代码说明：三个小问的代码写在一个工程中，使用 CMakeLists.txt 里面的 #define 来区分

```
add_definitions(-DPROBLEM_1)
# add_definitions(-DPROBLEM_2)
# add_definitions(-DPROBLEM_3)
```

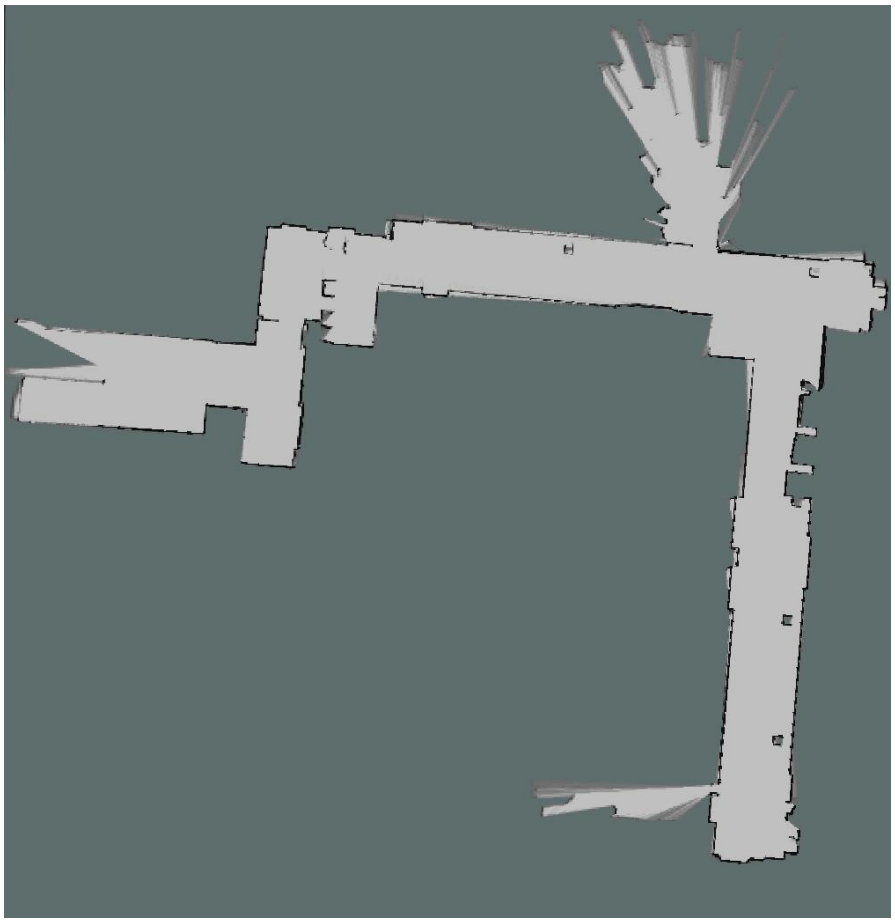
代码结构如下所示：

```
            //start of TODO 对对应的map的cell信息进行更新. (1,2,3题内容)

            GridIndex grid_x_y = ConvertWorld2GridIndex(world_x, world_y);

            if (isValidGridIndex(grid_x_y) == false)
                continue;

            GridIndex robotPose_grid = ConvertWorld2GridIndex(robotPose[0], robotPose[1]);

            //得到所有的被激光通过的index，并且更新栅格
            std::vector<GridIndex> miss_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, grid_x_y.x, grid_x_y.y);

#ifdef PROBLEM_1
#endif // PROBLEM_1

#ifdef PROBLEM_2
#endif //PROBLEM_2

#ifdef PROBLEM_3
#endif //PROBLEM_3

            //end of TODO
        }
    }
      You, 3 minutes ago • Uncommitted changes
    //start of TODO 通过计数建图算法或TSDF算法对栅格进行更新 (2,3题内容)
#ifdef PROBLEM_2
#endif //PROBLEM_2

#ifdef PROBLEM_3
#endif //PROBLEM_3

    //end of TODO
```

