期中报告

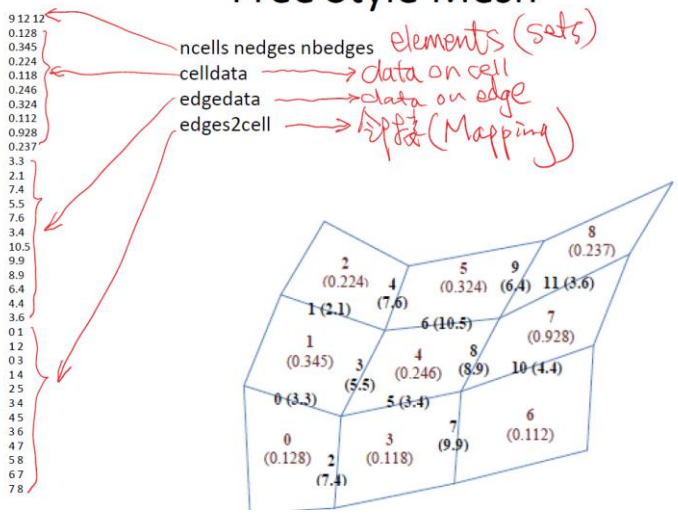
19335015 陈恩婷

Mesh01-03 读取和写入 Mesh

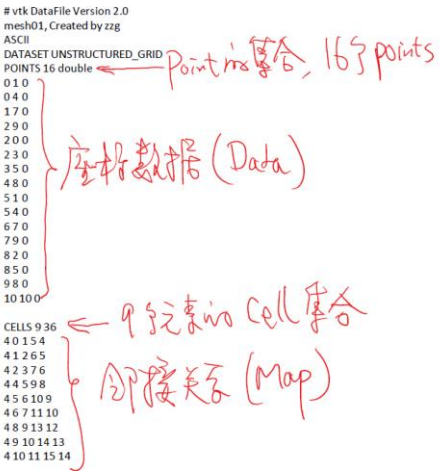
1. 项目目的

实现 Free Style Mesh 与 VTK Style Mesh 从文件的读入和写入文件，主要格式如下：

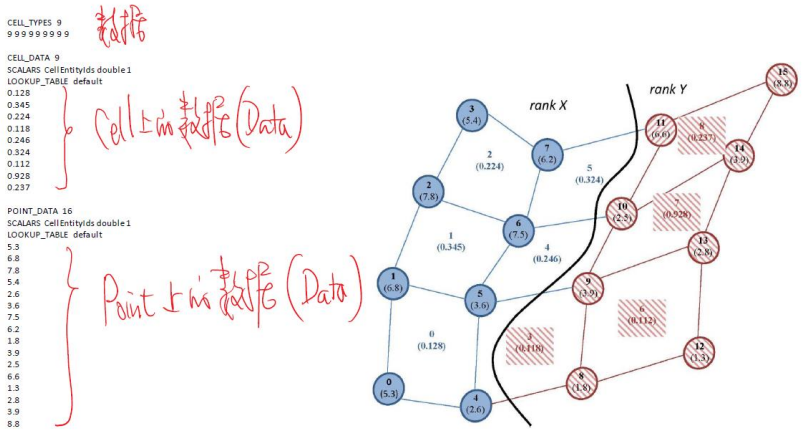
Free Style Mesh



VTK Mesh



VTK Mesh



2. 为实现目的存在的各种技术问题

动态分配内存、C 语言读取文件等。

3. 用什么算法、数据结构、语言机制解决这些问题

数据结构：本次实验中 mesh 中的数据结构采用数组存储，Mesh 01 实验中，

Mesh2DQuad 类中的数据成员包括：

```
int  nnode, ncell, nedge, nbedge;
int* becell, * ecell, * bound, * bedge, * edge, * cell, *cell2edge;
real* x;
```

Mesh 02实验中包括：

```
int  npoint, ncell,cellsize;
int* cells, *celltype;
double* xyz;
int * q;
```

