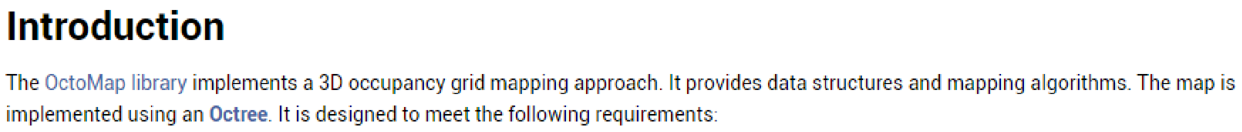
octomap 库的一点总结

前些天突然想起高博《视觉 SLAM 十四讲》提到的 octomap 八叉树地图，在网上百度又只有高博写的那么一片博文，于是就想自己看看这个库，了解其几个基本的类和接口，以下是我这几天来的总结，如果有错误请及时指出，或者联系我，我也会及时改正。（ yuanliudongdong@163.com ）

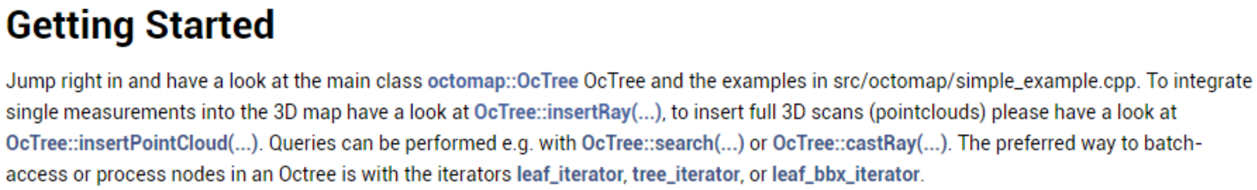
一，官方的 Introduction 部分翻译(http://octomap.github.io/octomap/doc/index.html)



Introduction

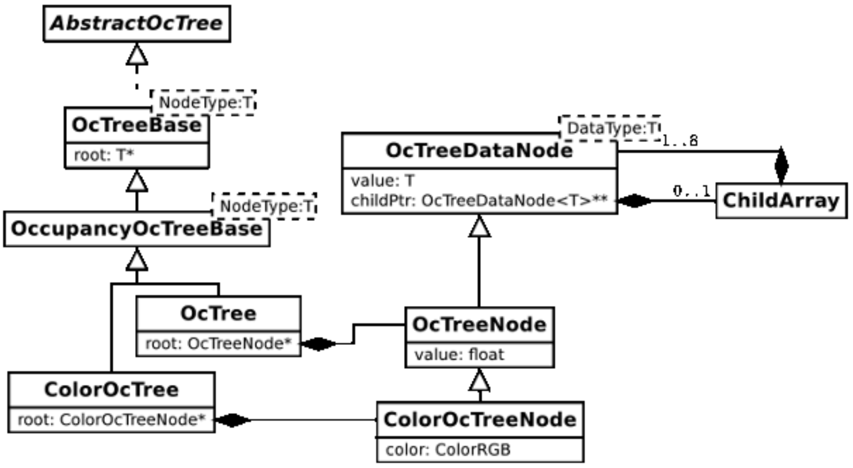
The OctoMap library implements a 3D occupancy grid mapping approach. It provides data structures and mapping algorithms. The map is implemented using an Octree. It is designed to meet the following requirements:

Octomap 库实现了一种填充 3D 栅格的构图方式，并提供了相应的数据结构和构图算法。整个地图利用一个 Octree 的类来实现。



Getting Started

Jump right in and have a look at the main class octomap::OcTree OcTree and the examples in src/octomap/simple_example.cpp. To integrate single measurements into the 3D map have a look at OcTree::insertRay(...), to insert full 3D scans (pointclouds) please have a look at OcTree::insertPointCloud(...). Queries can be performed e.g. with OcTree::search(...) or OcTree::castRay(...). The preferred way to batch-access or process nodes in an Octree is with the iterators leaf_iterator, tree_iterator, or leaf_bbx_iterator.



