GCC Code Coverage Report

Directory: ./
File: MathUtility.cpp
Date: 2021-11-17 01:36:45

	Exec	Total	Coverage
Lines:	83	88	94.3%
Branches:	75	134	56.0%

Line	Branch	Exec	Source
1			/**
2			* @file MathUtility.cpp
3			* @brief Math utility functions
4			* @author Dennis Ping
5			* @date 2021-11-13
6			**/
7			
8			#include <vector>
9			#include <cmath>
10			#include <iostream>
11			#include <queue>
12			#include <set>
13			
14			#include "MathUtility.hpp"
15			#include "App.hpp"
16			
17			/*! \brief Return a vector of intermediate pixels between (x1,y1) and (x2,y2).
18			* This is the Extremely Fast Line Algorithm (EFLA) Variation E
19			* Author: Po-Han Lin (2005)
20			* Source: http://www.edepot.com/algorithm.html
21			*/
22		9606	std::vector<std::pair<int, int>> MathUtility::ExtremelyFastLineAlgo(int x1, int y1, int x2, int y2) {
23		9606	std::vector<std::pair<int, int>> pixelsVector;
24		9606	bool yLonger = false;
25		9606	int shortLen = y2 - y1;
26		9606	int longLen = x2 - x1;
27	▶ 2/2	9606	if (abs(shortLen) > abs(longLen)) {
28		4377	int swap = shortLen;
29		4377	shortLen = longLen;
30		4377	longLen = swap;
31		4377	yLonger = true;
32		4377	}
33			int decInc;
34	▶ 1/2	9606	if (longLen == 0) decInc = 0;
35		9606	else decInc = (shortLen << 16) / longLen;
36			
37	▶ 2/2	9606	if (yLonger) {
38	▶ 1/2	4377	if (longLen > 0) {
39		4377	longLen += y1;
40	▶ 2/2	106908	for (int j = 0x8000 + (x1 << 16); y1 <= longLen; ++y1) {
41			// Make a new pair and push it back into the vector
42	▶ 2/4	102531	pixelsVector.emplace_back(std::make_pair(j >> 16, y1));
43		102531	j += decInc;
44		102531	}
45		4377	return pixelsVector;
46			}
47		✗	longLen += y1;
48		✗	for (int j = 0x8000 + (x1 << 16); y1 >= longLen; --y1) {
49		✗	pixelsVector.emplace_back(std::make_pair(j >> 16, y1));
50		✗	j -= decInc;
51			}
52		✗	return pixelsVector;
53			}
54			
55	▶ 2/2	5229	if (longLen > 0) {
56		5228	longLen += x1;
57	▶ 2/2	125602	for (int j = 0x8000 + (y1 << 16); x1 <= longLen; ++x1) {
58	▶ 2/4	120374	pixelsVector.emplace_back(std::make_pair(x1, j >> 16));
59		120374	j += decInc;
60		120374	}
61		5228	return pixelsVector;
62			}
63		1	longLen += x1;
64	▶ 2/2	10	for (int j = 0x8000 + (y1 << 16); x1 >= longLen; --x1) {
65	▶ 2/4	9	pixelsVector.emplace_back(std::make_pair(x1, j >> 16));
66		9	j -= decInc;
67		9	}
68		1	return pixelsVector;
69	▶ 1/2	9606	}
70			
71			/*! \brief Return a vector of a filled in circle using a modified version of Bresenham's Circle Algorithm.
72			* Cache this circle template so that App does not need to always compute this.
