CSE 145/237D FINAL REPORT

3D Reconstruction with Dynamic Fusion

Junyu Wang, Zeyangyi Wang

.

Contents

.

Abstract

This paper introduces a new integration of Dynamic Fusion with Intel Realsense camera to improve sample quality and reconstruction consistency. The background section includes a general introduction to this research topic. Next in the implementation section we'll discuss technical details of our attempts in length, divided as several stages of the project. The next section is a comparison to our old milestones at the beginning of this quarter and a discussion of difficulties we have met so far. In the end we'll conclude with our contributions to Dynamic Fusion and potential goals in the future.

Background

Dynamic 3D reconstruction technique has been a recent focus for real time medical augmented reality research that helps diagnosing and imaging during operations. 3D graphics model is mathematical representation of any three-dimensional object that can be displayed visually as a two-dimensional image through a process called 3D rendering or used in non-graphical computer simulations and calculations. The medical application of 3D reconstruction collects data from a tiny camera that receives both visual and depth information and then transmits to the backend processing unit, which outputs an accurate dynamic reconstruction of organs and their relative locations inside the patient's body. It is not only more accurate, but also much less intrusive therefore safer than traditional methods.

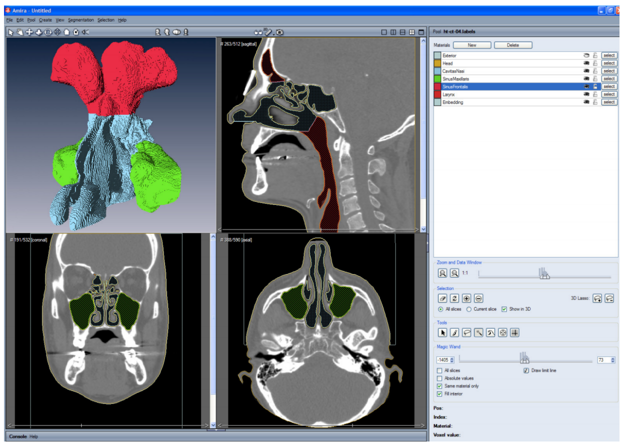


Fig.1 3D reconstruction of upper respiratory tract

The concept of dynamic reconstruction is based on static reconstruction by University of Washington in *KinectFusion: Real-Time Dense Surface Mapping and Tracking*[1] , which utilizes a simple depth camera and real time 3D mapping algorithms to build a 3D surface tracking system. KinectFusion is an open source project and requires minimal computing resources as commodity depth camera and graphical processing units will suffice. The picture below is an example of depth mapping of a desktop. In grayscale mapping the brightness indicates closeness to the camera and shades means the object is farther away.

.

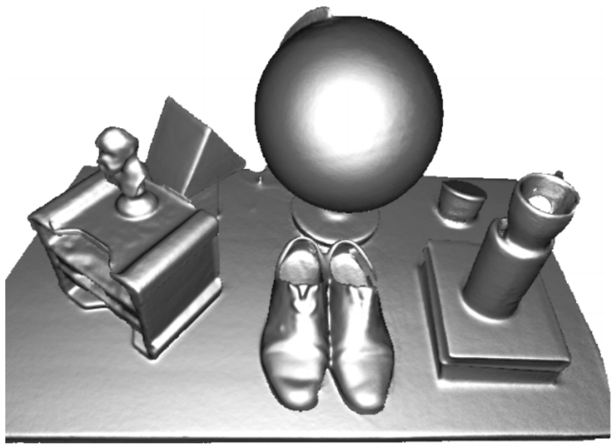


Fig.2 Static reconstructed scene by KinectFusion

Dynamic Fusion is essentially an extension of KinectFusion but includes reconstruction of objects in motion. We referred to the article: *DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time*[2] that implements a real time 3D registration system with almost the same hardware as KinectFusion. Dynamic Fusion introduces SLAM system that provides estimation of interstate of moving objects based on interpolation and reflecting lights in the scene. However unlike KinectFusion, the technical details of Dynamic Fusion are not all disclosed, bringing difficulties to replicating their results for further development. The following example captures a succession of movements by a man holding a cup. Although the depth camera has a relatively low sampling rate, the missing details in between scenes are interpolated by Dynamic Fusion and it actually outputs smooth scenes.