注意这里的数组为指针形式，需要为其动态分配内存。

语言机制：主要使用 malloc、fprintf 和 fscanf 等库函数。

malloc

Defined in header <stdlib.h>

```
void* malloc( size_t size );
```

fprintf

```
int fprintf ( FILE * stream, const char * format, ... );
```

fscanf

```
int fscanf ( FILE * stream, const char * format, ... );
```

fscanf 和 fprint 我在之前并不是很熟悉，不过它的主要语法和 printf、scanf 类似，比较容易上手。

4. 对应的程序框架和实现代码

参考 airfoil_seq.cpp，根据头文件中的声明，补充相关函数的实现。

在 readHeader 函数中，直接读入 nnode, ncell, nedge, nbedge 等数据，其余的数组数据成员则根据这些数据进行预分配内存和初始化。

在其余的读入函数中（写入函数同理），使用 for 循环分别完成以下数组的读入或初始化：

- readNode 函数：x
- readCell 函数：cell
- readEdge 函数：edge，ecell，cell2edge
- readBEdge：bedge

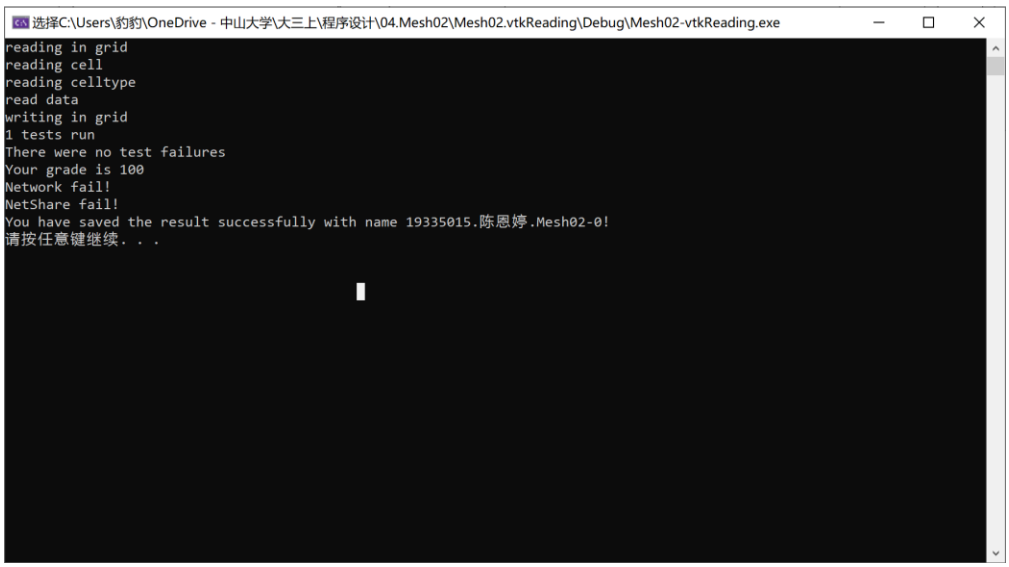
核心代码如下：

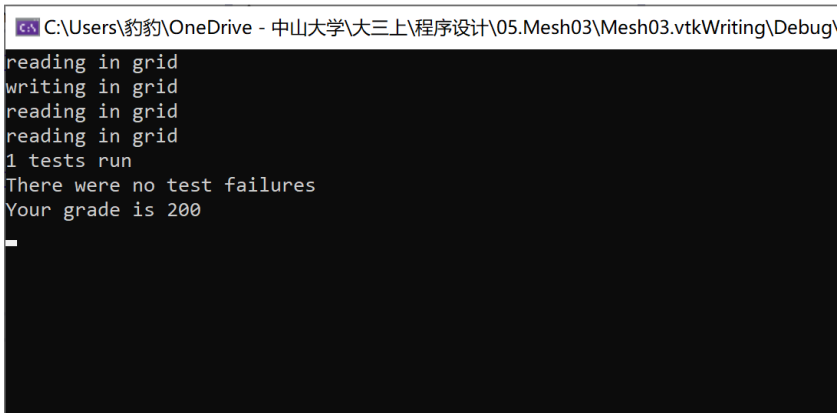
```
bool Mesh2Dquad::read(const char* fileName) {
    printf("reading in grid \n");

    FILE* fp;
    if ((fp = fopen(fileName, "r")) == NULL) {
        printf("can't open file\n");
        return 0;
    }
    readHeader(fp);
    init();
    readNode(fp);
    readCell(fp);
    readEdge(fp);
    readBEdge(fp);
    fclose(fp);
```

```
    return true;
}
bool Mesh2Dquad::readHeader(FILE* fp) {
    if ( fscanf( fp, "%d %d %d %d \n", &nnode, &ncell, &nedge,
&nbedge ) != 4 ) {
        printf( "error reading from new_grid.dat\n" );
        exit( -1 );
    }
    cell = (int*)malloc( 4 * ncell * sizeof( int ) );
    edge = (int*)malloc( 2 * nedge * sizeof( int ) );
    ecell = (int*)malloc( 2 * nedge * sizeof( int ) );
    bedge = (int*)malloc( 2 * nbedge * sizeof( int ) );
    becell = (int*)malloc( nbedge * sizeof( int ) );
    bound = (int*)malloc( nbedge * sizeof( int ) );

    x = (real*)malloc( 2 * nnode * sizeof( real ) );
