Document

1. Introduction

This code repository implements a special level of detail (LOD) method as part of a prediction framework. The method aims to reduce computational complexity through efficient subsampling and neighbor search.

2. File Structure

- **LOD4EED.h**: Contains declarations and related definitions for the special LOD method.
- **PCCTMC3Common.h**: Provides common PCC types and function definitions.
- **PCCMath.h**: Includes mathematical functions and data structure definitions.
- **PCCPointSet.h**: Defines the point cloud data structure and related functions.
- **constants.h**: Contains constant definitions.
- **hls.h**: Includes definitions for high-level synthesis (HLS) functionality.
- **nanoflann.hpp**: Includes the nanoflann library for fast k-nearest neighbor search.
- **Other System Header Files**: Used for standard C++ library and dependencies.

3. Main Functions

3.1 `subsampleForEED`

```
void subsampleForEED(
  const AttributeParameterSet& aps,
  const AttributeBrickHeader& abh,
  const PCCPointSet3& pointCloud,
  const std::vector<MortonCodeWithIndex>& packedVoxel,
  const std::vector<uint32_t>& input,
  const int32_t lodIndex,
  std::vector<uint32_t>& retained,
  std::vector<uint32_t>& indexes,
  MortonIndexMap3d& atlas);
```

- **Description**: Implements the special LOD method to reduce complexity through subsampling. This function use a MortonIndexMap3d structure to storage positions of points in local region and update them. In this way, the complexity will be reduced from n^2 to n. This function splite the point indexes in input into "retained" and "indexes", enabling each point in retained has at least one neighbour in indexes.
- **Parameters**:
  - `aps`: Attribute parameter set.
  - `abh`: Attribute brick header.
  - `pointCloud`: Input point cloud data.
  - `packedVoxel`: Point cloud data with Morton codes and indices.
  - `input`: Indices of input points.
  - `lodIndex`: Level of detail.
  - `retained`: Indices of retained points.
  - `indexes`: Final indices of subsampled points.
  - `atlas`: Data structure for storing Morton codes and updates in a local region.

3.2 `computeNearestEEDNeighbors`

```
void computeNearestEEDNeighbors(
  const AttributeParameterSet& aps,
  const AttributeBrickHeader& abh,
  const std::vector<MortonCodeWithIndex>& packedVoxel,
  std::vector<uint32_t>& indexes,
  std::vector<EEDPCCPredictor>& EEDpredictors,
  std::vector<uint32_t>& pointIndexToEEDPredictorIndex,
  int32_t pointCount);
```

- **Description**: Computes potential 18 neighbors for each point in the point cloud.
- **Parameters**:
  - `aps`: Attribute parameter set.
  - `abh`: Attribute brick header.
  - `packedVoxel`: Point cloud data with Morton codes and indices.
  - `indexes`: Indices of input point cloud.
  - `EEDpredictors`: Structure storing EED predictor information.
  - `pointIndexToEEDPredictorIndex`: Mapping from point index to EED predictor index.
  - `pointCount`: Number of points in the point cloud.

3.3 `buildPredictorsFastForEED`

```
void buildPredictorsFastForEED(
  const AttributeParameterSet& aps,
  const AttributeBrickHeader& abh,
  const PCCPointSet3& pointCloud,
  int32_t minGeomNodeSizeLog2,
  int geom_num_points_minus1,
  std::vector<PCCPredictor>& predictors,
  std::vector<uint32_t>& numberOfPointsPerLevelOfDetail,
  std::vector<uint32_t>& indexes,
  std::vector<EEDPCCPredictor>& EEDpredictors);
```

- **Description**: Builds EED predictors through subsampling. This function use `subsampleForEED` to divide layers and prepare `PCCPredictor` for the first layer
- **Parameters**:
  - `aps`: Attribute parameter set.
  - `abh`: Attribute brick header.
  - `pointCloud`: Input point cloud data.
  - `minGeomNodeSizeLog2`: Logarithm of the minimum geometric node size.
  - `geom_num_points_minus1`: Number of points in the geometric node.
  - `predictors`: Structure storing PCC predictor information.
  - `numberOfPointsPerLevelOfDetail`: Number of points for each detail level.
  - `indexes`: Indices of subsampled points.
  - `EEDpredictors`: Structure storing EED predictor information.

3.4 `createGussianWeight`

```
void createGussianWeight(float& sigma, std::vector<float>& neighboursw);
```

- **Description**: Creates Gaussian weights for Gaussian filtering of the point cloud.
- **Parameters**:
  - `sigma`: Standard deviation of the Gaussian distribution.
  - `neighboursw`: Vector storing Gaussian weights.

3.5 `AtlasIndexMaps3D`

```
// This is a structure used to help predictor search neighbors.
// It divides an ordered index sequence into segments and records the location of
the last lookup.
class AtlasIndexMaps3D {
public:
  // Structure to store start, search, and end indices.
  struct StartAndSearchIndex {
    int32_t startindex;
    int32_t searchindex;
    int32_t endindex;
  };

  // Insert a new entry with the given Morton code and index.
  void inseart(int64_t mortonCode, int32_t index);

  // Get the start index for a given Morton code.
  int32_t getStartIndex(int64_t mortonCode);

  // Get the search index for a given Morton code.
  int32_t getSearchIndex(int64_t mortonCode);

  // Get the end index for a given Morton code.
  int32_t getEndIndex(int64_t mortonCode);

  // Set the search index for a given Morton code.
  void setSearchIndex(int64_t mortonCode, int32_t index);

  // Set the end index for a given Morton code.
  void setEndIndex(int64_t mortonCode, int32_t index);

  // Reserve space in the unordered_map.
  void reserve(const uint32_t sz) ;

private:
  // Unordered map to store Morton codes and associated indices.
  std::unordered_map<int64_t, StartAndSearchIndex> SearchMap;
};
```