直接查看本库主要的类 octomap::OcTree，以及其示例程序 src/octomap/simple_example.cpp。如果你想要将单组测量数据整合到 3D 地图中，请查看 OcTree::insertRay(...)函数；如果你想要将整个 3D 扫描数据（点云）整合到地图中，请查看 OcTree::insertPointCloud(...)函数。同时可以通过 OcTree::search(...) 和 OcTree::castRay(...)函数来进行地图空间占用的查询。推荐使用迭代器的方式对 Octree 的节点等进行查询，例如 leaf_iterator，tree_iterator 和 leaf_bbx_iterator。

二，示例代码分析(Github 源码地址：https://github.com/OctoMap/octomap)

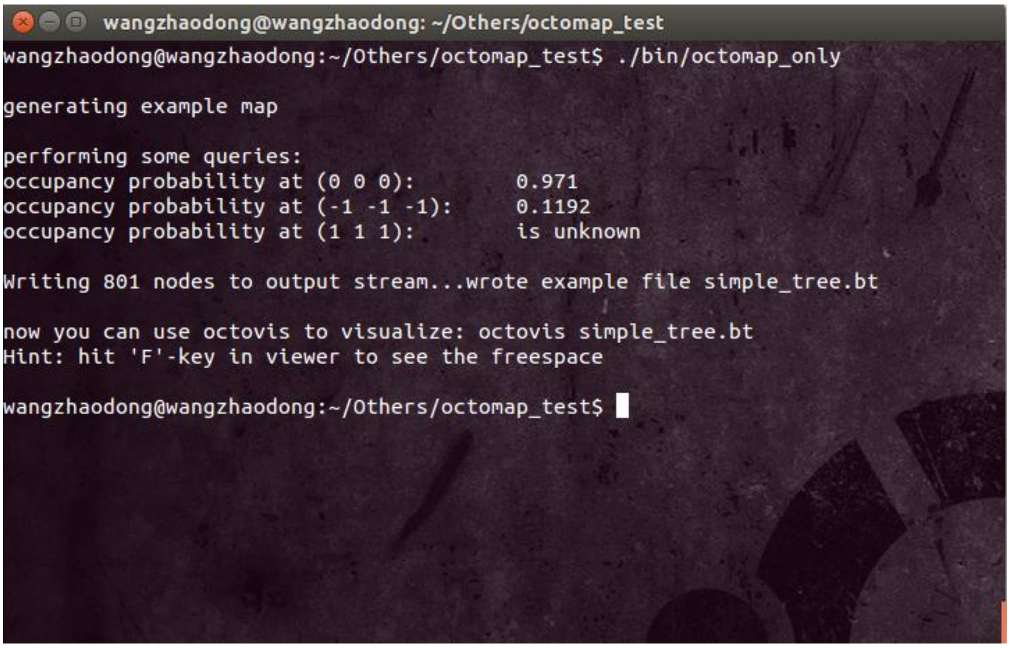
在使用该库时，主要使用了基本的 OcTree 类和带有颜色信息的 ColorOcTree 类，所以只简单分析了这两个类的示例代码。

1，octomap/octomap/src/simple_example.cpp

```
#include <octomap/octomap.h>
#include <octomap/OcTree.h>
using namespace std;
using namespace octomap;

//查询信息输出函数，输入为octomap 命名空间下的 3D 点位置和用 OcTree::search(...)得到的返回值
void print_query_info(point3d query, OcTreeNode* node)
{
    if (node != NULL)
    {
       cout << "occupancy probability at " << query << ":\t " << node->getOccupancy() << endl;
    }
    else
      cout << "occupancy probability at " << query << ":\t is unknown" << endl;
}
 // 主函数入口
int main(int argc, char** argv)
{
  cout << endl;
  cout << "generating example map" << endl;
  // 建立一个空的 OcTree 对象地图，分辨率为0.1
  OcTree tree (0.1);  // create empty tree with resolution 0.1

  // insert some measurements of occupied cells
  // 向地图中插入一些测量数据，这些数据点在地图中表现为已占用状态
  for (int x=-20; x<20; x++)
  {
    for (int y=-20; y<20; y++)
    {
      for (int z=-20; z<20; z++)
      {
        //建立空间点对象，并向地图中更新节点
        point3d endpoint ((float) x*0.05f, (float) y*0.05f, (float) z*0.05f);
        tree.updateNode(endpoint, true); // integrate 'occupied' measurement
      }
    }
  }

  // insert some measurements of free cells
// 向地图中插入一些测量数据，这些数据点在地图中表现为已未占用状态
  for (int x=-30; x<30; x++)
  {
    for (int y=-30; y<30; y++)
    {
      for (int z=-30; z<30; z++)
      {
        //建立空间点对象，并向地图中更新节点
        point3d endpoint ((float) x*0.02f-1.0f, (float) y*0.02f-1.0f, (float) z*0.02f-1.0f);
        tree.updateNode(endpoint, false);  // integrate 'free' measurement
      }
    }
  }
//上面已经完成了对地图的创建，之后的程序是对地图节点数据的访问
  cout << endl;
  cout << "performing some queries:" << endl;
 // 节点（0,0,0）
  point3d query (0., 0., 0.);
  OcTreeNode* result = tree.search (query);
  print_query_info(query, result);
// 节点（-1,-1,-1）
  query = point3d(-1.,-1.,-1.);
  result = tree.search (query);
  print_query_info(query, result);
// 节点（1,1,1）
  query = point3d(1.,1.,1.);
  result = tree.search (query);
  print_query_info(query, result);
  cout << endl;
// 将建好的地图保存
  tree.writeBinary("simple_tree.bt");
  cout << "wrote example file simple_tree.bt" << endl << endl;
  cout << "now you can use octovis to visualize: octovis simple_tree.bt"  << endl;
  cout << "Hint: hit 'F'-key in viewer to see the freespace" << endl  << endl;
```

```
}
```

