

HEY
WAKE UP!

VITO
RAY

D
E
S
I
G
N
E
D

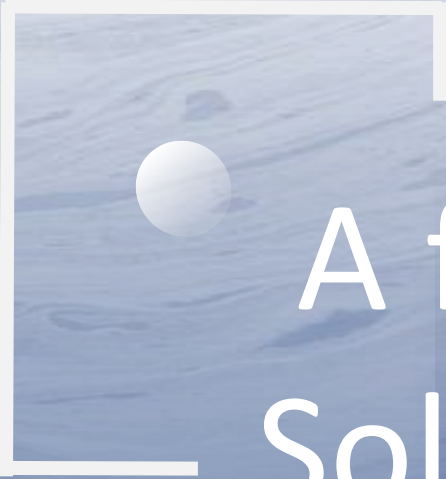
B
Y

I
B
O
T
U

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
U



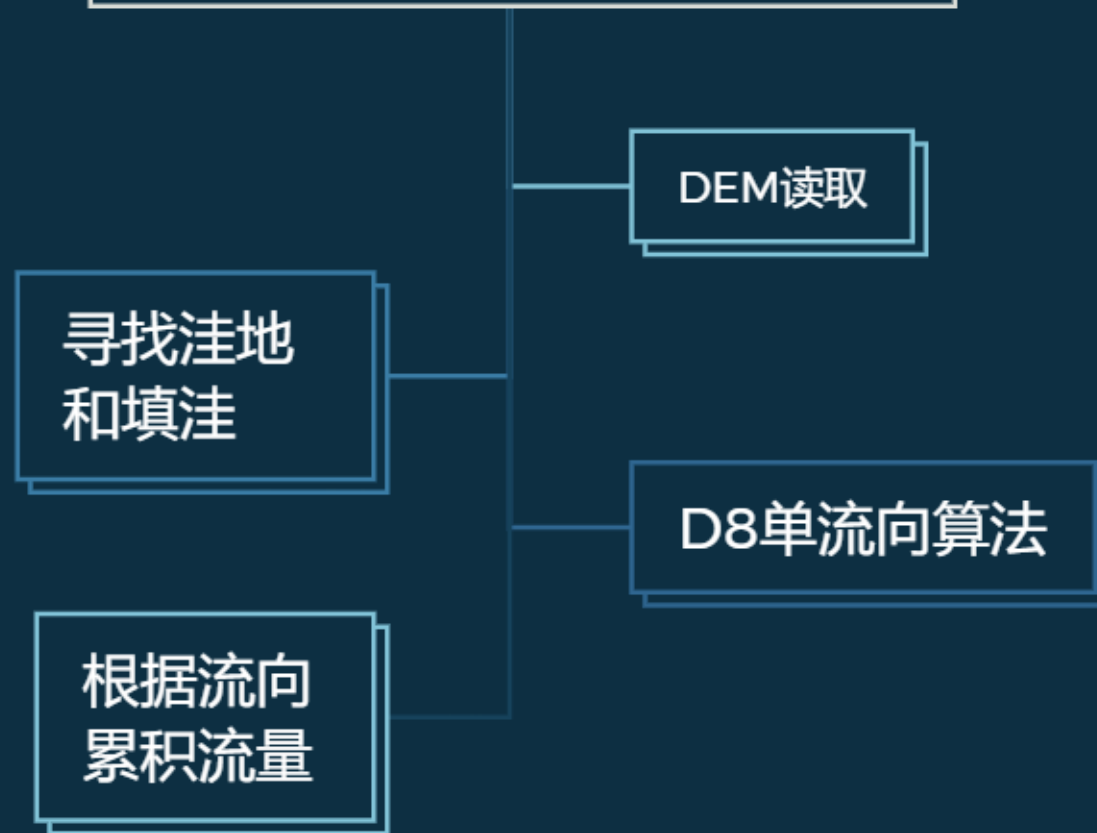
A fast and effective Solution to calculate Accumulation from DEM

DO SOME
THINGS!



四个 核心 组件

提取数字高程模型中可能存在的累计流量



Presented with **xmind**

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

PART .01

Fill

An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling

L. Wang & H. Liu

To cite this article: L. Wang & H. Liu (2006) An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling, International Journal of Geographical Information Science, 20:2, 193-213, DOI: 10.1080/13658810500433453

To link to this article: <https://doi.org/10.1080/13658810500433453>

15	15	14	15	12	6	12
14	13	10	12	15	17	15
15	15	9	11	8	15	15
16	17	8	16	15	7	5
19	18	19	18	17	15	14

(a)

15	15	14	15	12	6	12
14						15
15						15
16						5
19	18	19	18	17	15	14

(b)

15	15	14	15	12	6	12
14						15
15						15
16						7
19	18	19	18	17	15	14

(c)

15	15	14	15	12	6	12
14						15
15						15
16						7
19	18	19	18	17	15	14

(d)

15	15	14	15	12	6	12
14						15
15						15
16						15
19	18	19	18	17	15	14

(e)

15	15	14	15	12	6	12
14						15
15						15
16						15
19	18	19	18	17	15	14

(f)

15	15	14	15	12	6	12
14						15
15						15
16						7
19	18	19	18	17	15	14

(g)

15	15	14	15	12	6	12
14	13	11	12	15	17	15
15	15	11	11	8	15	15
16						7
19	18	19	18	17	15	14

(h)

15	15	14	15	12	6	12
14	13	11	12	15	17	15
15	15	11	11	8	15	15
16	17	11	16	15	7	5
19	18	19	18	17	15	14

(i)

