The Chinese University of Hong Kong

Final Year Project Report

Semester 2

TANG XUN 1155046996

TANG XUN 1155046996

2018-4-18

Abstract

In the first semester, we have built a game with AR, which utilizes the point cloud to find a horizontal plane and on where we place the game terrain and play our tower defense game.

In this semester, for better immersion and interaction with the environment, we have make more use of the point cloud to reach the deeper understanding of the features of the environment. So currently we can use the real-world environment as our game terrain and play the game. Also we have implemented the navigation in real-world to avoid the obstacles so that the enemies look like walking in the real-world.

Background

Background

Virtual reality is becoming more and more popular, since hardware performance has been able to make virtual images often indistinguishable from the real world. However, those images presented in computer games, science fiction films, or other media are disconnected with our physical surroundings.

Smartphones and tablets provide a convenient way to gain huge size of information. But it is usually detached from the physical world. Consumers have to do a lot to connect such information with world around us. There ‘re many attractive ways that try to connect the mobile computation with real world, such as GPS or barcode. But their applicable fields are too few and cannot handle emergency.

Augmented reality (AR) is thought to be able to build a direct, automatic and actionable connection between the real world and virtual information. Or in other way, increase the amount of information, simplify and accelerate the information communication in real world.

Tango Project was a pioneer in AR area by Google and it also has approved its power as a smart phone with AR ability. So it is thought to be the right way to combine smart phones with AR.

And afterwards as the popularization of dual-camera phones with more and more powerful computational performance, the Apple ARKit comes, which adapts the AR ability to almost all iOS devices, when the google noticed that an extra depth camera is redundant and costly, so they have released the ARCore, which is significantly based

on the code of Tango Project and planned to apply to all the Android devices. Therefore, though the Tango Project is deprecated and shut down by Google, the early effort is not wasted, and of course, neither our study in AR. Our project based on Tango is possible to be transplanted to ARCore.

Review of Last Semester

In last semester we have made a lot of efforts on research on Tango AR and implemented an AR Tower Defense game with Tango and Unity, where the player build towers to prevent the enemies from reaching the terminal. In that game, we have utilized the AR functions in finding a plane to place our game scene and in keeping motion tracking and area learning to make the scene bound with the real world. We have also utilized many Unity tools like Animator Controller, Line Renderer and Particle System to build the game with completed game mechanics and attractive special effects.

- Research on Tango

We have deeply studied the Tango technics in last semester. It is achieved by using three core technologies: Motion Tracking, Area Learning, and Depth Perception.

1. Motion Tracking

Motion Tracking in Tango is achieved through visual-inertial odometry. It uses camera images and inertial motion sensors to estimate both its position and rotation in 3D world more accurately.

Pose

A device's pose means the combination its position and orientation of the user's device in full six degrees of freedom. There're two methods in Tango API to get pose data: one is using callbacks to get the most recent pose updates, and another is using functions to get a pose estimate at a specific time. The data consists of two main parts: a vector in meters for translation and a quaternion for rotation.

Common use cases

Improved rotation sensing: It can replace Android Game Rotation Vector APIs with better performance.

Tracking movement: Tango allows you to track a device's movement in the real world.

Virtual camera: When you combine rotation and position tracking, you can use the device as a virtual camera in a 3D rendered environment.

Limitations

Through motion tracking, device does not truly understand the real world around.

Every time you start a new Motion Tracking session, it will forget the previous information and cannot tell you where you are now.

Over long distances and periods of time. Small errors will be accumulated to a large amount.

2. Area Learning

With Motion Tracking alone, device cannot keep memory of things it sees. But Area Learning ensures that it is able to “remember”. To do this, it generates a mathematical description of the edges, corners, other unique visual features. To improve Motion Tracking, it improves the accuracy of the trajectory by performing "drift corrections" and orients and locates itself within a previously learned area by performing "localization".

With Area Learning, the device is able to use the visual features in its memory to adjust its estimation on movement. This memory allows the system to perform drift corrections. When the device recognizes a visual feature it stored before, it knows a

loop is finished and adjusts its trajectory to be more consistent with previous information.

Also, the Area Description File can be stored and transmitted to other devices, so this allows multiple users to interact with each other in the same physical room.

3. Depth Perception

Depth Perception provides a device the ability to estimate the distance to objects. The way of implementation is depended on manufacturers. They can choose among several depth technologies, including Structured Light, Time of Flight, and Stereo.

