HEY
WAKE UP!

VITO
RAY

D E S I G N E D B Y I B O T U

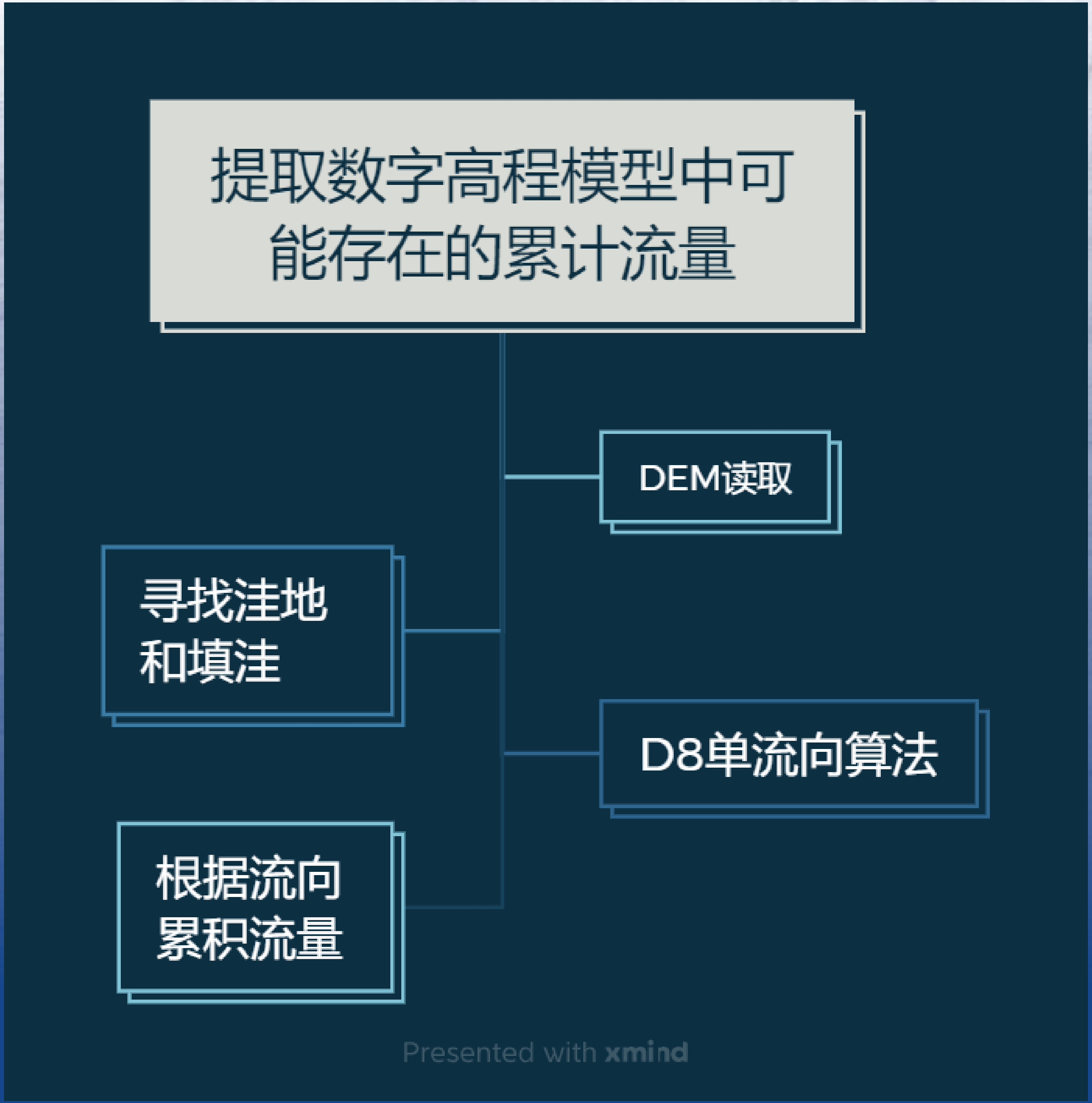
A fast and effective Solution to calculate Accumulation from DEM

D E S I G N E D B Y I B O T U

DO SOME
THINGS!

四个核心组件

DESIGNED BY IBOTU



DO SOME THINGS!

VITO RAY

DESIGNED BY IBOTU

HEY
WAKE UP!

