

基于点云的三维点描述

一、实验目的

- 1、学会使用 blender 软件构造三维点云模型，并修改属性。
- 2、学会使用 PCL 库读取三维点云数据。
- 3、学会使用 PCL 库中 viewer 类，从不同角度观测三维数据，输出三维视图。
- 4、学会使用 ICP 算法，对三维点云数据匹配。

二、实验内容

本次实验利用 blender 软件构建一个三维数据模型，利用 PCL 库读取该模型并绘制三维数据，使用最小化均方误差对模型的三维点云数据匹配。

三、实验环境

实验平台类型	实验所用软件	软件所在位置
Ubuntu16.04	Blender+PCL+Codeblocks	/data

四、实验原理

三维成像技术的发展，结构光测量、激光扫描、ToF 等技术趋于成熟，物体表面的三维坐标能够精准而快速的获取，从而生成场景的三维数据，能够更好地感知和理解周围环境。三维数据包含了场景的深度信息，能够表示物体的表面形状，在机器人、AR/VR、人机交互、遥感测绘等多个领域具有广阔的应用前景。

三维数据由传感器直接获得，可以表示为深度图、点云、网格、CAD 等不同形式。其中点云数据获取便捷，易于存储，具有离散和稀疏特性，方便扩展为高维的特征信息，是近年来的研究主流方向。

然而，与二维图像中像素的规则排列方式不同，点云数据是无序的，这使得它很难直接应用卷积来获取三维点之间的局部相关性信息。同时，由于采集方法的原因，点云数据常常是非均匀分布的，不同局部区域的点云密度常常不等，这会为特征提取时，数据点的采样带来困难。此外，三维空间中物体的形变较二维图像更为复杂，除三个维度的仿射变换外，还有非刚体形变需要考虑。

ICP 用于求解两堆点云之间的变种关系。假设有两堆点云数据，分别记为两个集合。假设两个点云之间的变换关系为，这两个就是需要求解的参数。问题描述为最小化均方误差。

五、实验步骤

- 1、新建/data 目录，打开网站 <https://www.blender.org/download>, 下载 blender 压缩包, 保存在/data 路径下。

```
root@0e1df42e9b81: /# cd ~
root@0e1df42e9b81: ~# mkdir /data
```

图 1 新建文件



图 2 blender 下载

2、解压缩以上 blender 压缩包，并打开 blender 软件。

```
root@0e1df42e9b81: /data# tar -xjf blender-2.79b-linux-glibc219-x86_64.tar.bz2
root@0e1df42e9b81: /data# cd blender-2.79b-linux-glibc219-x86_64
root@0e1df42e9b81: /data/blender-2.79b-linux-glibc219-x86_64# ls
2.79          copyright.txt      LICENSE-bmonofont-i18n.ttf.txt
blender       GPL3- license.txt  LICENSE-droidsans.ttf.txt
blender.desktop  GPL- license.txt  ocio- license.txt
blenderplayer  icons             Python- license.txt
blender- softwaregl  jemalloc- license.txt  readme.html
blender.svg     lib
blender- thumbnailer.py  LICENSE-bfont.ttf.txt
root@0e1df42e9b81: /data/blender-2.79b-linux-glibc219-x86_64# ./blender
ALSA lib confmisc.c:768:(parse_card) cannot find card '0'
ALSA lib conf.c:4292:(_snd_config_evaluate) function snd_func_card_driver return
```