Plan A 优先队列+最小值堆

VITO
RAY

插图



使用方法

- 凹陷点是指未定义流域方向的像元；其周围的像元均高于它。倾泻点是汇流区域中具有最低高程的边界像元。如果凹陷点中充满了水，则水将从该点倾泻出去。
- z 限制指定凹陷点深度和倾泻点间的最大允许差值并确定要填充的凹陷点和保持不变的凹陷点。z 限制并非一个凹陷点要填充的最大深度。

例如，假设一个凹陷点区域中倾泻点的高程为 210 英尺，凹陷点的最深点为 204 英尺（相差 6 英尺）。如果将 z 限制设置为 8，则会填充该特殊凹陷点。但是，如果将 z 限制设置为 4，则不会填充该凹陷点，因为该凹陷点的深度超过该限制值，将其视为有效凹陷点。

- 小于 z 限制且低于其最低相邻像元的所有凹陷点都将填充到其倾泻点的高度。
- 运行填洼工具非常占用内存、CPU 和磁盘空间。最多时可能要求磁盘空间为输入栅格的四倍。
- 包含的带有 z 限制的凹陷点数量将决定处理时间的长度。凹陷点越多，处理时间就越长。
- 汇工具可用于在使用填洼工具前查找凹陷点数量，并帮助识别凹陷点深度。了解凹陷点深度将有助于确定适用的 z 限制。

A fast, simple and versatile algorithm to fill the depressions of digital elevation models

Olivier Planchon^{a,*}, Frédéric Darboux^{b,c,1}

^a *Institut de Recherche pour le Développement — IRD, BP 1386, Dakar, Senegal*

^b *Géosciences – Rennes, Campus de Beaulieu, 35042 Rennes Cedex, France*

^c *National Soil Erosion Research Laboratory, 1196 SOIL Building, Purdue University, West Lafayette, IN 47907-1196, USA*

Dependence graph概念

栅格细胞Cell和它的邻居们 →

每当我们扫描到一个Cell，我们就自动地弹出他的邻居

细胞Cell

淹水法 (全部齐平)

逐个排水

找到洼地

填平

每当我们扫描到一个Cell, 我们就自动地弹出他的邻居

比较高程

逼近邻居

只需扫描一次
DEM矩阵, 时间
复杂度 $O(N^{1.5})$

三个目标:

- A. Fill中的任何一个栅格
 \geq 初始DEM
- B. 每个径流都有一个出口
- C. Fill中的栅格是符合AB条件下的极小值

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

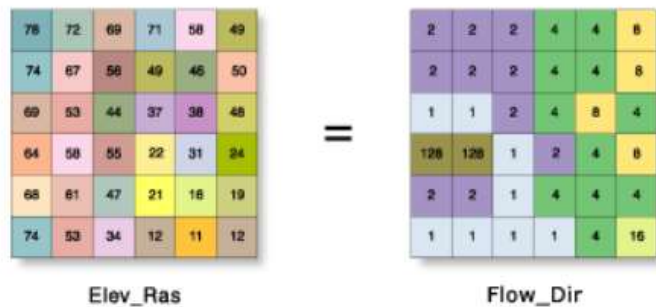
B
Y

I
B
O
T
T
O
M

PART .02

D8

插图



```
Flow_Dir = FlowDirection(Elev_Ras, #, #, D8)
```

使用方法

- 流向工具支持三种流向建模算法。分别为 D8、多流向 (MFD) 和 D-Infinity (DINF)。
- D8 流向法可对每个像元到其最陡下坡邻域的流向进行建模。

以 D8 流向类型运行的流向工具的输出是值范围介于 1 到 255 之间的整型栅格。从中心出发的各个方向值为：

32	64	128
16		1
8	4	2

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

PART .03

Accumulate

A fast and simple algorithm for calculating flow accumulation matrices from raster digital elevation models

Guiyun ZHOU (✉)^{1,2}, Hongqiang WEI², Suhua FU^{3,4}

1 Center for Information Geoscience, University of Electronic Science and Technology of China, Chengdu 611731, China

2 School of Resources and Environment, University of Electronic Science and Technology of China, Chengdu 611731, China

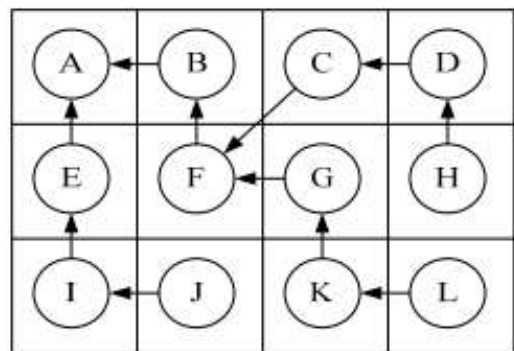
3

State Key Laboratory of Soil Erosion and Dryland Farming on the Loess Plateau, Institute of Soil and Water Conservation, Chinese Academy of Sciences, Yangling, Shaanxi 712100, China

4 Faculty of Geographical Science, Beijing Normal University, Beijing 100875, China

快
一
点
再
快
一
点

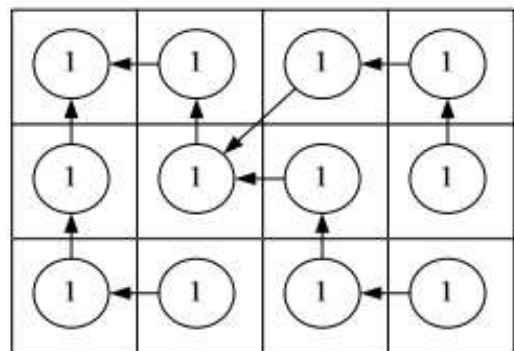
有向图的深度优先搜索



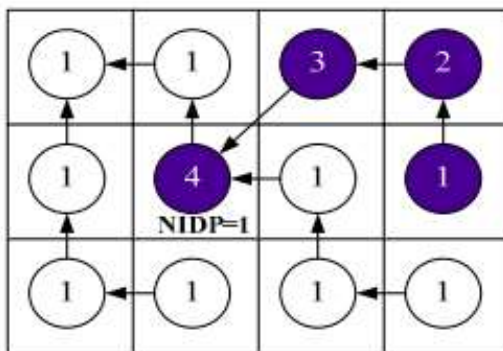
(a)

