussian splats.
    - The system renders each frame based on the interpolated camera poses.
5. **Object Detection**:
    - We integrated **YOLO (You Only Look Once)** to perform object detection on the rendered frames. This adds a layer of scene understanding, identifying objects like "chair", "potted plant", etc., during the fly-through.

2. Novelty and "Proud Tricks"

- **Lissajous Pathing**: Using a Lissajous curve (\(x = A\sin(at+\delta), y = B\sin(bt)\)) allows for a non-repetitive, room-filling path that looks much more interesting than a standard turntable rotation. It feels like a drone exploring the room.
- **The "Look-Ahead" Orientation**: A simple but effective trick we used is calculating the `look_at` point by evaluating the path function at \(t + \epsilon\). This guarantees that the camera anticipates turns, mimicking how a human pilot or cinematographer would operate.
- **Safe Margin Clamping**: To prevent the camera from flying outside the scene or clipping through outer walls, we implemented a "safe box" calculated as a percentage (15%) reduction of the scene's bounding box. All path points are clamped to this box. This simple geometric constraint proved robust for keeping the view focused on the content.

3. Challenges and Solutions

3.1. Installation Difficulties

Challenge: Installing `gsplat` and its CUDA dependencies locally on Windows proved to be extremely difficult due to version mismatches and compilation errors. **Solution**: We migrated the development and execution environment to **Kaggle Notebooks**. This provided a pre-configured environment with compatible CUDA drivers and PyTorch versions, allowing us to focus on the code rather than dependency hell.

3.2. Interactive Limitations

Challenge: Due to the limitations of the Kaggle environment (headless execution, no easy port forwarding for WebSocket/HTTP servers), we could not implement an interactive viewer or a 360-degree video player as originally intended. **Solution**: We pivoted to generating a high-quality, pre-rendered **cinematic video (MP4)**. This ensures the result is viewable anywhere without requiring the user to have a powerful GPU or complex software setup.

3.3. Obstacle Avoidance vs. Cinematic Quality

Challenge: Implementing effective obstacle avoidance (avoiding internal objects like tables or lamps) is difficult. A purely reactive avoidance system often results in jerky, unnatural camera movements that ruin the "cinematic" feel. **Solution**: We adopted a **balanced approach**. Instead of reactive collision avoidance, we used **preventative path planning**.

- We constrained the path to a "safe zone" derived from the scene's bounding box.
- We used **Cubic Spline interpolation** to smooth out any sharp changes caused by clamping.
- This trade-off ensures the camera stays generally safe while maintaining the smooth, flowing motion required for a pleasant viewing experience.

4. Results and Evaluation

The final system successfully generates 1080p videos at 24 FPS.

- **Visual Quality**: The rendered video is smooth and stable. The Gaussian splats are rendered correctly with appropriate depth and occlusion.
- **Path Quality**: The camera explores the scene comprehensively without exiting the valid volume. The movement feels intentional and directed.
- **Detection**: The YOLO integration successfully identifies common objects in the scene, overlaying bounding boxes that track with the camera movement.

5. Vision for Future Improvements

Technical Improvements

1. **Depth-Based Obstacle Avoidance**: Currently, we rely on bounding boxes. A significant improvement would be to render a **depth map** alongside the RGB image. We could use this depth map to detect imminent collisions with specific objects (not just walls) and dynamically adjust the spline path to "nudge" the camera away, preserving smoothness.
2. **Real-Time Interactive Viewer**: Moving away from Kaggle to a local or cloud-hosted WebGL/WebGPU viewer would allow users to control the camera themselves. Porting the Gaussian Splatting renderer to a web-compatible format (like Three.js or a custom WebGPU shader) would be the next logical step.
3. **Optimized Splat Culling**: To improve rendering speed, we could implement frustum culling to avoid processing splats that are behind the camera or outside the field of view.

Creative Directions

1. **"Director Mode"**: Instead of a random Lissajous path, we could implement a tool where the user selects 3-4 "Keypoints of Interest" (e.g., a specific painting or piece of furniture). The system would then generate a smooth path that visits these points in a logical order, lingering on them for a few seconds.
2. **Style Transfer**: Applying neural style transfer to the rendered frames could create artistic interpretations of the scene (e.g., "Van Gogh's Bedroom" in the style of actual Van Gogh).
3. **Audio Reactivity**: Syncing the camera speed or orbit radius to a music track would create a dynamic music video experience.