D E S I G N E D B Y I B O T U

VITO
RAY

PART .01

Fill

D E S I G N E D B Y I B O T U

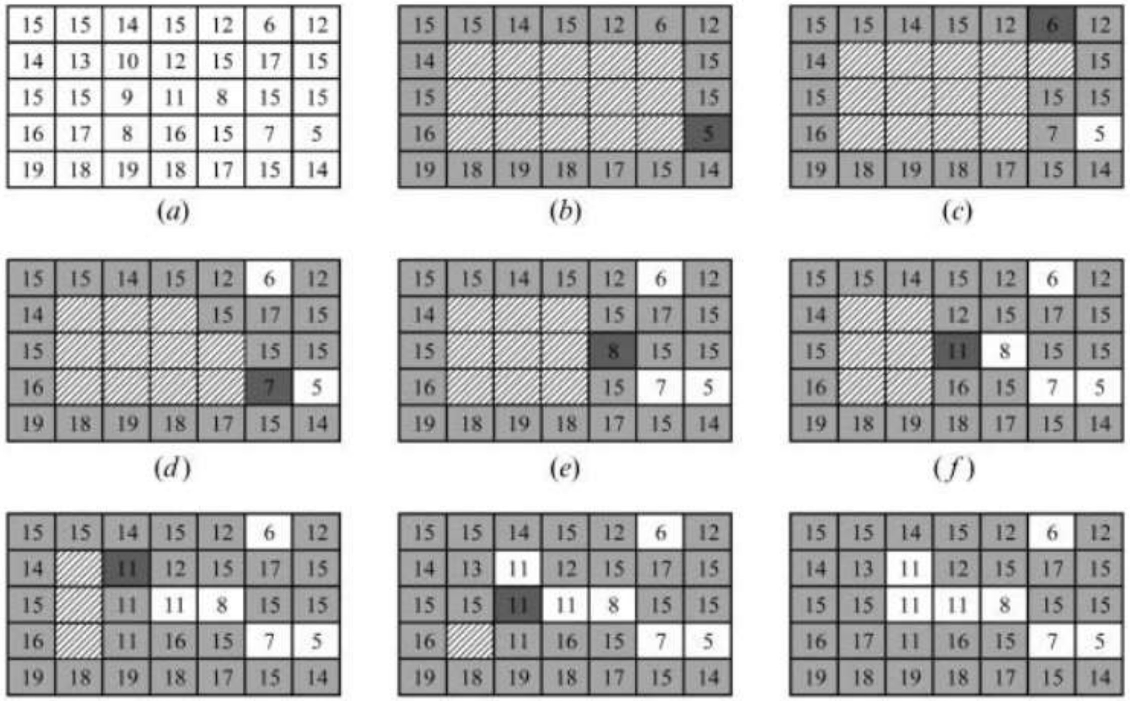
DO SOME
THINGS!

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



(a) (b) (c)

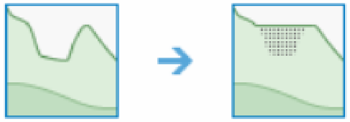
(d) (e) (f)

(g) (h) (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。

底层算法

ELSEVIER

Catena 46 (2001) 159–176

CATENA

www.elsevier.com/locate/catena

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

Olivier Planchon [a, *], Frédéric Darboux [b,c,1]

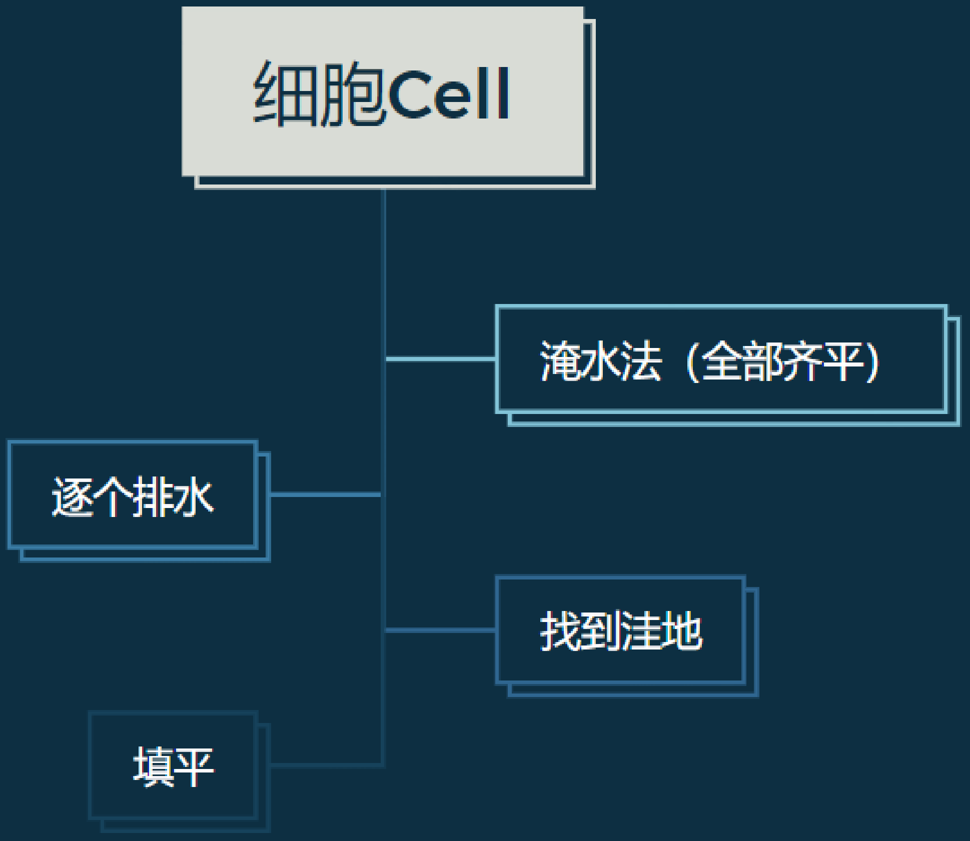
[a] *Institut de Recherche pour le Développement — IRD, BP 1386, Dakar, Senegal*
[b] *Géosciences – Rennes, Campus de Beaulieu, 35042 Rennes Cedex, France*
[c] *National Soil Erosion Research Laboratory, 1196 SOIL Building, Purdue University, West Lafayette, IN 47907-1196, USA*