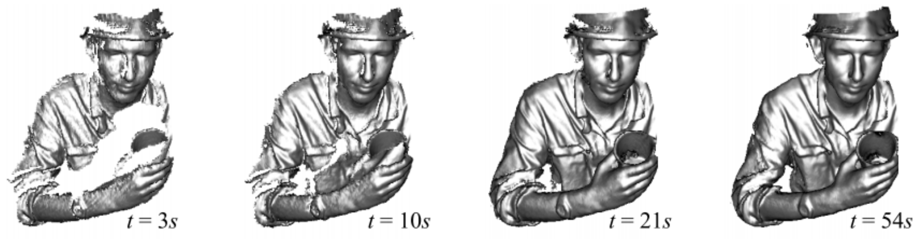


Fig.3 Example scene snapshots by Dynamic Fusion

Current research & development around Dynamic Fusion has all been implemented on Microsoft Kinect camera system, as there is no other official support package provided for other systems. This article explores a possible improvement by creating an interface between Intel Realsense camera and Dynamic Fusion. We believe an integration with Intel Realsense camera brings better results and opens new research possibilities for the following advantages:

- Better scanning depth. The camera is recording at a very close distance from the objects in our application and Realsense is optimized to capture gestures/facial expressions of close objects, while Kinect focus on body movements for gaming.

- More flexible. Realsense has power/data cords combined in one USB cable. Also more new support packages/post processing libraries provided.

.

Fig.4 Comparison between Realsense(left) and Kinect(right) cameras.

Implementation

Since Realsense does not support Dynamic Fusion directly, we have to gather data from Realsense and port into a popular computer vision development platform - Point Cloud Library (PCL) then convert it into .pcd file which Dynamic Fusion can read. This section includes discussion of technical procedures of this project, design decisions and implementation details, with five subsections organized step by step in chronological order.

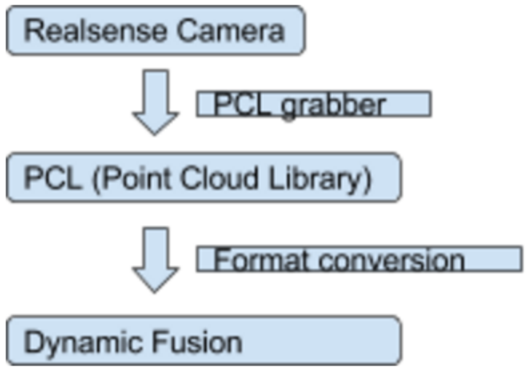


Fig.5 Diagram of implementation flow

Development setup

The development environment is Thinkpad W530 + Linux Debian 8.0/Windows 10. We built and installed Dynamic Fusion project from BitBucket. The project compiles on Nvidia Eclipse, an IDE specifically designed for computer vision developments. Other dependencies include CUDA libraries and Realsense SDK package.

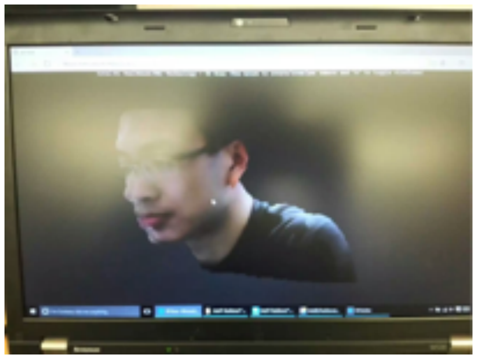


Fig.6 Realsense depth image

.

Real time capturing

To run initial tests with Dynamic Fusion and save input data for comparison with Realsense, we set up a testing bench and collected a set of samples with KinectFusion. The testbench used a filled water balloon fixed on a flat table with strings attached to the surface of the balloon. The balloon is used to simulate soft tissue with noticeable change of shape under external pressure. During recording the string will be stretched which causes deformation of the surface of the balloon. The collected data in .pcd format are visually represented in three dimensions as the picture shows below.

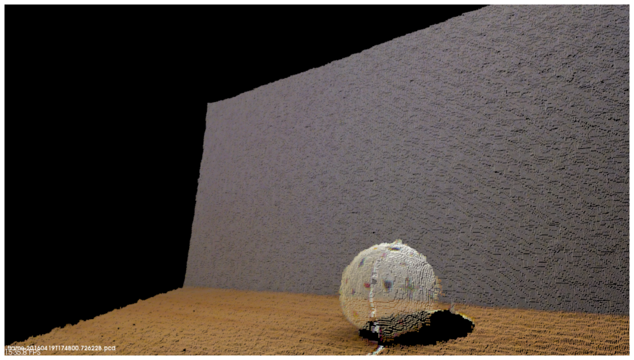


Fig.7 Data set 3 reconstruction image

For better comparison with other environmental variables, we conducted the experiments multiple times under different conditions as explained the table below.

