

地图块存储加载流程设计

|  |  |
| --- | --- |
| 文档版本 | V 1.0.0 |
| 发布日期 | 2022-07-04 |

**版权所有©舵敏智能2021，保留一切权力。**

非经本公司书面许可，任何单位或个人不得擅自摘抄、复制部分或全部本文档，不得以任何形式传播。

**商标声明**

、舵敏、舵敏智能和其他舵敏商标均为舵敏智能科技的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人所有。

**上海舵敏智能科技有限公司**

**舵敏智能科技（苏州）有限公司**

|  |  |
| --- | --- |
| 网址: | https://www.dominant-tech.com/ |
| 客户服务邮箱: | support@dominant-tech.com |

**目 录**

[1、 地图分块存储 2](#_Toc512985802)

[1.1创建关键帧块与地图点块 2](#_Toc1310184905)

[1.2判断是否存储关键帧块与地图点块 5](#_Toc1023181860)

[1.3存储地图点块、关键帧块 8](#_Toc1554334090)

[1.4更新与存储关键帧块信息表与地图点块信息表 12](#_Toc2064760703)

[1.5删除已存储的关键帧、地图点 15](#_Toc1798126412)

[2、 地图分块加载 17](#_Toc2001872829)

[2.1加载关键帧块与地图点块信息表 17](#_Toc2036252844)

[2.2查询并加载需要加载的关键帧块与地图点块 18](#_Toc701721520)

[2.3加载地图点块与关键帧块 20](#_Toc558192559)

[2.4删除关键帧块和地图点块 22](#_Toc314431595)

[3、 Atlas主函数 25](#_Toc251596106)

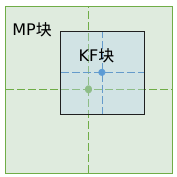
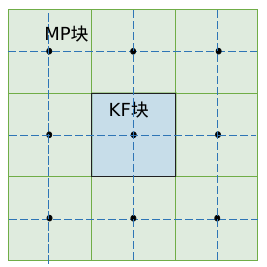
**分为关键帧块和地图点块的原因**

如果将关键帧块和地图点块存放在一起，那么块交界处的关键帧之间会存在大量共视的地图点。如图，一个关键帧观测到的地图点位于其他地图块，则加载时需要把这些子地图加载，但是并不需要加载其中的关键帧。所以将关键帧块和地图点块分开存储。

**子地图概念说明**

子地图包含一个关键帧块 以及 该关键帧块观测到的所有地图点块。

**关键帧块与地图点块的尺寸说明**



是为了减少地图块的索引。对关键帧块来说，他所观测到的地图点范围广，如果关键帧块的尺寸和地图点块一样的话，加载一个关键帧块，不仅要加载当前地图点块，还要加载其周围的的地图点块。因此，地图点块的尺寸大于关键帧块的尺寸。

# 地图分块存储

## 1.1创建关键帧块与地图点块

1. **创建关键帧块**
2. 函数签名

void Atlas::CreateKeyFremeBlock(KFBlockPos &, int )

1. 实现功能

针对该范围的KF创建临时的KF块

1. 输入输出

输入：KFBlockPos关键帧块中心坐标, KFBlockScale关键帧块尺度

输出：void

1. 细节描述
   1. 在三维世界坐标系（暂以二维表示）将关键帧按照N\*N（\*N）（N为整数，单位：m）分割为块，如图1所示；

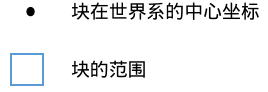
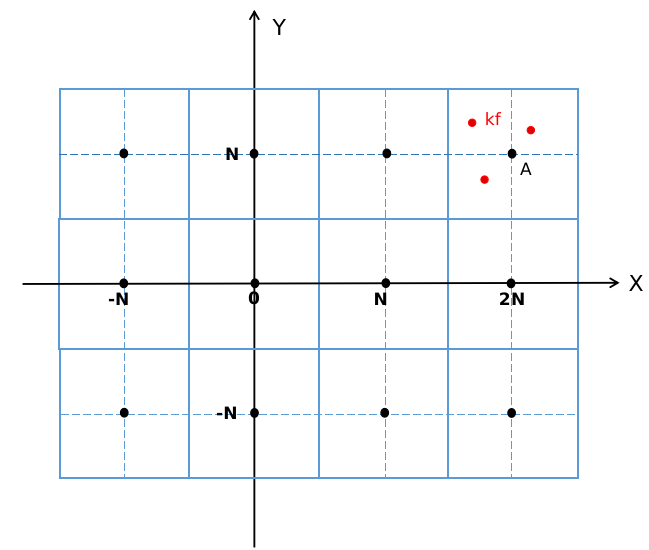


图1关键帧块示意图

* 1. 关键帧块的编号：关键帧块中心坐标/N；如关键帧块A的中心坐标（2N，N），关键帧块中心坐标为（2，1），则其编号为2\_1；（注：关键帧块坐标以vector存储，可扩展至三维）
  2. 关键帧分块依据：关键帧块范围内的所有关键帧作为一个关键帧块；
  3. 关键帧所属块中心坐标的计算：

如果KF.x > 0 则所属块的x坐标为：(KF.x + N/2) / N

KF.x < 0 则所属块的x坐标为：(KF.x - N/2) / N

如果KF.y > 0 则所属块的y坐标为：(KF.y + N/2) / N

KF.y < 0 则所属块的y坐标为：(KF.y - N/2) / N

* 1. 程序流程图

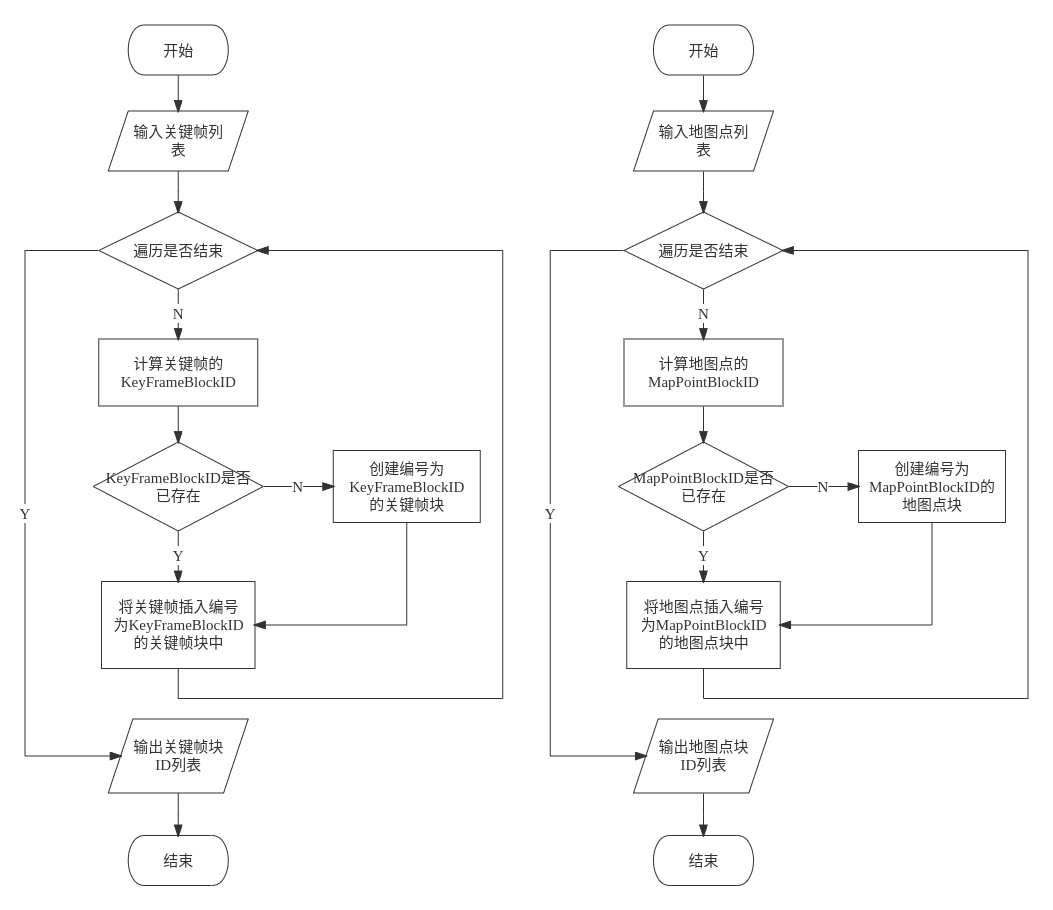


图2关键帧分块流程图

1. **创建地图点块**
2. 函数名

void Atlas::CreateMapPointBlock(MPBlockPos &, int)