这是执行该代码后的运行结果：

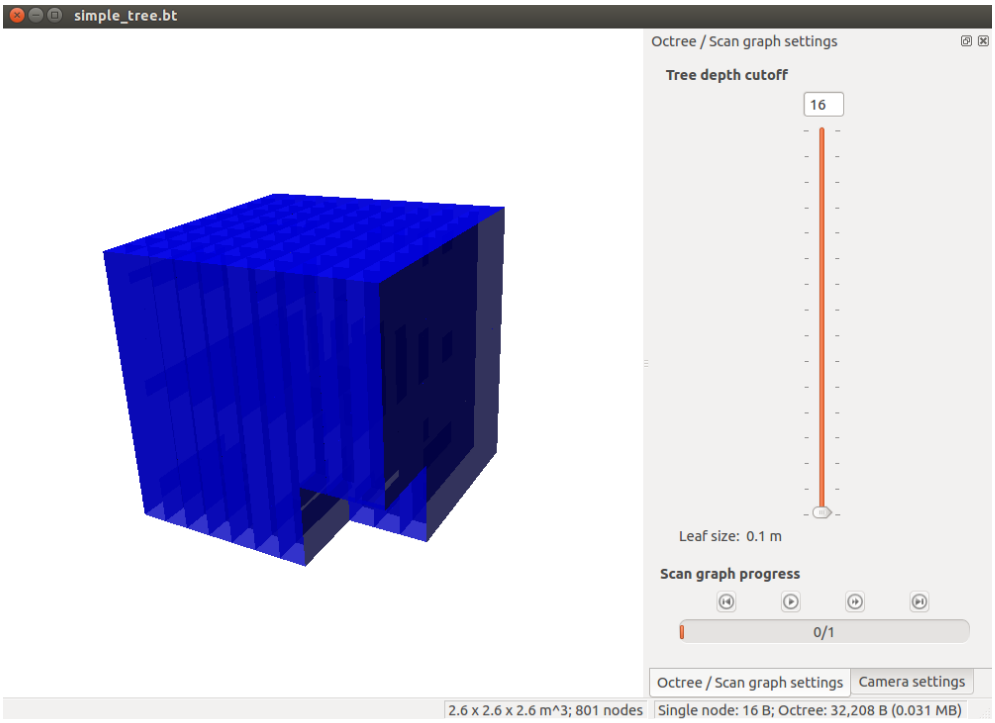


wangzhaodong@wangzhaodong: ~/Others/octomap_test

```
wangzhaodong@wangzhaodong:~/Others/octomap_test$ ./bin/octomap_only

generating example map

performing some queries:
occupancy probability at (0 0 0):        0.971
occupancy probability at (-1 -1 -1):     0.1192
occupancy probability at (1 1 1):        is unknown

Writing 801 nodes to output stream...wrote example file simple_tree.bt

now you can use octovis to visualize: octovis simple_tree.bt
Hint: hit 'F'-key in viewer to see the freespace

wangzhaodong@wangzhaodong:~/Others/octomap_test$ 
```

图表 1

这是用 octovis 查看所生成的地图 simple_tree.bt：



Tree depth cutoff 16

Leaf size: 0.1 m

0/1

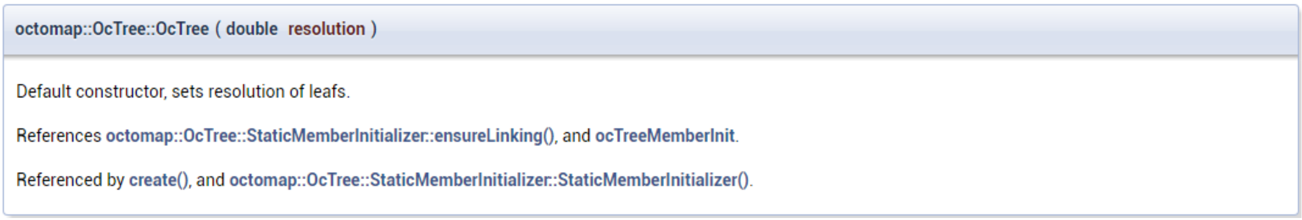
2.6 x 2.6 x 2.6 m^3; 801 nodes

Single node: 16 B; Octree: 32,208 B (0.031 MB)

图表 2

下面针对一些关键语句进行分析：

- OcTree tree(0.1);建立 OcTree 地图对象。这个函数的声明是这样的：



octomap::OcTree::OcTree (double resolution)

Default constructor, sets resolution of leafs.

