```cpp
#pragma once
#define MAX_UINT8_VALUE 256
#define VECTOR_CALC 1
#define STATS_VERSION 1

#include <stdint.h>
#include <utility>
#include <vector>
#include <cstdlib>
#include <cmath>
#include <limits>
#include <typeinfo>
#include <string>

#include <fstream>

#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>

#include "MathException.h"
#include "SoilMathTypes.h"

using namespace std;

namespace SoilMath
{
    template <typename T1, typename T2, typename T3>
    class Stats
    {
    public:
        bool isDiscrete = true;

        T1 *Data;
        uint32_t *bins;
        bool Calculated = false;
        float Mean = 0.0;
        uint32_t n = 0;
        uint32_t noBins = 0;
        T1 Range;
        T1 min;
        T1 max;
```

```
42          T1 Startbin;
43          T1 EndBin;
44          T1 binRange;
45          float Std = 0.0;
46          T3 Sum = 0;
47          uint16_t Rows = 0;
48          uint16_t Cols = 0;
49
50          Stats(int noBins = 256, T1 startBin = 0, T1 endBin = 255)
51          {
52              min = numeric_limits<T1>::max();
53              max = numeric_limits<T1>::min();
54              Range = numeric_limits<T1>::max();
55              Startbin = startBin;
56              EndBin = endBin;
57              this->noBins = noBins;
58              bins = new uint32_t[noBins] {};
59
60              if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) || typeid(T1) == typeid(long double))
61              {
62                  isDiscrete = false;
63                  binRange = static_cast<T1>((EndBin - Startbin) / noBins);
64              }
65              else
66              {
67                  isDiscrete = true;
68                  binRange = static_cast<T1>(round((EndBin - Startbin) / noBins));
69              }
70          }
71
72          Stats(T1 *data, uint16_t rows, uint16_t cols, int noBins = 256, T1 startBin = 0, T1 endBin = 255)
73          {
74              min = numeric_limits<T1>::max();
75              max = numeric_limits<T1>::min();
76              Range = max - min;
77
78              Startbin = startBin;
79              EndBin = endBin;
80
81              if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) || typeid(T1) == typeid(long double))
82              {
```

```
83              isDiscrete = false;
84              binRange = static_cast<T1>((EndBin - Startbin) / noBins);
85          }
86          else
87          {
88              isDiscrete = true;
89              binRange = static_cast<T1>(round((EndBin - Startbin) / noBins));
90          }
91
92          Data = data;
93          Rows = rows;
94          Cols = cols;
95          bins = new uint32_t[noBins] {};
96          this->noBins = noBins;
97          if (isDiscrete) { BasicCalculate(); }
98          else { BasicCalculateFloat(); }
99      }
100
101     /// <summary>
102     /// Constructor when the data is given as an histogram
103     /// </summary>
104     /// <param name="binData">A histogram of [256] bins</param>
105     /// <param name="offset">offset when the data starts</param>
106     Stats(T2 *binData, uint16_t startC, uint16_t endC)
107     {
108         noBins = endC - startC;
109         Startbin = startC;
110         EndBin = endC;
111         uint32_t i = noBins;
112
113         if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) || typeid(T1) == typeid(long double))
114         {
115             isDiscrete = false;
116             throw Exception::MathException("Calculations using histogram not supported with floating-type!");
117         }
118         else
119         {
120             isDiscrete = true;
121             binRange = static_cast<T1>(round((EndBin - Startbin) / noBins));
122         }
123
```

```
124             bins = new uint32_t[noBins] {};
125             while (i-- > 0)
126             {
127                 bins[i] = binData[i];
128                 n += binData[i];
129             }
130             BinCalculations(startC, endC);
131         }
132
133         ~Stats() {};
134
135         void BasicCalculateFloat()
136         {
137             float sum_dev = 0.0;
138
139             // Make copy of the starting pointer
140             T1 *StartDataPointer = Data;
141
142             // Get number of samples
143             n = Rows * Cols;
144             uint32_t i = n;
145
146             // Get sum , min, max, fill histogram
147             for (uint32_t i = 0; i < n; i++)
148             {
149                 if (Data[i] > max) { max = Data[i]; }
150                 else if (Data[i] < min) { min = Data[i]; }
151                 Sum += Data[i];
152             }
153
154             //while (i-- > 0)
155             //{
156             //  if (*Data > max) { max = *Data; }
157             //  else if (*Data < min) { min = *Data; }
158             //  Sum += *Data++;
159             //}
160
161             binRange = (max - min) / noBins;
162             uint32_t index = 0;
163             T1 shift = -min;
164
165             i = n - 1;
```

```
166             Data = StartDataPointer;
167             while (i > 0)
168             {
169                 index = (shift + Data[i]) / binRange;
170                 bins[index]++;
171                 i--;
172             }