1. 实现功能

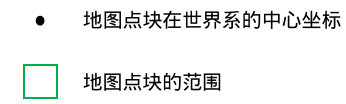
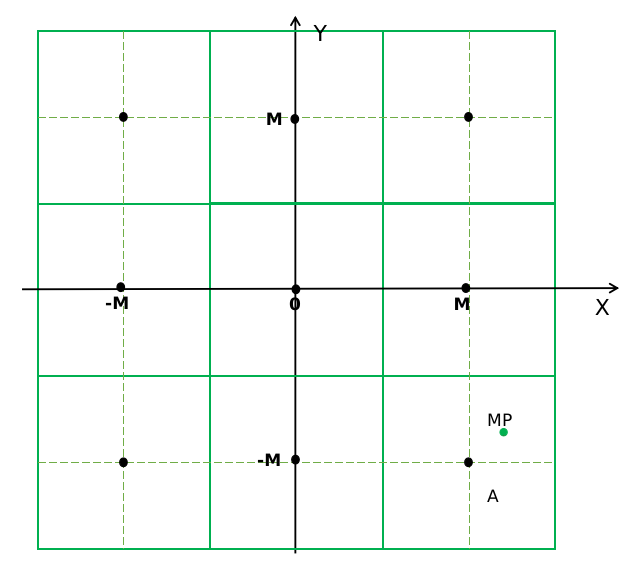
创建地图点块 存放 某范围内的所有地图点

1. 输入输出

输入：MPBlockPos地图点块中心坐标, MPBlockScale地图点块尺度

输出：void

1. 细节描述
   1. 在三维世界坐标系（暂以二维表示）将地图点按照M\*M（\*M）（M为整数，单位：m）分割为块，如图2所示；



* 1. 地图点块的编号：地图点块中心坐标/M；如地图点块A的中心坐标（M，-M），关键帧块中心坐标为（1，-1），则块编号为1\_-1；（注：地图点块中心坐标以vector存储，可扩展至三维）
  2. 地图点分块依据：块范围内的所有地图点作为一个地图点块；
  3. 地图点所属块的计算：

如果MP.x > 0 则所属块的x坐标为：(MP.x + M/2) / M

MP.x < 0 则所属块的x坐标为：(MP.x - M/2) / M

如果MP.y > 0 则所属块的y坐标为：(MP.y + M/2) / M

MP.y < 0 则所属块的y坐标为：(MP.y - M/2) / M

* 1. 程序流程图

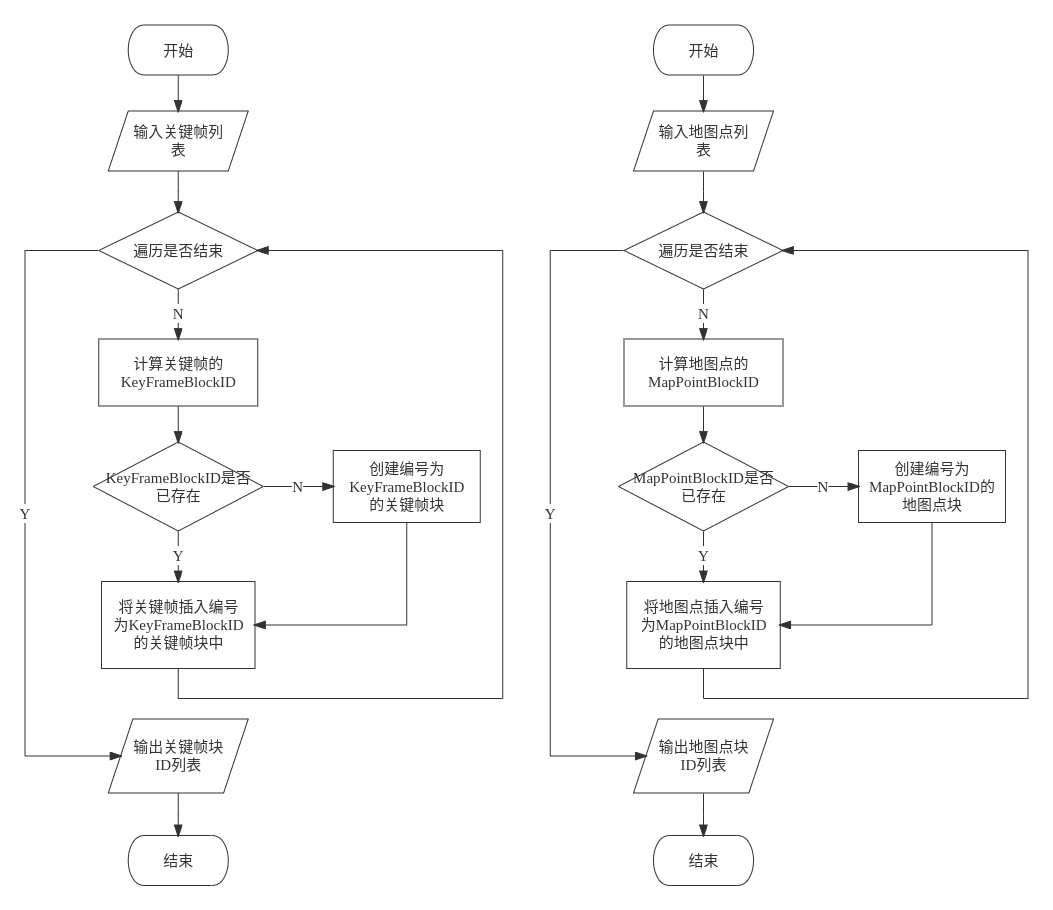


图3地图点分块流程图

## 1.2判断是否存储关键帧块与地图点块

一、判断是否存储关键帧块

1. 函数名

vector<KeyFrame\*> Atlas::CheckKeyFrameSave(KeyFrame\* , bool)