2	1	1	1
1	2	1	0
1	0	1	0

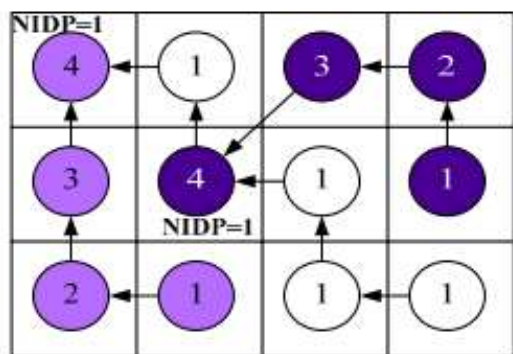
(b)



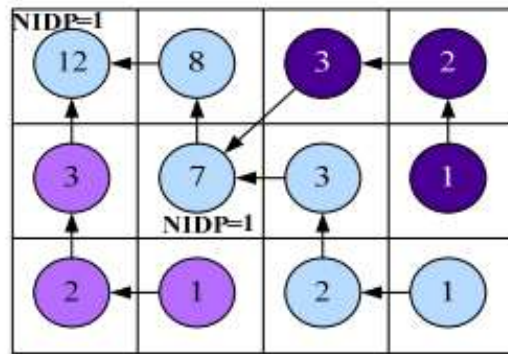
(c)



(d)



(e)



(f)

Cells visited in tracing 1
 Cells visited in tracing 2
 Cells visited in tracing 3

三种细胞

源

交叉口

通道

Depth-First Search (DFS)

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
U

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

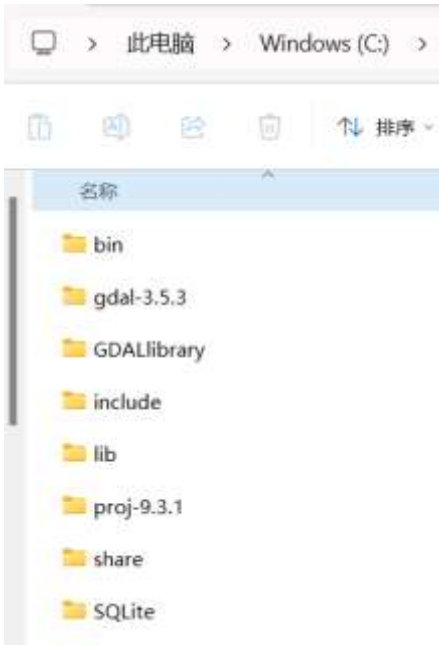
I
B
O
T
U

PART .04

Talk is CHEAP

开发环境

附加依赖项



常规属性		
输出目录	<不同选项>	
中间目录	<不同选项>	
目标文件名	\$(ProjectName)	
配置类型	应用程序(.exe)	
Windows SDK 版本	10.0 (最新安装的版本)	
平台工具集	Visual Studio 2022 (v143)	
C++ 语言标准	ISO C++20 标准 (/std:c++20)	
C 语言标准	默认(旧 MSVC)	



CMake (cmake-gui)
应用



C with C++20

栅格细胞Cell

```
107 // 定义一个细胞结构体装载DEM中的元素
108 struct Cell {
109     int row, col;
110     float dem;
111     bool operator<(const Cell& rhs) const {
112         return dem > rhs.dem; // 优先级队列需要最小堆
113     }
114 };
115
```

Cell的邻居

```
132 //定义八个方向的neighbour
133 std::vector<std::pair<int, int>> directions = {
134     {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1} };
135
257 // 定义8个方向的偏移量
258 std::vector<int> dx = { -1, -1, 0, 1, 1, 1, 0, -1 };
259 std::vector<int> dy = { 0, 1, 1, 1, 0, -1, -1, -1 };
260
```

有向图的边

```
187
188 // 定义方向查询表
189 std::vector<std::vector<int>> dir = {
190     {8, 16, 32}, {4, 0, 64}, {2, 1, 128} }; //流出查询表
191 std::vector<std::vector<int>> dir_receive = {
192     {128, 1, 2}, {64, 0, 4}, {32, 16, 8} }; //流入查询表
261 // 定义8个方向的值
262 std::vector<int> d8 = { 16, 32, 64, 128, 1, 2, 4, 8 };
263
```