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

VITO RAY

底层算法



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 >= 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

VITO RAY

HEY
WAKE UP!

D E S I G N E D B Y I B O T U

PART .02

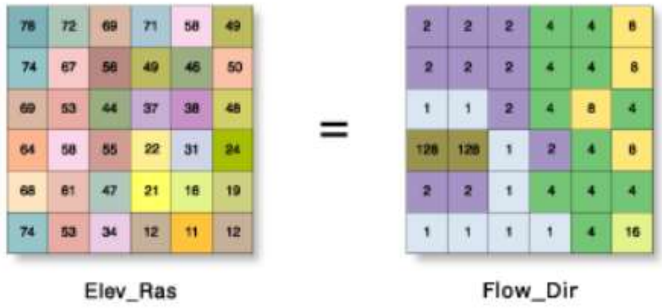
D8

DO SOME
THINGS!

VITO
RAY

D E S I G N E D B Y I B O T U

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

偷师ESRI

VITO RAY

HEY
WAKE UP!

D E S I G N E D B Y I B O T U

VITO
RAY

PART .03

Accumulate

D E S I G N E D B Y I B O T U

DO SOME
THINGS!

Frontiers of Earth Science
https://doi.org/10.1007/s11707-018-0725-9
RESEARCH ARTICLE

CN 11-5982/P
ISSN 2095-0195

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

Guiyun ZHOU (✉)[1,2], Hongqiang WEI[2], 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

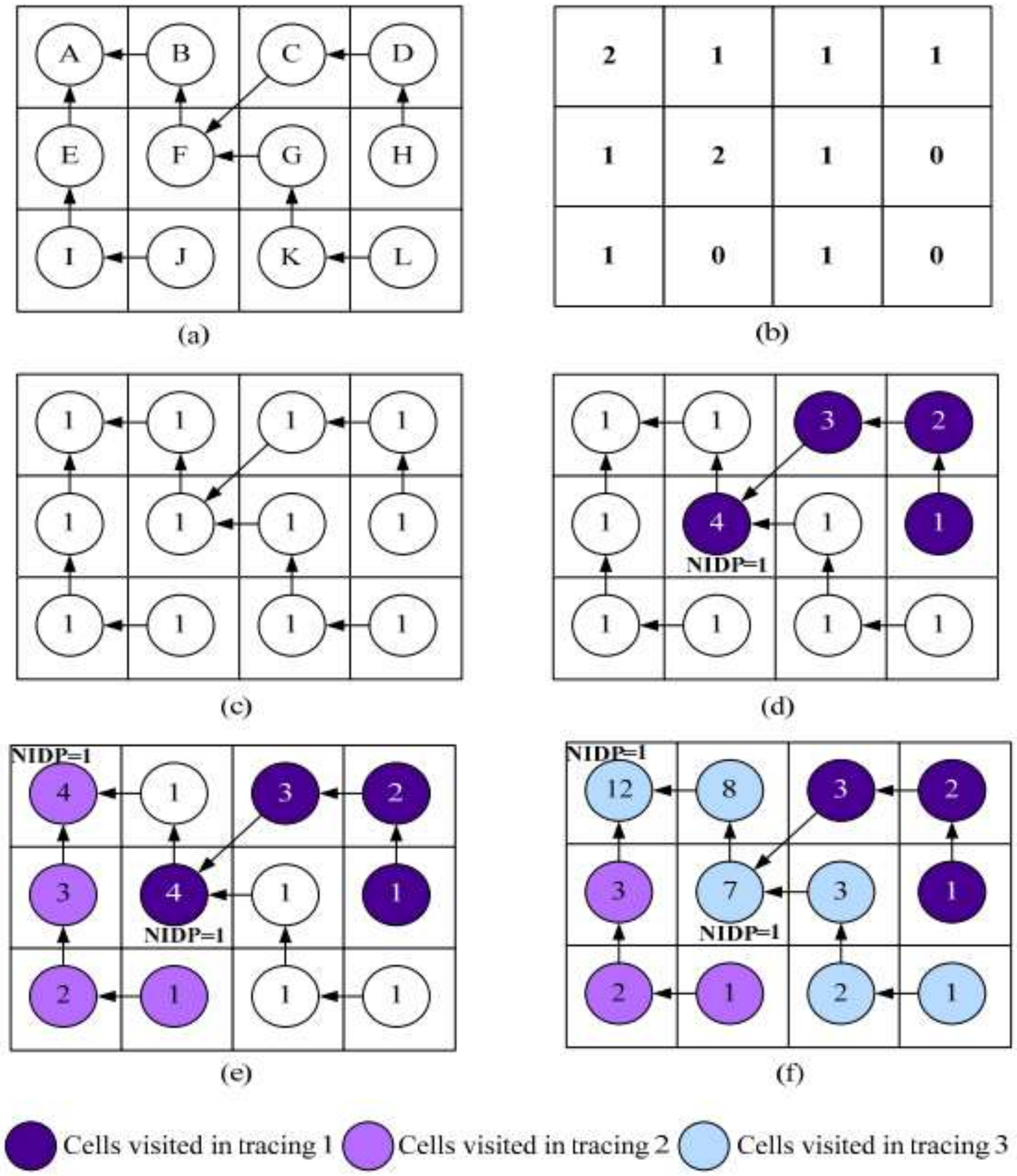
4 Faculty of Geographical Science, Beijing Normal University, Beijing 100875, China