1. 功能描述

检测地图中的关键帧序列

检测窗口内的关键帧 中的是否符合存储条件。

1. 输入输出

输入：当前正在使用的关键帧，系统是否停止

输出：vector<KeyFrame\*>能够存储的关键帧序列；

1. 细节描述
2. 要存储的关键帧序列窗口：

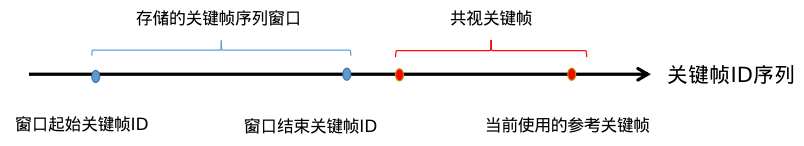
窗口起始关键帧ID：mStratKeyFrameID

窗口结束关键帧ID：mStratKeyFrameID+th

1. 存储满足条件：

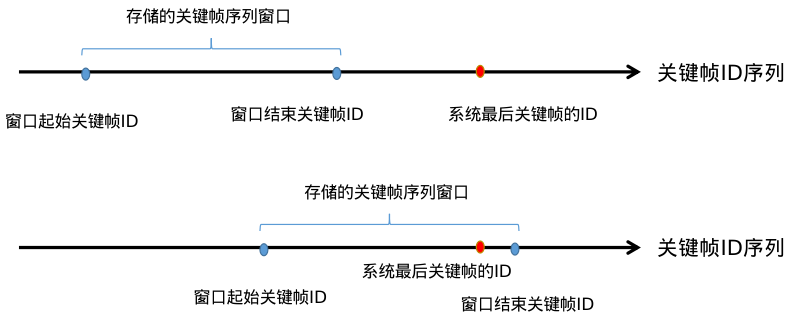
i系统正常运行时，判断条件如下：

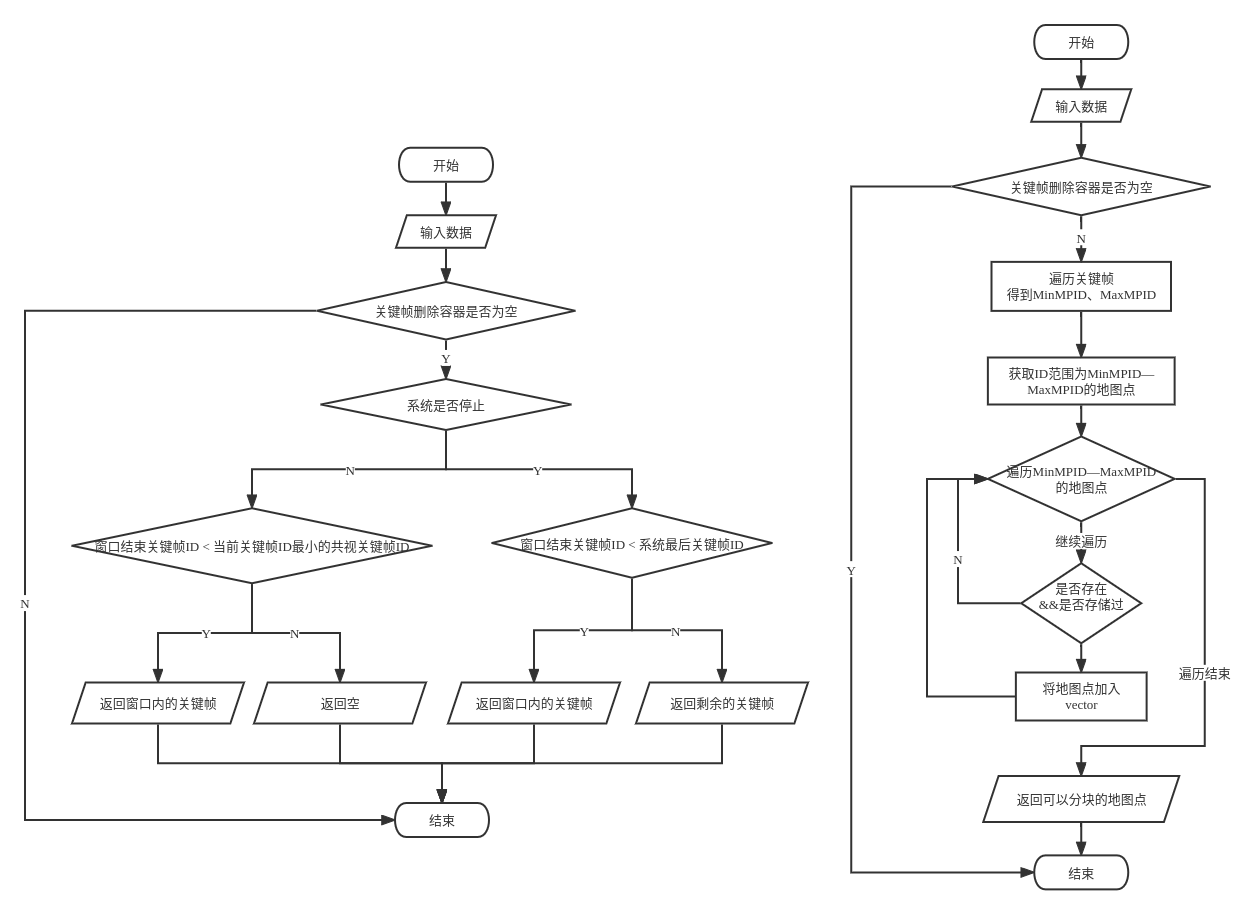
窗口结束关键帧ID < 当前关键帧所有共视关键帧最小的ID



ii系统停止运行时，判断条件如下：

窗口结束关键帧ID < 系统最后(最大)的关键帧ID





二、判断是否存储地图点块

1. 函数名

vector<MapPoint\*> Atlas::CheckMapPointSave(vector<KeyFrame\*>)

1. 功能描述

检测地图点中是否符合存储条件。

1. 输入输出

输入：vector<KeyFrame\*>符合存储条件的关键帧；

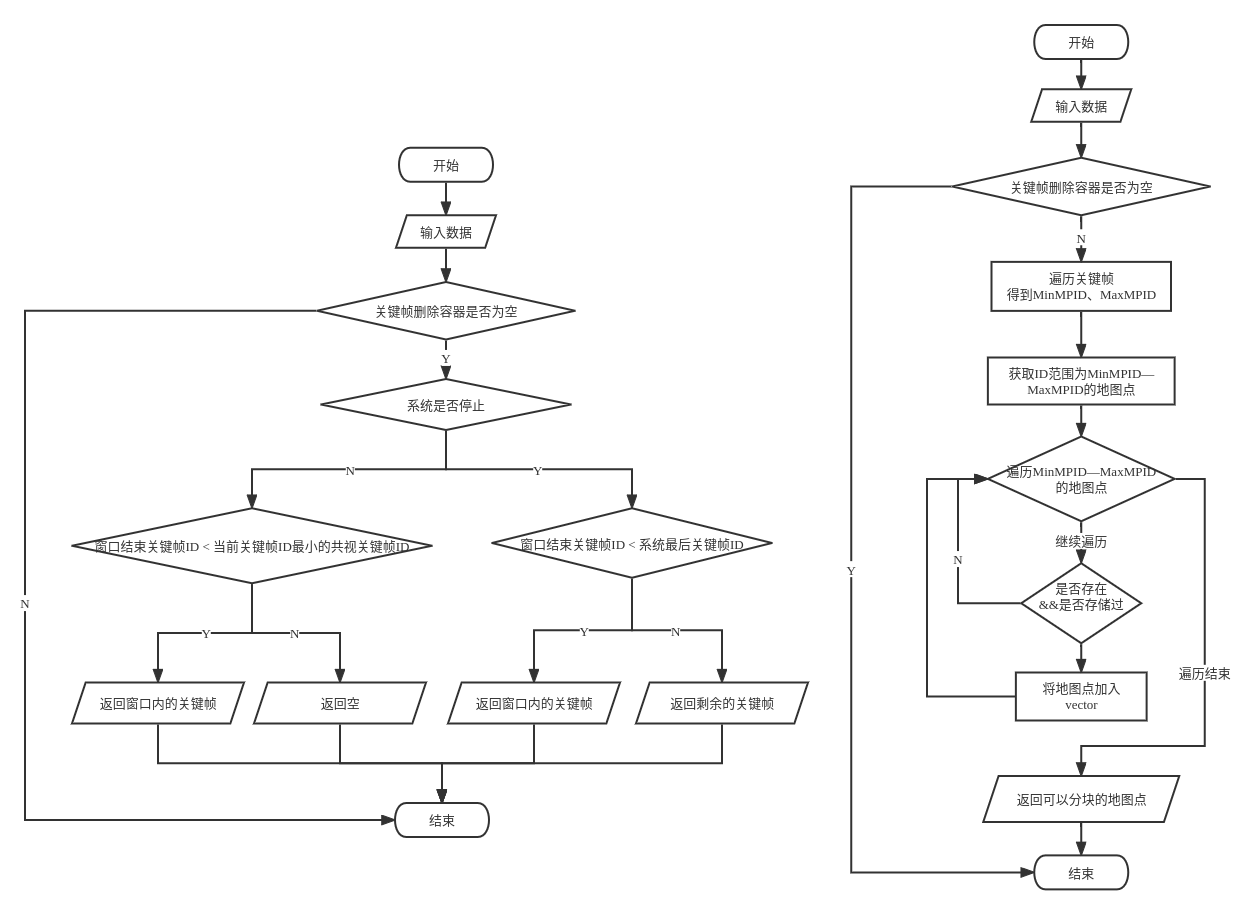
输出：vector<MapPoint\*>能够存储的地图点序列；

1. 细节描述
2. 遍历关键帧

获取ID最小地图点对应的ID：MinMPID；

ID最大地图点对应的ID：MaxMPID；

1. 获取ID范围为MinMPID—MaxMPID的地图点；
2. 筛选出存在且没有被保存过的地图点；



## 1.3存储地图点块、关键帧块

1. **存储地图点块**
2. 函数名

vector<MPBlockPos> Atlas::AddMapPointToBlock(vector<MapPoint\*>)

1. 功能描述

将地图点放入临时地图点块

1. 输入输出

输入：vector<MapPoint\*>

输出：vector<MPBlockPos> 地图点块的中心坐标

1. 函数名

void Atlas::MergeMapPointBlock()

1. 功能描述

将临时地图点块和已经存储的地图点块融合，并把融合后的地图点块放入待存储地图点块。

1. 输入输出

输入：void

输出：void

1. 函数名

void Atlas::MapPointBlockSave()