➢ Game Elements

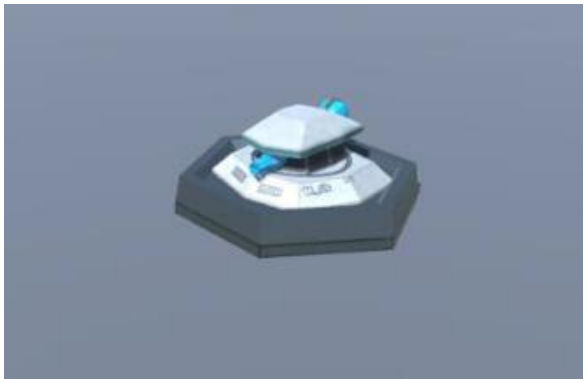
■ Map Generator

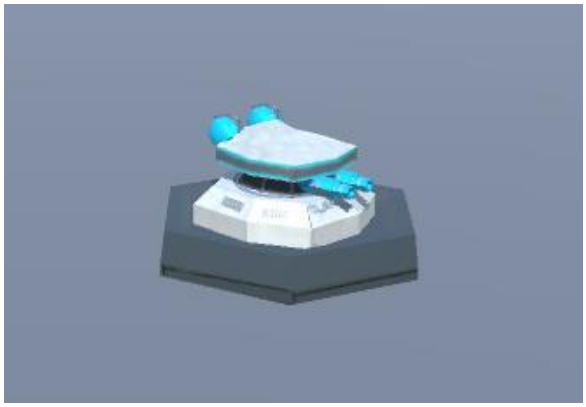
The map is generated above a horizontal plane in real world. It is randomized to make sure it will get different challenges each time. The map size is about 1 meter * 1 meter with no more than 21 * 21 blocks of floors or obstacles. It is reasonable to have some rooms and some narrow paths. We adopt a Depth-First-Search(DFS) algorithm to generate the map with fixed start and end point and no dead ends.

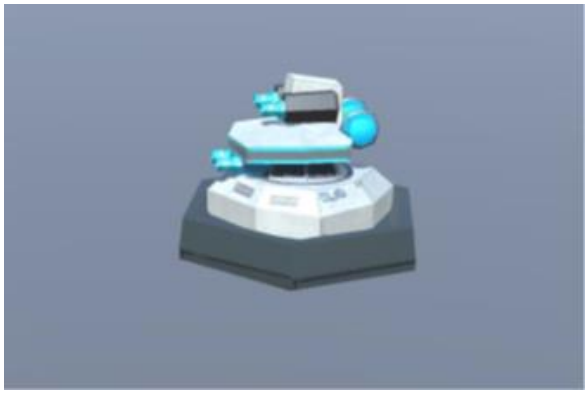
■ Towers

There are 4 types of towers, each type has 3 levels:

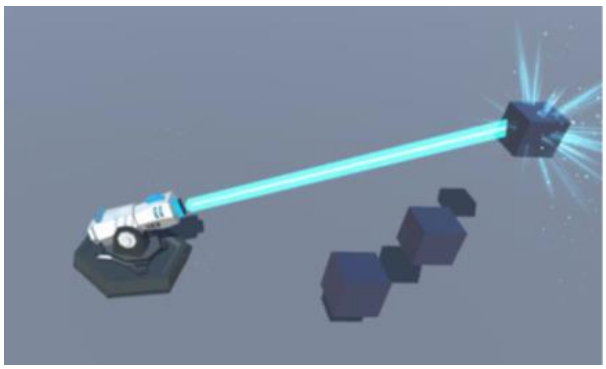
- Basic Towers can rotate smoothly along Y-axis to aim an enemy and shoot an explosive shell to attack. The higher level one has higher-damage shells.

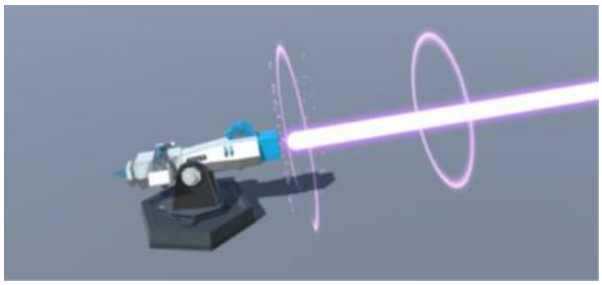




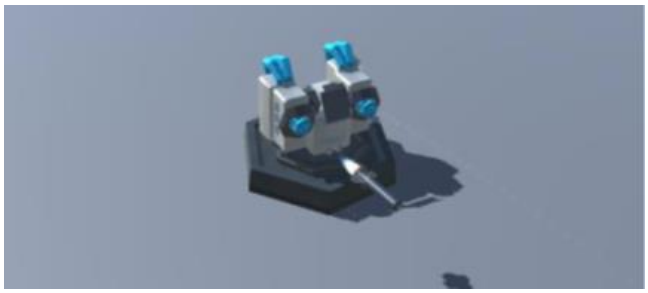


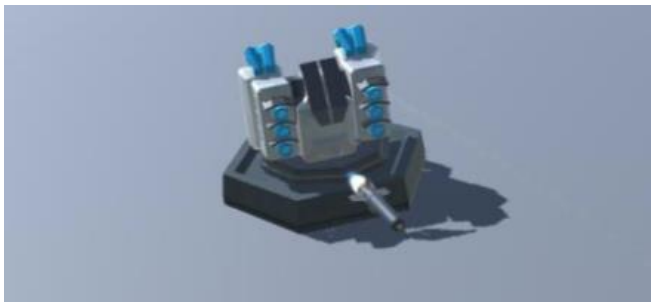
- Laser Towers can rotate smoothly along Y-axis and X-axis to aim an enemy and emit laser beam to attack. The higher level one has more powerful beam.