快一点 再快一点

VITO RAY

有向图的深度优先搜索

Depth-First Search (DFS)



VITO RAY

HEY
WAKE UP!

VITO
RAY

D E S I G N E D B Y I B O T U

PART .04

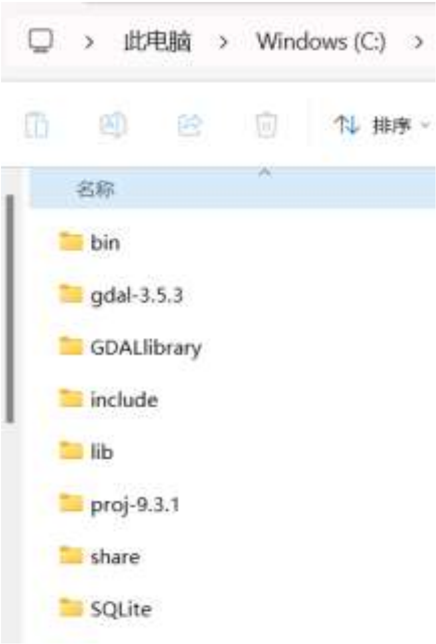
Talk is CHEAP

D E S I G N E D B Y I B O T U

DO SOME
THINGS!

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)

C with C++20

Cell 和Neighbour 构建有向图

栅格细胞Cell

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

Cell的邻居

```cpp
//定义八个方向的neighbour
std::vector<std::pair<int, int>> directions = {
    {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1} };
```

```cpp
// 定义8个方向的偏移量
std::vector<int> dx = { -1, -1, 0, 1, 1, 1, 0, -1 };
std::vector<int> dy = { 0, 1, 1, 1, 0, -1, -1, -1 };
```

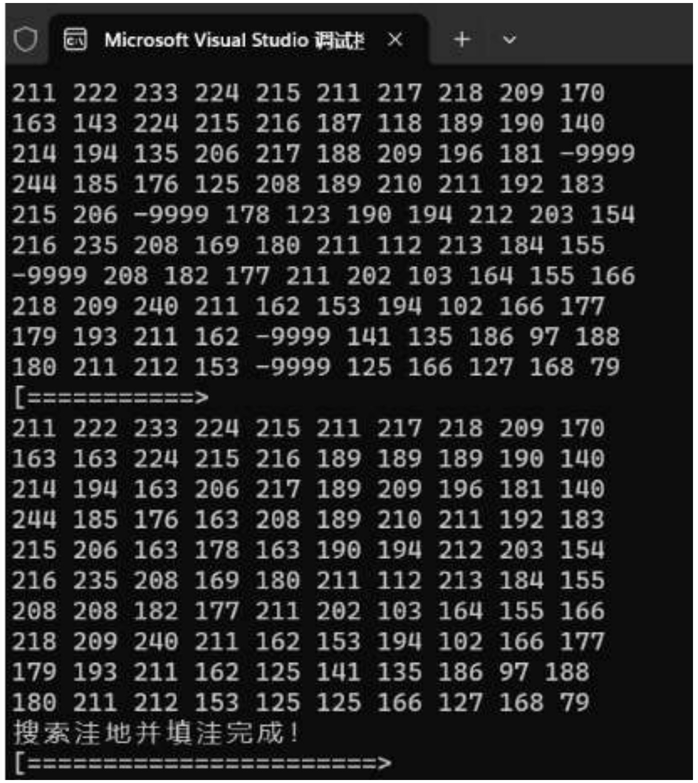
有向图的边

```cpp
// 定义方向查询表
std::vector<std::vector<int>> dir = {
    {8, 16, 32}, {4, 0, 64}, {2, 1, 128} };//流出查询表
std::vector<std::vector<int>> dir_receive = {
    {128, 1, 2}, {64, 0, 4}, {32, 16, 8} };//流入查询表
```