1. 功能描述

存储地图点块存储容器中的地图点块。

1. 输入输出

输入：void

输出：void

存储地图点块细节描述

* 1. 地图点块融合

判断临时地图点块是否存在（也就是是否存储过）

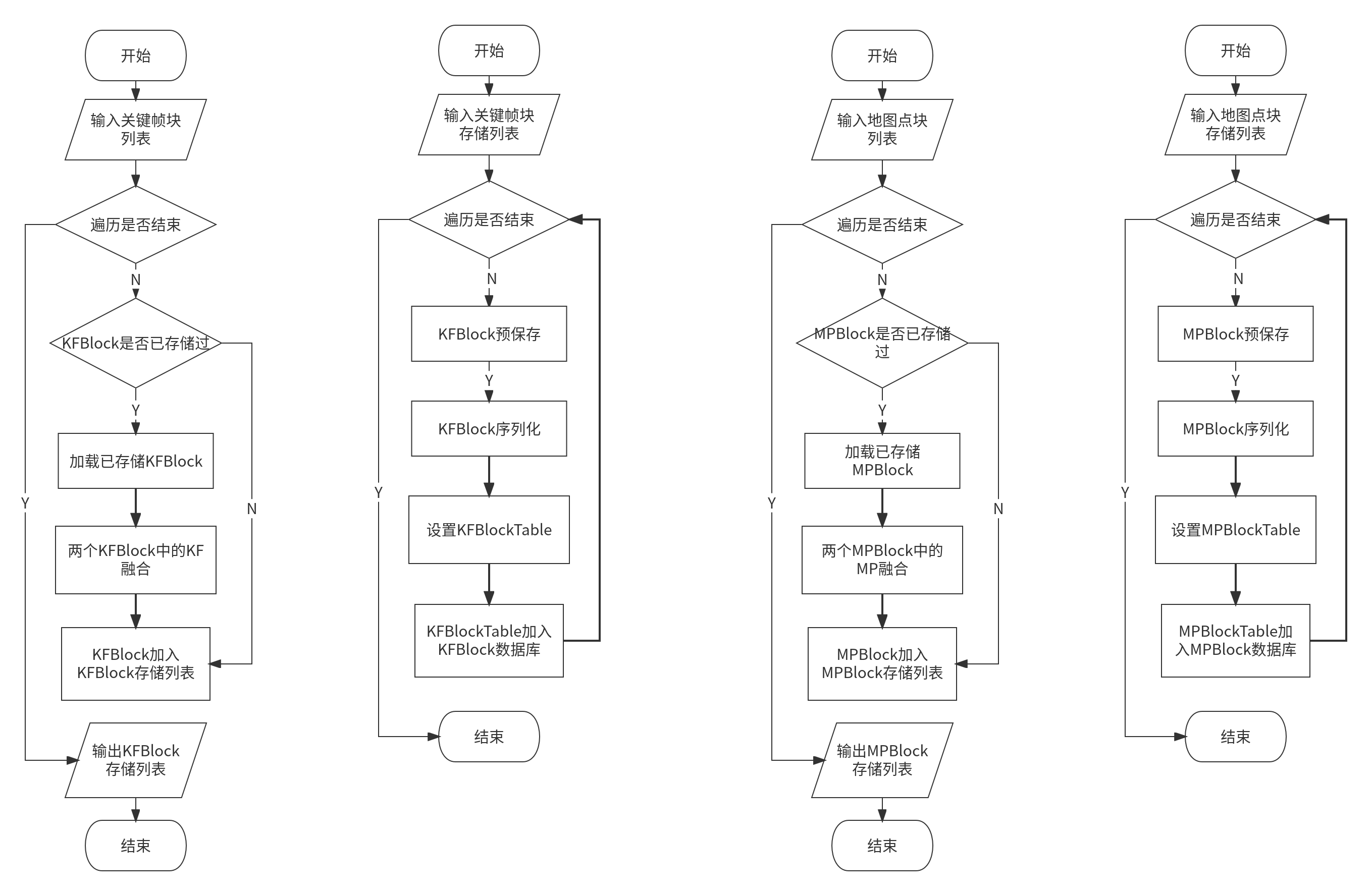
存储过，则将存储过的块加载，将其中的所有地图点加入临时地图点块，然后将临时地图点块加入待存储列表；

没存储过，临时地图点块加入待存储列表；

* 1. 地图点坐标转换

把地图点坐标从世界坐标系转到地图点块坐标系（float到float）；

* 1. 序列化地图点数据



地图点块融合流程 地图点块存储流程

1. **存储关键帧块**
2. 函数名

vector<KFBlockPos> Atlas::AddKeyFrameToBlock(vector<KeyFrame\*>)

1. 功能描述

将关键帧放入临时关键帧块

1. 输入输出

输入：vector<KeyFrame\*>

输出：vector<KFBlockPos> 关键帧块的中心坐标

1. 函数名

void Atlas::MergeKeyFrameBlock()

1. 功能描述

将临时关键帧块和已经存储的关键帧块融合，并把融合后的关键帧块放入待存储关键帧块。

1. 输入输出

输入：void

输出：void

1. 函数名

void Atlas::MapPointBlockSave()

1. 功能描述

存储关键帧块存储容器中的关键帧块。

1. 输入输出

输入：void

输出：void

存储关键帧块细节描述

* 1. 关键帧块融合

判断临时关键帧块是否存在（也就是是否存储过）

存储过，则将存储过的块加载，将其中的所有关键帧加入临时关键帧块，然后将临时关键帧块加入待存储列表；

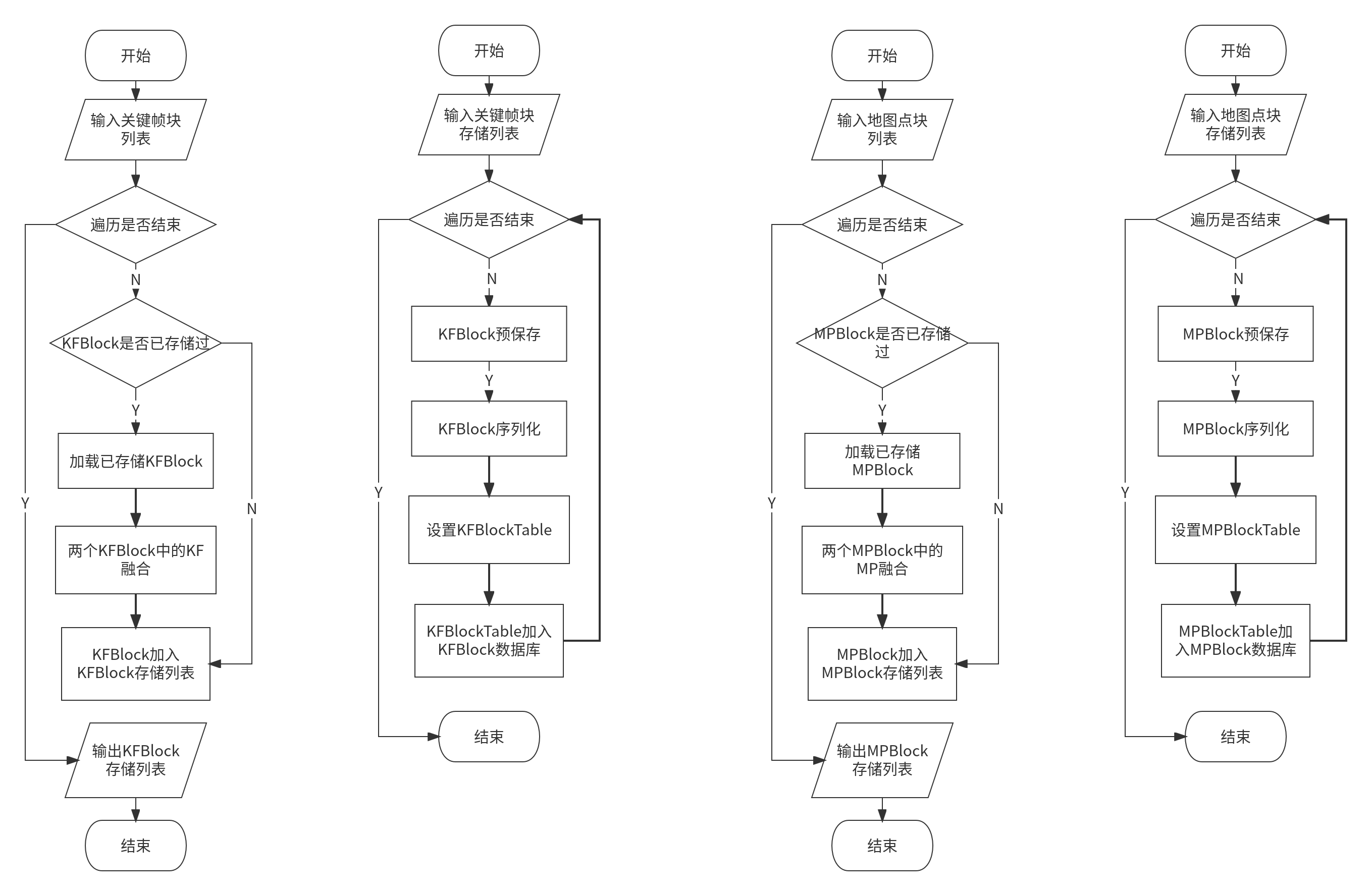
没存储过，临时关键帧块加入待存储列表；

* 1. 关键帧坐标转换（压缩）

把关键帧坐标从世界坐标系转到关键帧块坐标系（float到float）；

然后将float映射到到short int保存（精度0.001）。

* 1. 序列化关键帧
  2. 列表移除关键帧块



关键帧块融合流程 关键帧块存储流程

## 1.4更新与存储关键帧块信息表与地图点块信息表

（1）函数名

void AddKFBlockTable(KFBlockTable\* KFBTable);

void AddMPBlockTable(MPBlockTable\* MPBTable);