1. 补充代码，通过覆盖栅格建图算法进行栅格地图构建；（3 分）

```
            //start of TODO 对对应的map的cell信息进行更新. (1,2,3题内容)

            GridIndex grid_x_y = ConvertWorld2GridIndex(world_x, world_y);

            if (isValidGridIndex(grid_x_y) == false)
                continue;

            GridIndex robotPose_grid = ConvertWorld2GridIndex(robotPose[0], robotPose[1]);

            //得到所有的被激光通过的index，并且更新栅格
            std::vector<GridIndex> miss_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, grid_x_y.x, grid_x_y.y);

#ifdef PROBLEM_1
            // 更新被经过的点
            for (size_t j = 0; j < miss_grids.size(); j++) {
                GridIndex tmpIndex = miss_grids[j];
                int linear_index = GridIndexToLinearIndex(tmpIndex);
                pMap[linear_index] += mapParams.log_free;
                pMap[linear_index] = max(mapParams.log_min, double(pMap[linear_index]));
            }

            //更新被击中的点
            int linear_index = GridIndexToLinearIndex(grid_x_y);
            pMap[linear_index] += mapParams.log_occ;
            pMap[linear_index] = min(mapParams.log_max, double(pMap[linear_index]));
#endif // PROBLEM_1

#ifdef PROBLEM_2…
#endif //PROBLEM_2

#ifdef PROBLEM_3…
#endif //PROBLEM_3

            //end of TODO
        }
    }
    You, a few seconds ago • Uncommitted changes
```