References octomap::OcTree::StaticMemberInitializer::ensureLinking(), and ocTreeMemberInit.

Referenced by create(), and octomap::OcTree::StaticMemberInitializer::StaticMemberInitializer().

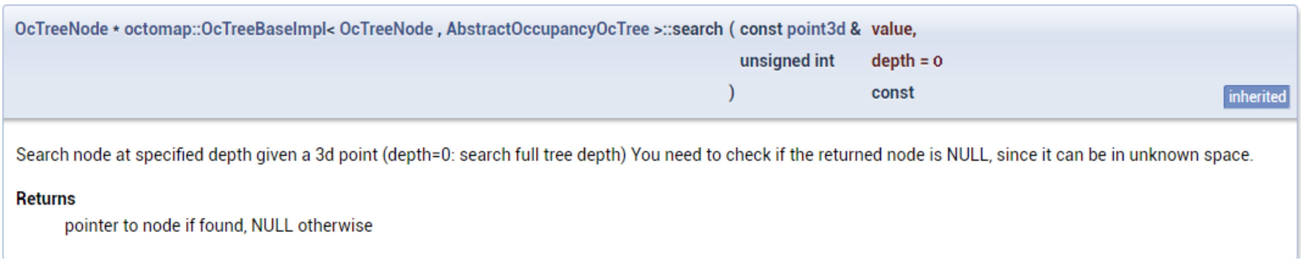
resolution 译为 “分辨率” ，即访问地图的最小单位；

- point3d endpoint((float)x*0.02f-1.0f, (float)y*0.02f-1.0f, (float)z*0.02f-1.0f);

  tree.updateNode(endpoint, false);

  建立一个三维空间点，并将其添加到地图当中；point3d 构造函数的三个参数分别是三维点的 x/y/z 坐标；使用.updateNode()函数将空间点信息更新到地图当中，第二个参数表示该空间点对应的节点未被占用(false)

- OcTreeNode* result = tree.search (query);节点信息查询函数；函数声明是这样的



```
OcTreeNode * octomap::OcTreeBaseImpl< OcTreeNode , AbstractOccupancyOcTree >::search ( const point3d &  value,
                                                                                       unsigned int    depth = 0
                                                                                     )                 const
```

Search node at specified depth given a 3d point (depth=0: search full tree depth) You need to check if the returned node is NULL, since it can be in unknown space.

Returns
pointer to node if found, NULL otherwise

传入参数为一个三维点对象，如果在这个地图 tree 中，这个三维点对应的位置有节点（不管占用与否），那么将返回该位置的节点指针，否则将返回一个空指针；

- cout << "occupancy probability at " << query << ":\t " << node->getOccupancy() << endl；如果查询到有该位置上节点的信息，则使用 getOccupancy()函数输出该节点占用情况，那为什么这个 Occupancy 是个小数呢？这是因为 Octomap 在描述一个栅格是否被占用时，并不是单一的只描述为占用和被占用，而是用一个概率（Occupancy probability）来描述它，即这个栅格被占用的概率是多少，通过这个概率来确定这个栅格被占用的可能性。

2，Occupancy probability

关于这个占用概率值，高博在一篇博客中做出了和通俗易懂的解释，我就不再复制粘贴了，下面附上连接(https://www.cnblogs.com/gaoxiang12/p/5041142.html)。

我要说的是，对于一个最小单位的栅格，如何判断 updateNode()函数对其做出了贡献或者说更新。就比如说

```
        point3d endpoint( 4.05f, 4.05f, 4.05f );
        tree.updateNode( endpoint, true );
```

这两行代码，由于分辨率是 0.1，并不能精确到 0.01，所以要把这个（4.05,4.05,4.05）归到（4.0,4.0,4.0）这个节点呢，还是（4.1,4.1,4.1）这个节点或者其他节点呢？经过我的多次测试，应当归到（4.0,4.0,4.0）节点当中，也就是说，对于节点（4.0,4.0,4.0）来说，凡是同时满足 x=[4.0,4.1)，y=[4.0,4.1)，z=[4.0,4.1)的 point，如果使用 updateNode()函数将这个 point 更新到地图当中，那么必然会影响到节点（4.0,4.0,4.0）的 Occupancy probability。