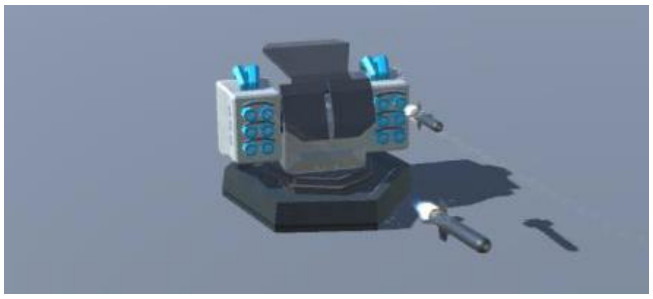




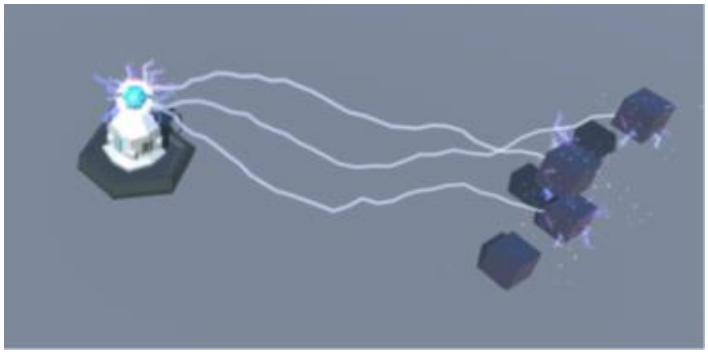
- Rocket Towers can rotate smoothly along Y-axis and X-axis to aim an enemy and shoot rockets (guided missiles) to attack. The higher level one has more missiles and shorter shoot interval and a bit longer filling time.

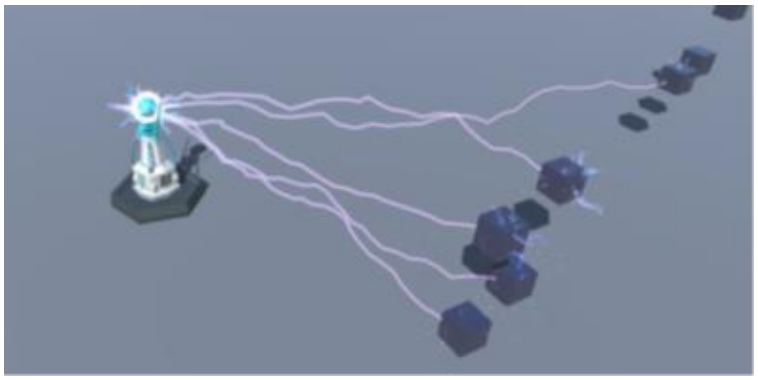


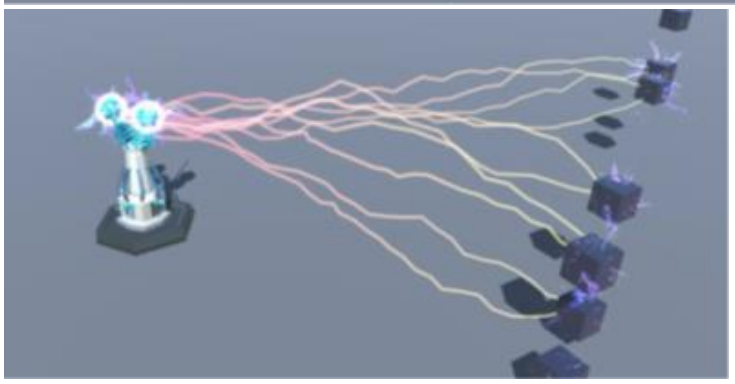




- Tesla Towers can emit electrical arcs to attack multiple enemies in the range.



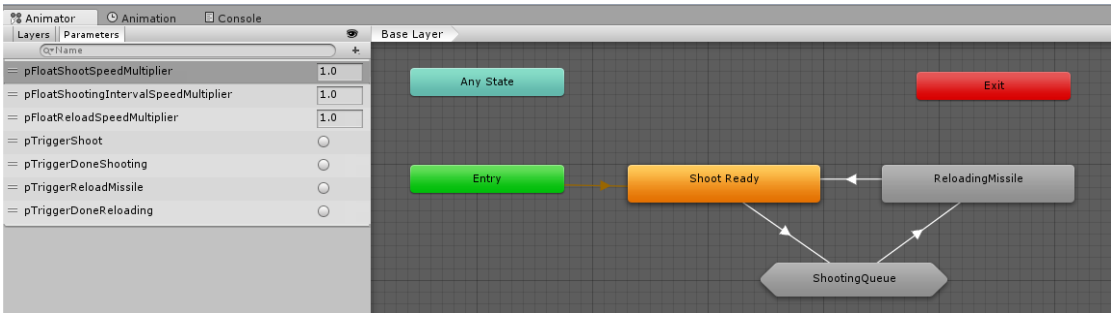




- Animation & Animator Controller

We use the Animator Controller to control the animation states of the towers. For example, when the Rocket Tower launches a missile it will be affected by the