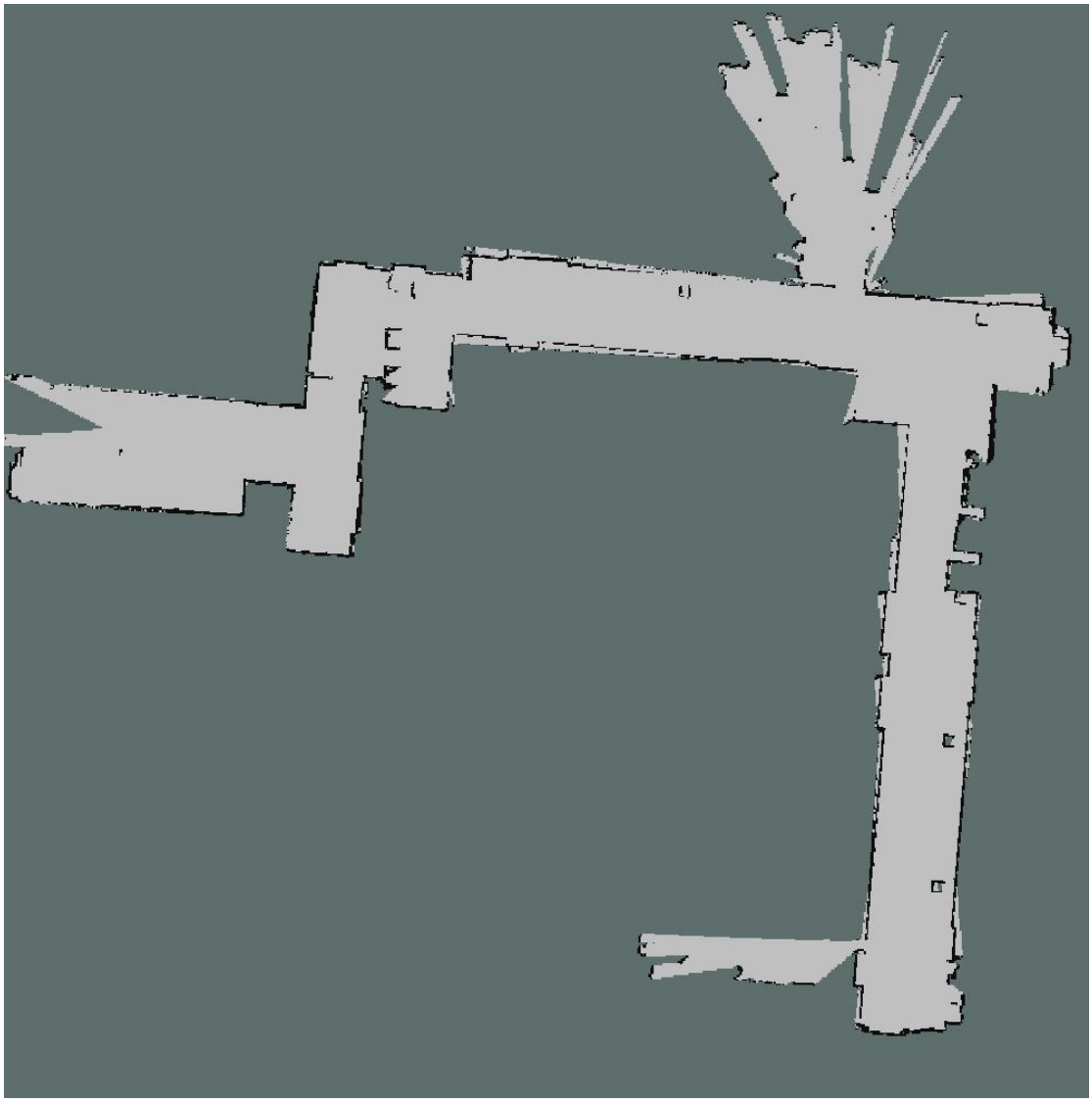
效果如下所示：



输出结果中间有些区域颜色比较淡，修改 pub 地图的函数如下所示，将 pMap 中小于 50 的栅格赋值为 0，大于 50 的栅格赋值为 100.

```
for (int i = 0; i < mapParams.width * mapParams.height; i++) {
    if (pMap[i] == 50) {
        rosMap.data[i] = -1.0;
    } else if (pMap[i] < 50) {
        rosMap.data[i] = 0;
    } else if (pMap[i] > 50) {
        rosMap.data[i] = 100;
    }
}
```

输出结果如下所示，障碍物与非障碍物区域的对比效果更清晰。



不过这样做一些区域的障碍物被忽略掉了，比如左下角这里：

修改地图输出效果前：

修改地图输出效果后：

2. 将第 1 题代码改为通过计数建图算法进行栅格地图构建；（3 ）

代码：

计数

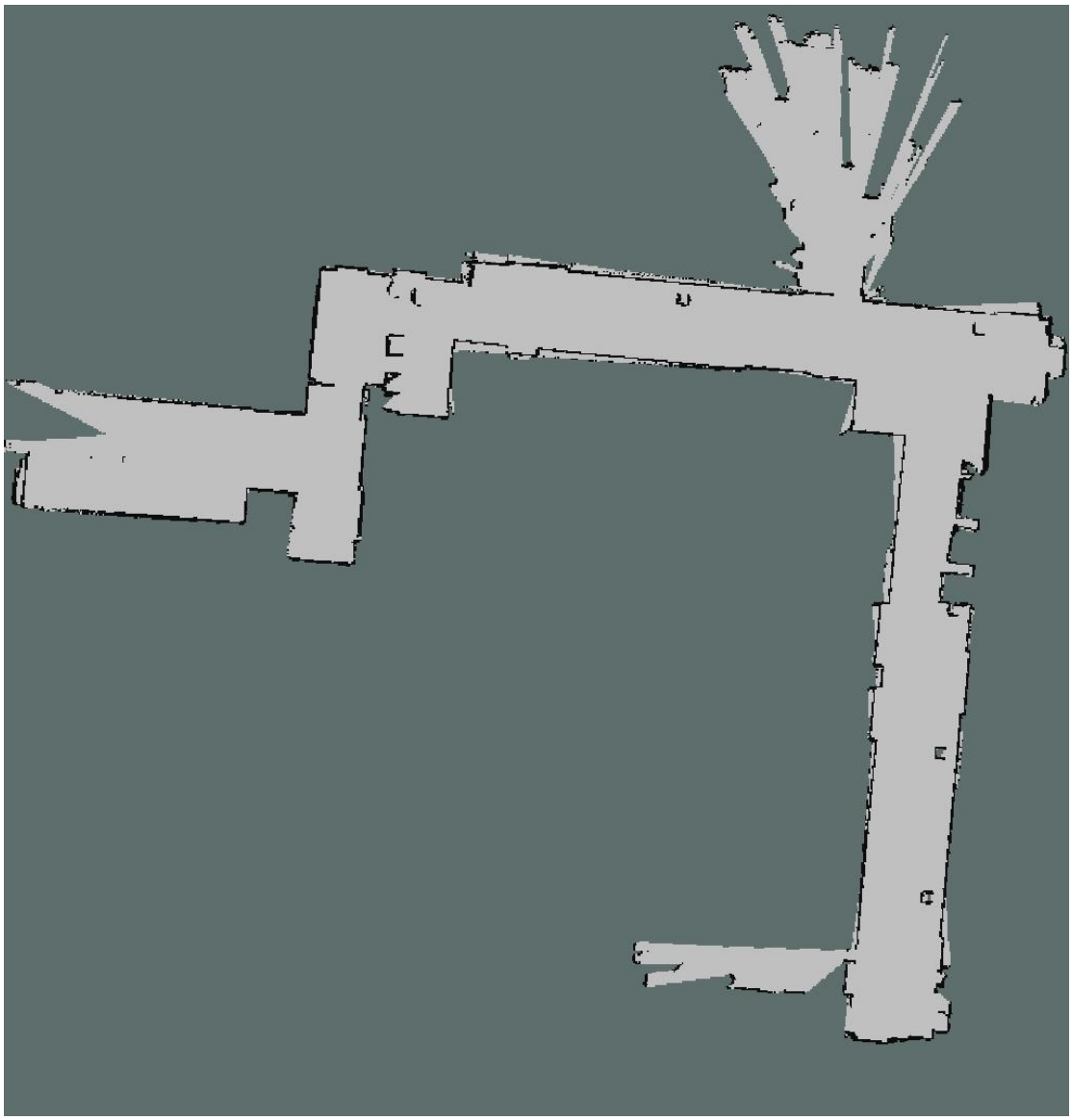
```
            GridIndex grid_x_y = ConvertWorld2GridIndex(world_x, world_y);

            if (isValidGridIndex(grid_x_y) == false)
                continue;

            GridIndex robotPose_grid = ConvertWorld2GridIndex(robotPose[0], robotPose[1]);

            //得到所有的被激光通过的index，并且更新栅格
            std::vector<GridIndex> miss_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, grid_x_y.x, grid_x_y.y);
```

```
#ifdef PROBLEM_1…
#endif // PROBLEM_1

#ifdef PROBLEM_2
            // 更新被经过的点
            for (size_t j = 0; j < miss_grids.size(); j++) {
                GridIndex tmpIndex = miss_grids[j];
                int linear_index = GridIndexToLinearIndex(tmpIndex);
                pMapMisses[linear_index]++;
            }

            //更新被击中的点
            int linear_index = GridIndexToLinearIndex(grid_x_y);
            pMapHits[linear_index]++;
#endif //PROBLEM_2
```