失败的PlanA

```
1. // 定义一个细胞结构体装载DEM中的元素
2. struct Cell {
3.     int row, col;
4.     float dem;
5.     bool operator<(const Cell& rhs) const {
6.         return dem > rhs.dem; // 优先级队列需要最小堆
7.     }
8. };

9. std::vector<std::vector<float>>> Fill(const std::vector<std::vector<float>>>& DEM) {
10.     int rows = DEM.size();
11.     int cols = DEM[0].size();

12.     std::priority_queue<Cell> OPEN;
13.     std::vector<std::vector<bool>>> CLOSED(rows, std::vector<bool>(cols, false));
14.     std::vector<std::vector<float>>> Spill = DEM;

15.     // 用边界细胞初始化优先级队列
16.     for (int i = 0; i < rows; ++i) {
17.         for (int j = 0; j < cols; ++j) {
18.             if (i == 0 || j == 0 || i == rows - 1 || j == cols - 1) {
19.                 if (DEM[i][j] != -9999) { // Only add the cell to the queue if its value is not -9999
20.                     OPEN.push({ i, j, Spill[i][j] });
21.                 }
22.             }
23.         }
24.     }

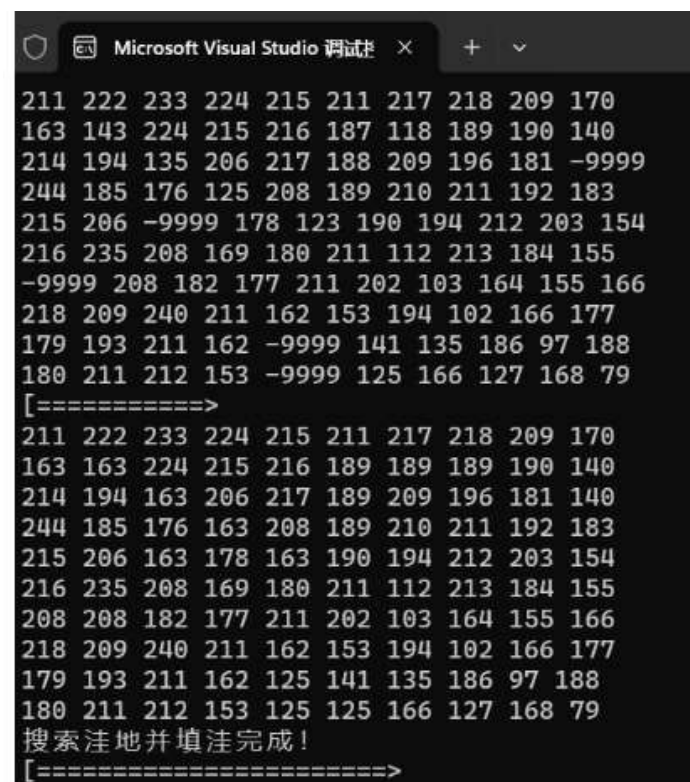
25.     // 定义8个方向的偏移量
26.     std::vector<std::pair<int, int>>> directions = { {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1} };

27.     while (!OPEN.empty()) {
28.         Cell c = OPEN.top();
29.         OPEN.pop();
30.         CLOSED[c.row][c.col] = true;

31.         for (const auto& dir : directions) {
32.             int newRow = c.row + dir.first;
33.             int newCol = c.col + dir.second;

34.             if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < cols
35.                 && !CLOSED[newRow][newCol]) {
36.                 Spill[newRow][newCol] = std::max(DEM[newRow][newCol], Spill[c.row][c.col]);
37.                 OPEN.push({ newRow, newCol, Spill[newRow][newCol] });
38.             }
39.         }

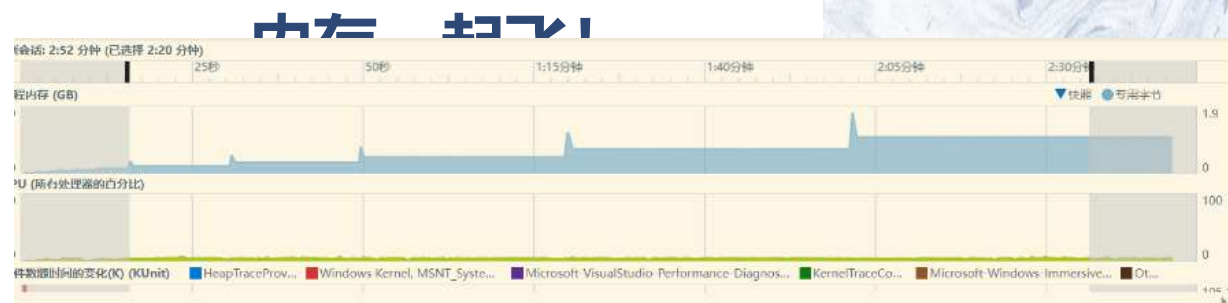
40.     // 返回Spill矩阵
41.     return Spill;
42. }
```



Microsoft Visual Studio 调试

211	222	233	224	215	211	217	218	209	170
163	143	224	215	216	187	118	189	190	140
214	194	135	206	217	188	209	196	181	-9999
244	185	176	125	208	189	210	211	192	183
215	206	-9999	178	123	190	194	212	203	154
216	235	208	169	180	211	112	213	184	155
-9999	208	182	177	211	202	103	164	155	166
218	209	240	211	162	153	194	102	166	177
179	193	211	162	-9999	141	135	186	97	188
180	211	212	153	-9999	125	166	127	168	79
[=====>									
211	222	233	224	215	211	217	218	209	170
163	163	224	215	216	189	189	189	190	140
214	194	163	206	217	189	209	196	181	140
244	185	176	163	208	189	210	211	192	183
215	206	163	178	163	190	194	212	203	154
216	235	208	169	180	211	112	213	184	155
208	208	182	177	211	202	103	164	155	166
218	209	240	211	162	153	194	102	166	177
179	193	211	162	125	141	135	186	97	188
180	211	212	153	125	125	166	127	168	79
搜索洼地并填洼完成!									
[=====>									