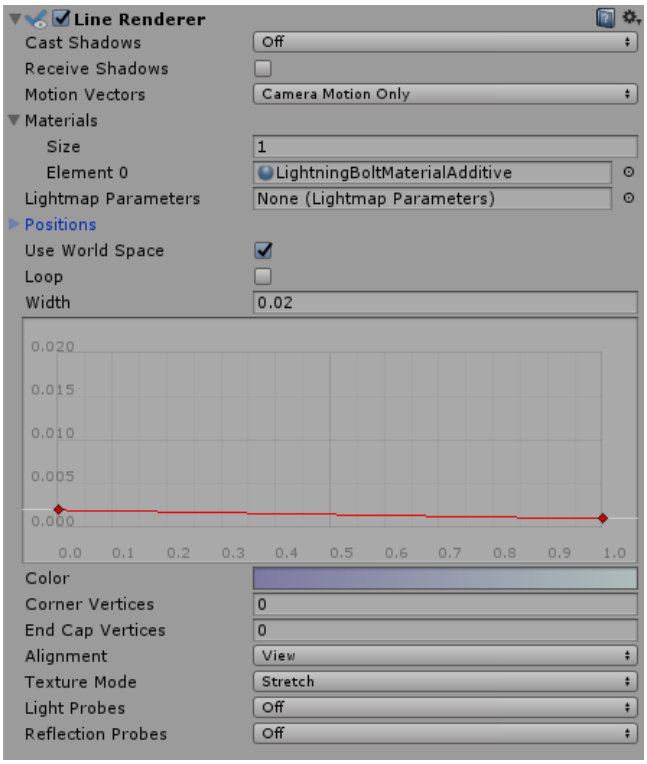
recoil. With the help of Animation Event we can create the missile object at the right time of the animation of the recoil effect.



We use script to control the rotation of the towers to make it smooth and reasonable.

■ Line Renderer Special Effect

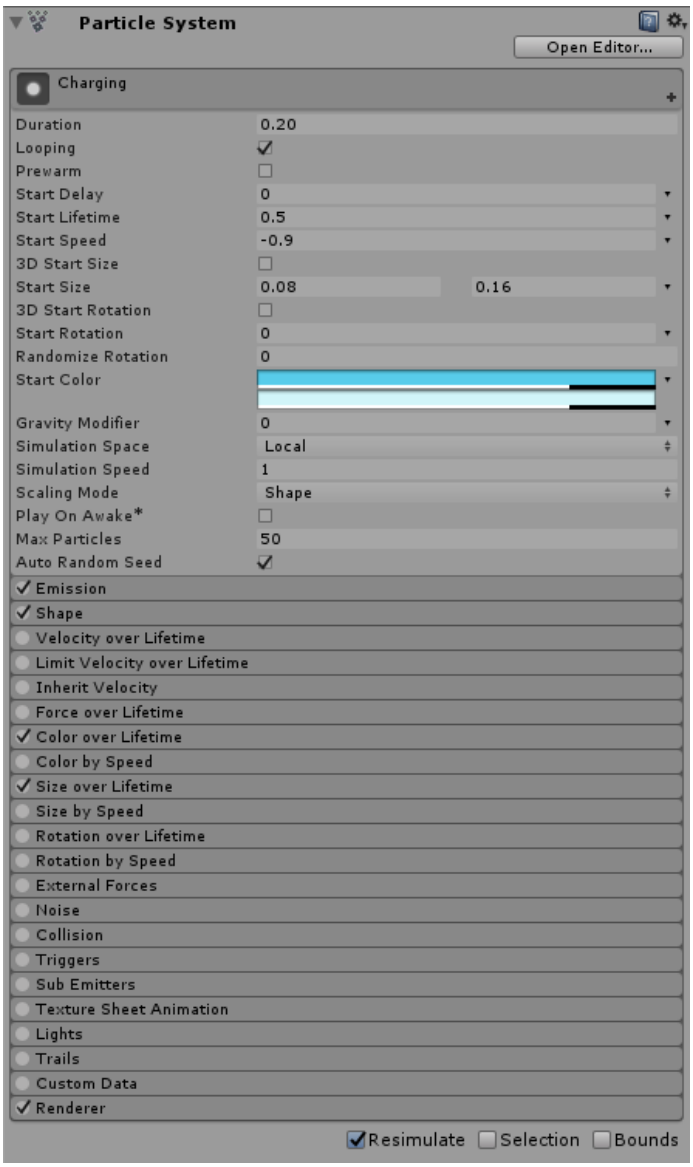
We use the LineRenderer to simulate the electrical arcs. This is a powerful tool to draw polylines in the scene. We have used program to randomly and reasonably update the points of the polyline every several frames to make it behave like a lightning.