图 3 解压

3、单击编辑区域，并按“Delete”键，点击下图中 Delete X 删除方形模型，在 Add Mesh 下添加 Monkey 模型。

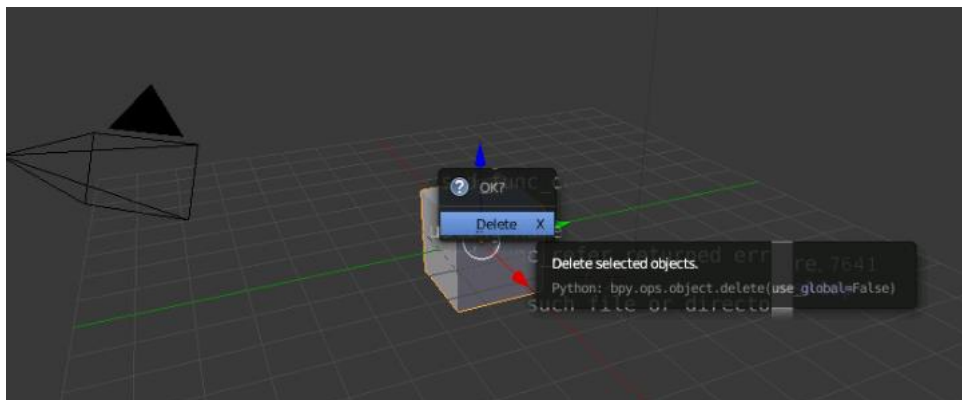


图 4 模型添加

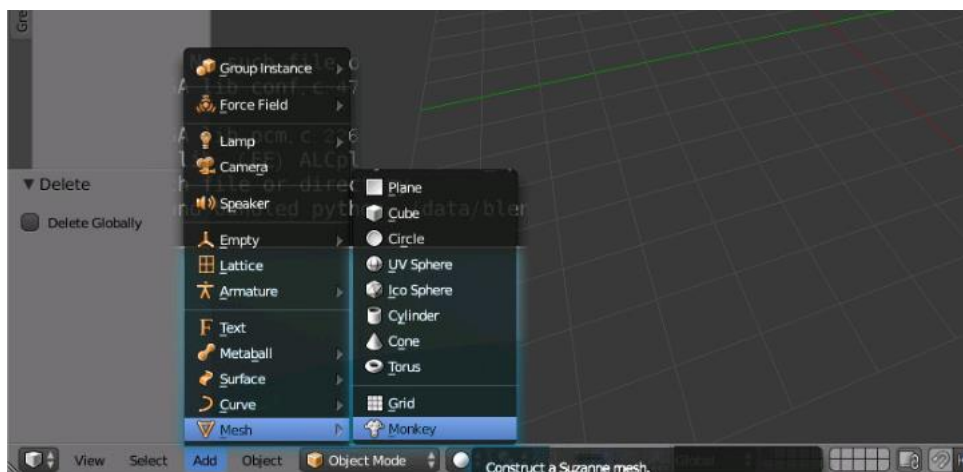


图 5 模型添加

4、在右侧点击设置按钮后，点击 Add Modifier,修改密度属性，将 View 的值设置为 3。并点击“Apply”应用设置。

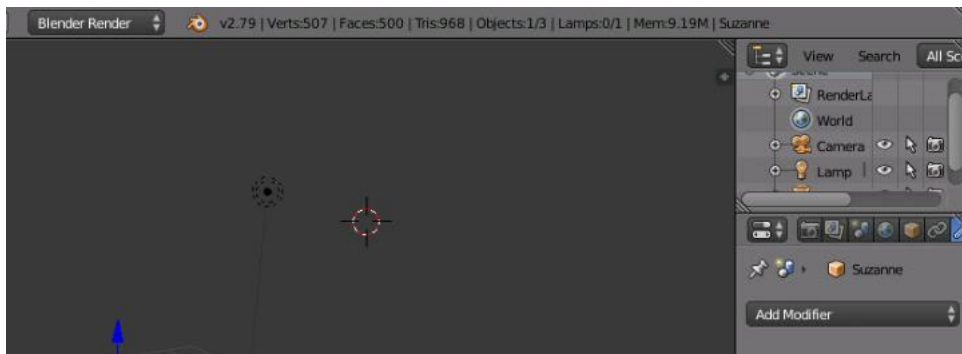


图 6 密度设置

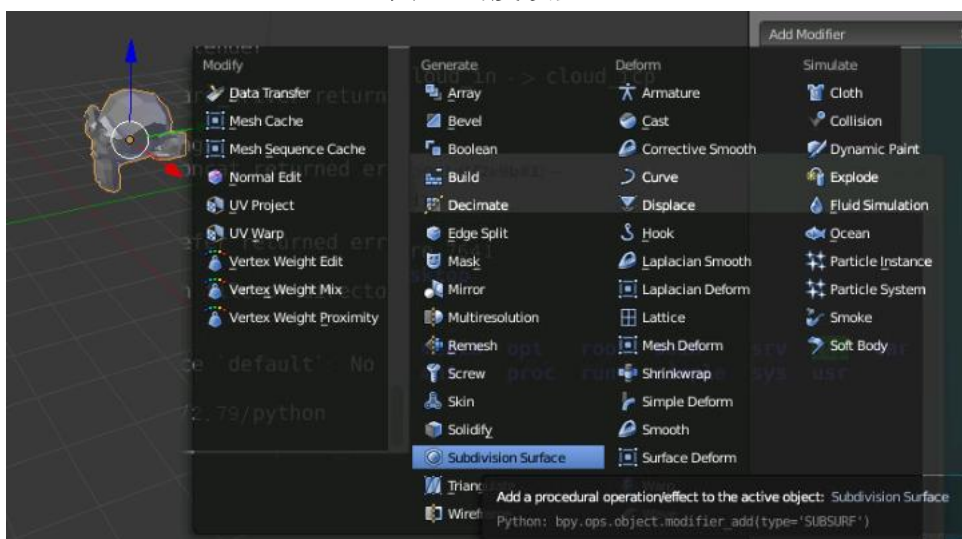


图 7 密度设置

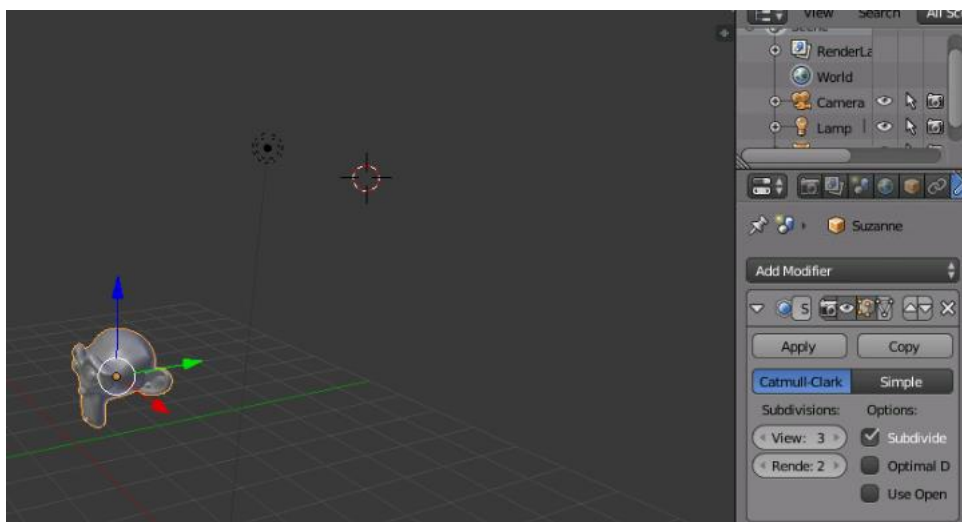


图 8 密度设置

5、点击 File,将该模型到处到/data 目录下，选择 Stanford (.ply)。在弹出的窗口选择/data 目录，命名为 monkey.ply。点击 Export PLY 保存。关闭该软件。

