```
// MeshData.h: MeshData クラスのインターフェイス
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MESHDATA_H__958F025D_682B_4971_A477_DC00A2379F28__INCLUDED_)
#define AFX_MESHDATA_H__958F025D_682B_4971_A477_DC00A2379F28__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "math.h"
#include "Node.h"

class MeshData
{
public:
/*
	class Edges{
		int index1;
		int index2;
		Edges* next;
	}
*/
	//The number of vertex and face (File data)
	int face_N, vertex_N;

	//Vertex coordinates
	float (*vertex)[3];
	//Face index
	int (*face)[3];

	//Vertex link data (vertex to 1-ring neighbor verticse)
	//This data is not sorted (random order)
	Node** vertex_link;

	//Face link data (3 faces, linked along common edge)
	int (*face_link_E)[3];

	//Face link data (# = face_ad+N, linkd at common vertex)
	int *face_link_V_N;
	int **face_link_V;

	//Is the vertex specified a index boundary?
	//value = 0 if not boundary
	//velue = index of adjacent boundary face
	int *isBound;

	//Is the vertex specified a index ridge or ravine?
	BOOL *isRidge, *isRavine;

	//Is the face specified a index ridge or ravine?
	BOOL *ridge_tri, *ravine_tri;

	//Vertex normal
	float (*normal)[3];
```

```
    //Face normal
    float (*normal_f)[3];

    //Mean curvature, median, varience
    float *K, midK, varience;

    //principal curvature
    float *k_max, *k_min;
    //principal direction
    float (*t_max)[3], (*t_min)[3];

    //ridge and ravine direction
    float (*ridge_dir)[3], (*ravine_dir)[3];

    //ridge and ravines
    //edge* edge_list;

public:
    void getConsistentNRings(int index, int n, int **&nei, int *&nei_N);
    float getVolume();
    int previousFace(int face_id, int vertex_id);
    int nextVertex(int face_id, int vertex_id);
    int nextFace(int face_id, int vertex_id);
    void getConsistent1Ring(int index, int *&nei, int &nei_N);
    void computePrincipal();
    void meanCurvatureNormal(int index, float* Hn);
    void laplacian(int index, float *L);
    void computeNormal();
    void computeFaceNormal();
    void get1Ring(int index, int *&nei, int &nei_N);
    //Generate face link (face_link_E)
    void generateFaceLink();
    //Generate vertex link (vertex_link)
    void generateVertexLink();

    //Count valid # face
    int countValidFace();
    //Count valid # vertex (eliminated vertex is not included)
    int countValidVertex();

    //Scale mesh size into 20x20x20 cube
    void normalizeScale();

    //get & set data (These data can be acessed directly, so no need)
    void getFace(int index, int vertex_index[3]);
    void getVertex(int index, float coordinates[3]);
    void setFace(int index, int vertex_index[3]);
    void setVertex(int index, float coordinates[3]);
    int getFaceCount();
    int getVertexCount();
    void setFaceCount(int face_N);
    void setVertexCount(int vertex_N);

    MeshData();
    virtual ~MeshData();
```

```
    static inline float MeshData::LENGTH(float v[3]){
        return (float)sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    }

    static inline void MeshData::VEC(float v[3], float p1[3], float p2[3]){
        v[0] = p2[0]-p1[0];
        v[1] = p2[1]-p1[1];
        v[2] = p2[2]-p1[2];
    }

    static inline void MeshData::TIMES(float kv[3], float k, float v[3]){
        kv[0] = k*v[0];
        kv[1] = k*v[1];
        kv[2] = k*v[2];
    }

    static inline float MeshData::DIST(float p1[3], float p2[3]){
        float v[3];
        VEC(v,p1,p2);
        return LENGTH(v);
    }

    static inline float MeshData::DOT(float v1[3], float v2[3]){
        return v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2];
    }

    static inline void MeshData::CROSS(float n[3], float v1[3], float v2[3]){
        n[0] = v1[1]*v2[2] - v1[2]*v2[1];
        n[1] = v1[2]*v2[0] - v1[0]*v2[2];
        n[2] = v1[0]*v2[1] - v1[1]*v2[0];
    }

    static inline float MeshData::AREA(float p1[3], float p2[3], float p3[3])
    {
        float n[3];
        n[0] = (p2[1]-p1[1])*(p3[2]-p1[2])-(p2[2]-p1[2])*(p3[1]-p1[1]);
        n[1] = (p2[2]-p1[2])*(p3[0]-p1[0])-(p2[0]-p1[0])*(p3[2]-p1[2]);
        n[2] = (p2[0]-p1[0])*(p3[1]-p1[1])-(p2[1]-p1[1])*(p3[0]-p1[0]);

        return 0.5f*LENGTH(n);
    }
};

#endif // !defined(AFX_MESHDATA_H__958F025D_682B_4971_A477_DC00A2379F28__INCLU
DED_)
```