总结就是，对于一个点 point(x,y,z)，使用 updateNode()函数将其更新到地图当中，那么 Occupancy probability 受到影响的节点将是

$$node(x_0, y_0, z_0), (x_0 \le x, x_0 \ge x - resolution, y_0 \le y, y_0 \ge y - resolution, z_0 \le z, z_0 \ge z - resolution)$$

即小于 point 坐标值的最大节点。同时当我们用 search ()函数对 point 进行查询时，返回的信息也将是小于 point 坐标值的最大节点信息。

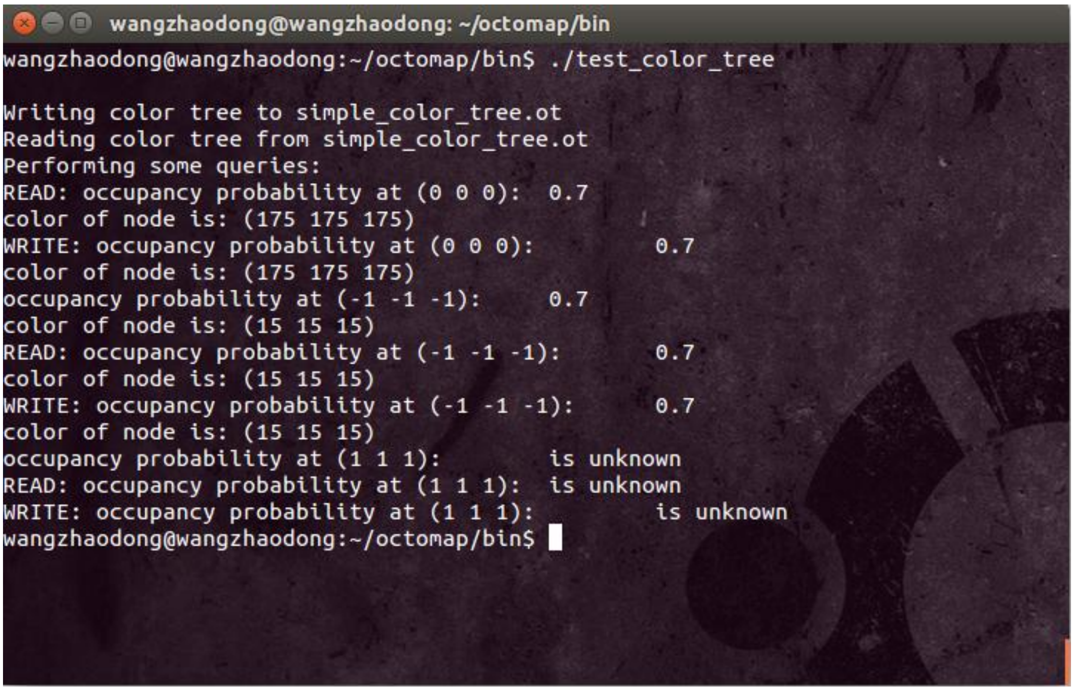
另外，在对某个节点进行不同次数的更新之后，发现 Occupancy probability 最大值为 0.971，最小值为 0.1192，这也验证了高博那篇博文中提到的最大值和最小值限制。

3，octomap/src/testing/test_color_tree.cpp

带有色彩的八叉树地图 ColorOcTree 与基本的 OcTree 类似，需要知道如何向节点当中添加颜色信息就好了。下面是 test_color_tree.cpp 的部分代码：