    cell2edge = (int*)malloc( 4 * ncell * sizeof( int ) );
    for ( int i = 0; i < 4 * ncell; ++i ) {
        cell2edge[ i ] = -1;
    }
    return 1;
}
bool Mesh2Dquad::readNode(FILE* fp) {
    for ( int n = 0; n < nnode; n++ ) {
        if ( fscanf( fp, "%lf %lf \n", &x[ 2 * n ], &x[ 2 * n + 1 ] ) !=
2 ) {
            printf( "error reading from new_grid.dat\n" );
            exit( -1 );
        }
    }
    return 1;
}
```

5. 实验结果和结论

运行结果如下：





如图所示，成功完成了 Mesh 结构的读入和写入。

6. 可能的各种可选方法的对比

1. 不封装函数，直接将所有代码放入 read 函数中

   这种方法可能会导致代码可读性下降，框架不明显。

2. 使用 C++中的 fstream 进行文件读取和写入

   个人觉得如果 mesh 存储于文本文件之中，这种方法会比使用 C 语言中的 FILE *要简洁，但是限于原有代码的框架没有尝试。

7. 遇到的问题和解决办法

1. 读入文件时，同一行既有字符串又有整数会导致读入失败

   解决方法：仔细检查发现，由于文件中有空行，读取时需要跳过这些空行，否则实

际读到的与预想的不是同一行，就会读不到想要的数据。

2. 写入文件时，cells 数组每个数字占一行，没有像原文件一样一个 cell 占一行

   解决方法：celltype 中存储了每个 cell 是哪个类型，1 表示 cell 由两个 point 构成，3 表示由三个构成，5 表示由 4 个构成。根据这些信息。可以按照原文件的格式写入 cells 数组中的信息。

3. 将文件中 mesh 的名字读到 dataset 字符串中时，由于文件中 mesh 的名字后面紧跟着一个逗号，读取时会将逗号也读进去

   解决方法：参考 stackoverflow 上面的回答，使用%[^,]就可以解决了

   （https://stackoverflow.com/questions/16014859/sscanf-until-it-reaches-a-comma）

   If the order of your variables in the string is fixe, I mean It's always:

   ```
   string, string, string, int, float
   ```

   the use the following format specifier in `sscanf()`:

   ```
   int len = strlen(str);
   char a[len];
   char b[len];
   char c[len];
   unsigned long d;
   float e;

   sscanf(" %[^,] , %[^,] , %[^,] , %lu , %lf", a, b, c, &d, &e);
   ```

Mesh 04 从 VTK 文件中恢复数据结构

1. 项目目的

参考 mesh01-kernels 项目，完成 mesh04-vtkAlg 项目，要求重新构建数据结构，相关课件截图如下：

- 任务是构造核函数
- 从一个定义核函数的沅文件转换到VtK项目

- 参考的沅项目： mesh01-kernels 基于自由式的mesh.
- 今天编程项目: mesh04-vtkAlg 基于vtk格式的mesh.

