3D Reconstruction Using Intel RealSense L515

Mr. Sagar Tyagi[1], Mr. Shishir .K Singh[2]
[1](SVKM's MPSTME NMIMS University)
[2](Bhabha Atomic Research Centre, BARC)
(E-mail: sagartyagi@hotmail.com)

***Abstract*—**3D reconstruction involves a digital representation of real world objects. We have developed a working model of real-time 3D reconstruction of objects using Open3D module for python and Intel RealSense L515 RGBD (Red-Green-Blue-Depth) Li-Dar camera the algorithm deals with 3D data with 3 color channels and 1 depth channel as input from camera and provide 3D Modeling in visualizer which can be exported to use in creation of virtual reality, animation, preservation of object in soft form.

***Keywords*—** *Li-Dar Camera,Intel RealSense, RGBD, 3D Model, Reconstruction*

I. INTRODUCTION

The Field of 3D-reconstruction involves creating a digital modelling of real life objects using RGB camera with depth sensor which provides color and depth information to algorithm. The module used are: Open3D, Numpy, Time, and Pyrealsense2. The function of algorithm flows starting from capturing RGBD images, aligning and synchronizing the pair of images, point cloud formation, voxel down sampling, outlier detection and surface reconstruction. As processing time is crucial for real-time reconstruction system the paper also discuss the efficiency of the different methodology of outlier detection and surface reconstruction. In this paper we have developed a real time efficient 3D reconstruction system using pipeline with study of difference in processing time to generate effective and reliable reconstruction system.

II. SYSTEM DESCRIPTION AND FLOW

A. Pipeline

The pipeline eases the accessibility of computer vision modules. Pipeline manages computer vision modules, camera configuration, streaming, vision modules triggering and threading. It can regulate the output from the camera depending upon camera hardware specification, in our case RealSense L515 we are varying the frame rates, initializing configuration of camera, and controlling type of frame from color to depth. Pipeline configuration code [4]:

```
import pyrealsense2 as rs
# Configure Intel RealSense camera
pipeline = rs.pipeline()
config = rs.config()
#streaming color and depth frame from camera
#streaming format as bgr8 and z16
config.enable_stream(rs.stream.color, rs.format.bgr8)
config.enable_stream(rs.stream.depth, rs.format.z16)

pipeline.start(config)

# ....process from algorithim ....

pipeline.stop()
```

Figure: Pipeline Initialization

B. Data

Initial aim of the input data is to convert the depth and color information to point cloud information. Therefore, the color and depth frames are aligned and synchronized so, they can be matched with each other and a RGBD frame can be created the frames are captured using frames.get_<type of frame>_frames.[1]

```
frames = pipeline.wait_for_frames()
color_frame = frames.get_color_frame()
depth_frame = frames.get_depth_frame()
```

Figure: Getting Frames (color and depth)

C. Frame

Converting the color and depth frame to numpy array using Open3D and numpy module as:

```
# Convert frames to numpy arrays
color_image = np.asanyarray(color_frame.get_data())
depth_image = np.asanyarray(depth_frame.get_data())
```

Figure: Converting Images to Numpy Array

D. Visualization

Visualizing the information needs the algorithm to convert the numpy array of data points extracted from frames to RGBD image. This process is done multiple times to create multiple images which is then used to register and form point cloud. RGBD image is formed as following two methods depending upon nature of algorithm where tensor (t) is generally used for GPU processing and geometry for CPU processing:

```
if rgbd_video:  # Video file
    self.video = o3d.t.io.RGBDVideoReader.create(rgbd_video)
```

Or

```
rgbd_image = o3d.geometry.RGBDImage.create_from_color_and_depth(
    o3d.geometry.Image(color_image),
    o3d.geometry.Image(depth_image),
    convert_rgb_to_intensity=False
)
return rgbd_image
```