■ Particle System Special Effect

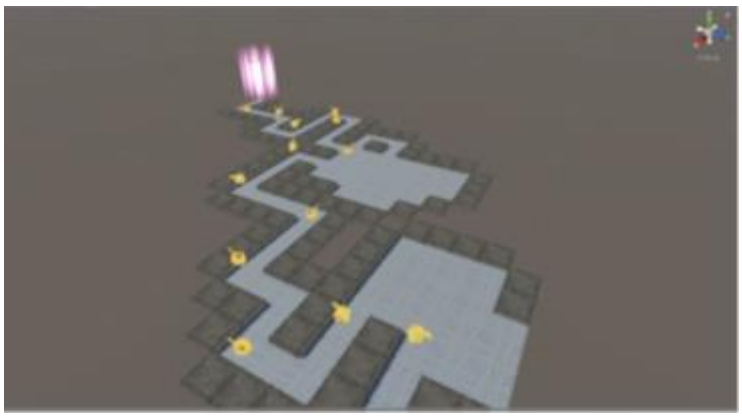
A lot of effects are created with Particle System, e.g. the explosion, the electro spark and the rings representing the blast waves of beam.

We have combined several types of the particle systems to make the effects more attractive and realistic.



■ Enemy Generator

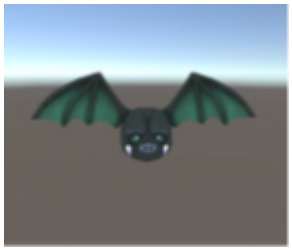
The enemy generator generates different types of enemy waves. We also make the generator more completed in having ability to make a single wave carrying different enemies of certain numbers to improve the difficulty of the defense.



- Enemies

The enemy models are from Unity Assets Store. We have 3 types of enemies and 2 of them are land enemies walking along the routes, 1 of them is air enemies flying over the map.





- Real-time Enemy Navigation

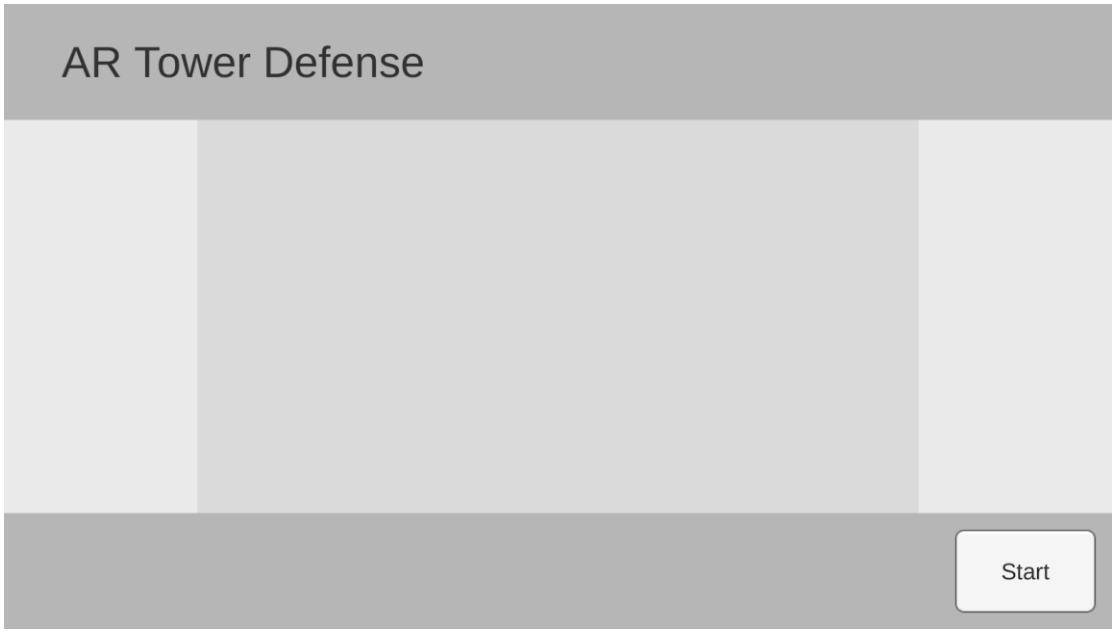
The tower can be placed on the path to block the route, so that we allow player to lead the enemies to take long detours. But The built-in navigation system cannot tolerate dynamic obstacle changes in the map. Therefore, we implement our own algorithm for the navigation system.

The most commonly used algorithm for navigation is A*, but it is too slow in our game since we require multiple enemies to react to the variation in a possibly large map very quickly. So we implement the D* Lite algorithm, which is faster and more suitable for dynamic environment.