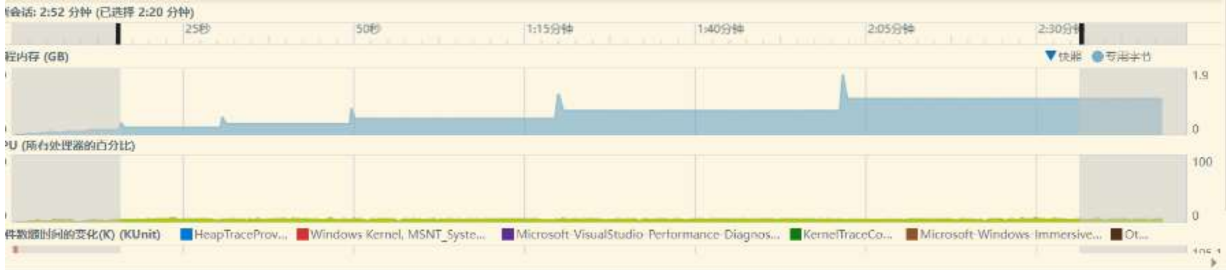
```cpp
// 定义8个方向的值
std::vector<int> d8 = { 16, 32, 64, 128, 1, 2, 4, 8 };
```

失败的PlanA

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

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

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

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

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

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

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

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

    // 返回Spill矩阵
    return Spill;
}
```



合理的结果，失败的内存管理



```cpp
//使用排水法实现填洼(Olivier Planchon & Frederic Darboux, 2002)
std::vector<std::vector<float>> Fill(const std::vector<std::vector<float>>& DEM) {
    int rows = DEM.size();
    int cols = DEM[0].size();
    // 遍历整个二维vector找到最大值
    int max_val = DEM[0][0];  // 初始化为第一行第一列的值
    for (const auto& row : DEM) {
        for (int val : row) {
            if (val > max_val) {
                max_val = val;
            }
        }
    }

    //初始化 Fill矩阵 (淹水)
    std::vector<std::vector<float>> Fill(rows, std::vector<float>(cols, max_val+1));
    //定义八个方向的neighbour
    std::vector<std::pair<int, int>> directions = {
        {-1, 0}, {-1, 1}, {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1} };

    //保留边界和Nodata原始值
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            if (i == 0 || j == 0 || i == rows - 1 || j == cols - 1 || DEM[i][j] == -9999) {
                Fill[i][j] = DEM[i][j];
            }
        }
    }
```

偷师ESRI

利用依赖图扫描

VITO RAY

偷师ESRI

利用依赖图扫描

```
//逐步排水
bool Fill_done = false;
while (!Fill_done) {
    Fill_done = true;
    //跳过边界
    for (int i = 1; i < rows-1; ++i) {
        for (int j = 1; j < cols-1; ++j) {
            //找到可能的洼地
            if (Fill[i][j] > DEM[i][j]) {
                for (const auto& dir : directions) {
                    int newRow = i + dir.first;
                    int newCol = j + dir.second;
                    //边界问题
                    if (newRow >= 0 && newRow < rows
                        && newCol >= 0 && newCol < cols && DEM[newRow][newCol] != -9999) {
                        //确保有出口的情况下，保证Fill 逼近 DEM
                        if (DEM[i][j] > DEM[newRow][newCol] + 0.0001) {
                            Fill[i][j] = DEM[i][j];
                            Fill_done = false;
                            break;
                        }
                        //如果没有已知的出口，逐渐排水保证至少有一个出口
                        if (Fill[i][j] > Fill[newRow][newCol] + 1) {
                            Fill[i][j] = Fill[newRow][newCol] + 1;
                            Fill_done = false;
                        }
                    }
```

VITO RAY

D8不那么完善的

```
// 检查所有8个邻居，从左下开始遍历
for (int di = -1; di <= 1; ++di) {
    for (int dj = -1; dj <= 1; ++dj) {
        int ni = i + di;
        int nj = j + dj;

        // 跳过中心单元格和没有邻居的方向
        if ((di == 0 && dj == 0) || ni < 0 || ni >= rows || nj < 0 || nj >= cols) continue;

        //对角线方向水平距离为根号2，其余方向水平距离为1
        if ((di == -1 && dj == -1) || (di == -1 && dj == 1) || (di == 1 && dj == -1) || (di == 1 && dj == 1))
        {
            if (fill[ni][nj] == -9999) continue;
            float slope = ( fill[ni][nj] - fill[i][j]) / 1.414;
            if (slope < down_slope) {
                down_slope = slope;
                min_dir = dir[di + 1][dj + 1];
            }
        }
        else if (fill[ni][nj] - fill[i][j] == 0 && flowDirection[ni][nj] == 0) {
            float slope = fill[ni][nj] - fill[i][j];
            // 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
            if (slope < down_slope) {
                min_dir = dir[di + 1][dj + 1];
            }
        }
```