Figure: Creating RGBD Image

E. Point Cloud

To create point cloud, ICP registration algorithm has to be considered. Therefore, the selection of algorithm from point-to-point and point-to-plane has to be made, for this case the considered method uses point to plane. The RGBD image pair are registered on to each other in loop so that we can gradually form point cloud image as input information from camera is received.

```
def point_to_plane_icp(source, target, threshold, trans_init):
    print("Apply point-to-plane ICP")
    reg_p2l = o3d.pipelines.registration.registration_icp(source, target,
        threshold, trans_init,o3d.pipelines.registration.TransformationEstimationPointToPlane())
    print(reg_p2l)
    print("Transformation is:")
    print(reg_p2l.transformation, "\n")
    draw_registration_result(source, target, reg_p2l.transformation)
```

Figure: Point-To-Point ICP Registration

F. Voxel Down Sampling, Normal Estimation And Outlier Detection

The voxel down sampling is used to reduce the number of points in point cloud for effective usage of computing capacity as computing time is crucial for real time reconstruction [1].

The process of Normal estimation is necessary for plane segmentation and ball pivoting algorithm as the algorithm visualizes the ball of certain radius moving on point cloud for surface reconstruction.

Outlier detection is used to detect and remove points that can possibly deform the reconstruction, there are two types of outlier detection we have used statistical outlier detection it removes points that are away from their neighbors compared to the average point cloud.

```
print("Downsample the point cloud with a voxel of 0.01")
downpcd = pcd.voxel_down_sample(voxel_size=0.01)
o3d.visualization.draw_geometries([downpcd])
```

```
downpcd.estimate_normals(
    search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.1, max_nn=30))
```

```
cl, ind = voxel_down_pcd.remove_statistical_outlier(nb_neighbors=20,
                                                    std_ratio=2.0)
```

Figure: Voxel Down Sampling, Normal Estimation, Statistical Outlier

G. Alpha Surface reconstruction

Alpha reconstruction refers to a specific algorithm implemented in the library of depth map estimation for multiple images. It's aim to construct a dense depth map by fusing the depth information from multiple viewpoints [5].

```
o3d.visualization.draw_geometries([pcd])
alpha = 0.03
print(f"alpha={alpha:.3f}")
mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(pcd, alpha)
mesh.compute_vertex_normals()
o3d.visualization.draw_geometries([mesh], mesh_show_back_face=True)
```

Figure: Alpha Surface Reconstruction

H. Ball Pivoting Surface reconstruction

In ball pivoting reconstruction algorithm triangular mesh is created iteratively by pivoting a ball over point cloud where the ball is defined by seed point and radius. Initially nearest point is detected from ball's radius then check the neighbor for presence of triangle to form mesh.

```
mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(
    downpcd, o3d.utility.DoubleVector(radii))
```

Figure: Ball Pivoting Surface Reconstruction

I. Surface reconstruction Poission

Poisson surface reconstruction process creates the surface over low point density areas, which on real time functioning of algorithm can create deformed surface reconstruction thereby a threshold value have to be fixed so that it do not affect the shape and structures of objects in the scene. After constructing the surface density plotting is to be done so that low point density areas can be seen visually and can be removed to refine the reconstruction as follows:

```
print('run Poisson surface reconstruction')
with o3d.utility.VerbosityContextManager(
        o3d.utility.VerbosityLevel.Debug) as cm:
    mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(
        downpcd, depth=9)
print(mesh)
o3d.visualization.draw_geometries([mesh])
print('visualize densities')
densities = np.asarray(densities)
density_colors = plt.get_cmap('plasma')(
    (densities - densities.min()) / (densities.max() - densities.min()))
density_colors = density_colors[:, :3]
density_mesh = o3d.geometry.TriangleMesh()
density_mesh.vertices = mesh.vertices
density_mesh.triangles = mesh.triangles
density_mesh.triangle_normals = mesh.triangle_normals
density_mesh.vertex_colors = o3d.utility.Vector3dVector(density_colors)
o3d.visualization.draw_geometries([density_mesh])
print('remove low density vertices')
vertices_to_remove = densities < np.quantile(densities, 0.01)
mesh.remove_vertices_by_mask(vertices_to_remove)
print(mesh)
o3d.visualization.draw_geometries([mesh])
```

Figure: Poisson Surface Reconstruction, Density Map, Reduced Poisson Reconstruction