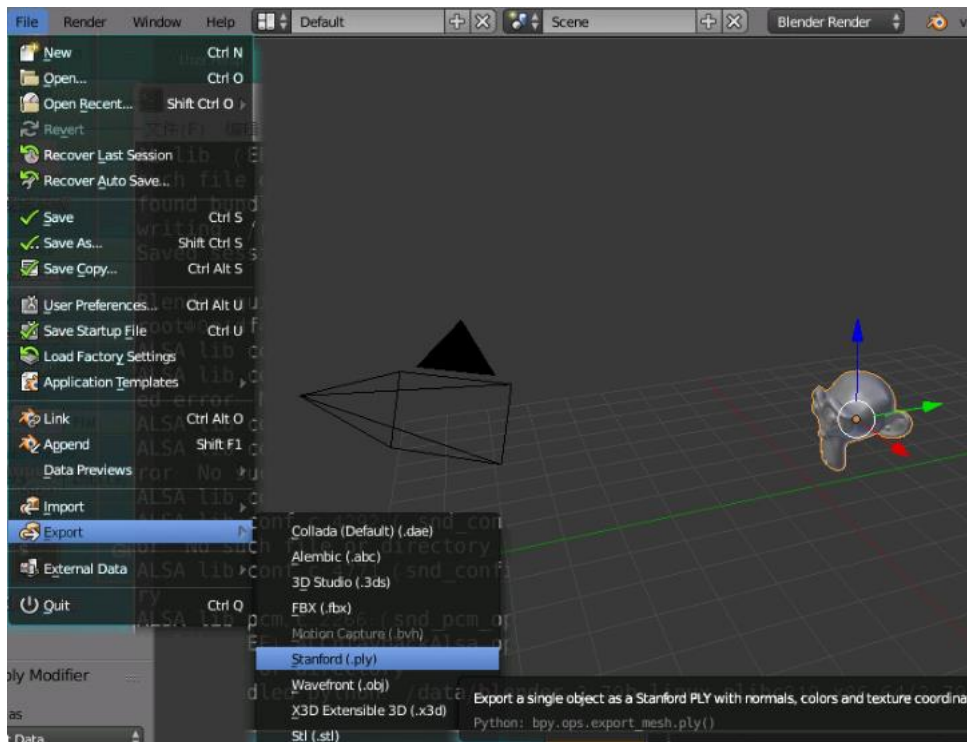


图 9 模型导出

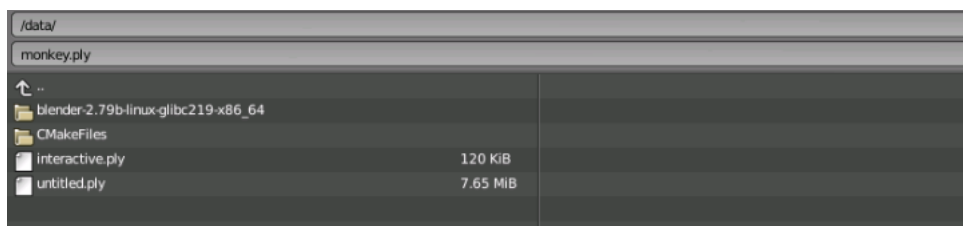


图 10 模型导出

6、检测/data 路径下是否存在 monkey.ply 文件，并打开 Codeblocks 软件。

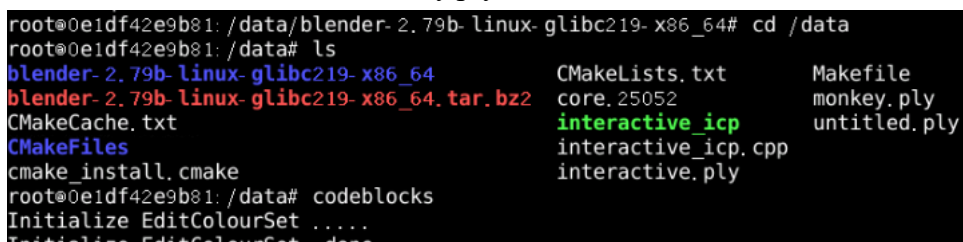


图 11 打开编程软件

7、在 File 下选择 C/C++ source,点击 Go,保存默认设置，点击 Next,路径设置为 /data,文件命名为 interactive_icp.cpp，点击 Finish 完成设置。