Conditions	Camera Moving Vertically	Camera Moving Horizontally	Camera Static
Balloon Deforming	Data set 1	Data set 2	Data set 3
Balloon Static	Data set 4	Data set 5	Data set 6

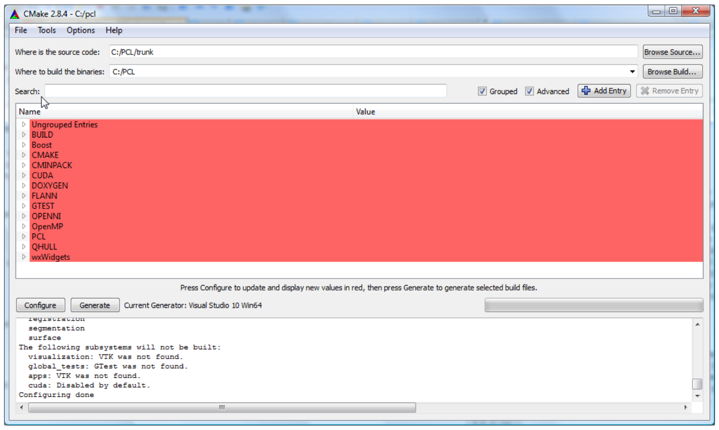
Build Point Cloud Library

Before installing PCL, we need the following dependencies:

Name	Description & Usage
Boost	Threading and shared pointer library for C++
Eigen	Matrix backend for SSE optimized computation
FLANN	Library for fast approximate nearest neighbors search
VTK	3D point cloud rendering and visualization package

Then we used Cmake to build PCL.After configuration, build the project in Microsoft Visual Studio.

.



Where is the source code: C:/PCL/trunk

Where to build the binaries: C:/PCL

Search:

☑ Grouped ☑ Advanced

Name	Value
Ungrouped Entries	
BUILD	
Boost	
CMAKE	
CMINPACK	
CUDA	
DOXYGEN	
FLANN	
GTEST	
OPENNI	
OpenMP	
PCL	
QHULL	
wxWidgets	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 10 Win64

```
  registration
  segmentation
  surface
The following subsystems will not be built:
  visualization: VTK was not found.
  global_tests: GTest was not found.
  apps: VTK was not found.
  cuda: Disabled by default.
Configuring done
```

Fig.8 Cmake configure & generate PCL

Build PCL grabber

The PCL grabber [3] integrates with Realsense soft development package and is able to capture and save video stream in real time. The grabber automatically converts format to PointCloud.

Building PCL grabber follows almost the same procedure as we did in Cmake & Visual Studio with PCL. However it has a few additional dependencies.

Name	Description & Usage
Gtest	Testbench by Google for C++ development
OpenNI	Grab point clouds from OpenNI compliant devices
Qt	Provide graphical user interface for PCL_Viewer

```
void
pcl::RealSenseGrabber::run ()
{
  const int WIDTH = mode_selected_.depth_width;
  const int HEIGHT = mode_selected_.depth_height;
  const int SIZE = WIDTH * HEIGHT;

  PXCProjection* projection = device_->getPXCDevice ().CreateProjection ();
  PXCCapture::Sample sample;
  std::vector<PXCPoint3DF32> vertices (SIZE);
  createDepthBuffer ();

  while (is_running_)
  {
    pcl::PointCloud<pcl::PointXYZ>::Ptr xyz_cloud;
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr xyzrgba_cloud;
```

Fig.9 PCL grabber.cpp

Fig.9 is a snippet of source code from PCL grabber. The system works in a few steps:

a. Push depth image to the depth buffer.
b. Pull filtered depth image from the depth buffer.
c. Project (filtered) depth image into 3D.
d. Fill XYZ point cloud with computed points.

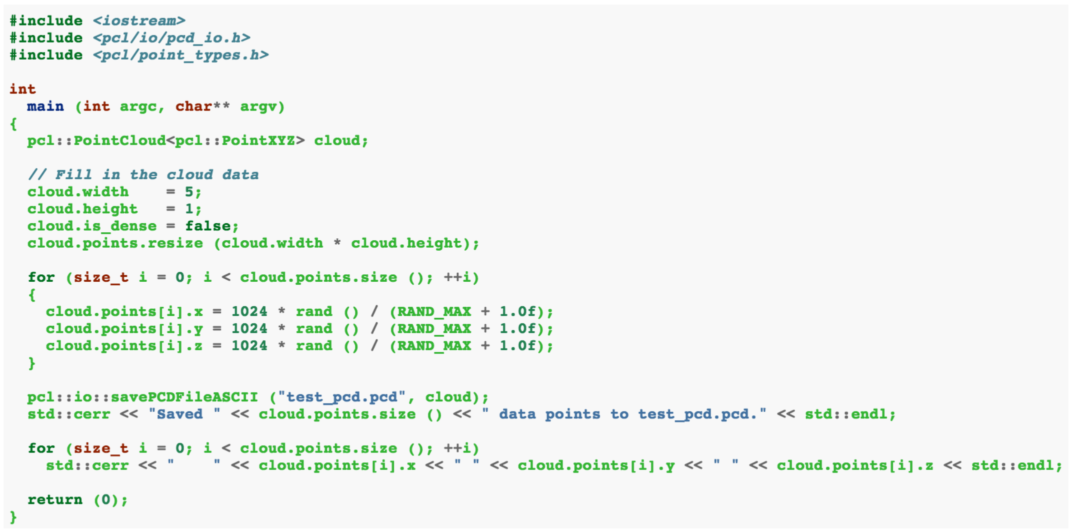
.

e. Fill XYZ-RGBA point cloud with computed points.
f. Assign colors to points in XYZRGBA point cloud.