```
            }
            else if (fill[ni][nj] - fill[i][j] == 0 && flowDirection[ni][nj] != 0) {
                float slope = fill[ni][nj] - fill[i][j];
                // 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
                if (slope < down_slope) {
                    min_dir = dir_receive[di + 1][dj + 1];
                }
            }
            else
            {
                if (fill[ni][nj] == -9999) continue;
                float slope = fill[ni][nj] - fill[i][j];
                if (slope < down_slope) {
                    down_slope = slope;
                    min_dir = dir[di + 1][dj + 1];
                }
            }
        }
    }
```

对自然的汇和开阔河道考虑不足

VITO RAY

DFS 构建有向图

```
    // 检查每个方向
    for (int k = 0; k < 8; ++k) {
        // 如果 c 单元格的流向是这个方向
        if (FlowDir[c.row][c.col] == d8[k]) {
            // 计算下游单元格的位置
            int newRow = c.row + dx[k];
            int newCol = c.col + dy[k];

            // 检查下游单元格是否在 DEM 的范围内
            if (newRow >= 0 && newRow < FlowDir.size() && newCol >= 0
                && newCol < FlowDir[0].size() && FlowDir[newRow][newCol] != -9999) {
                // 返回下游单元格，溢出量为 FlowDir 中相应位置的值
                return { newRow, newCol, static_cast<float>(FlowDir[newRow][newCol]) };
            }
            else {
                return { -1, -1, -1 };  // 假设行和列为-1表示没有下游单元格
            }
        }
    }

    // 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
    return { -1, -1, -1 };  // 假设行和列为-1表示没有下游单元格
}

```

根据流向返回下游细胞，并捕获异常

DFS 构建有向图

```cpp
    // 定义8个方向的偏移量
    std::vector<int> dx = { -1, -1, 0, 1, 1, 1, 0, -1 };
    std::vector<int> dy = { 0, 1, 1, 1, 0, -1, -1, -1 };
    std::vector<int> d8 = { 1, 2, 4, 8,16, 32, 64, 128 };

    // 遍历每一个单元格;
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            // 检查每个方向上的邻居
            for (int k = 0; k < 8; ++k) {
                int ni = i + dx[k];
                int nj = j + dy[k];

                // 检查邻居是否在 DEM 的范围内
                if (ni >= 0 && ni < rows && nj >= 0 && nj < cols) {
                    // 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
                    if (FlowDir[ni][nj] == d8[k]) {
                        NIDP[i][j]++;
                    }
                }
            }
        }
    }