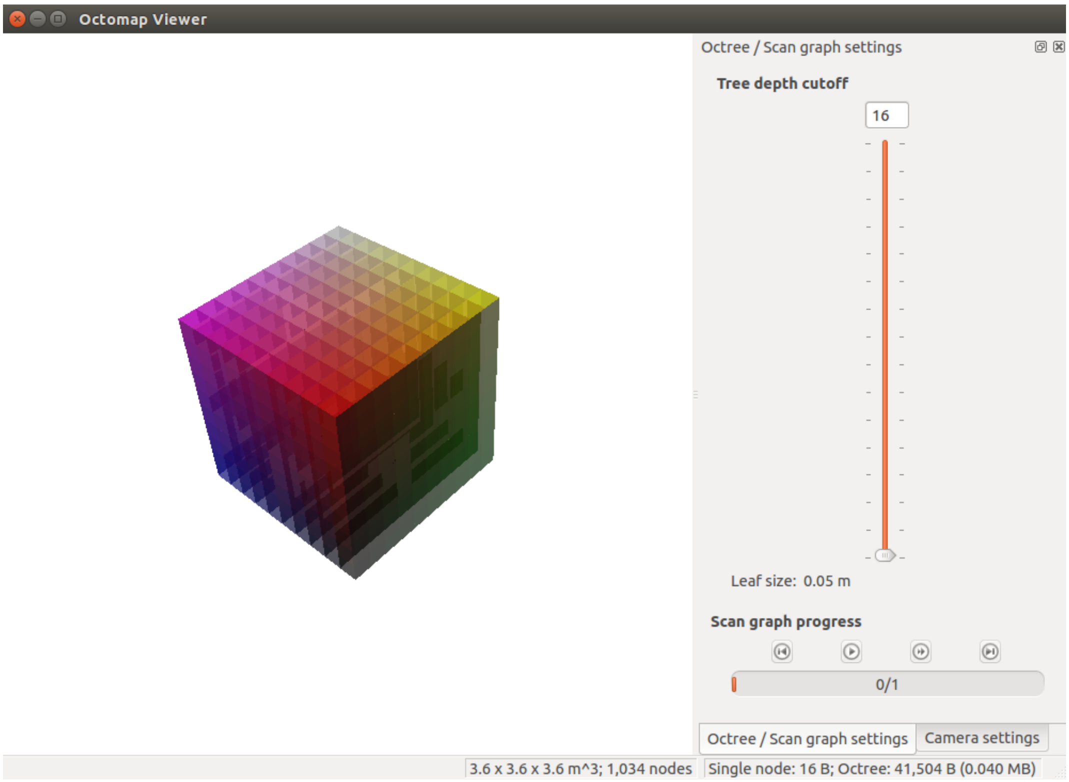
```
int main(int argc, char** argv)
{
  //分辨率
  double res = 0.05;  // create empty tree with resolution 0.05 (different from default 0.1 for test)
  //建立彩色地图对象
  ColorOcTree tree (res);
  // insert some measurements of occupied cells
  for (int x=-20; x<20; x++)
  {
    for (int y=-20; y<20; y++)
    {
      for (int z=-20; z<20; z++)
      {
        point3d endpoint ((float) x*0.05f+0.01f, (float) y*0.05f+0.01f, (float) z*0.05f+0.01f);
        ColorOcTreeNode* n = tree.updateNode(endpoint, true);
        //设置节点颜色信息的函数，每个地图节点差五个像素大小，渐变
        n->setColor(z*5+100,x*5+100,y*5+100);
      }
    }
  }

  // insert some measurements of free cells
  for (int x=-30; x<30; x++)
  {
    for (int y=-30; y<30; y++)
    {
      for (int z=-30; z<30; z++)
      {
        point3d endpoint ((float) x*0.02f+2.0f, (float) y*0.02f+2.0f, (float) z*0.02f+2.0f);
        ColorOcTreeNode* n = tree.updateNode(endpoint, false);
        //不被占用的节点设置为黄色
        n->setColor(255,255,0); // set color to yellow
      }
    }
  }
  // set inner node colors
  tree.updateInnerOccupancy();
}
```

这是执行该代码后的运行结果：



```
wangzhaodong@wangzhaodong:~/octomap/bin$ ./test_color_tree

Writing color tree to simple_color_tree.ot
Reading color tree from simple_color_tree.ot
Performing some queries:
READ: occupancy probability at (0 0 0):  0.7
color of node is: (175 175 175)
WRITE: occupancy probability at (0 0 0):        0.7
color of node is: (175 175 175)
occupancy probability at (-1 -1 -1):     0.7
color of node is: (15 15 15)
READ: occupancy probability at (-1 -1 -1):      0.7
color of node is: (15 15 15)
WRITE: occupancy probability at (-1 -1 -1):     0.7
color of node is: (15 15 15)
occupancy probability at (1 1 1):        is unknown
READ: occupancy probability at (1 1 1):  is unknown
WRITE: occupancy probability at (1 1 1):        is unknown
wangzhaodong@wangzhaodong:~/octomap/bin$
```