从 Mesh04.vtkAlgorithm 目录下的 meshvtkAlg.sln 开始启动程序.

- 注意两个项目Mesh结构数据不同
- 所以在工作项目中新定义的核函数可能需要新的数结结构
- 两种做法
  - 从VTK恢复所需的数据结构
  - 重新定义自己的数据结构或重新修改核函数的代码

2. 为实现目的存在的各种技术问题

如何遍历所有的边

如何确定一条边是否边界边

将这些信息存储于什么临时数据结构

如何将临时数据结构中的信息写入 mesh 中的相关数组

3. 用什么算法、数据结构、语言机制解决这些问题

1. 算法:

for cells 数组中所有 cell 的所有边

if 该边在两个 cell 中都有出现

将它存入 edge 数组

将其所属的两个 cell 的编号存入 ecell 数组

if 该边只出现在一个 cell 中

将它存入 bedge 数组

将其所属的 cell 存入 becell 数组

将其所属的边界的编号存入 bound 数组。

2. 数据结构：

在具体实现中，本人使用了 std::map 数据结构来维护已经遍历过的边，以此来检查一条边是否已经在之前遍历其他 cell 时遍历过。

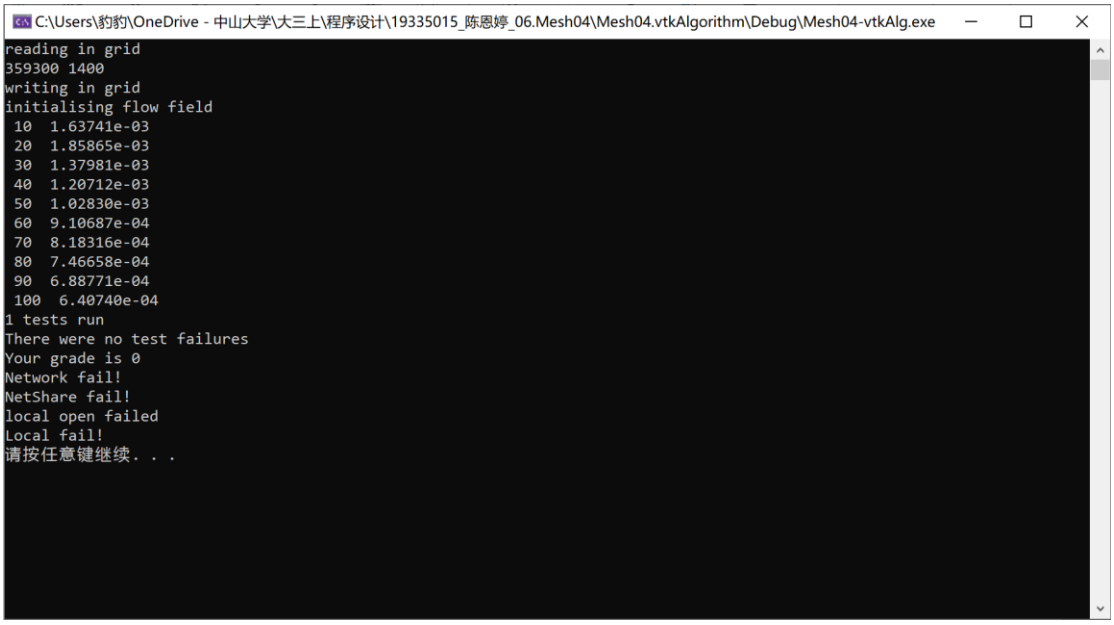
4. 对应的程序框架和实现代码

Read 函数结构和之前几次实验大体相同，主要不同在于调用的 Init 函数：

```
void MeshVTK::init() {
    //cout << ncell << endl;
    map<pair<int, int>, int> bedge_map;
    map<pair<int, int>, pair<int, int>> edge_map;
    pair<int, int> p, p1, p2;
    for ( int i = 0; i < ncell; i++ ) {
        for ( int j = 0; j < 4; j++ ) {
            p2.second = p.first = cells[ 4 * i + j % 4 ];
            p2.first = p.second = cells[ 4 * i + ( j + 1 ) % 4 ];
            if ( bedge_map.find( p ) != bedge_map.end() ) {
                p1.first = bedge_map[ p ];
                bedge_map.erase( p );
                p1.second = i;
                edge_map.insert( make_pair( p, p1 ) );
            }
            else if ( bedge_map.find( p2 ) != bedge_map.end() ) {
                p1.first = bedge_map[ p2 ];
                bedge_map.erase( p2 );
                p1.second = i;
                edge_map.insert( make_pair( p, p1 ) );
            }
            else {
                bool flag = false;
                for (auto& e : bedge_map) {
                    double a = xyz[3 * e.first.first];
                    double b = xyz[3 * e.first.first + 1];
                    double c = xyz[3 * p.first];
                    double d = xyz[3 * p.first + 1];
```

```
                    double e1 = xyz[3 * e.first.second ];
                    double f = xyz[3 * e.first.second + 1];
                    double g = xyz[3 * p.second];
                    double h = xyz[3 * p.second + 1];
                    if ( (make_pair(a, b) == make_pair(c, d) &&
make_pair(e1, f) == make_pair(g, h) )