（2）功能描述

增加：向数据库中增加关键帧块信息表、地图点块信息表；

（3）输入输出

输入：块信息表

输出：void

（1）函数名

int FindKFBlockTable(KFBlockTable\* KFBTable);

int FindMPBlockTable(MPBlockTable\* MPBTable);

（2）功能描述

查询块信息表在数据库中的位置

（3）输入输出

输入：块信息表指针

输出：int块信息表的位置

（1）函数名

string SelectKFBlock(const KFBlockPos &blockpos);

string SelectMPBlock(const MPBlockPos &blockpos);

（2）功能描述

根据块的中心坐标获得块的ID

（3）输入输出

输入：块的中心坐标

输出：块的ID

（1）函数名

KFBlockTable\* GetKFBlockTable(const string KFBID);

MPBlockTable\* GetMPBlockTable(const string MPBID);

（2）功能描述

加载时根据块ID获得块表中的地图点块ID

（3）输入输出

输入：关键帧块ID

输出：地图点块ID

（1）函数名

void UpdateKFBlockTable(KFBlockTable\* , int );

void UpdateMPBlockTable(MPBlockTable\* , int );

（2）功能描述

更新关键帧块信息表、地图点块信息表数据；

（3）输入输出

输入：块信息表，int（块信息表的位置）

输出：void

（1）函数名

void SaveDatabase(string path);

（2）功能描述

将所有关键帧块信息表和地图点块信息表整体存储；

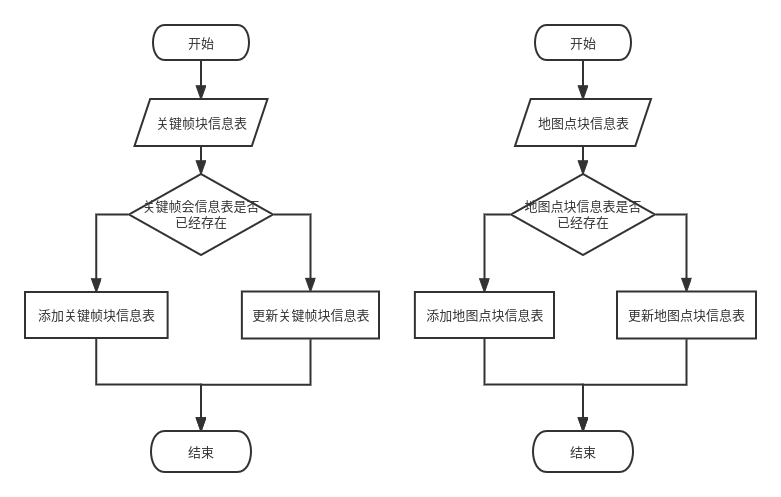
（3）输入输出

输入：string 存储路径

输出：void

细节描述

|  |  |  |
| --- | --- | --- |
| **地图块** | | |
| **地图点块信息表** | **关键帧块信息表** | **类型** |
| 地图点块ID | 关键帧块ID | string（MPB\_4\_31、KFB\_1\_12） |
| 地图点总数 | 关键帧总数 | int |
| — | vector<地图块信息表> | vector |
| 创建时间 | 创建时间 | double |
| 更新时间 | 更新时间 | double |



## 1.5删除已存储的关键帧、地图点

**一、地图点的删除**

1. 功能描述

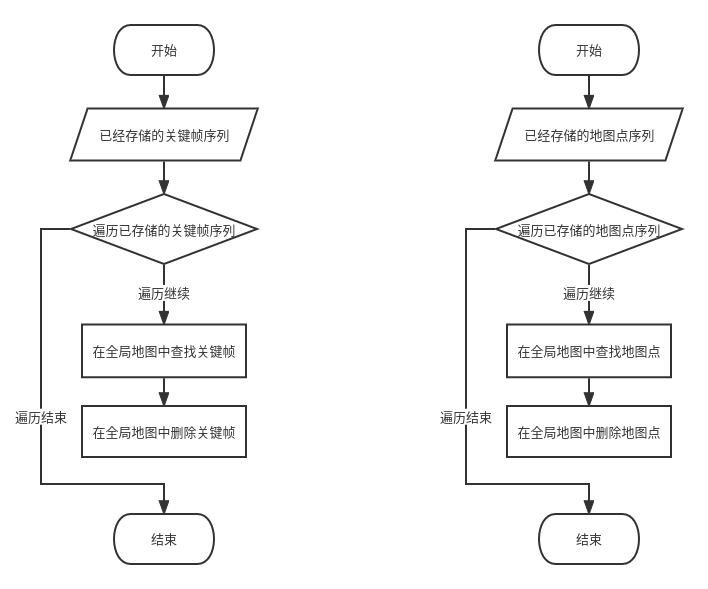
删除已经存储的地图点；

1. 输入输出

输入：vector<MapPoint\*> 已经存储的地图点

输出：void

1. 细节描述

逐一删除：首先在地图中查询到要删除的地图点，然后删除

**二、关键帧的删除**

1. 功能描述

删除已经存储的关键帧；

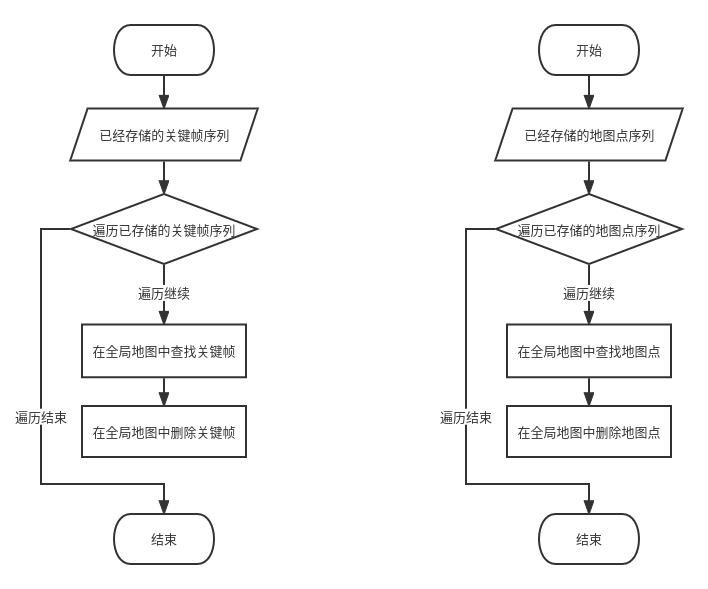
1. 输入输出

输入：vector<KeyFrame\*> 已经存储的关键帧

输出：void

1. 细节描述

逐一删除：首先在地图中查询到要删除的关键帧，然后删除



# 地图分块加载

## 2.1加载关键帧块与地图点块信息表

1. **关键帧块信息表与地图点块信息表加载**
2. 函数名

void LoadDatabase(string path);

1. 功能描述

将块信息表的到数据库

1. 输入输出

输入：块信息表路径

输出：void

## 2.**2查询并加载需要加载的关键帧块与地图点块**

1. 函数名

void Atlas::BlocksLoad(vector<float> &pos, int th, string path)