这是用 octovis 查看所生成的地图 simple_color_tree.ot：



Tree depth cutoff 16

Leaf size: 0.05 m

Scan graph progress 0/1

3.6 x 3.6 x 3.6 m^3; 1,034 nodes

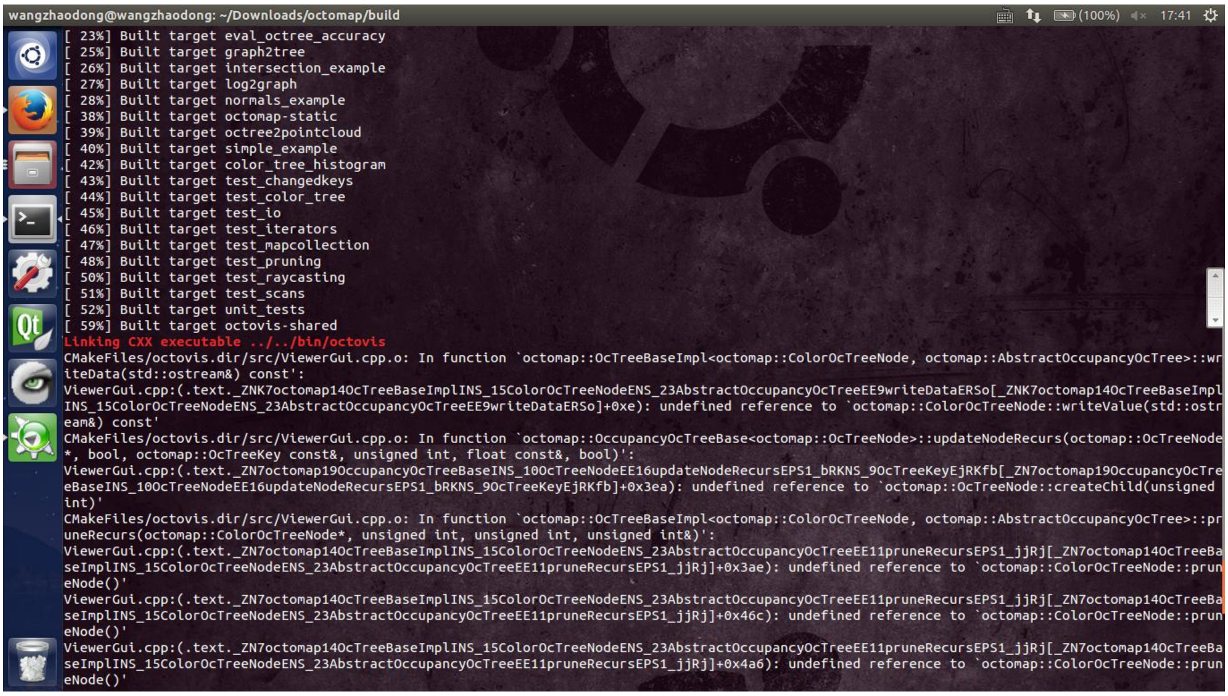
Single node: 16 B; Octree: 41,504 B (0.040 MB)

三，安装和编译 octomap 库时的一个小问题

第一次编译安装 octomap 库时，直接按照网上的教程进行的操作：

```
git clone https://github.com/OctoMap/octomap
cd octomap
mkdir build
cd build
cmake ..
make
sudo make install
```