                        || (make_pair(a, b) == make_pair(g, h) &&
make_pair(e1, f) == make_pair(c, d)) ){
                            p1.first = bedge_map[e.first];
                            bedge_map.erase(e.first);
                            p1.second = i;
                            edge_map.insert(make_pair(p, p1));
                            flag = true;
                            break;
                    }
                }
                if (!flag) {
                    bedge_map.insert( make_pair( p2, i ) );
                }
            }
        }
    }
```

5. 实验结果和结论



程序的最终输出结果如图所示。仔细对比了示例程序和工作程序的代码和输出后发现引起错误的主要原因是边界边的 bound 数组所存数据有误导致的，修改代码后，程序输出了正确结果。

6. 可能的各种可选方法的对比

两种做法

1. 从 VTK 恢复所需的数据结构

   这种思路比较简单直接，利用 std::map 和 std::set 按部就班完成即可，难度较低。

2. 重新定义自己的数据结构或重新修改核函数的代码

   这个比较复杂，要涉及到重新设计数据结构，或者按照原有核函数的算法重新编写代码，难度比较大，没有想到比较好的思路所以没有尝试。

7. 遇到的问题和解决办法

1. 实现过程中，测试运行时经常出现内存错误

   解决方案：这些大多是没有正确分配或者索引数组引起的。利用 Visual Studio 的调试功能，检查具体是哪一处内存出错，修改相关代码即可。

Mesh 05-06 Mesh 的存储

1. 项目目的

根据课件和程序中给出的函数原型和提示，完成 Elements、Map 和 Data 的创建函数，并实现 mesh 在二进制文件中的存储与读取。相关课件截图如下：

- Define In-memory mesh
  - Elements
  - Map
  - Data
- Related Functions
  - Elements makeElements(int, char const*);
  - Map makeMap(Elements, Elements, int, int*, char const*);
  - Data makeData(Elements, int, char const*, char*, char const*);

同时要将构造基础结构加入对应的测试中

Notice

- You may change or modify any part of the code
- You have to make sure that your code is consistent in logic.
- You may not change the test logic I have put in code.

工程中原有的代码可以根据实际情况进行改动。

2. 为实现目的存在的各种技术问题

理解清楚 Elements，Maps 和 Data 结构体中各数据成员的具体含义

如何读写二进制文件

3. 用什么算法、数据结构、语言机制解决这些问题

数据结构：Elements，Maps 和 Data 结构体

语言机制：MeshMemory 类中构造各结构体的成员函数

function

fopen

```
FILE * fopen ( const char * filename, const char * mode );
```

C string containing a file access mode. It can be:

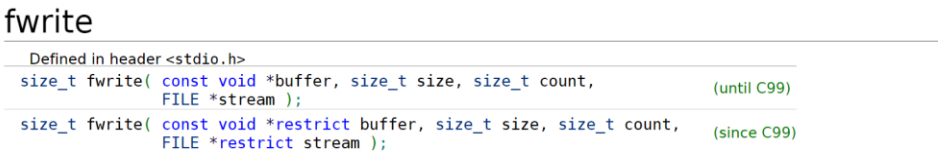
"r"	**read:** Open file for input operations. The file must exist.
"w"	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist.
"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
"a+"	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist.