    return NIDP;
}
```

把细胞作为节点Node，根据入边数量确定节点类型

VITO
RAY

DFS 利用有向图

```cpp
    // 遍历每一个单元格
    for (int row = 0; row < FlowDir.size(); ++row) {
        for (int col = 0; col < FlowDir[0].size(); ++col) {
            Cell c = { row, col , FlowDir[row][col] };
            if (NIDP[row][col] != 0) continue;
            Cell n = c;
            int nAccu = 0;
            do {
                FlowAccu[n.row][n.col] += nAccu;
                nAccu = FlowAccu[n.row][n.col];
                //交叉细胞已被访问过一次，NIDP-1
                if (NIDP[n.row][n.col] > 1) {
                    NIDP[n.row][n.col]--;
                    break;
                }
                //从源细胞开始向下游遍历
                n = NextCell(n, FlowDir);
            } while (n.row != -1 && n.row < rows
                && n.col != -1 && n.col < cols);//直到这条流线流出DEM范围
        }
    }
    return FlowAccu;
}
```

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 U

PART .05

数据验证

VITO
RAY

D E S I G N E D B Y I B O T U

DO SOME
THINGS!

数据集验证

72行，57列

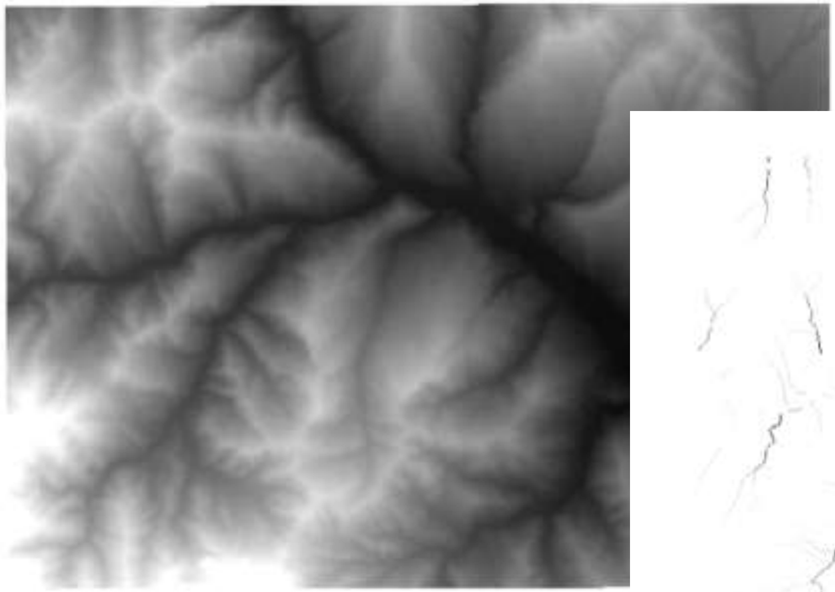
Property	Value
Raster Information	
Columns and Rows	72, 57
Number of Bands	1
Cell Size (X, Y)	28.93513394, 28.93513394
Uncompressed Size	16.03 KB
Format	GRID
Source Type	Generic
Pixel Type	signed integer
Pixel Depth	16 Bit

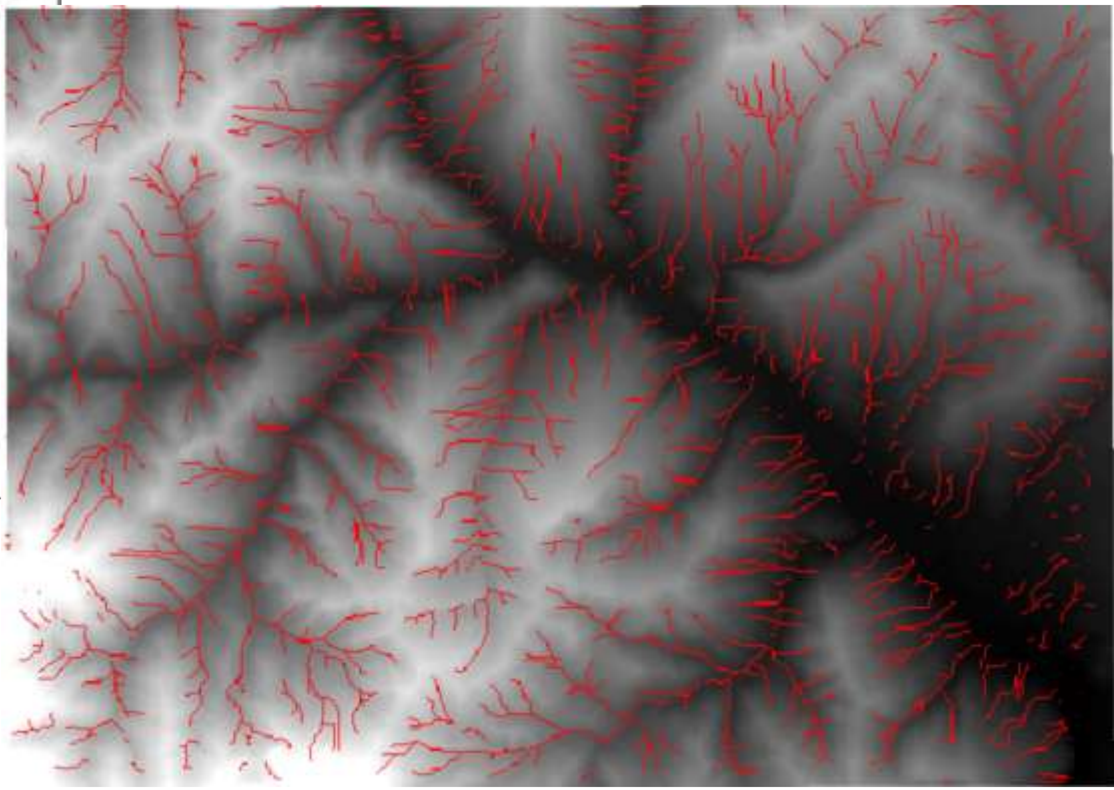
VITO RAY

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!

VITO
RAY

D E S I G N E D B Y I B O T U

PART .06

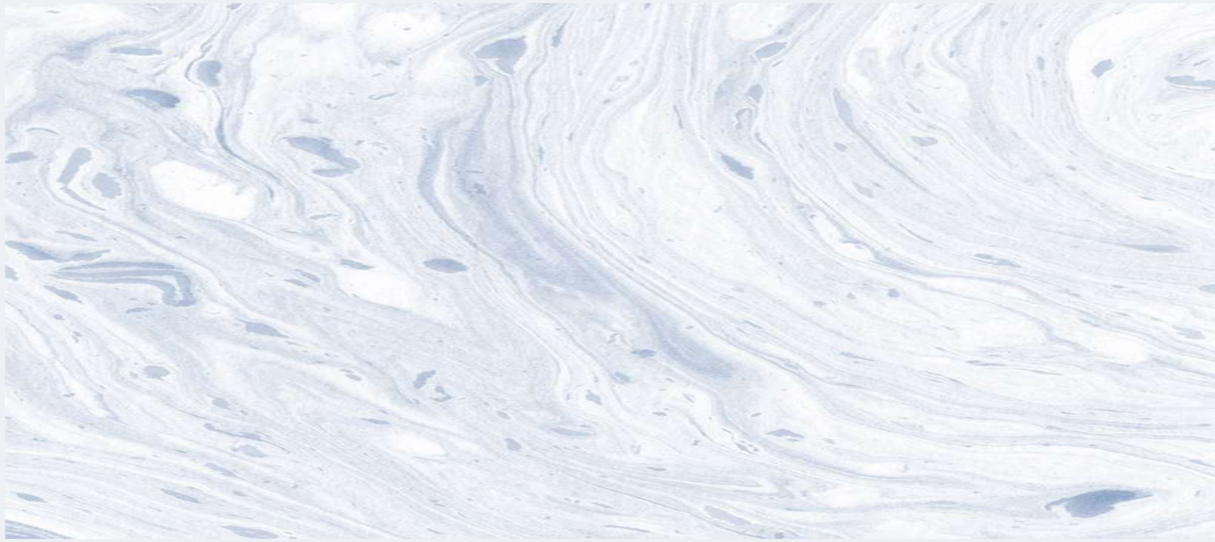
性能

D E S I G N E D B Y I B O T U

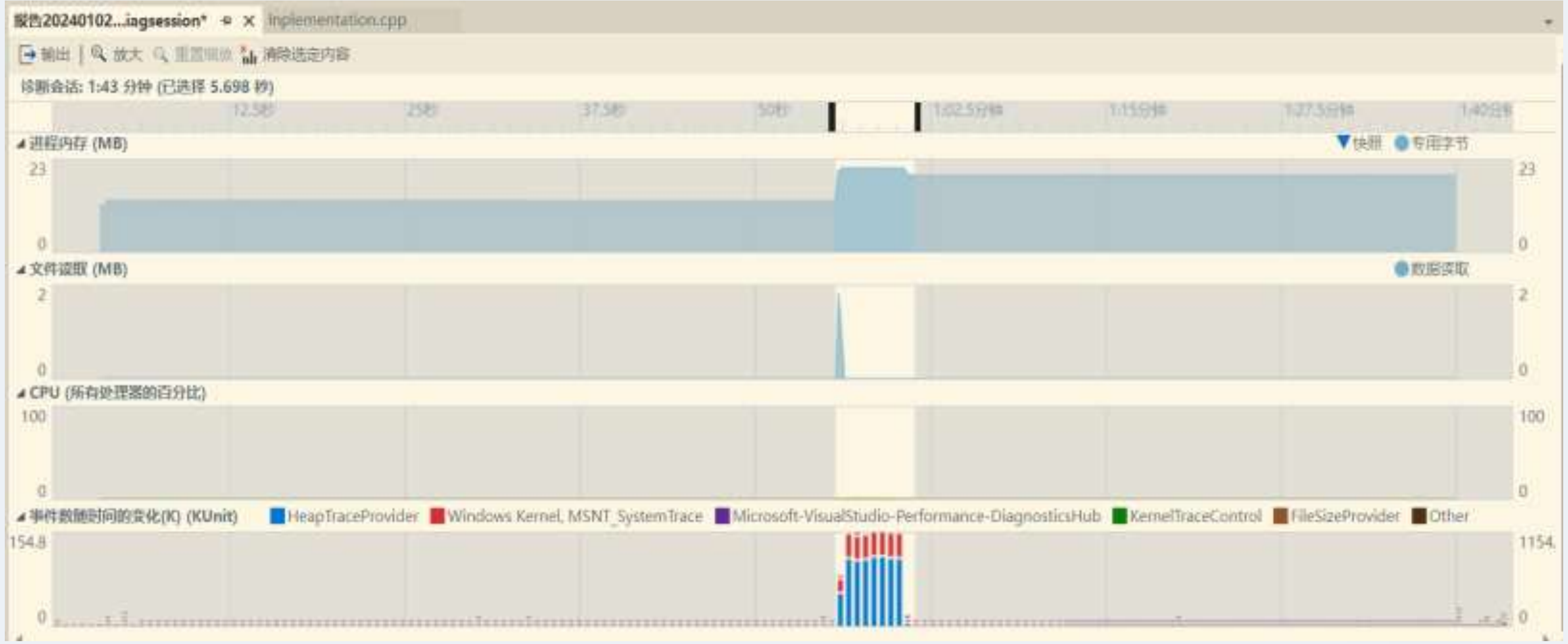
DO SOME
THINGS!

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!

VITO
RAY

DESIGNED BY IBOTU

谢谢观看

DESIGNED BY IBOTU

DO SOME
THINGS!