可是在执行 make 指令时却出现了 undefined reference to 的错误

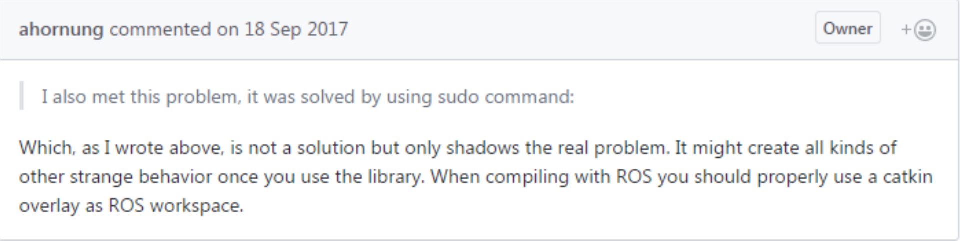


```
[ 23%] Built target eval_octree_accuracy
[ 25%] Built target graph2tree
[ 26%] Built target intersection_example
[ 27%] Built target log2graph
[ 28%] Built target normals_example
[ 38%] Built target octomap-static
[ 39%] Built target octree2pointcloud
[ 40%] Built target simple_example
[ 42%] Built target color_tree_histogram
[ 43%] Built target test_changedkeys
[ 44%] Built target test_color_tree
[ 45%] Built target test_io
[ 46%] Built target test_iterators
[ 47%] Built target test_mapcollection
[ 48%] Built target test_pruning
[ 50%] Built target test_raycasting
[ 51%] Built target test_scans
[ 52%] Built target unit_tests
[ 59%] Built target octovis-shared
Linking CXX executable ../../bin/octovis
CMakeFiles/octovis.dir/src/ViewerGui.cpp.o: In function `octomap::OcTreeBaseImpl<octomap::ColorOcTreeNode, octomap::AbstractOccupancyOcTree>::wr
iteData(std::ostream&) const':
ViewerGui.cpp:(.text._ZNK7octomap14OcTreeBaseImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE9writeDataERSo[_ZNK7octomap14OcTreeBaseImpl
INS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE9writeDataERSo]+0xe): undefined reference to `octomap::ColorOcTreeNode::writeValue(std::ostr
eam&) const'
CMakeFiles/octovis.dir/src/ViewerGui.cpp.o: In function `octomap::OccupancyOcTreeBase<octomap::OcTreeNode>::updateNodeRecurs(octomap::OcTreeNode
*, bool, octomap::OcTreeKey const&, unsigned int, float const&, bool)':
ViewerGui.cpp:(.text._ZN7octomap19OccupancyOcTreeBaseINS_10OcTreeNodeEE16updateNodeRecursEPS1_bRKNS_9OcTreeKeyEjRKfb[_ZN7octomap19OccupancyOcTre
eBaseINS_10OcTreeNodeEE16updateNodeRecursEPS1_bRKNS_9OcTreeKeyEjRKfb]+0x3ea): undefined reference to `octomap::OcTreeNode::createChild(unsigned
int)'
CMakeFiles/octovis.dir/src/ViewerGui.cpp.o: In function `octomap::OcTreeBaseImpl<octomap::ColorOcTreeNode, octomap::AbstractOccupancyOcTree>::pr
uneRecurs(octomap::ColorOcTreeNode*, unsigned int, unsigned int, unsigned int&)':
ViewerGui.cpp:(.text._ZN7octomap14OcTreeBaseImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj[_ZN7octomap14OcTreeBa
seImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj]+0x3ae): undefined reference to `octomap::ColorOcTreeNode::prun
eNode()'
ViewerGui.cpp:(.text._ZN7octomap14OcTreeBaseImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj[_ZN7octomap14OcTreeBa
seImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj]+0x46c): undefined reference to `octomap::ColorOcTreeNode::prun
eNode()'
ViewerGui.cpp:(.text._ZN7octomap14OcTreeBaseImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj[_ZN7octomap14OcTreeBa
seImplINS_15ColorOcTreeNodeENS_23AbstractOccupancyOcTreeEE11pruneRecursEPS1_jjRj]+0x4a6): undefined reference to `octomap::ColorOcTreeNode::prun
eNode()'
```

在网上百度了好久，找到了关于这个问题的帖子（https://github.com/OctoMap/octomap/issues/171）

1， 我首先用的方法是 sudo make，确实，编译通过，也安装成功了，但是我在编译自己写的 octomap_test 程序时，仍然会出现类似的 undefined reference to 的错误，于是在编译时还是需要 sudo；

2， 后来我觉得有一个对方法 1 的评价特别对：

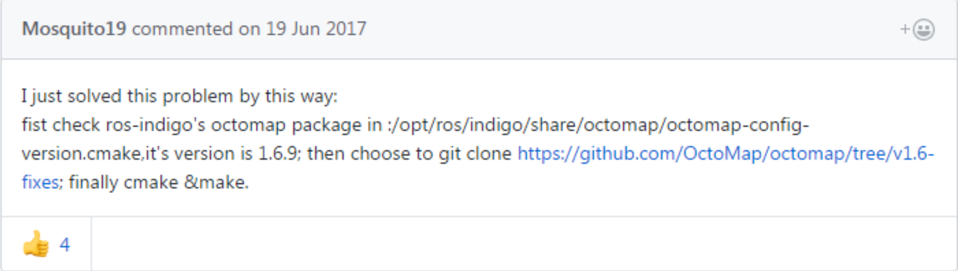


ahornung commented on 18 Sep 2017

> I also met this problem, it was solved by using sudo command:

Which, as I wrote above, is not a solution but only shadows the real problem. It might create all kinds of other strange behavior once you use the library. When compiling with ROS you should properly use a catkin overlay as ROS workspace.

于是我尝试了第二种做法：



Mosquito19 commented on 19 Jun 2017

I just solved this problem by this way:
fist check ros-indigo's octomap package in :/opt/ros/indigo/share/octomap/octomap-config-version.cmake,it's version is 1.6.9; then choose to git clone https://github.com/OctoMap/octomap/tree/v1.6-fixes; finally cmake &make.

打开查询了这个/opt/ros/indigo/share/octpmap/octomap-config-version 文件，我的版本是 1.6.9，在重新下载了 octomap 源码之后，校对了版本，具体指令如下：

```
git clone https://github.com/OctoMap/octomap
cd octomap
git tag    //列出所有版本，查看是否有自己的版本
git checkout v1.6.9   //校验为自己的版本
mkdir build
cd build
cmake ..
make
sudo make install
```

感觉第二种做法才是真正的解决方案。