ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computational Engineering

Spring semester 2014

Prof. P. Koumoutsakos, G, Tauriello
ETH Zentrum, CLT F 12, C 11
CH-8092 Zürich

Solution 8

Issued: 20.5.2014

In this set of exercises, you work with the code you developed in the previous set of exercises and you reshape the code in order to use templates. Later on, you improve your background on C++ STL.

Question 1: Generic P2M method

As you have noticed in Set 7, every particle-to-mesh interpolation kernel (Λ_2 and M_4' kernels) was a class with its own implementation of the `p2m` method. The main idea behind this exercise is to guarantee a minimal amount of redundant code which exists in Set 7 between the two `p2m` member methods.

In this exercise, you will have create a generic standalone method, called `p2m`, which is not part of any class and is templatized over the "particle-to-mesh interpolation kernel type".

There are at least two possible ways to implement this code. In one, an object of particle-to-mesh interpolation kernel type can be passed to the generic method `p2m` as an argument. Therefore, before calling the `p2m` methods, this object must be instantiated. This approach requires minimal change to the solution code of Set 7.

In the other approach, all the kernel-specific constant member variables e.g. `support_start` – which must not change – and methods e.g. `kernel` – which does not depend on the state of the class – can be made static. No object of the kernel type is therefore needed and these members are accessible inside `p2m` by using `::` operator.

For each approach, we provide the code of the updated `Lambda2` class and then we show the implementation of the templatized `p2m` method and the `TestP2M` function which demonstrates the usage of the method. The code changes to the `MP4` class are similar.

The first approach, which passes an object of particle-to-mesh interpolation kernel type to the generic method `p2m`, is as follows:

```
1  class Lambda2
2  {
3  //private:
4  public:
5      const int support_start;
6      const int support_end;
7      const int support;
8
9      double kernel(double x)
10     {
11         double absx = fabs(x);
12
13         if (absx<1.5 && absx>=.5)
```

```
14              return .5*(2.-absx)*(1.-absx);
15          else if (absx<.5)
16              return 1. - absx*absx;
17          else // outside of support
18              return 0.;
19      }
20
21  public:
22      // Constructor
23      Lambda2()
24      : support_start(-1), support_end(2), support(support_end-
            support_start)
25      {
26      }
27
28      void get_neighbors(const double px, const double py, int &idx, int &
            idy)
29      {
30          idx = (px-floor(px))<.5 ? (int)floor(px) : (int)ceil(px);
31          idy = (py-floor(py))<.5 ? (int)floor(py) : (int)ceil(py);
32      }
33
34  };
35
36  class MP4
37  {
38  //private:
39  public:
40      const int support_start;
41      const int support_end;
42      const int support;
43
44      double kernel(double x)
45      {
46          // kernel for the interpolation
47          double absx = fabs(x);
48
49          if (absx<2. && absx>=1.) {
50              return .5*(2.-absx)*(2.-absx)*(1.-absx);
51          } else if (absx<1.) {
52              return 1. - 2.5*absx*absx + 1.5*absx*absx*absx;
53          } else { // outside of support
54              return 0.;
55          }
56      }
57
58  public:
59      // Constructor
60      MP4(): support_start(-1), support_end(3), support(support_end-
            support_start)
61      {
62      }
63
64      void get_neighbors(const double px, const double py, int &idx, int &
            idy)
65      {
```

```
66          idx = (int)floor(px);
67          idy = (int)floor(py);
68      }
69
70  };
71
72
73  // perform 2 dimensional P2M of mass contained in particles to the mesh
74  template<typename TKernel>
75  void p2m(TKernel &k, vector<Particle>& particles, Grid2D& mesh)
76  {
77      const double h = mesh.getH();
78
79      // since we are accumulating, we have to make sure the mesh is empty
80      mesh.clear();
81
82      for (int p=0; p<particles.size(); p++)
83      {
84          // find the two closest mesh point indices
85          const double px = particles[p].x/h;
86          const double py = particles[p].y/h;
87
88          int idx, idy;
89          k.get_neighbors(px, py, idx, idy);
90
91          // compute weights
92          double wx[k.support], wy[k.support];
93          for (int i=k.support_start; i<k.support_end; i++)
94          {
95              wx[i-k.support_start] = k.kernel(px-(idx+i));
96              wy[i-k.support_start] = k.kernel(py-(idy+i));
97          }
98
99          // distribute mass
100         for (int j=k.support_start; j<k.support_end; j++)
101             for (int i=k.support_start; i<k.support_end; i++)
102                 mesh(idx+i,idy+j) += wx[i-k.support_start]*wy[j-k.
                        support_start]*particles[p].m;
103     }
104 }
```