1. 功能描述

根据当前地图块中心坐标，查询周围临近的关键帧块，并筛选需要加载的关键帧块ID；

根据关键帧块ID，加载地图点块与关键帧块；

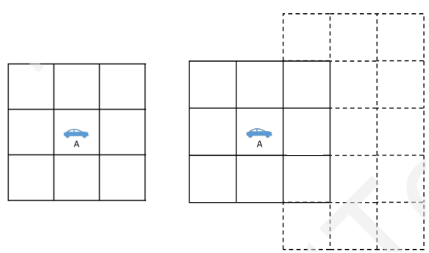
1. 输入输出

输入：当前的位置坐标，加载范围，块存放路径

输出：void

1. 细节描述
   1. 确定加载范围

移动速度、地图块尺寸、加载耗时等因素确定加载的范围。例如



* 1. 确定关键帧块集合

根据当前位置所在地图方格的中心坐标、加载范围，确定待筛选的关键帧块集合（查询8个方向关键帧块）。两个条件进行筛选：

* + 1. 块必须存储过

通过数据库中查询其是否存储过；

* + 1. 块没在当前地图中

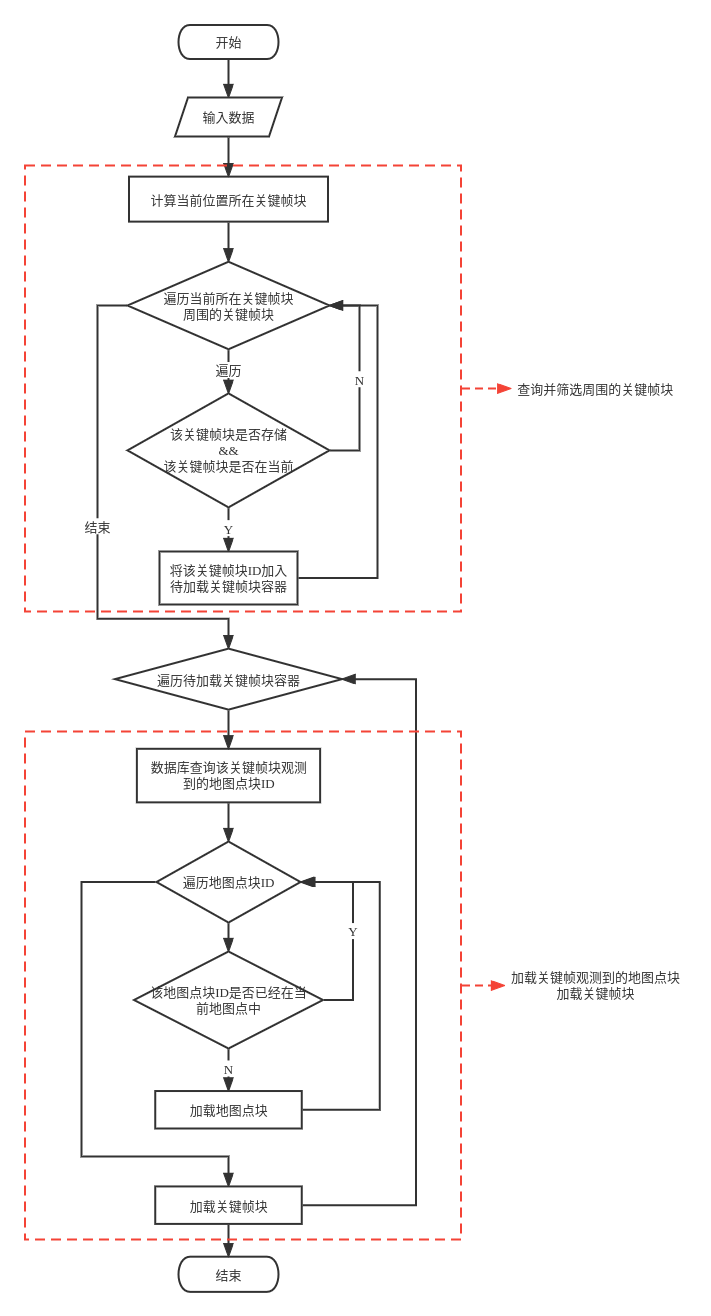
将关键帧块ID插入mCurrUsingKFBlocksID，如果插入成功，可以加载该块；

* 1. 确定地图点块集合

根据已经确定要加载的关键帧块，确定其中关键帧观测到的地图点块（可以认为观测到地图点块的th个地图点 则观测成立），地图点块筛选条件如下：

* + 1. 块没在当前地图中

将地图点块ID插入mCurrUsingMPBlocksID，如果插入成功，可以加载该块；



## 2.3加载地图点块与关键帧块

1. **关键帧块加载（存储需要）**
2. 函数名

bool KeyFrameBlockLoad(string filename);

1. 功能描述

加载关键帧块内的所有关键帧；

1. 输入输出

输入：关键帧块ID（string）

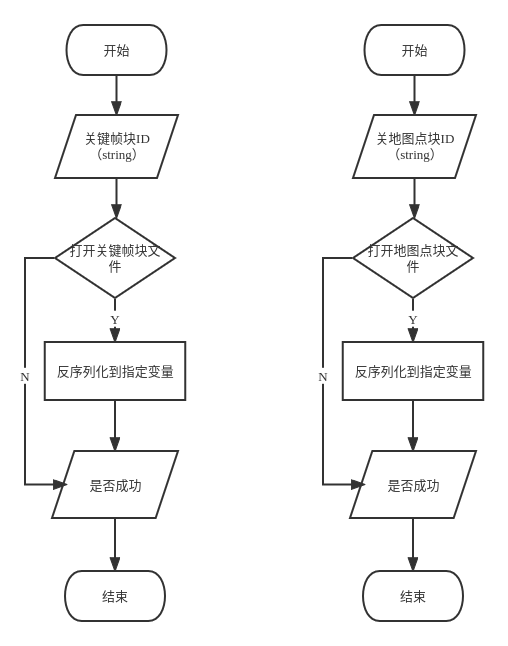
输出：bool 是否加载成功

1. 细节描述
   1. 反序列化

Boost库进行反序列化

* 1. 坐标恢复

关键帧块坐标系转世界坐标系

~~~~

1. **地图点块加载（存储）**
2. 函数名

bool MapPointBlockLoad(string filename);

1. 功能描述

加载地图点块内的所有地图点；

1. 输入输出

输入：地图点块ID（string）

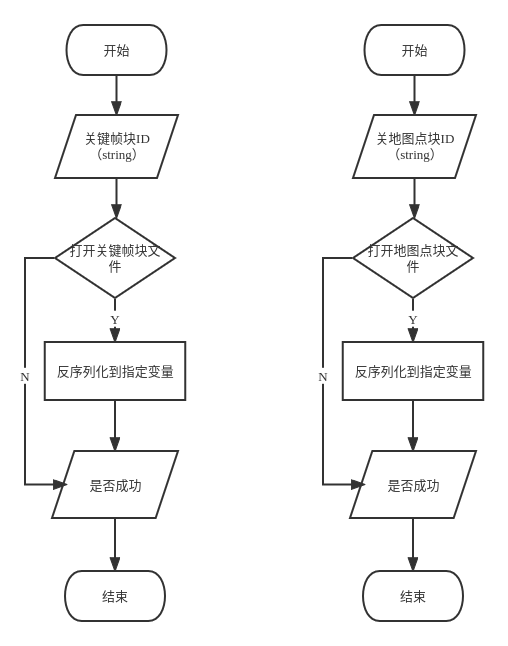
输出：bool 是否加载成功

1. 细节描述
   1. 反序列化

Boost库进行反序列化

* 1. 坐标恢复

地图点块坐标系转世界坐标系

~~~~

## 2.4删除关键帧块和地图点块

1. **删除关键帧块**
2. 函数名

void Atlas::DeleteKeyFrameBlocks(vector<float> &pos, int th)

1. 功能描述

从内存中删除当前位置不需要的关键帧块；

1. 输入输出

输入：pos 当前位置、th 范围系数

输出：void

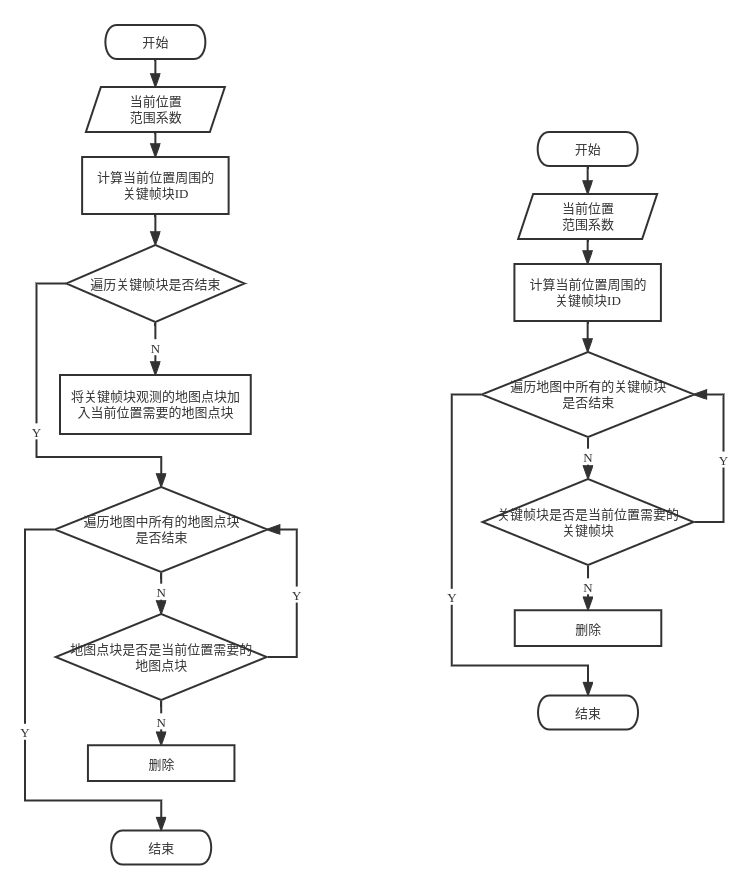
1. 细节描述
   1. 删除的关键帧块条件

地图中的关键帧块如果不在当前位置需要的关键帧块中，从内存中删除该关键帧块

* 1. 获取当前需要删除的关键帧块

首先根据当前位置获取当前位置需要的关键帧块

其次删除内存中不是当前位置需要的关键帧块



1. **删除地图点块**
2. 函数名

void Atlas::DeleteMapPointBlocks(vector<float> &pos, int th)