计算比例，并赋值得 pMap

```
    //start of TODO 通过计数建图算法或TSDF算法对栅格进行更新（2,3题内容）
#ifdef PROBLEM_2
    for (int i = 0; i < mapParams.width * mapParams.height; i++) {
        if (pMapHits[i] + pMapMisses[i] == 0) {
            pMap[i] = 50;
            continue;
        }

        double ratio = pMapHits[i] * 1.0 / (pMapHits[i] + pMapMisses[i]);
        double ratio_threshold = 0.3;

        if (ratio > ratio_threshold) {
            pMap[i] = 100;
        } else if (ratio <= ratio_threshold) {
            pMap[i] = 0;
        }
    }
#endif //PROBLEM_2

#ifdef PROBLEM_3…
#endif //PROBLEM_3

    //end of TODO
    std::cout << "建图完毕" << std::endl;
```

输出效果如下所示：



3. 将第 1 题代码改为通过 TSDF 建图算法进行栅格地图构建；（4 ）

代码：

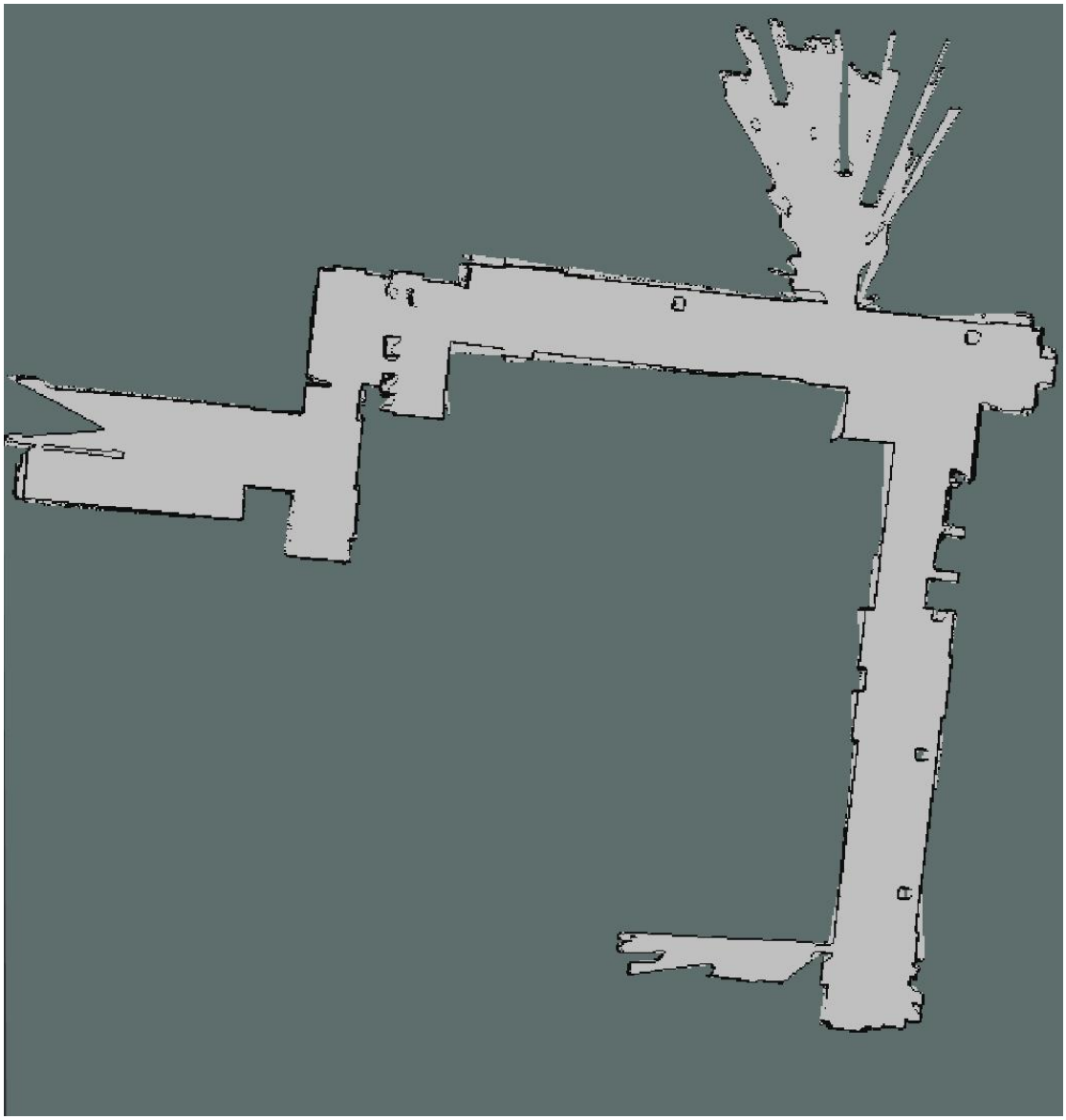
计算 TSDF

```
    //start of TODO 对对应的map的cell信息进行更新. (1,2,3题内容)

    GridIndex grid_x_y = ConvertWorld2GridIndex(world_x, world_y);

    if (isValidGridIndex(grid_x_y) == false)
        continue;

    GridIndex robotPose_grid = ConvertWorld2GridIndex(robotPose[0], robotPose[1]);

    //得到所有的被激光通过的index，并且更新栅格
    std::vector<GridIndex> miss_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, grid_x_y.x, grid_x_y.y);

› PROBLEM_1 ⋯
  // PROBLEM_1

› PROBLEM_2 ⋯
  //PROBLEM_2

  PROBLEM_3
    // tsdf 截断距离
    double cut_off_dis = 2 * mapParams.resolution;
    double far_dis;

    // 计算远点      You, a few seconds ago • Uncommitted changes
    far_dis = dist + 3 * cut_off_dis;

    double far_laser_x = far_dis * cos(angle);
    double far_laser_y = far_dis * sin(angle);

    double far_world_x = cos(theta) * far_laser_x - sin(theta) * far_laser_y + robotPose[0];
    double far_world_y = sin(theta) * far_laser_x + cos(theta) * far_laser_y + robotPose[1];

    GridIndex far_grid_x_y = ConvertWorld2GridIndex(far_world_x, far_world_y);

    std::vector<GridIndex> near_grids;
    // 如果增加的距离是远点超出地图范围，那么就是用激光点作为远点
    if (isValidGridIndex(far_grid_x_y) == false) {
        near_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, grid_x_y.x, grid_x_y.y);
    } else {
        near_grids = TraceLine(robotPose_grid.x, robotPose_grid.y, far_grid_x_y.x, far_grid_x_y.y);
    }

    // 更新laser_dist附近的栅格
    for (size_t j = 0; j < near_grids.size(); j++) {
        GridIndex tmpIndex = near_grids[j];
        double grid_dis = sqrt(pow(tmpIndex.x - robotPose_grid.x, 2) + pow(tmpIndex.y - robotPose_grid.y, 2));

        // 从gridmap尺度转化为实际地图尺度
        grid_dis *= mapParams.resolution;

        // 计算tsdf
        double tsdf = max(-1.0, min(1.0, (dist - grid_dis) / cut_off_dis));

        int linearIndex = GridIndexToLinearIndex(tmpIndex);

        // 更新TSDF
        pMapTSDF[linearIndex] = (pMapW[linearIndex] * pMapTSDF[linearIndex] + tsdf) / (pMapW[linearIndex] + 1);
        pMapW[linearIndex] += 1;
        // cout << "sdf: " << dist - grid_dis << endl;
        // cout << "tsdf:     " << tsdf << endl;
        // cout << "grid_dis: " << grid_dis << endl;
    }
    // cout << "laser_dis: " << dist << endl;
    // cout << "----------" << endl;
  //PROBLEM_3

    //end of TODO
  }
```