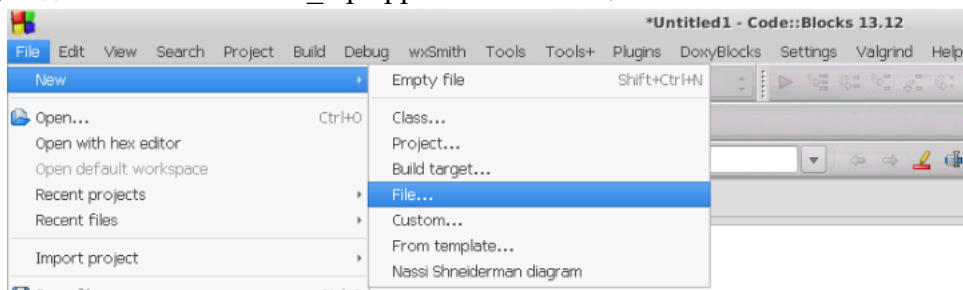


图 12 新建文件

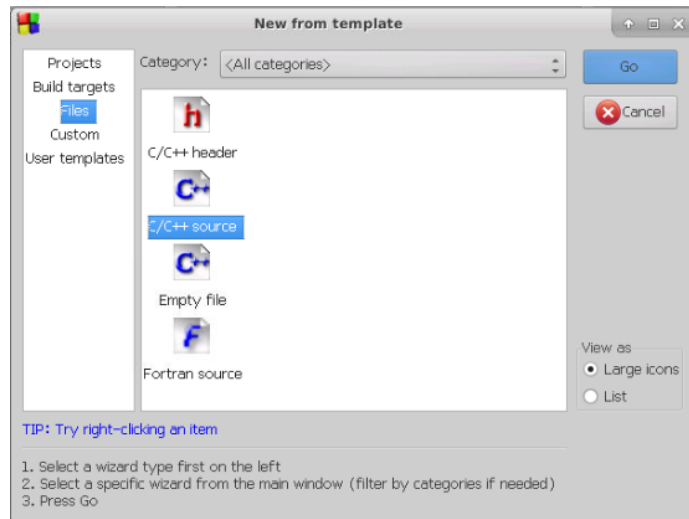


图 13 新建源文件

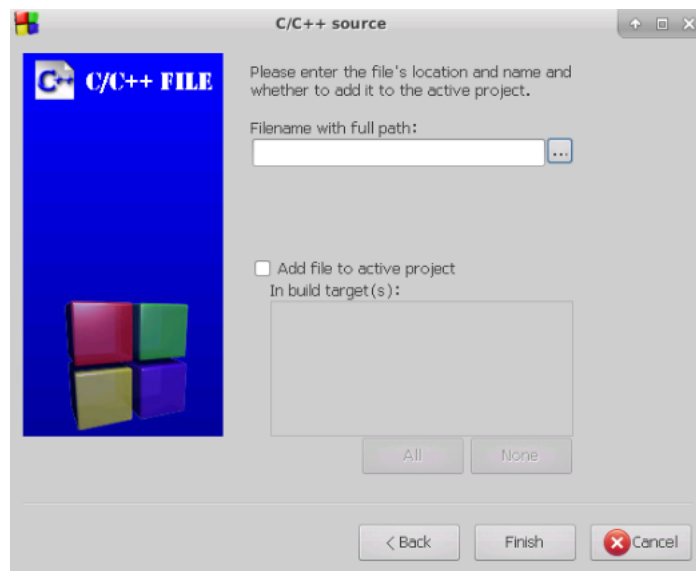


图 14 新建元文件

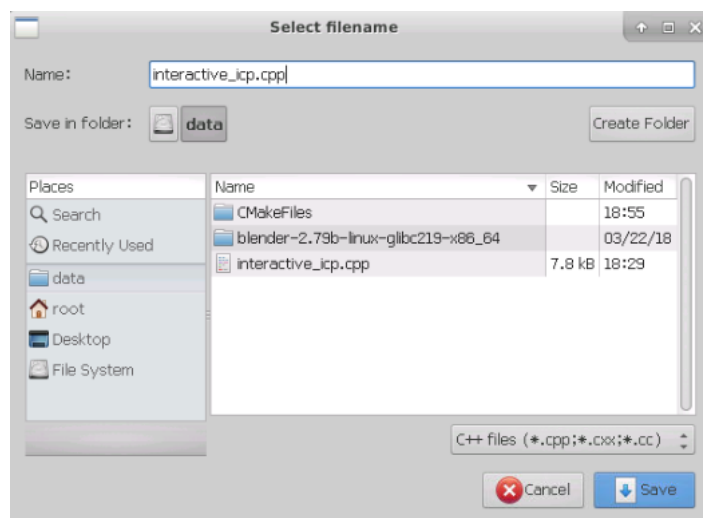


图 15 新建源文件

8、添加如下所示的库文件，PCL 库已经预安装到系统，不需要再次安装。

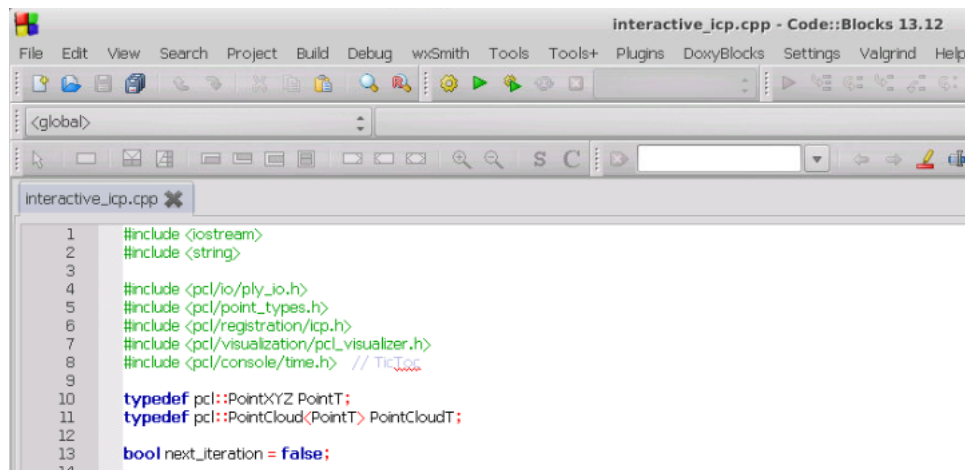


图 16 添加头文件

9、添加如下所示的宏定义，全局变量和 print 信息。

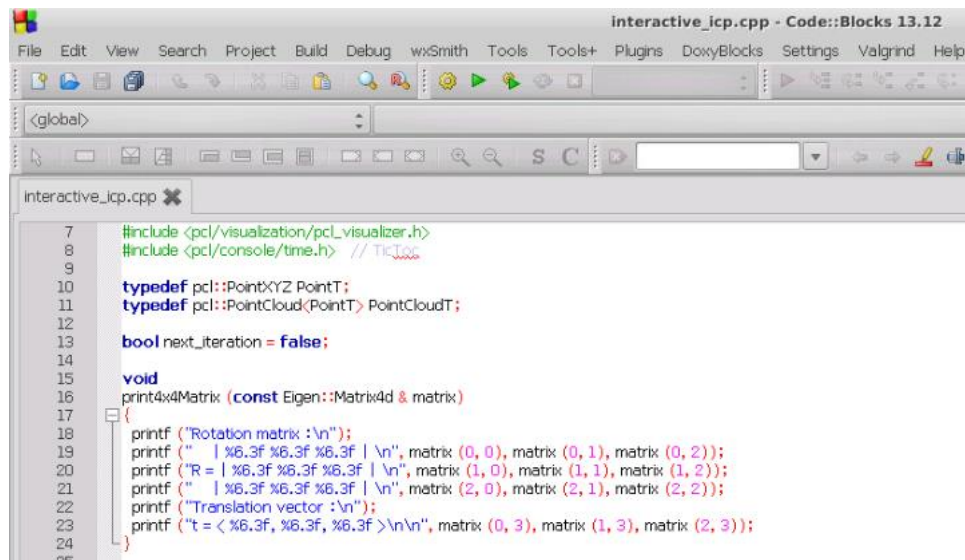


图 17 宏定义

10、keyboardEventOccurred 函数用于监听键盘输入，当输入为“Space”空格键，下次迭代 next_iteration 值被设置为“True”。

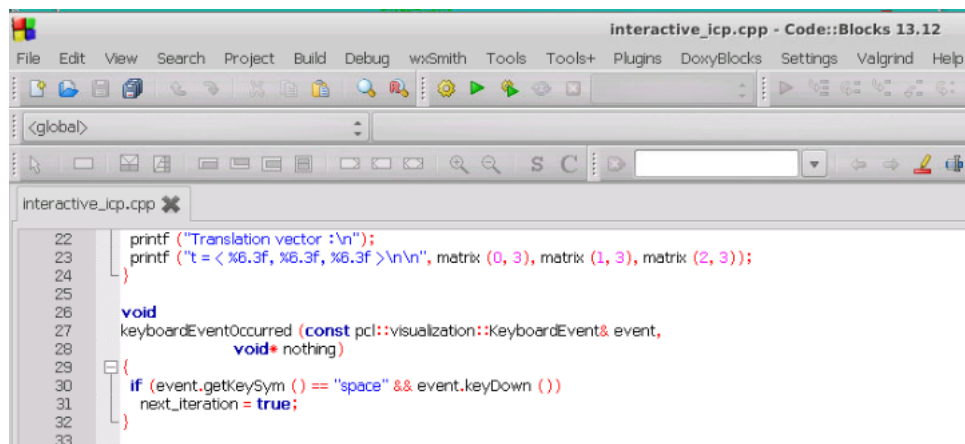
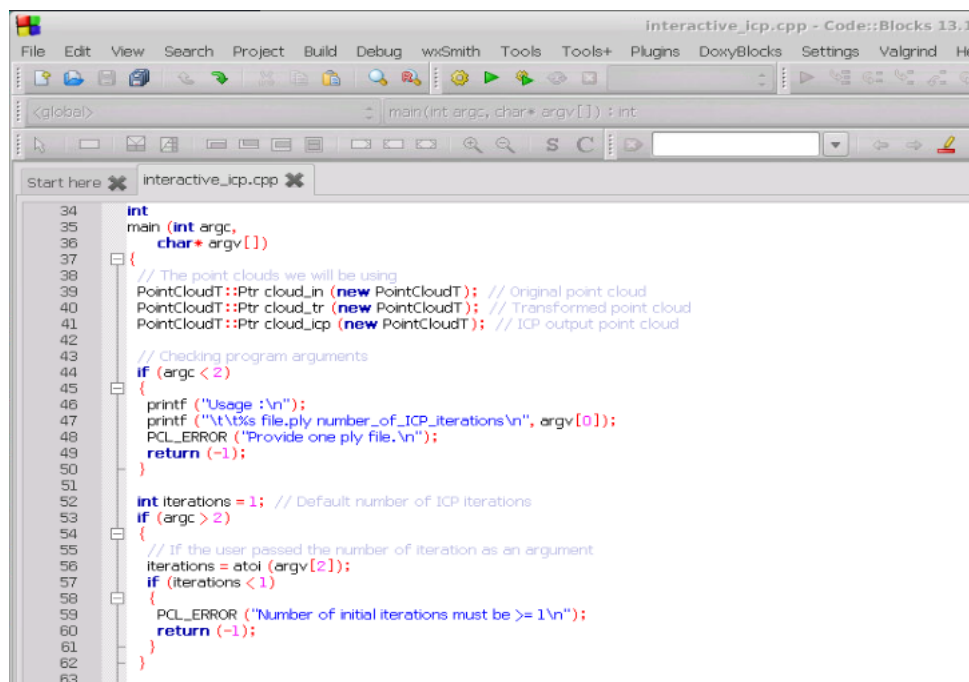