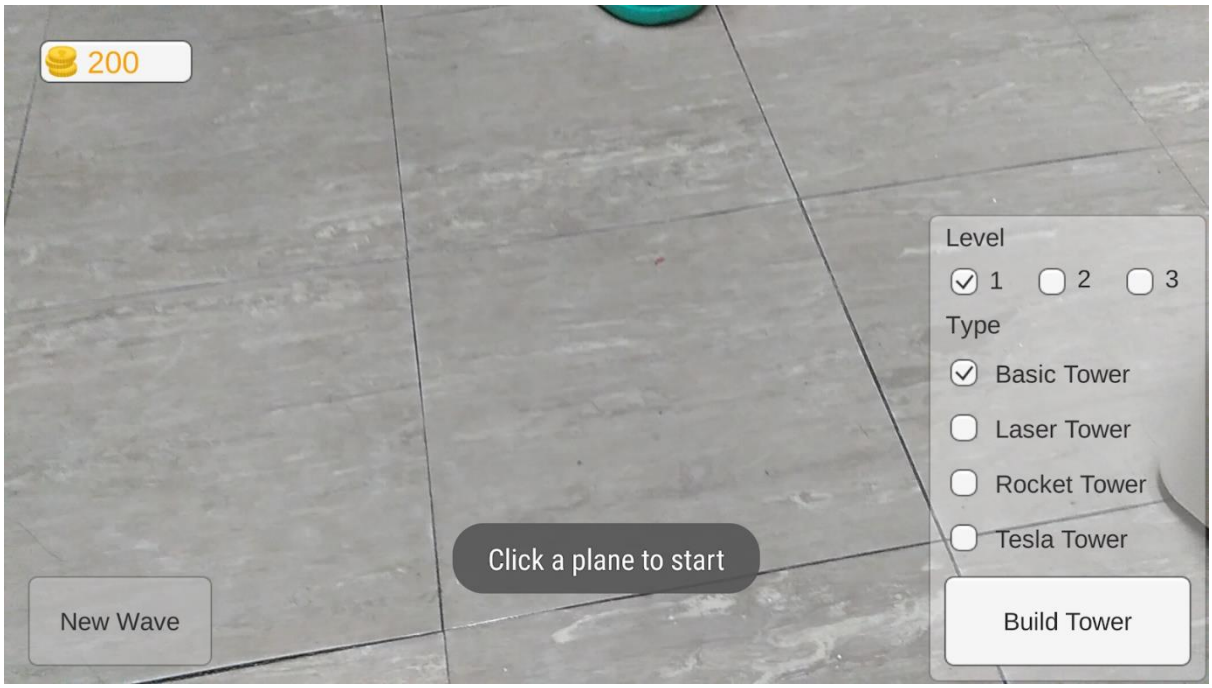
- UI

We have used the “Canvas” in Unity to build the UI overlays.