The Point Cloud Data is saved in C++ std::vector array. It consists of multiple variables of space, distance and colors on the captured scene. The basic variables are XYZ that gives orientation of each point in space. RGBA on the other hand defines color of each point.

Write to Dynamic Fusion

When we have gathered input data from Realsense that is readable to PCL, the last step is to write it as .pcd file for Dynamic Fusion. Fortunately PCL has included a writer file as below:



```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>

int
  main (int argc, char** argv)
{
  pcl::PointCloud<pcl::PointXYZ> cloud;

  // Fill in the cloud data
  cloud.width    = 5;
  cloud.height   = 1;
  cloud.is_dense = false;
  cloud.points.resize (cloud.width * cloud.height);

  for (size_t i = 0; i < cloud.points.size (); ++i)
  {
    cloud.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
    cloud.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
    cloud.points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
  }

  pcl::io::savePCDFileASCII ("test_pcd.pcd", cloud);
  std::cerr << "Saved " << cloud.points.size () << " data points to test_pcd.pcd." << std::endl;

  for (size_t i = 0; i < cloud.points.size (); ++i)
    std::cerr << "    " << cloud.points[i].x << " " << cloud.points[i].y << " " << cloud.points[i].z << std::endl;

  return (0);
}
```

Fig.10 pcd_write.cpp

Milestones & Problems

We spent the first four weeks working on background research, initial setup and sample collection. Then for the next two weeks we tried creating a valid representation of warp-field, an essential part of Dynamic Fusion's rendering algorithm. In the sixth week our project advisor recommended a new set of milestones because we were unlikely to finish all tasks in time. We actually started working on most of this project very late into the quarter since we lost contact with him midway through the quarter for about a week. We have initially attempted to implement PCL grabber on Linux but found out later it was not feasible.

The lack of basic functioning resources and tools also contributed to our slow start. We spent a lot of time setting up work environment from installing operating system to finding dependencies for software packages. The Intel Realsense camera came with unstable connections. Since there is only a single laptop with proper graphical processing unit we could not develop separately so if one of us was stuck we had to wait till the issue was solved. Realsense integration with Dynamic Fusion is a new topic therefore there is little previous experience or solutions online when we face certain technical difficulties.

Milestones	Results	Details
Background research based on	Completed	Finished on time.

.

Microsoft and UWashington papers.		
Initial setup of development environment.	Completed	Took more time than expected. Did another setup for new set of milestones.
Gather input data for DynamicFusion.	Completed	Finished on time.
Implement a representation of a warp field based on the KinFu TSDF.	Did not complete	The object takes more time than we expected and is out of scope of this quarter.
Implement the rest of Dynamic Fusion based on our own modification of KinectFusion.	Did not complete	The object takes more time than we expected and is out of scope of this quarter.
Modify or add image grabber to Intel Realsense camera API then output depth map to pcl format.	Did not complete	Revised after week 6. First attempt on Linux failed because PCL grabber requires RSSDK (Windows only). Cmake dependency issue when building PCL grabber.
Convert pcl formatted data into .pcd files.	Completed	Revised after week 6. Finished on time.

We are unable to produce a final demo because the PCL grabber installation failed to build. The error message suggests Cmake can not link to the correct version of VTK library even when we have placed it under the right path. The problem seems platform independent and we tried with all versions of VTK package we can find. There are no similar problems online and our project instructor also helped debugging. If time permits we'd rebuild everything from the start on a clean system.

Conclusion

Realsense integration with Dynamic Fusion provides a better alternative compared to Kinect implementation and is well suited for medical reconstruction. Our implementation creates a reliable interface between two platforms. It makes Dynamic Fusion more powerful now it has higher resolution in close up and also makes object tracking easier. Currently this project only works on Windows platform but the same PCL grabber could work on Linux and MacOS when Intel releases its support package & driver for more platforms.

References

[1] http://grail.cs.washington.edu/projects/dynamicfusion/papers/DynamicFusion.pdf
[2] http://homes.cs.washington.edu/~newcombe/papers/newcombe_etal_ismar2011.pdf

.

[3] https://github.com/taketwo/rs
Fig.1 https://opus4.kobv.de/opus4-zib/files/1044/ZR_07_41.pdf
Fig.2 http://homes.cs.washington.edu/~newcombe/papers/newcombe_etal_ismar2011.pdf
Fig.3 http://grail.cs.washington.edu/projects/dynamicfusion/papers/DynamicFusion.pdf