图 18 迭代设置

11、定义 main 函数，对输入的参数判断。且输入的第二个参数为迭代次数。

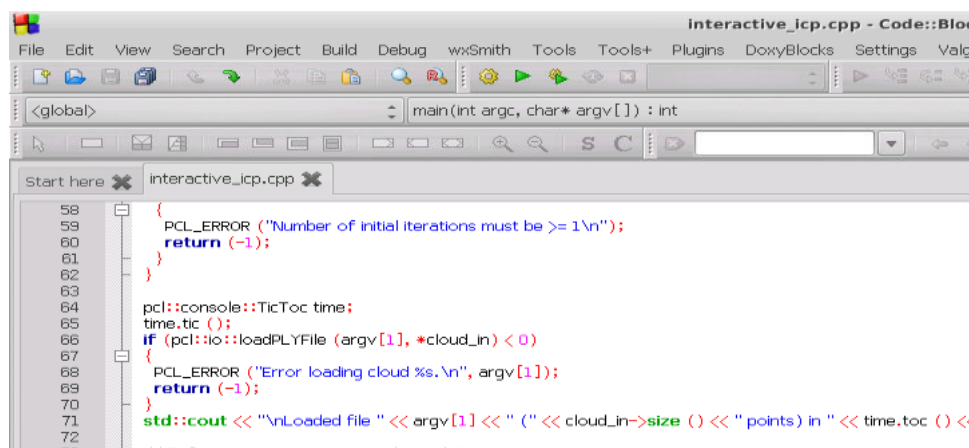


```
int
main (int argc,
      char* argv[])
{
    // The point clouds we will be using
    PointCloudT::Ptr cloud_in (new PointCloudT); // Original point cloud
    PointCloudT::Ptr cloud_tr (new PointCloudT); // Transformed point cloud
    PointCloudT::Ptr cloud_icp (new PointCloudT); // ICP output point cloud

    // Checking program arguments
    if (argc < 2)
    {
        printf ("Usage : \n");
        printf ("t\t%s file.ply number_of_ICP_iterations\n", argv[0]);
        PCL_ERROR ("Provide one ply file.\n");
        return (-1);
    }

    int iterations = 1; // Default number of ICP iterations
    if (argc > 2)
    {
        // If the user passed the number of iteration as an argument
        iterations = atoi (argv[2]);
        if (iterations < 1)
        {
            PCL_ERROR ("Number of initial iterations must be >= 1\n");
            return (-1);
        }
    }
}
```

图 19 参数定义

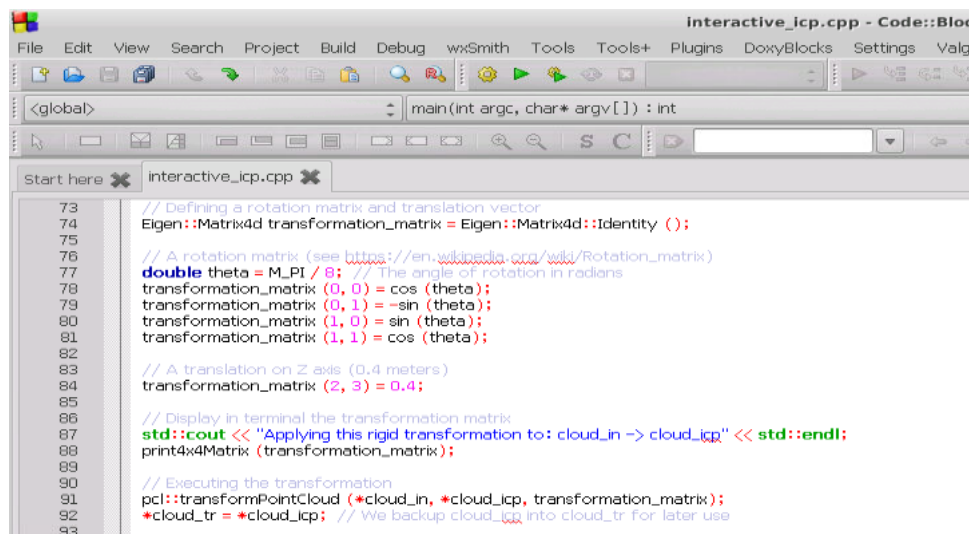


```
        PCL_ERROR ("Number of initial iterations must be >= 1\n");
        return (-1);
    }

    pcl::console::TicToc time;
    time.tic ();
    if (pcl::io::loadPLYFile (argv[1], *cloud_in) < 0)
    {
        PCL_ERROR ("Error loading cloud %s.\n", argv[1]);
        return (-1);
    }
    std::cout << "\nLoaded file " << argv[1] << " (" << cloud_in->size () << " points) in " << time.toc () << "\n";
    // Definition of rotation matrix and translation vector
}
```

