Point Cloud Scanning and Saving in Unity

Overview

The point cloud scanning system in Unity captures and stores 3D environment data by shooting rays, detecting surface hits, and recording positional and color information. The data is saved in a structured format for further visualization or processing.

How It Works

1. Point Structure

Each scanned point consists of:

- Position: A Vector3 representing the 3D coordinates of the point.
- Color: Stored as a 32-bit RGBA value for compatibility with Unity's VFX Graph.

2. Scanning Process

- The camera emits multiple rays in random directions (NumRays adjustable for accuracy).
- Each ray detects collisions with scene objects within the MaxDistance.
- If a valid surface is hit:
    - The position and color of the hit point are recorded.
    - The system avoids capturing unnecessary objects like the player or known scene meshes (e.g., "SeenMesh").
    - For objects with LOD groups, the closest mesh in LOD0 is used for scanning.
- The SeenMeshManager manages the hit triangle for future use by the AR games

3. Point Cloud Accumulation

- A scan is triggered when the camera moves significantly (CameraTranslationThreshold) or rotates (CameraRotationThreshold).
- Collected points are stored in a list (_points) for long-term storage and _scannedPoints for the current scan.

4. Visualization

- Captured points can be displayed using the PointCloudVisualizer component.
- A GameObject named PointCloud holds the visual representation.
- Materials can be applied for rendering and highlighting points.
- Visualization can be toggled on or off using the KeyCode.U.

5. Data Storage

- For each ScanObject() call:
    - One camera entry is written.
    - All point cloud entries of the current scan are stored.
- Data is written to text files asynchronously to avoid blocking the main thread.
- File Structure:
    - Point Cloud Data: Stored in pointcloudstr.txt with each line containing:
        - x y z r g b timestamp
    - Camera Data: Stored in camerastr.txt with each line containing:
        - x y z α β γ timestamp
- Files are saved in the folder:
    - PointCloudDataSets/{timestamp}_{identifier}/
- The system uses StringBuilder to buffer data and flushes it at regular intervals (_flushStringsTime).

Code Flow Overview

1. **Initialization (Start):**
    - Sets up the SeenMeshManager and PointCloudVisualizer.
    - Creates necessary directories and files for saving data.
    - Initializes variables like _points, _scannedPoints, and timers.
2. **Scanning (Update and ScanObject):**
    - Checks if the camera has moved or rotated enough to trigger a scan.
    - Shoots rays, detects hits, and records point data.
    - Stores hit triangles using SeenMeshManager.
3. **Saving (SavePointCloud and SavePoint):**
    - Saves camera and point data to StringBuilder.
    - Flushes data to files asynchronously to prevent performance issues.
4. **Visualization (TogglePointCloudDisplay):**
    - Toggles the visibility of the point cloud in the scene.
5. **Cleanup (OnDestroy and OnDisable):**
    - Ensures all buffered data is flushed and files are closed properly.

Key Components

- PointCloudVisualizer:
    - Handles the visual representation of the point cloud.
    - Adds points to a graphics buffer for rendering.
- SeenMeshManager:
    - Manages and updates triangles that have been scanned.
- PointCloud GameObject:
    - Holds the visualized point cloud in the scene.

Error Handling

- If the PointCloud GameObject is missing, a warning is logged, and visualization is disabled.
- If the MainCamera is not found, scanning is skipped, and an error is logged.

Why Asynchronous Saving?

- Writing data to files on the main thread can cause performance issues, especially in real-time applications.
- By using asynchronous saving (FlushStringBuildersCoroutine), the system ensures smooth operation without freezing the game.

Summary

The system efficiently scans and saves a 3D point cloud in Unity, leveraging structured storage and visualization. It is designed for real-time data collection, making it suitable for AR applications, simulations, and research projects. The use of asynchronous saving and modular components ensures high performance and flexibility.