1. 功能描述

从内存中删除当前位置不需要的地图点块；

1. 输入输出

输入：pos 当前位置、th 范围系数

输出：void

1. 细节描述
   1. 删除的地图点块条件

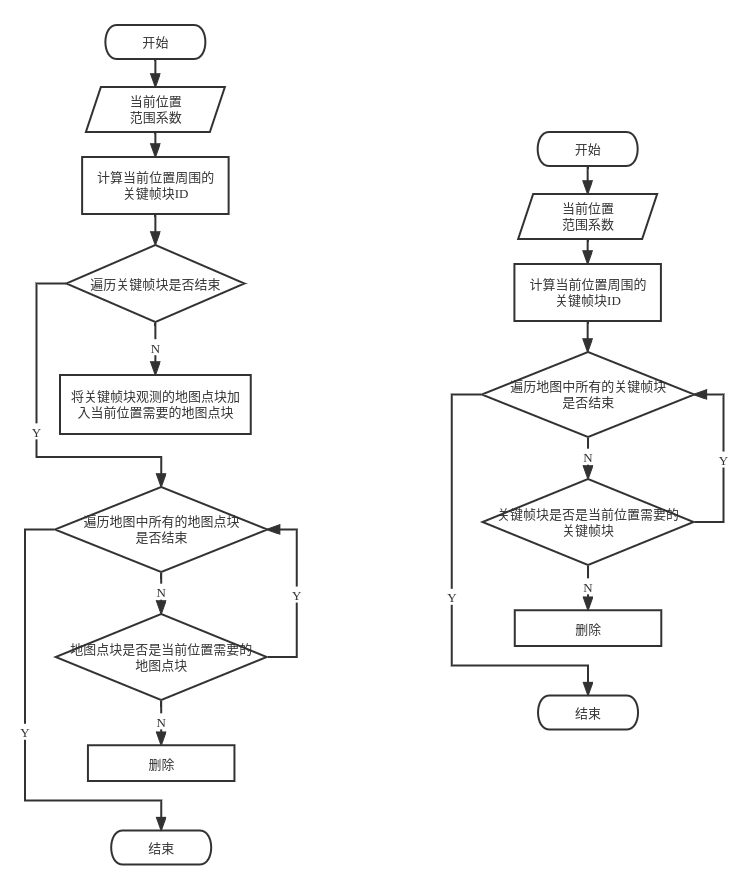
如果地图中的地图点块不被当前位置周围的关键帧块观测到，从内存中删除该地图点块

* 1. 获取当前需要删除的地图点块

首先根据当前位置获取当前位置需要的关键帧块

其次通过关键帧块获取当前位置需要的地图点块

最后删除内存中不是当前位置需要的地图点块



# Atlas主函数

1. 函数名

void Atlas::Run()