图 20 参数定义

12、对点云数据转化，即旋转移位。



```
// Defining a rotation matrix and translation vector
Eigen::Matrix4d transformation_matrix = Eigen::Matrix4d::Identity ();

// A rotation matrix (see https://en.wikipedia.org/wiki/Rotation\_matrix)
double theta = M_PI / 8; // The angle of rotation in radians
transformation_matrix (0, 0) = cos (theta);
transformation_matrix (0, 1) = -sin (theta);
transformation_matrix (1, 0) = sin (theta);
transformation_matrix (1, 1) = cos (theta);

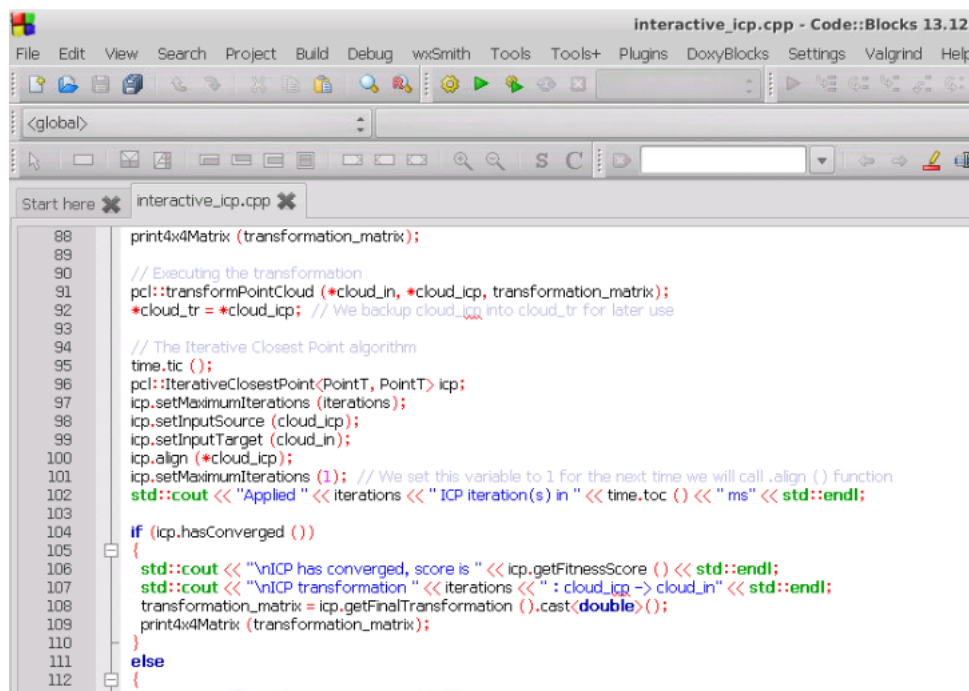
// A translation on Z axis (0.4 meters)
transformation_matrix (2, 3) = 0.4;

// Display in terminal the transformation matrix
std::cout << "Applying this rigid transformation to: cloud_in -> cloud_icp" << std::endl;
print4x4Matrix (transformation_matrix);

// Executing the transformation
pcl::transformPointCloud (*cloud_in, *cloud_icp, transformation_matrix);
*cloud_tr = *cloud_icp; // We backup cloud_icp into cloud_tr for later use
```

图 21 点云数据转化

14、构建如下所示 ICP 迭代参数。



```

interactive_icp.cpp - Code::Blocks 13.12
File Edit View Search Project Build Debug wxSmith Tools Tools+ Plugins DoxyBlocks Settings Valgrind Help

<global>

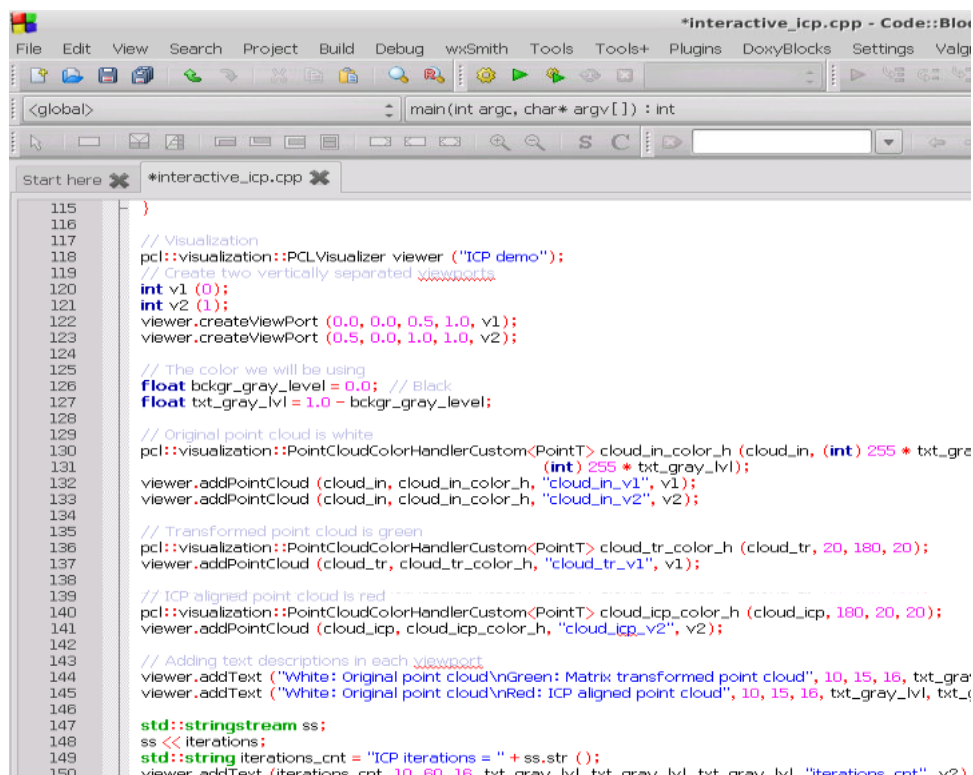
Start here X interactive_icp.cpp X

88 print4x4Matrix (transformation_matrix);
89
90 // Executing the transformation
91 pcl::transformPointCloud (*cloud_in, *cloud_icp, transformation_matrix);
92 *cloud_tr = *cloud_icp; // We backup cloud_icp into cloud_tr for later use
93
94 // The Iterative Closest Point algorithm
95 time.tic ();
96 pcl::IterativeClosestPoint<PointT, PointT> icp;
97 icp.setMaximumIterations (iterations);
98 icp.setInputSource (cloud_icp);
99 icp.setInputTarget (cloud_in);
100 icp.align (*cloud_icp);
101 icp.setMaximumIterations (1); // We set this variable to 1 for the next time we will call .align () function
102 std::cout << "Applied " << iterations << " ICP iteration(s) in " << time.toc () << " ms" << std::endl;
103
104 if (icp.hasConverged ())
105 {
106     std::cout << "\nICP has converged, score is " << icp.getFitnessScore () << std::endl;
107     std::cout << "\nICP transformation " << iterations << " : cloud_icp -> cloud_in" << std::endl;
108     transformation_matrix = icp.getFinalTransformation ().cast<double>();
109     print4x4Matrix (transformation_matrix);
110 }
111 else
112 {