合理的结果，失败的内存管理



VITO
RAY


```
116 //使用排水法实现填洼(Olivier Planchon & Frederic Darboux, 2002)
117 std::vector<std::vector<float>> Fill(const std::vector<std::vector<float>>& DEM) {
118     int rows = DEM.size();
119     int cols = DEM[0].size();
120     // 遍历整个二维vector找到最大值
121     int max_val = DEM[0][0]; // 初始化为第一行第一列的值
122     for (const auto& row : DEM) {
123         for (int val : row) {
124             if (val > max_val) {
125                 max_val = val;
126             }
127         }
128     }
129
130     //初始化 Fill矩阵 (淹水)
131     std::vector<std::vector<float>> Fill(rows, std::vector<float>(cols, max_val+1));
132     //定义八个方向的neighbour
133     std::vector<std::pair<int, int>> directions = {
134         {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1} };
135
136     //保留边界和Nodata原始值
137     for (int i = 0; i < rows; ++i) {
138         for (int j = 0; j < cols; ++j) {
139             if (i == 0 || j == 0 || i == rows - 1 || j == cols - 1 || DEM[i][j] == -9999) {
140                 Fill[i][j] = DEM[i][j];
141             }
142         }
143     }
```

```
145 //逐步排水
146 bool Fill_done = false;
147 while (!Fill_done) {
148     Fill_done = true;
149     //跳过边界
150     for (int i = 1; i < rows-1; ++i) {
151         for (int j = 1; j < cols-1; ++j) {
152             //找到可能的洼地
153             if (Fill[i][j] > DEM[i][j]) {
154                 for (const auto& dir : directions) {
155                     int newRow = i + dir.first;
156                     int newCol = j + dir.second;
157                     //边界问题
158                     if (newRow >= 0 && newRow < rows
159                         && newCol >= 0 && newCol < cols && DEM[newRow][newCol] != -9999) {
160                         //确保有出口的情况下, 保证Fill 逼近 DEM
161                         if (DEM[i][j] > DEM[newRow][newCol] + 0.0001) {
162                             Fill[i][j] = DEM[i][j];
163                             Fill_done = false;
164                             break;
165                         }
166                         //如果没有已知的出口, 逐渐排水保证至少有一个出口
167                         if (Fill[i][j] > Fill[newRow][newCol] + 1) {
168                             Fill[i][j] = Fill[newRow][newCol] + 1;
169                             Fill_done = false;
170                         }
171                     }
172                 }
173             }
174         }
175     }
176 }
```


不那么完善的

对自然的
汇和开阔
河道考虑
不足

```
206 // 检查所有8个邻居, 从左下开始遍历
207 for (int di = -1; di <= 1; ++di) {
208     for (int dj = -1; dj <= 1; ++dj) {
209         int ni = i + di;
210         int nj = j + dj;
211
212         // 跳过中心单元格和没有邻居的方向
213         if ((di == 0 && dj == 0) || ni < 0 || ni >= rows || nj < 0 || nj >= cols) continue;
214
215         // 对角线方向水平距离为根号2, 其余方向水平距离为1
216         if ((di == -1 && dj == -1) || (di == -1 && dj == 1) || (di == 1 && dj == -1) || (di == 1 && dj == 1))
217         {
218             if (fill[ni][nj] == -9999) continue;
219             float slope = (fill[ni][nj] - fill[i][j]) / 1.414;
220             if (slope < down_slope) {
221                 down_slope = slope;
222                 min_dir = dir[di + 1][dj + 1];
223             }
224         }
225         else if (fill[ni][nj] - fill[i][j] == 0 && flowDirection[ni][nj] == 0) {
226             float slope = fill[ni][nj] - fill[i][j];
227             // 如果邻居的DEM值与当前单元格相同且邻居的流向为0, 则流向指向邻居
228             if (slope < down_slope) {
229                 min_dir = dir[di + 1][dj + 1];
230             }
231         }
232         else if (fill[ni][nj] - fill[i][j] == 0 && flowDirection[ni][nj] != 0) {
233             float slope = fill[ni][nj] - fill[i][j];
234             // 如果邻居的DEM值与当前单元格相同且邻居的流向不为0, 则邻居被指向
235             if (slope < down_slope) {
236                 min_dir = dir_receive[di + 1][dj + 1];
237             }
238         }
239         else
240         {
241             if (fill[ni][nj] == -9999) continue;
242             float slope = fill[ni][nj] - fill[i][j];
243             if (slope < down_slope) {
244                 down_slope = slope;
245                 min_dir = dir[di + 1][dj + 1];
246             }
247         }
248     }
249 }
```

```
266 // 检查每个方向
267 for (int k = 0; k < 8; ++k) {
268     // 如果 c 单元格的流向是这个方向
269     if (FlowDir[c.row][c.col] == d8[k]) {
270         // 计算下游单元格的位置
271         int newRow = c.row + dx[k];
272         int newCol = c.col + dy[k];
273
274         // 检查下游单元格是否在 DEM 的范围内
275         if (newRow >= 0 && newRow < FlowDir.size() && newCol >= 0
276             && newCol < FlowDir[0].size() && FlowDir[newRow][newCol] != -9999) {
277             // 返回下游单元格, 溢出量为 FlowDir 中相应位置的值
278             return { newRow, newCol, static_cast<float>(FlowDir[newRow][newCol]) };
279         }
280         else {
281             return { -1, -1, -1 }; // 假设行和列为-1表示没有下游单元格
282         }
283     }
284 }
285
286 // 如果 c 单元格的流向不是任何一个方向, 返回一个特殊的 Cell 对象用于判别
287 return { -1, -1, -1 }; // 假设行和列为-1表示没有下游单元格
288 }
289
```