73			* Author: Linus Arver (2021)
74			* Source: https://funloop.org/post/2021-03-15-bresenham-circle-drawing-algorithm.html
75			*/
76		16	std::vector<std::pair<int,int>> MathUtility::BresenhamCircleAlgo(int radius) {

```
77                  16      std::set<std::pair<int, int>> outerSet;
78                  16      int x = 0;
79                  16      int y = -radius;
80                  16      int F_M = 1 - radius;
81                  16      int dir_east = 3;
82                  16      int dir_northeast = -(radius << 1) + 5;
83                          // Emplace all the mirror points of (x,y)
84  ▶ 2/4           16      outerSet.emplace(std::make_pair(x, y));
85  ▶ 2/4           16      outerSet.emplace(std::make_pair(x, -y));
86  ▶ 2/4           16      outerSet.emplace(std::make_pair(-x, y));
87  ▶ 2/4           16      outerSet.emplace(std::make_pair(-x, -y));
88  ▶ 2/4           16      outerSet.emplace(std::make_pair(y, x));
89  ▶ 2/4           16      outerSet.emplace(std::make_pair(y, -x));
90  ▶ 2/4           16      outerSet.emplace(std::make_pair(-y, x));
91  ▶ 2/4           16      outerSet.emplace(std::make_pair(-y, -x));
92
93  ▶ 2/2           79      while (x < -y) {
94  ▶ 2/2           63          if (F_M <= 0) {
95                  33              F_M += dir_east;
96                  33          } else {
97                  30              F_M += dir_northeast;
98                  30              dir_northeast += 2;
99                  30              y += 1;
100                             }
101                 63          dir_east += 2;
102                 63          dir_northeast += 2;
103                 63          x += 1;
104                             // Emplace all the mirror points of (x,y)
105 ▶ 2/4           63          outerSet.emplace(std::make_pair(x, y));
106 ▶ 2/4           63          outerSet.emplace(std::make_pair(x, -y));
107 ▶ 2/4           63          outerSet.emplace(std::make_pair(-x, y));
108 ▶ 2/4           63          outerSet.emplace(std::make_pair(-x, -y));
109 ▶ 2/4           63          outerSet.emplace(std::make_pair(y, x));
110 ▶ 2/4           63          outerSet.emplace(std::make_pair(y, -x));
111 ▶ 2/4           63          outerSet.emplace(std::make_pair(-y, x));
112 ▶ 2/4           63          outerSet.emplace(std::make_pair(-y, -x));
113                         }
114
115                         // O(n^2) loop to fill pixels in because math is hard.
116                         // It's ok, we will cache this circle template anyway.
117 ▶ 2/2           96      for (int j=0; j < radius; j++) {
118 ▶ 2/2           504         for (int k=0; k < radius; k++) {
119 ▶ 2/2           424             if (pow(j, 2) + pow(k, 2) < pow(radius, 2)) {
120 ▶ 2/4           373                 outerSet.emplace(std::make_pair(j,k));
121 ▶ 2/4           373                 outerSet.emplace(std::make_pair(j,-k));
122 ▶ 2/4           373                 outerSet.emplace(std::make_pair(-j,k));
123 ▶ 2/4           373                 outerSet.emplace(std::make_pair(-j,-k));
124                 373             }
125                 424         }
126                 80      }
127                         // Convert our set to a vector for easier access
128                 16      std::vector<std::pair<int,int>> circleTemplate;
129                         // Reserve memory in vector to get 4x performance: https://github.com/facontidavide/CPP_Optimizations_Diary/blob/maste
130 ▶ 1/2           16      circleTemplate.reserve((outerSet.size()*8));
131                         // Use std::transform to fill the vector
132 ▶ 2/4           1640    std::transform(outerSet.begin(), outerSet.end(), std::back_inserter(circleTemplate), [](const std::pair<int,int>& p) {
133                 16      return circleTemplate;
134 ▶ 1/2           16  }
135
136                 /*! \brief  A pairing function that maps two values to a single unique value.
137                 *           Essentially a hash function for pairs of signed integers.
138                 *           The C++ std lib pair hash only works for combinations, not permutations where order matters.
139                 *           Author: Matthew Szudzik (2006)
140                 *           Source: https://www.vertexfragment.com/ramblings/cantor-szudzik-pairing-functions/
141                 */
142                 namespace std {
143                     template<>
144                     struct std::hash<std::pair<int,int>> {
145                         size_t operator()(const std::pair<int,int>& p) const {
146                             int one = p.first;
147                             int two = p.second;
148                             int const a = (one >= 0.0 ? 2.0 * one : (-2.0 * one) - 1.0);
149                             int const b = (two >= 0.0 ? 2.0 * two : (-2.0 * two) - 1.0);
150                             return (a >= b ? (a * a) + a + b : (b * b) + a) * 0.5;
151                         }
152                     };
153                 }
154
155                 // /*! \brief   Return a vector of pairs of (x,y) coordinates that within inside the radius of the paintbrush center.