173
174             // Get Mean
175             Mean = Sum / (float)n;
176
177             // Get Max;
178             Range = max - min;
179
180             // Calculate Standard Deviation
181             Data = StartDataPointer;
182             i = n;
183             while (i-- > 0) { sum_dev += pow((*Data++ - Mean), 2); }
184             Std = sqrt((float)(sum_dev / n));
185             Calculated = true;
186
187             // Reset the pointer
188             Data = StartDataPointer;
189         }
190
191         void BasicCalculate()
192         {
193             float sum_dev = 0.0;
194
195             // Make copy of the starting pointer
196             T1 *StartDataPointer = Data;
197
198             // Get number of samples
199             n = Rows * Cols;
200
201             // fills the histogram
202             uint32_t i = n;
203
204             while (i-- > 0) { bins[(uint32_t)*Data++]++; }
205
206             // Depending on the data size choose between using the histogram or
207             // actual data for efficient calculations
```

```
208             if (n > MAX_UINT8_VALUE) { BinCalculations(0, 256); }
209             else
210             {
211                 Data = StartDataPointer;
212
213                 // Get sum , min, max
214                 i = n;
215                 while (i-- > 0)
216                 {
217                     if (*Data > max) { max = *Data; }
218                     else if (*Data < min) { min = *Data; }
219                     Sum += *Data++;
220                 }
221
222                 // Get Mean
223                 Mean = Sum / (float)n;
224
225                 // Get Max;
226                 Range = max - min;
227
228                 // Calculate Standard Deviation
229                 Data = StartDataPointer;
230                 i = n;
231                 while (i-- > 0) { sum_dev += pow((*Data++ - Mean), 2); }
232                 Std = sqrt((float)(sum_dev / n));
233                 Calculated = true;
234             }
235
236             // Reset the pointer
237             Data = StartDataPointer;
238         }
239
240         /// <summary>
241         /// Make the calculations using the histogram
242         /// </summary>
243         void BinCalculations(uint16_t startC, uint16_t endC)
244         {
245             float sum_dev = 0.0;
246             uint32_t lastC = endC - startC;
247             int32_t i = lastC;
248             // Get sum
```

```
249             while (i-- > 0) { Sum += bins[i] * (startC + i); }
250
251             // Get Mean
252             Mean = Sum / (float)n;
253
254             // Get max
255             i = lastC;
256             while (i-- > 0)
257             {
258                 if (bins[i] != 0)
259                 {
260                     max = i;
261                     break;
262                 }
263             }
264             max += startC;
265
266             // Get min
267             i = 0;
268             while (i < lastC)
269             {
270                 if (bins[i] != 0)
271                 {
272                     min = i;
273                     break;
274                 }
275                 i++;
276             }
277             min += startC;
278
279             // Get Max;
280             Range = max - min;
281
282             // Calculate Standard Deviation
283             i = lastC;
284             while (i-- > 0) { sum_dev += bins[i] * pow(((i + startC) - Mean), 2); }
285             Std = sqrt((float)(sum_dev / n));
286             Calculated = true;
287         }
288     private:
289         friend class boost::serialization::access;
290         template <class Archive>
```

```
291         void serialize(Archive & ar, const unsigned int version)
292         {
293             ar & BOOST_SERIALIZATION_NVP(isDiscrete);
294             ar & BOOST_SERIALIZATION_NVP(n);
295             for (size_t dc = 0; dc < n; dc++) {
296                 std::stringstream ss;
297                 ss << "Data_" << dc;
298                 ar & boost::serialization::make_nvp(ss.str().c_str(), Data[dc]);
299             }
300             ar & BOOST_SERIALIZATION_NVP(noBins);
301             for (size_t dc = 0; dc < noBins; dc++) {
302                 std::stringstream ss;
303                 ss << "Bin_" << dc;
304                 ar & boost::serialization::make_nvp(ss.str().c_str(), bins[dc]);
305             }
306             ar & BOOST_SERIALIZATION_NVP(Calculated);
307             ar & BOOST_SERIALIZATION_NVP(Mean);
308             ar & BOOST_SERIALIZATION_NVP(Range);
309             ar & BOOST_SERIALIZATION_NVP(min);
310             ar & BOOST_SERIALIZATION_NVP(max);
311             ar & BOOST_SERIALIZATION_NVP(Startbin);
312             ar & BOOST_SERIALIZATION_NVP(EndBin);
313             ar & BOOST_SERIALIZATION_NVP(binRange);
314             ar & BOOST_SERIALIZATION_NVP(Std);
315             ar & BOOST_SERIALIZATION_NVP(Sum);
316             ar & BOOST_SERIALIZATION_NVP(Rows);
317             ar & BOOST_SERIALIZATION_NVP(Cols);
318         }
319
320     };
321 }
322
323 typedef SoilMath::Stats<float, double, long double> floatStat_t;
324 typedef SoilMath::Stats<uchar, uint32_t, uint64_t> ucharStat_t;
```