根据流向返回下游细胞，并捕获异常


```
299 // 定义8个方向的偏移量
300 std::vector<int> dx = { -1, -1, 0, 1, 1, 1, 0, -1 };
301 std::vector<int> dy = { 0, 1, 1, 1, 0, -1, -1, -1 };
302 std::vector<int> d8 = { 1, 2, 4, 8, 16, 32, 64, 128 };
303
304 // 遍历每一个单元格;
305 for (int i = 0; i < rows; ++i) {
306     for (int j = 0; j < cols; ++j) {
307         // 检查每个方向上的邻居
308         for (int k = 0; k < 8; ++k) {
309             int ni = i + dx[k];
310             int nj = j + dy[k];
311
312             // 检查邻居是否在 DEM 的范围内
313             if (ni >= 0 && ni < rows && nj >= 0 && nj < cols) {
314                 // 如果邻居的流向指向当前单元格, 增加 NIDP 值, NIDP>1代表交叉细胞, 0代表源细胞, 1代表内部细胞
315                 if (FlowDir[ni][nj] == d8[k]) {
316                     NIDP[i][j]++;
317                 }
318             }
319         }
320     }
321 }
322
323 return NIDP;
324 }
325
```

把细胞作为节点Node, 根据入边数量确定节点类型

```
338 // 遍历每一个单元格
339 for (int row = 0; row < FlowDir.size(); ++row) {
340     for (int col = 0; col < FlowDir[0].size(); ++col) {
341         Cell c = { row, col, FlowDir[row][col] };
342         if (NIDP[row][col] != 0) continue;
343         Cell n = c;
344         int nAccu = 0;
345         do {
346             FlowAccu[n.row][n.col] += nAccu;
347             nAccu = FlowAccu[n.row][n.col];
348             //交叉细胞已被访问过一次, NIDP-1
349             if (NIDP[n.row][n.col] > 1) {
350                 NIDP[n.row][n.col]--;
351                 break;
352             }
353             //从源细胞开始向下游遍历
354             n = NextCell(n, FlowDir);
355         } while (n.row != -1 && n.row < rows
356                && n.col != -1 && n.col < cols); //直到这条流线流出DEM范围
357     }
358 }
359 return FlowAccu;
360 }
361
```

根据我们的规则，对流向图进行深度优先搜索

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

PART .05

数据验证

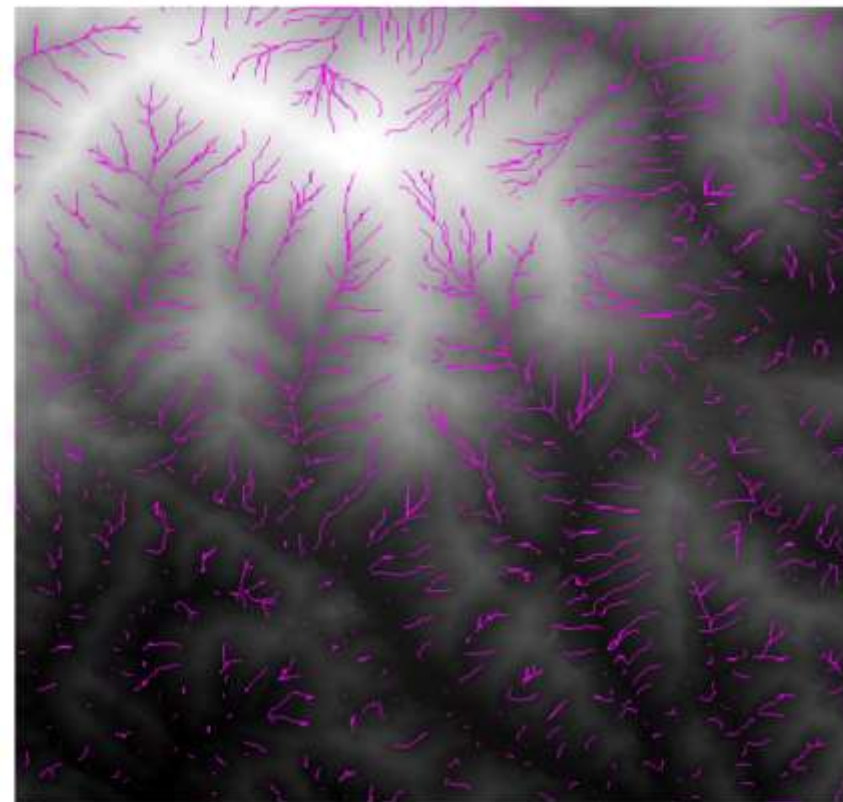
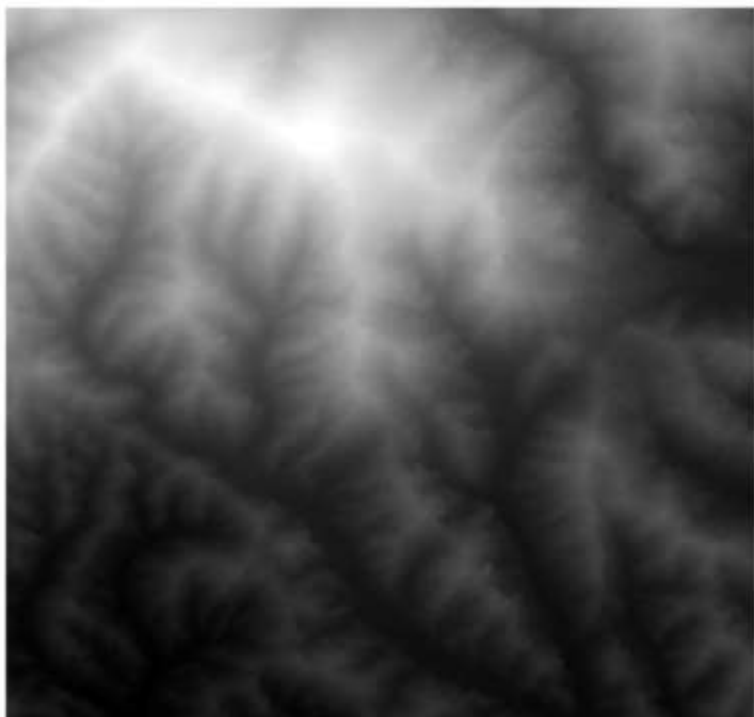
数据集验证