```

图 22 ICP 迭代

14、输出三维点云视图，如下图所示。并使用 while 循环迭代匹配。完成后保存文件，关闭该软件。



```

*interactive_icp.cpp - Code::Blocks
File Edit View Search Project Build Debug wxSmith Tools Tools+ Plugins DoxyBlocks Settings Valgrind Help

<global> main(int argc, char* argv[]) : int

Start here X *interactive_icp.cpp X

115 }
116
117 // Visualization
118 pcl::visualization::PCLVisualizer viewer ("ICP demo");
119 // Create two vertically separated viewports
120 int v1 (0);
121 int v2 (1);
122 viewer.createViewport (0.0, 0.0, 0.5, 1.0, v1);
123 viewer.createViewport (0.5, 0.0, 1.0, 1.0, v2);
124
125 // The color we will be using
126 float bckgr_gray_level = 0.0; // Black
127 float txt_gray_lvl = 1.0 - bckgr_gray_level;
128
129 // Original point cloud is white
130 pcl::visualization::PointCloudColorHandlerCustom<PointT> cloud_in_color_h (cloud_in, (int) 255 * txt_gray_lvl);
131 viewer.addPointCloud (cloud_in, cloud_in_color_h, "cloud_in_v1", v1);
132 viewer.addPointCloud (cloud_in, cloud_in_color_h, "cloud_in_v2", v2);
133
134 // Transformed point cloud is green
135 pcl::visualization::PointCloudColorHandlerCustom<PointT> cloud_tr_color_h (cloud_tr, 20, 180, 20);
136 viewer.addPointCloud (cloud_tr, cloud_tr_color_h, "cloud_tr_v1", v1);
137
138 // ICP aligned point cloud is red
139 pcl::visualization::PointCloudColorHandlerCustom<PointT> cloud_icp_color_h (cloud_icp, 180, 20, 20);
140 viewer.addPointCloud (cloud_icp, cloud_icp_color_h, "cloud_icp_v2", v2);
141
142 // Adding text descriptions in each viewport
143 viewer.addText ("White: Original point cloud\nGreen: Matrix transformed point cloud", 10, 15, 16, txt_gray_lvl);
144 viewer.addText ("White: Original point cloud\nRed: ICP aligned point cloud", 10, 15, 16, txt_gray_lvl, txt_gray_lvl);
145
146 std::stringstream ss;
147 ss << iterations;
148 std::string iterations_cnt = "ICP iterations = " + ss.str ();
149 viewer.addText (iterations_cnt, 10, 10, 16, txt_gray_lvl, txt_gray_lvl, "iterations_cnt", v2);
150

```

图 23 三维点云视图

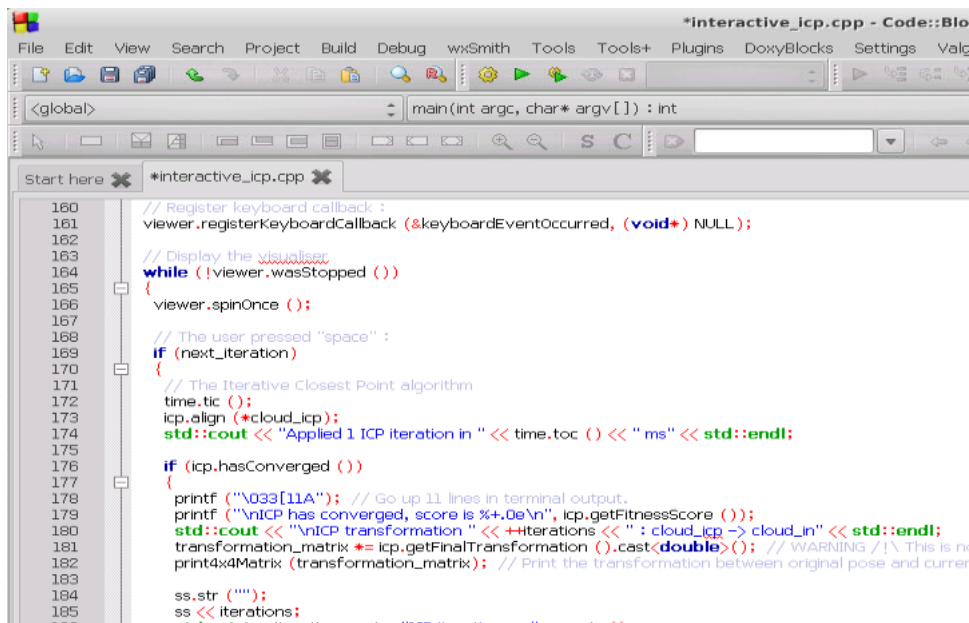


图 24 三维点云视图

15、在/data 目录下，输入“vim CMakeLists.txt.”。并添加如下所示的内容，并保存。（如果不会使用 vim，可使用其他编辑器，如 gedit 等）。