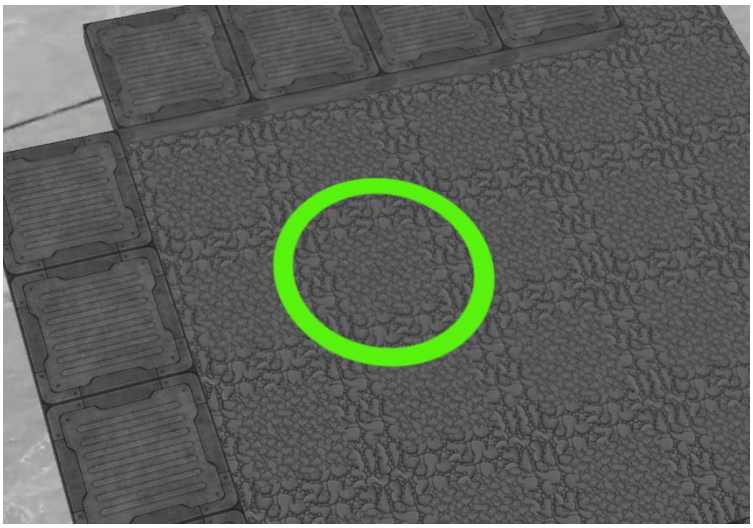
- The Initial Interface



- The Play Interface



- The circle to highlight the selected tile



Previous Research

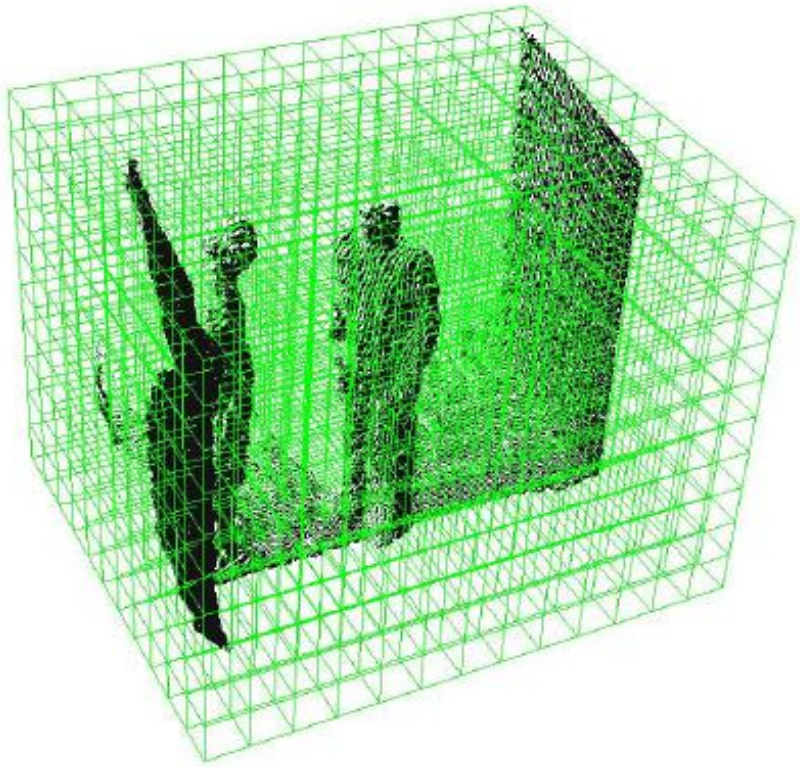
In this term, what we want to do is using Tango Device to actually interact with real world. The key point is the processing of point cloud and image information, so we did a lot of research in this topic.

1. Empty Space

   Point clouds contain a lot of potential information about scanned objects and surfaces, as well as the space information that can used to move through. We can use empty space to effectively help applications like 3D navigation, because it focuses on available and usable space instead of boundary points or objects. Olivier Rodenberg et al. (2016) used empty space to do indoor navigation. There’re several options to store and represent the empty space, and below are two most important and common methods.

   - Voxel

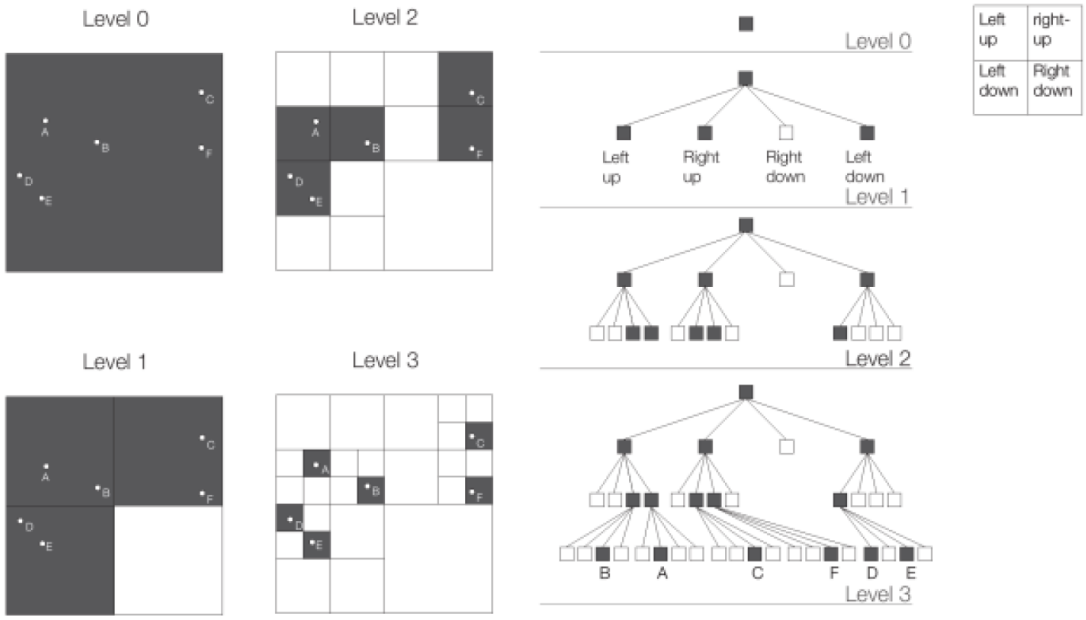
   The voxel approach takes the bounding box of the point cloud and voxelizes the entire space. The points in the point cloud are then intersected with the voxels to find the points inside each voxel. The voxels which do not contain any points after the procedure make up the empty space and thus get this as an attribute. A similar approach was used by Bienert et al. (2010) or by Nourian and Zlatanova (2015) to voxelize a point cloud.



■ Octree

A linear octree was used to derive and structure the empty space in the point cloud. An octree recursively subdivides the point cloud into eight equal-sized octants, up to a predefined maximum resolution. Octants are called black (containing points) o5r white (empty) leaf nodes if they do not contain smaller octants. A locational code is generated for every leaf node using bitwise interleaving: a method that combines the x, y, and z coordinates into a single binary string. The entire octree structure is implicitly stored in the resulting set of locational codes.

Black leaf nodes always have the maximum specified resolution, thus the octree always reaches maximum resolution around points in the point cloud. The white leaf nodes are not further subdivided for efficient storage of empty space.