广度优先搜索找到所有的边界处，并赋值对应 pMap

```
    //start of TODO 通过计数建图算法或TSDF算法对栅格进行更新（2,3题内容）
#ifdef PROBLEM_2…
#endif //PROBLEM_2

#ifdef PROBLEM_3
    // BFS查找边界
    pMapIsAccessed = new int[mapParams.width * mapParams.height];
    std::vector<std::pair<int, int>> directions = { { 1, 0 }, { 0, 1 }, { -1, 0 }, { 0, -1 } };
    queue<GridIndex> q;
    q.push(GridIndex(0, 0));

    while (!q.empty()) {
        int size = q.size();
        for (size_t i = 0; i < size; i++) {
            GridIndex node = q.front();
            q.pop();
            for (size_t k = 0; k < 4; k++) {
                pair<int, int> dir = directions[k];
                int tmpx = node.x + dir.first;
                int tmpy = node.y + dir.second;
                int linearIndex = tmpy + tmpx * mapParams.width;
                int centerIndex = node.y + node.x * mapParams.width;

                if (tmpx >= 0 && tmpx < mapParams.height && tmpy >= 0 && tmpy < mapParams.width && pMapIsAccessed[linearIndex] == 0) {
                    // 对于边界点的处理
                    if (pMapTSDF[linearIndex] * pMapTSDF[centerIndex] < 0) {
                        // 选择pMapTSDF绝对值较小的一个栅格，认为是障碍物所在的栅格
                        if (abs(pMapTSDF[linearIndex]) < abs(pMapTSDF[centerIndex])) {
                            pMap[linearIndex] = 100;
                        } else {
                            pMap[centerIndex] = 100;
                        }
                    }
                    // 访问过的点不再进行访问
                    pMapIsAccessed[linearIndex] = 1;
                    q.push(GridIndex(tmpx, tmpy));
                }
            }
        }
    }
#endif //PROBLEM_3

    //end of TODO
```

输出效果如下所示：



4. 简答题，开放性答案：总结比较课堂所学的 3 种建图算法的优劣 （2 ）

1）覆盖栅格建圖算法对栅格进行更新只需要进行加法操作，具有较高的更新速度。

2）计数建圖算法实现简单，但是缺点是每个栅格需要两个计数变量 Hits 和 Misses，内存占用较多。

3）上述两个方法没有考虑激光噪声，而 TSDF 建圖算法将多帧数据一起考虑

如果传感器的噪声服从高斯分布，那么通过 TSDF 进行融合，等价于通过最小二乘来进行融合，能比较好的进行曲面重构。

能够插值出确切的曲面位置，构建的地图最多只有一个栅格的厚度。不过该算法相比于

前两种算法，计算复杂度较高