1. 功能描述

主函数；

1. 输入输出

输入：void

输出：void

1. 程序流程图

下面两个流程图中，图一图二仅在不复用地图模式存在差别；

图二 程序已经实现的，但存在频繁merge的缺陷；

图一为新的流程图，但程序中暂时未实现

**图三为最新的流程图（2023.06.05）**

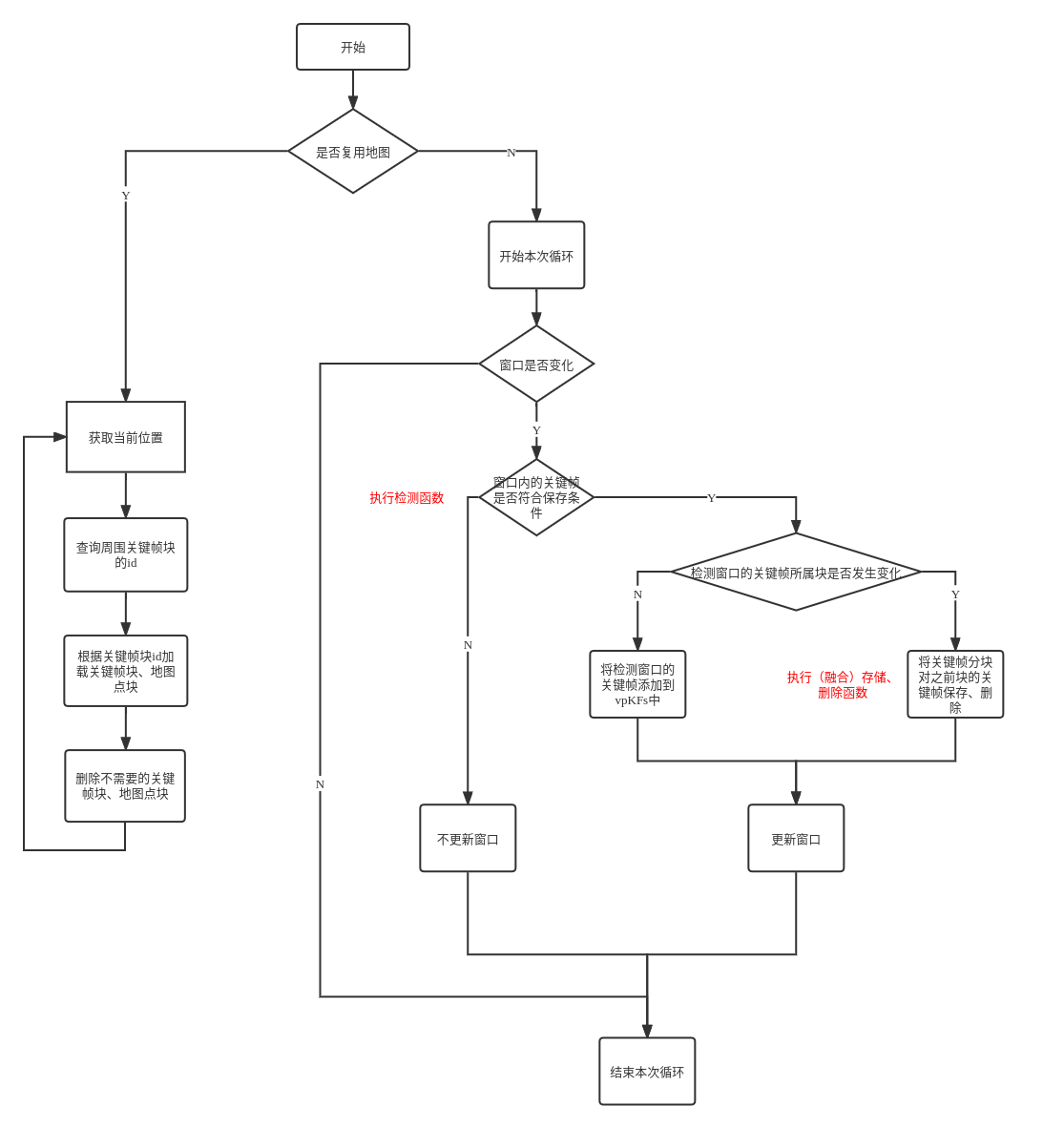


图1

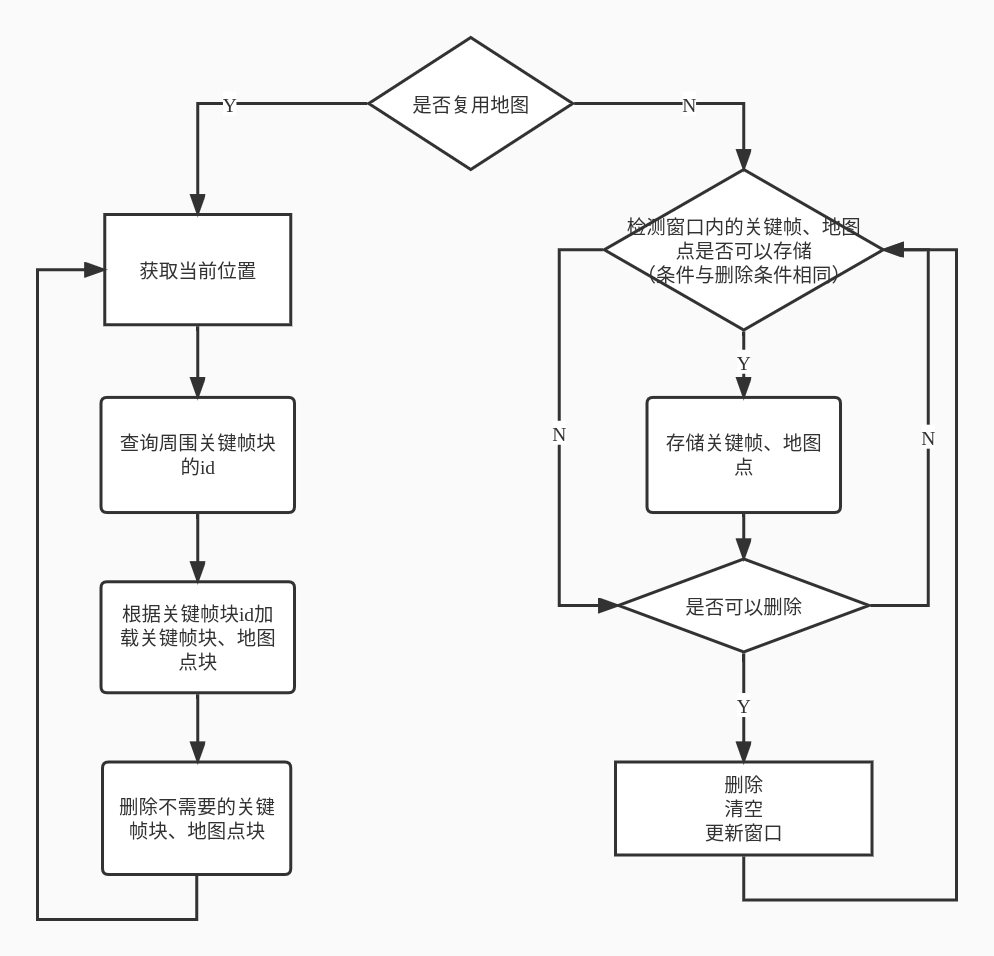


图2

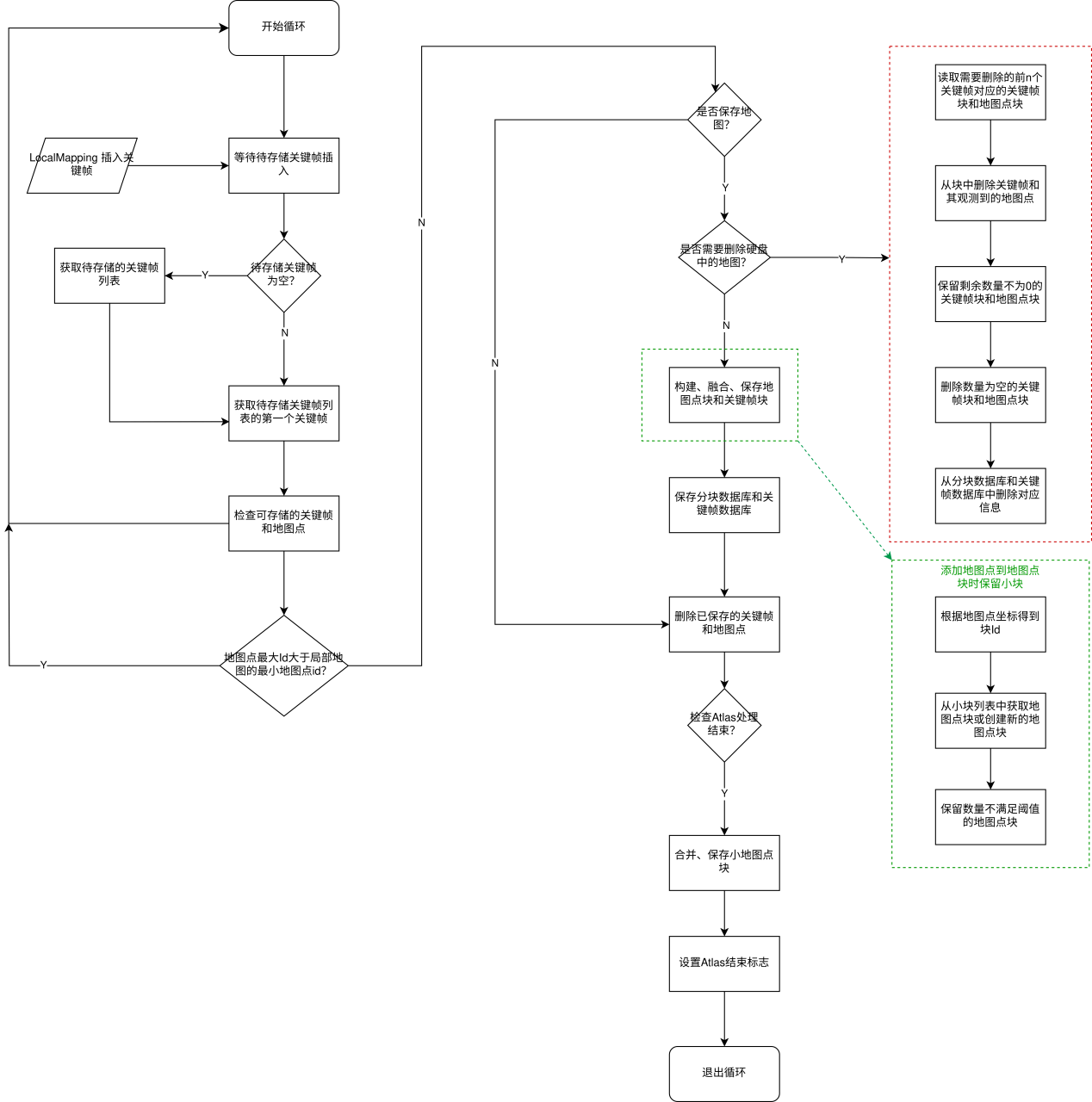


图3

# 小地图点块保留与存储

（1）函数名

void Atlas::AddSmallBlock(MPBlockPos &MPB\_Pos)

（2）功能描述

在根据地图点创建地图点块时，先查找对应的地图点块是否在保留的小地图点块列表中。如果存在，使用小地图点块代替新地图点块的创建。并从保留的小地图点块中删除对应的块。

（3）输入输出

输入：需要创建的地图点块

输出：void

（1）函数名

void Atlas::ReserveSmallBlock()

（2）功能描述

1.将mMapPointBlockTemp中地图点数量小于阈值的地图点块移入mMPBlocksReserve，并将其中包含的地图点从vpDeleteMPs中移除。设置保留地图点标志位（mbReserved=true）

2. 将mMPBlocksReserve中超过阈值的地图点块移入mMapPointBlockTemp，并将相应的地图点加入vpDeleteMPs。设置保留地图点标志位（mbReserved=false）

（3）输入输出

输出：void

（1）函数名

void Atlas::SaveSmallBlock()

（2）功能描述

将所有的小地图点块合并为一个大地图点块并保存

（3）输入输出

输出：void

# 删除地图块

（1）函数名

void Atlas::DeleteMapBlock(int num)

（2）功能描述

当存储的地图块文件大小（Blocks文件夹大小）超过系统最大地图存储阈值后，根据关键帧Id，删除磁盘中存储的前n个关键帧和对应的地图点。并删除分块数据库中的相应信息。

（3）输入输出

输入：需要创建的地图点块

输出：void

# 关键帧数据库的保存与加载

（1）函数名

void Atlas::AddBowDatabase(vector<KeyFrame \*> &vpDeleteKFs)

（2）功能描述

添加关键帧数据库中的关键帧对应的bow信息、特征描述符及对应的关键帧块Id到待保存的数据结构。

（3）输入输出

输入：待删除的关键帧列表

输出：void

（1）函数名

void Atlas::SaveBowDatabase()

（2）功能描述

保存关键帧数据库数据

（3）输入输出

输出：void

（1）函数名

void Atlas::LoadBowDatabase()

（2）功能描述

加载关键帧数据库数据

（3）输入输出

输出：void

（1）函数名

void Atlas::RestoreBowDatabase()

（2）功能描述

从保存的数据中恢复出关键帧数据库结构，用于定位。

（3）输入输出

输出：void