With the *mode* specifiers above the file is open as a *text file*. In order to open a file as a *binary file*, a "b" character has to be included in the *mode* string. This additional "b" character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the "+" sign for the mixed modes ("rb+", "wb+", "ab+").

fread

Defined in header <stdio.h>

```
size_t fread( void          *buffer, size_t size, size_t count,
              FILE          *stream );                              (until C99)
size_t fread( void *restrict buffer, size_t size, size_t count,
              FILE *restrict stream );                              (since C99)
```



4. 对应的程序框架和实现代码

三个构造结构体的函数：

```
Elements MeshMemory::makeElements(int ele_size, char const* ele_name) {
    Elements ele = (Elements)malloc(sizeof(elements));
    ele->name = ele_name;
    ele->size = ele_size;
    ele->index = element_list_index;
    element_list[element_list_index++] = ele;
    return ele;
}
Map MeshMemory::makeMap(Elements map_from, Elements map_to, int
map_dim, int* map_map, char const* map_name) {
    printf("hello\n");
    int arr_cnt = map_from->size;
    Map mapi = (Map)malloc(sizeof(map));
    mapi->index = map_list_index;
    mapi->from = map_from; mapi->to = map_to;
    mapi->dim = map_dim; mapi->name = map_name;
    mapi->map = map_map;
    map_list[map_list_index++] = (Map)mapi;
    return (Map)mapi;
}
Data MeshMemory::makeData(Elements data_set, int data_dim, char const*
data_type, char* data_data, char const* data_name) {
    Data data = (Data)malloc(sizeof(dat));
    data->index = dat_list_index;
    data->set = data_set;
    data->dim = data_dim;
    data->size = 8;
    data->data = data_data;
    data->name = data_name;
    dat_list[dat_list_index++] = data;
    return data;
}
```