III. RESULT AND TESTING

Result of the 3D reconstruction algorithm in detail will require to visualize the multiple steps of process that have been executed in a synchronized manner. The shown process is performed on eagle-dataset present in Open3D module itself but the final reconstruction is directly showing the reconstructed 3D model as in real time execution the only visualization is of reconstruction to increase the computing efficiency of the system. The input information from the camera is displayed on Intel SDK which is a developer tool [3]:

A. Point Cloud formation

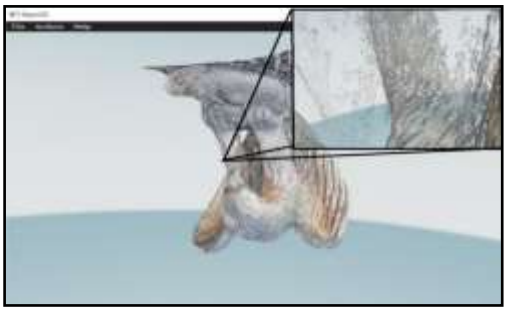

Figure: Point Cloud (Dataset)

B. Voxel Down Sampling

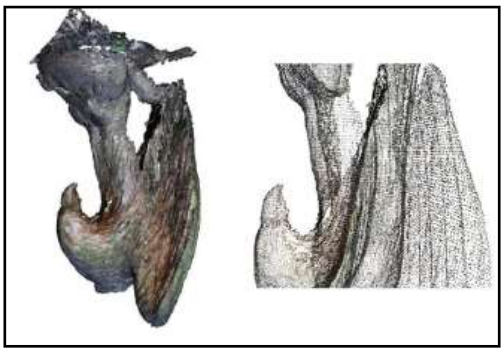

Figure: Voxel Down Sampling

C. Statastical Outlier Removal

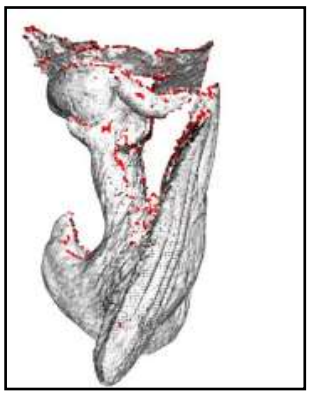

Figure: Statistical Outlier

D. Surface Reconstruction Alpha

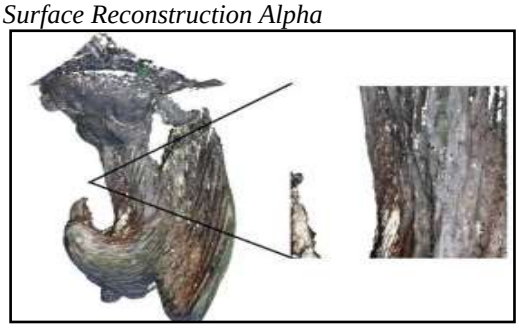

Figure: Alpha Surface Reconstruction

E. Surface Reconstruction Ball Pivoting

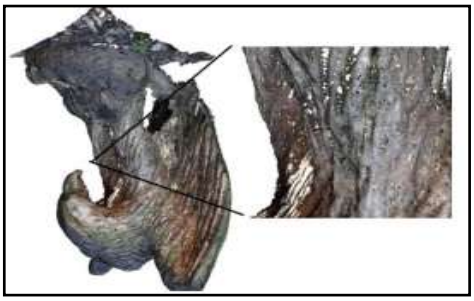

Figure: Ball Pivoting Surface Reconstruction

F. Surface Reconstruction Poission

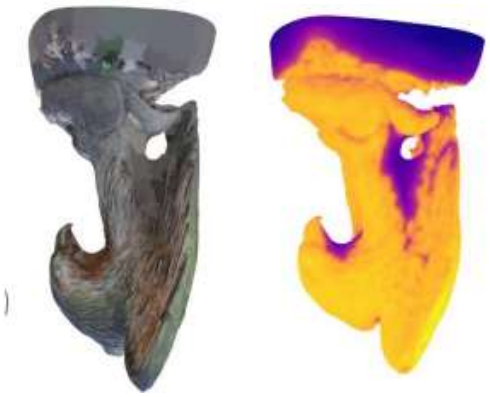

Figure: Density Map

G. Camera Output processed 3D Reconstruction

Figure: Human Reconstruction (Camera Input)