The second approach, which uses static member variables and methods, is implemented as follows:

```
1   class Lambda2
2   {
3   public:
4       static const int support_start = -1;
5       static const int support_end = 2;
6
7       static double kernel(double x)
8       {
9           double absx = fabs(x);
10
11          if (absx<1.5 && absx>=.5)
12              return .5*(2.-absx)*(1.-absx);
13          else if (absx<.5)
```

```
14              return 1. - absx*absx;
15          else // outside of support
16              return 0.;
17      }
18
19      static void get_neighbors(const double px, const double py, int &idx,
            int &idy)
20      {
21          idx = (px-floor(px))<.5 ? (int)floor(px) : (int)ceil(px);
22          idy = (py-floor(py))<.5 ? (int)floor(py) : (int)ceil(py);
23      }
24  };
25
26  class MP4
27  {
28  public:
29      static const int support_start = -1;
30      static const int support_end = 3;
31
32      static double kernel(double x)
33      {
34          // kernel for the interpolation
35          double absx = fabs(x);
36
37          if (absx<2. && absx>=1.) {
38              return .5*(2.-absx)*(2.-absx)*(1.-absx);
39          } else if (absx<1.) {
40              return 1. - 2.5*absx*absx + 1.5*absx*absx*absx;
41          } else { // outside of support
42              return 0.;
43          }
44      }
45
46      static void get_neighbors(const double px, const double py, int &idx,
            int &idy)
47      {
48          idx = (int)floor(px);
49          idy = (int)floor(py);
50      }
51  };
52
53  // perform 2 dimensional P2M of mass contained in particles to the mesh
54  template<typename TKernel>
55  void p2m(vector<Particle>& particles, Grid2D& mesh)
56  {
57      const int support = TKernel::support_end - TKernel::support_start;
58
59      const double h = mesh.getH();
60
61      // since we are accumulating, we have to make sure the mesh is empty
62      mesh.clear();
63
64      for (int p=0; p<particles.size(); p++)
65      {
66          // find the two closest mesh point indices
67          const double px = particles[p].x/h;
```

```
68          const double py = particles[p].y/h;
69
70          int idx, idy;
71          TKernel::get_neighbors(px, py, idx, idy);
72
73          // compute weights
74          double wx[support], wy[support];
75          for (int i=TKernel::support_start; i<TKernel::support_end; i++)
76          {
77              wx[i-TKernel::support_start] = TKernel::kernel(px-(idx+i));
78              wy[i-TKernel::support_start] = TKernel::kernel(py-(idy+i));
79          }
80
81          // distribute mass
82          for (int j=TKernel::support_start; j<TKernel::support_end; j++)
83              for (int i=TKernel::support_start; i<TKernel::support_end; i
                ++)
84                  mesh(idx+i,idy+j) += wx[i-TKernel::support_start]*wy[j-
                    TKernel::support_start]*particles[p].m;
85      }
86  }
```

Question 2: C++ STL

Let us consider the following program:

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main (int argc, char * const argv[]) {
7      vector<int> v;
8
9      for (int i=0; i<10; ++i) v.push_back(i);
10
11     vector<int>::reverse_iterator it;
12     for (it = v.rbegin(); it != v.rend(); it++)
13         cout << *it << " ";
14
15     cout << endl;
16
17     return 0;
18 }
```

What is the output of the program?

The output is: 9 8 7 6 5 4 3 2 1 0.

Question 3: C++ STL

Given the following program:

```
1  #include <vector>
2  #include <iostream>
```

```
3
4  using namespace std;
5
6  int main()
7  {
8      vector<int> v(3,2);
9
10     for (int i=1; i<=3; i++)
11         v.push_back(i);
12
13     for (int i=0; i<v.size(); i++)
14         cout << v[i] << " ";
15
16     return 0;
17 }
```

a) What is the output?

2 2 2 1 2 3

b) Rewrite lines 13-14 so that it uses the iterator `vector<int>::iterator`.

```
1  vector<int>::iterator it;
2  for (it=v.begin(); it!=v.end(); it++)
3  cout << *it << " ";
```

Question 4: C++ STL and templates

The following code does not compile.

1. Fix the `print_vec` using templates.
2. What is the output?

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  void print_vec(const std::vector<int> &myvector)
6  {
7    for (std::vector<int>::const_iterator it=myvector.begin(); it!=myvector
       .end(); ++it)
8      std::cout << ' ' << *it;
9    std::cout << '\n';
10 }
11
12 int main () {
13
14   int myints[] = {32,71,53,45,26,80,54,33};
15
16   std::vector<int> myvector(myints, myints+8);
17
18   print_vec(myvector);
19
20   std::sort (myvector.begin()+2, myvector.end()-1);
21
```

```
22    print_vec(myvector);
23
24    std::vector<float> myvector2(2, 1.234);
25
26    print_vec(myvector2);
27
28    return 0;
29  }
```

1. Templatized `print_vec`:

```
1  template<typename TElement>
2  void print_vec(const std::vector<TElement> &myvector)
3  {
4    for (typename std::vector<TElement>::const_iterator it=myvector.
       begin(); it!=myvector.end(); ++it)
5      std::cout << ' ' << *it;
6    std::cout << '\n';
7  }
```

2. Output:
   32 71 53 45 26 80 54 33
   32 71 26 45 53 54 80 33
   1.234 1.234