72行, 57列



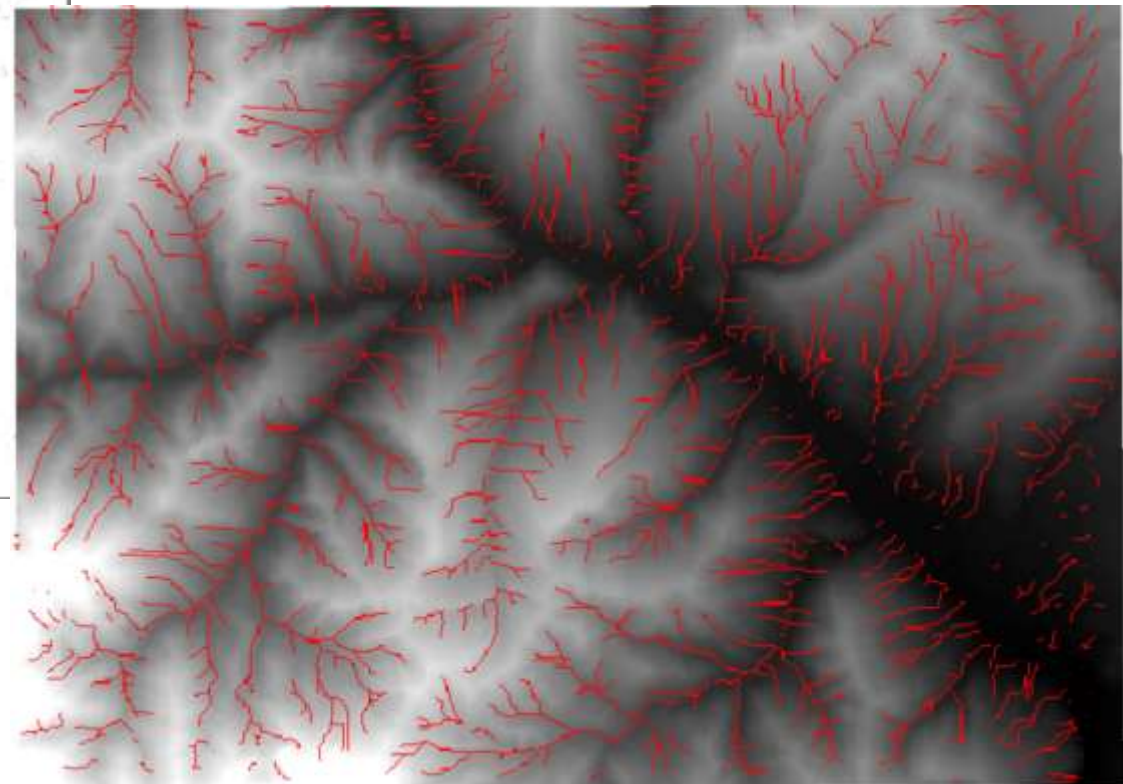
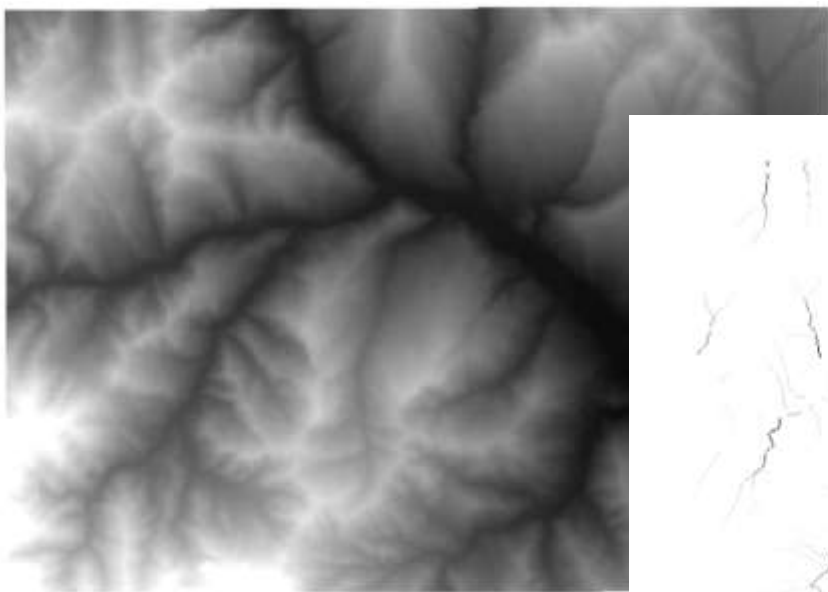
Property	Value
<input type="checkbox"/> Raster Information	
Columns and Rows	72, 57
Number of Bands	1
Cell Size (X, Y)	28.93513394, 28.93513394
Uncompressed Size	16.03 KB
Format	GRID
Source Type	Generic
Pixel Type	signed integer
Pixel Depth	16 Bit