Figure: Human Reconstruction (Camera Input)

IV. SYSTEM ANALYSIS(EFFICENCY AND PROCESSING TIME)

The application of 3D reconstruction using RGBD camera are diverse and span over various fields, the cameras are employed for environment and surrounding monitoring tasks like navigation and object manipulation. The processes of surface creation have to be analyzed to formulate the efficient system, selecting surface reconstruction method is crucial as surface formation utilizes the most amount of time and computing power, there are total of three methods whose processing time is plotted in the figure below.

Alpha reconstruction system is less time consuming but the surface have missing patches of data points that are fail to form surface thereby using interpolation becomes necessity increasing the processing time further, Poisson Surface Reconstruction creates surface on low point areas that have to be controlled the thereby varying the processing time. Ball Pivoting consumes more time but provides accurate surface reconstruction [6], Therefore, the selection of algorithm is based many factors including hardware specification, operating system, available computing power, and application of the system.

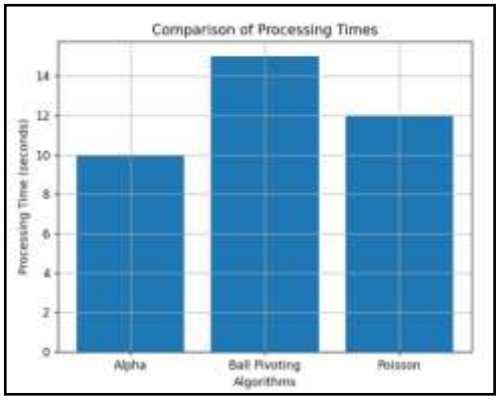

Figure: Comparison of surface reconstruction algorithms

Camera Limitation

The range of data that Intel realsense L515 get the data points from is from 0.25m to 9m and average of failed points ratio per image by distance in meters and standard ratio of outliers with error of +10cm or -10cm ratio per image by distance in meters are shown in the table:

TABLE I.

Sr. No.	Camera Details		
	Intel RealSense L515	***d***	***Std ratio of outliers***
1	0.0020%	0.3	0%
2	0.0001%	0.4	0.0034%
3	0.0023%	0.5	1.2715%
4	0.0001%	0.75	0%
5	0.0014%	1	0%
6	0.0017%	1.25	0.0025%
7	0.0320%	1.5	0.0008%
8	0.1648%	-----	0.0018%
9	0.8263%	-----	0.0027%
10	0.5456%	------	0.0001%

a. *(Average ratio of failed points and standard ratio of outliers from Intel RealSense L515 Camera)*

b.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Mr. Shishir .K Singh for their valuable support and guidance throughout the duration of the "3D Reconstruction Using RGBD Camera" project. Mr. S.K Singh's expertise in the field of computer vision and his continuous guidance have been instrumental in shaping the project's direction and methodology. His insightful feedback, constructive suggestions, and unwavering support have significantly contributed to the successful completion of the project. Lastly, I would like to show my gratitude to my parents for encouraging me and providing moral values and support.

REFERENCES

[1] Qian-Yi Zhou, Jaesik Park, Vladlen Koltun, "Open3D: A Modern Library for 3D Data Processing", Intel Labs, pp. 02-05

[2] Intel® RealSense™ LiDAR Camera L515 Datasheet

[3] Intel® SDK Applications developer Guide

[4] pyrealsense2 (2.33.1) documentation

[5] On 3D Reconstruction Using RGB-D Cameras Authors: Kyriaki A. Tychola *, Ioannis Tsimperidis and George A. Papakostas *

[6] Computer vision based 3D reconstruction : A review Hanry Ham, Julian Wesley, Hendra

Mr.Sagar Tyagi
MBA Tech NMIMS University (2020-2025 Batch)