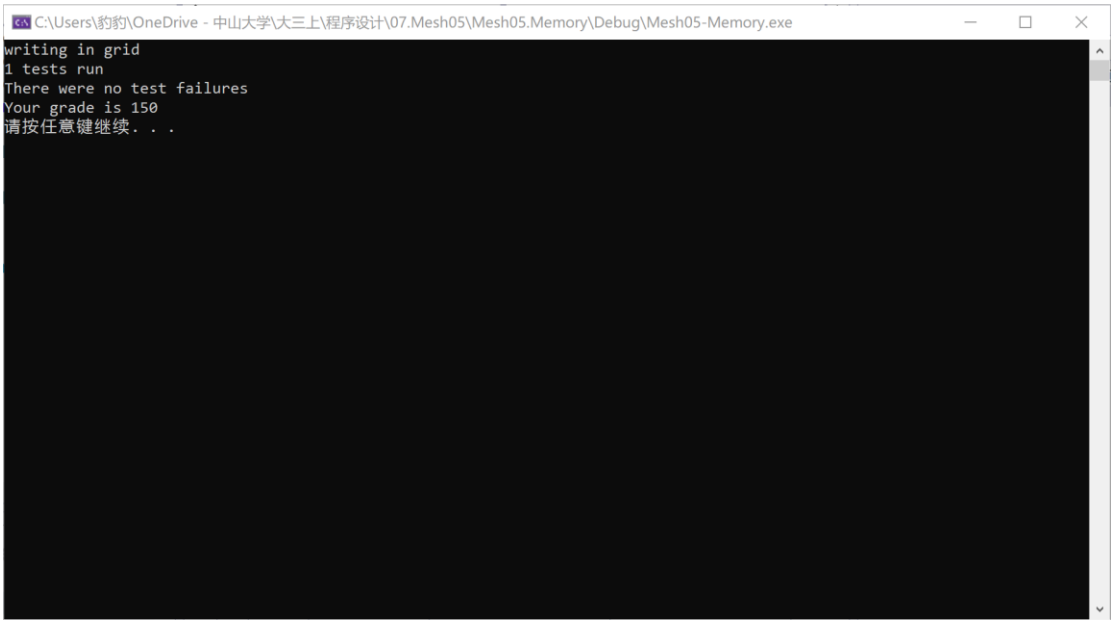
从二进制文件中读取 mesh 的代码：

```
bool MeshMemory::readfromfile(const char* fileName) {
    FILE* fp;
```

```
    if ((fp = fopen(fileName, "rb")) == NULL) {
        printf("can't open file\n");
        return 0;
    }

    readHeader(fp);
    readElements(fp);
    readMaps(fp);
    readData(fp);

    fclose(fp);
    return true;
}

bool MeshMemory::readHeader(FILE* fp) {
    fscanf(fp, "#emd %d %d %d %d %d %d\n", &element_list_size,
&map_list_size, &dat_list_size,
        &element_list_index, &map_list_index, &dat_list_index);
    return 1;
}

bool MeshMemory::readElements(FILE* fp) {
    fread(element_list, sizeof(elements), element_list_index, fp);
    return 1;
}

bool MeshMemory::readMaps(FILE* fp) {
    fread(map_list, sizeof(map), map_list_index, fp);
    for (int i = 0; i < map_list_index; i++) {
        //fwrite(map_list[i], sizeof(map), 1, fp);
        Map map = map_list[i];
        map->map = (int*)malloc(sizeof(int) * map->from->size *
map->dim);
        fread(map->map, sizeof(int), map->from->size * map->dim, fp);
    }
    return 1;
}
bool MeshMemory::readData(FILE* fp) {
    fread(dat_list, sizeof(dat), dat_list_index, fp);
    for (int i = 0; i < dat_list_index; i++) {
        //fwrite(dat_list[i], sizeof(dat), 1, fp);
        Data d = dat_list[i];
        d->data = (char*)malloc(sizeof(double) * d->dim * d->set->size);
        fread(d->data, sizeof(double), d->dim * d->set->size, fp);
```

```
    }
    double* pd = (double*)(dat_list[3]->data);
    return 1;
}
```

5. 实验结果和结论



如图所示，成功完成了 mesh 的读写和存储。

6. 可能的各种可选方法的对比

makeMap 中，可以选择申请 arr_cnt 个 map 的空间，将数据分别存储与各个 map 中，

或者只申请一个 map 的空间，将数据存储于它的 map 数据成员中。如下所示：

```
Map makeMap(Elements map_from, Elements map_to, int map_dim, int* map_map,
char const* map_name) {
 int arr_cnt = map_from->size;
 Map mapi = (Map)malloc(sizeof(map) );
 mapi->index = map_list_index;
 mapi->from = map_from; mapi->to = map_to;
 mapi->dim = map_dim;   mapi->map = map_map;
 mapi->name = map_name;
 map_list[map_list_index++] = (Map)mapi;
 return (Map)mapi;
}
Map makeMap(Elements map_from, Elements map_to, int map_dim, int* map_map,
char const* map_name) {
 int arr_cnt = map_from->size;
 Map mapi = (Map)malloc(sizeof(map) *arr_cnt);
 for (int j= 0; j< arr_cnt; j++) {
     mapi[j].index = map_list_index;    mapi[j].from = map_from;
     mapi[j].to = map_to;       mapi[j].dim = map_dim;
     for (int i = 0; i < map_dim; i++)
         mapi[j].map[i] = map_map[j * map_dim+i];
     mapi->name = map_name;
 }
 map_list[map_list_index++] = (Map)mapi;
 return (Map)mapi;
}
```

个人认为只申请一个 map 的方法比较好，因为这样节省了存储空间，也简化了申请和释放内存的工作量。