156                 // *           This is the brute force version of Bresenham's Circle Algorithm
157                 // *           Source: https://stackoverflow.com/questions/1201200/fast-algorithm-for-drawing-filled-circles
158                 // *           Author: Daniel Earwicker (2009)
159                 // * */
160                 // std::vector<std::pair<int, int>> Utility::MidpointCircleAlgo(int xCenter, int yCenter, int radius) {
161                     // std::vector<std::pair<int, int>> pixelsVector;
162                     // int radius_sqr = radius * radius;
```

```
163                     // for (int x = -radius; x < radius ; x++) {
164                     //     int hh = (int)std::sqrt(radius_sqr - x * x);
165                     //     int rx = xCenter + x;
166                     //     int ph = yCenter + hh;
167                     
168                     //     for (int y = yCenter-hh; y < ph; y++) {
169                     //         pixelsVector.push_back(std::make_pair(rx, y));
170                     //     }
171                     // }
172                 // }
173                 
174                 // std::vector<std::pair<int, int>> Utility::MidpointCircleAlgo(int xCenter, int yCenter, int radius) {
175                 //     std::vector<std::pair<int, int>> pixelsVector;
176                 //     int x = radius;
177                 //     int y = 0;
178                 //     int radiusError = 1 - x;
179                 //     while (x >= y) {  // iterate to the circle diagonal
180                 //         // use symmetry to draw the two horizontal lines at this Y with a special case to draw
181                 //         // only one line at the centerY where y == 0
182                 //         int startX = -x + xCenter;
183                 //         int endX = x + xCenter;
184                 //         for (int i = startX; i <= endX; i++) {
185                 //             pixelsVector.push_back(std::make_pair(i, y + yCenter));
186                 //         }
187                 //         if (y != 0) {
188                 //             for (int i = startX; i <= endX; i++) {
189                 //                 pixelsVector.push_back(std::make_pair(i, -y + yCenter));
190                 //             }
191                 //         }
192                 //         // move Y one line
193                 //         y++;
194                 //         // calculate or maintain new x
195                 //         if (radiusError<0) {
196                 //             radiusError += 2 * y + 1;
197                 //         }
198                 //         else {
199                 //             // we're about to move x over one, this means we completed a column of X values, use
200                 //             // symmetry to draw those complete columns as horizontal lines at the top and bottom of the circle
201                 //             // beyond the diagonal of the main loop
202                 //             if (x >= y)
203                 //             {
204                 //                 startX = -y + 1 + xCenter;
205                 //                 endX = y - 1 + xCenter;
206                 //                 for (int i = startX; i <= endX; i++) {
207                 //                     pixelsVector.push_back(std::make_pair(i, x + yCenter));
208                 //                     pixelsVector.push_back(std::make_pair(i, -x + yCenter));
209                 //                 }
210                 //             }
211                 //             x--;
212                 //             radiusError += 2 * (y - x + 1);
213                 //         }
214                 //     }
215                 //     return pixelsVector;
216                 // }
217                 
218                 // Source: https://en.wikipedia.org/wiki/Flood_fill
219                 // std::vector<std::pair<int, int>> Utility::FloodFill(sf::Image& image, int x, int y, sf::Color color) {
220                 //     std::vector<std::pair<int, int>> pixelsVector;
221                 //     std::queue<std::pair<int, int>> myQueue;
222                 //     std::unordered_set<std::pair<int, int>> mySet;
223                 //     myQueue.push(std::make_pair(x, y));
224                 //     while (!myQueue.empty()) {
225                 //         std::pair<int, int> pixel = myQueue.front();
226                 //         myQueue.pop();
227                 //         if (image.getPixel(pixel.first, pixel.second) != color) {
228                 //             mySet.insert(pixel);
229                 //             myQueue.push(std::make_pair(pixel.first + 1, pixel.second));
230                 //             myQueue.push(std::make_pair(pixel.first - 1, pixel.second));
231                 //             myQueue.push(std::make_pair(pixel.first, pixel.second + 1));
232                 //             myQueue.push(std::make_pair(pixel.first, pixel.second - 1));
233                 //         }
234                 //     }
235                 //     for (auto& pixel : mySet) {
236                 //         pixelsVector.push_back(pixel);
237                 //     }
238                 //     return pixelsVector;
239                 // }
240                 //
241
```