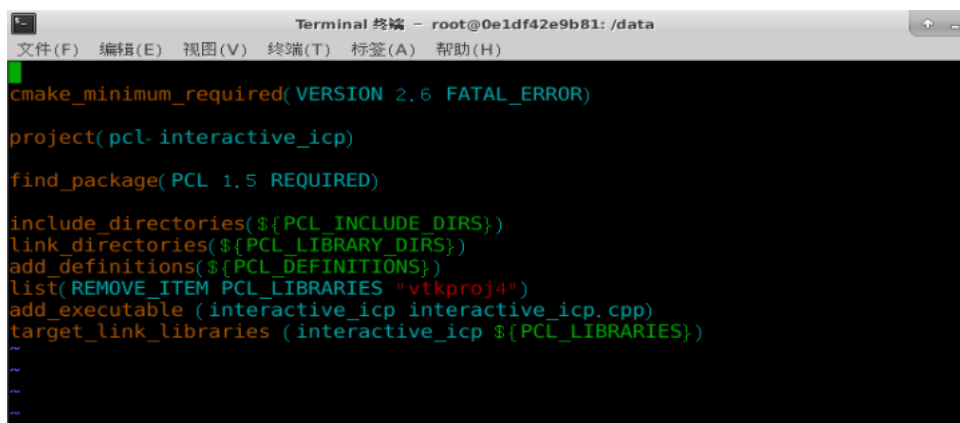


图 25 设置

16、运行“cmake ./”生成编译文件。完成后在/data 路径下存在 Makefile 文件。输入 make,等待编译完成。

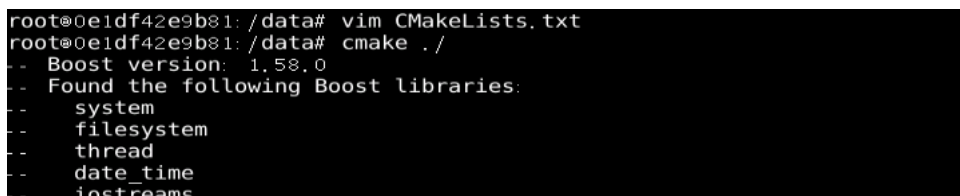


图 26 编译

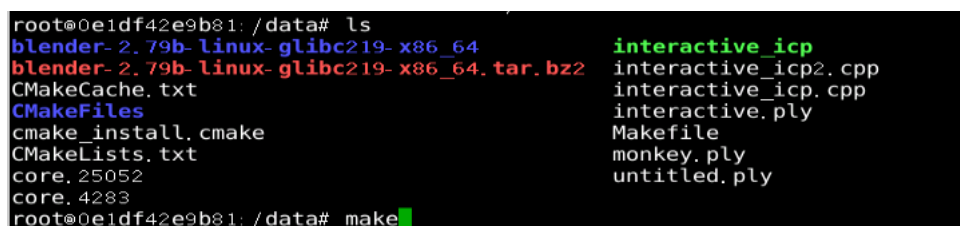


图 27 编译

17、执行程序，得到如下图所示的三维点云效果图。

```
root@0e1df42e9b81:/data# ./interactive_icp monkey.ply 1
[pcl::PLYReader] monkey.ply: 12: property 'list uint8 uint32 vertex_indices' of element 'face' is not handled

Loaded file monkey.ply (125952 points) in 396 ms

Applying this rigid transformation to: cloud_in -> cloud_icp
Rotation matrix :
R = | 0.924 -0.383 0.000 |
    | 0.383 0.924 0.000 |
    | 0.000 0.000 1.000 |
```

图 28 执行

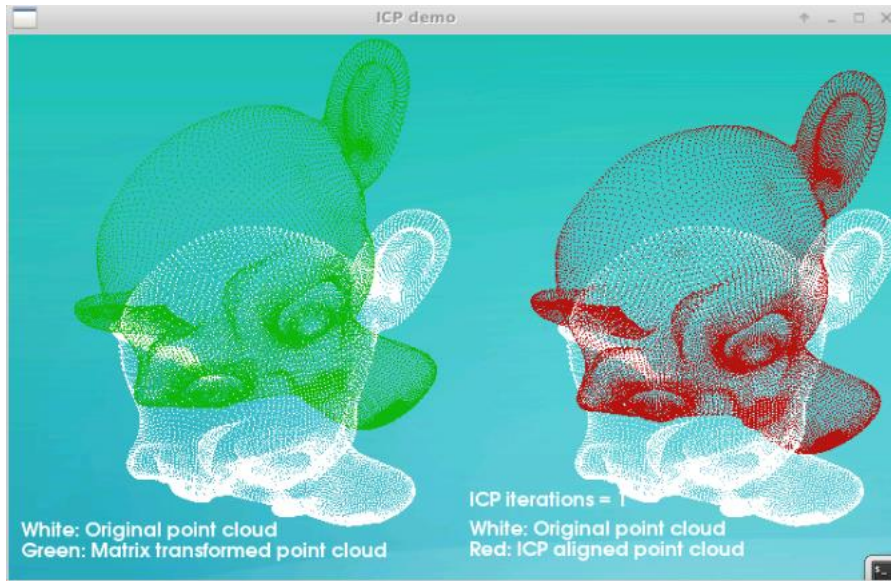


图 29 匹配结果

18、再次执行程序，需改迭代次数为 70，右图得到更好的匹配效果。

```
root@0e1df42e9b81:/data# ./interactive_icp monkey.ply 70
[pcl::PLYReader] monkey.ply: 12: property 'list uint8 uint32 vertex_indices' of element 'face' is not handled

Loaded file monkey.ply (125952 points) in 367 ms

Applying this rigid transformation to: cloud_in -> cloud_icp
Rotation matrix :
R = | 0.924 -0.383 0.000 |
    | 0.383 0.924 0.000 |
    | 0.000 0.000 1.000 |
```

图 30 执行

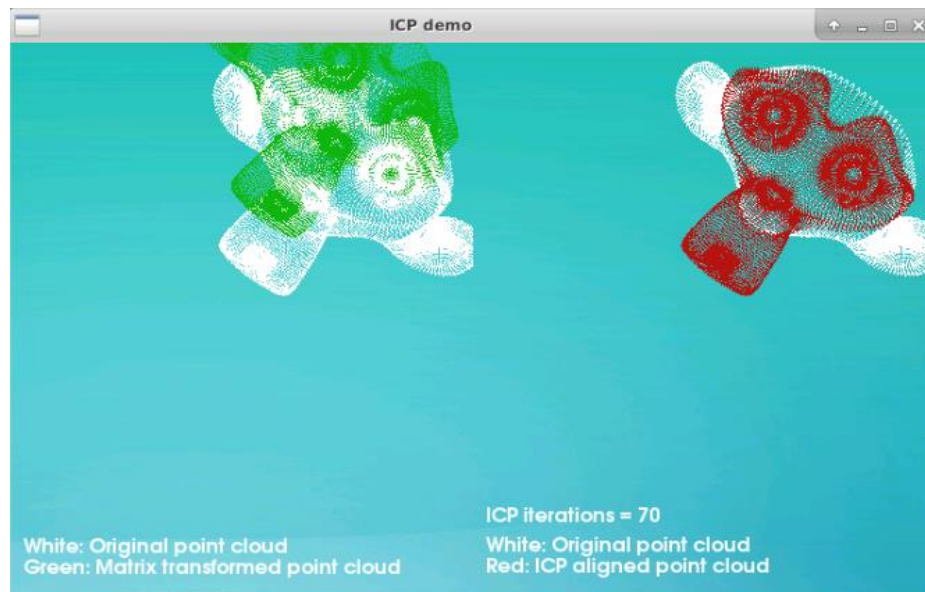


图 31 匹配结果