数据集验证



天目山（部分），600行，571列，16 Bit

数据集验证



四川某地, 696行, 493列, 32 Bit

HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
T
O
M

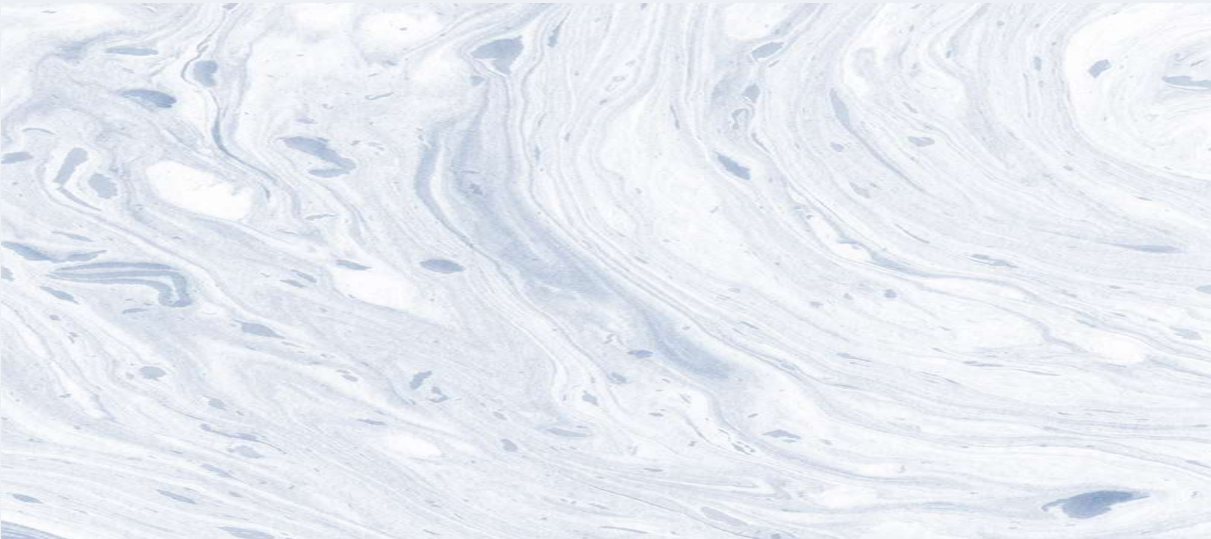
PART .06

性能

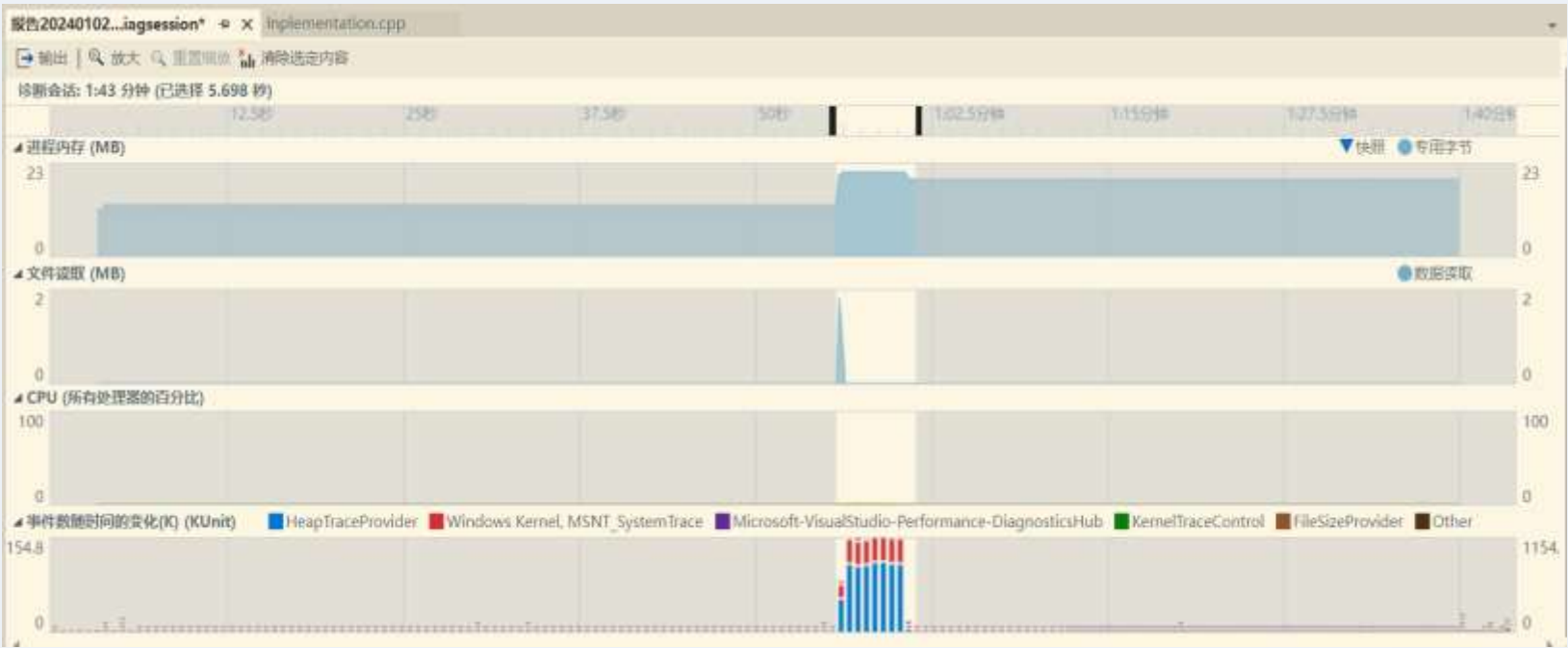
Visual Studio性能探查器



四川某地，696行，
493列，32 Bit



笔记本Intel i7-13700HX, 2100MHz, 16GB, 单线程，本地堆栈



HEY
WAKE UP!

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
U

DO SOME
THINGS!

VITO
RAY

D
E
S
I
G
N
E
D

B
Y

I
B
O
T
U

谢谢观看