Quad tree in 2D space

2. Normal Estimation / Plane Detection

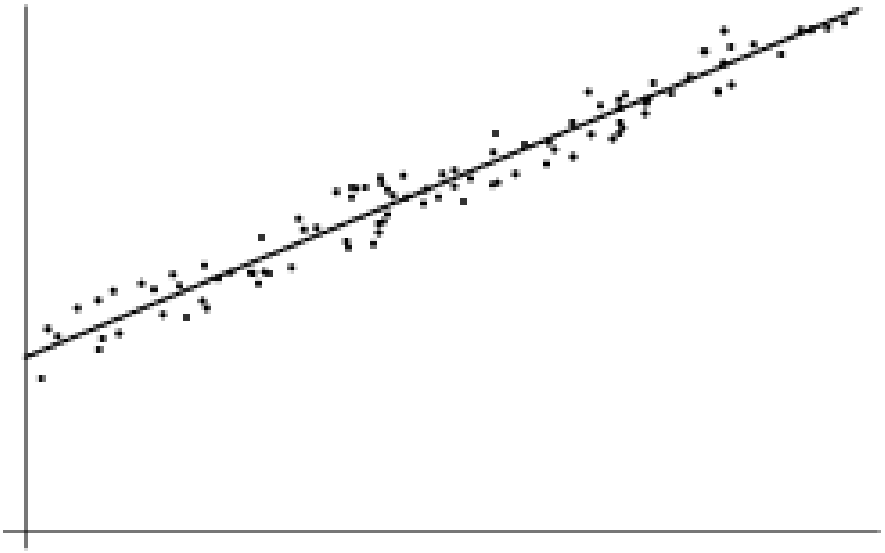
The aim of normal estimation is to estimation the normal vector of each point or a set of points in point cloud. When applied to a set of points, it is same as plane detection where the normal vector of points is same as the normal vector of plane to extract. This is important since we can get the shape information, and thus do more meaningful jobs.

There’re many ways to do normal estimation.

- Least Squares

Least square fitting is used to find the best curve for point set through minimizing the sum of the squares of the offset from point from the curve. The sum of the

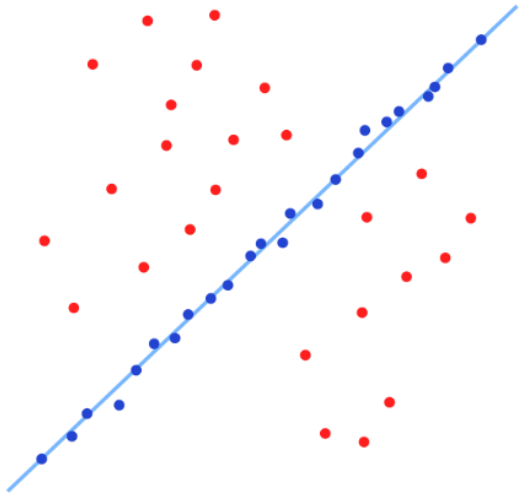
squares of the offsets is used instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. However, because squares of the offsets are used, outlying points can have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand.



Finding a plane in 3D space is similar as finding a line in 2D space.

■ RANSAC

RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data. Given a dataset whose data elements contain both inliers and outliers, RANSAC uses the voting scheme to find the optimal fitting result. Data elements in the dataset are used to vote for one or multiple models. The implementation of this voting scheme is based on two assumptions: that the noisy features will not vote consistently for any single model (few outliers) and there are enough features to agree on a good model (few missing data).



Fitted line with RANSAC, outliers have no influence on the result.

When applied to normal estimation or plane detection, RANSAC can be concluded as the following steps:

Algorithm 1 RANSAC for plane detection

bestSupport = 0; *bestPlane*(3,1) = [0, 0, 0]
bestStd = ∞; $i = 0$
ϵ = 1 - *forseeable-support*/length(*point-list*)
N = round($log(1-\alpha)/log(1-(1-\epsilon)^3)$)
while $i \leq N$ **do**
  j = pick 3 points randomly among (*point-list*)
  pl = pts2plane(j)
  dis = dist2plane(pl, *point-list*)
  s = find($abs(dis) \leq t$)
  st = Standard-deviation(s)
  if (length(s) > *bestSupport*) or (length(s) = *bestSupport* and $st <$ *bestStd*) **then**
    bestSupport = length(s)
    bestPlane = pl; *bestStd* = st
  end if
  $i = i + 1$
end while

3. Segmentation

Segmentation is the process of grouping point clouds into multiple homogeneous regions with similar properties.

- Edge-based

As described by Rabbani et al. (2006), edge-based segmentation algorithms have two main stages: (i) edge detection to outlines the borders of different regions and (ii) grouping of points inside the boundaries to deliver the final segments. Edges in a given depth map are defined by the points where changes in the local surface properties exceed a given threshold. The mostly used local surface properties are normals, gradients, principal curvatures or higher order derivatives. Methods based on edge- based segmentation techniques are reported by Bhanu et al.(1986), Sappa and Devy (2001), Wani and Arabnia (2003). Although such methods allow a fast segmentation, they may produce not accurate results in case of noise and uneven density of point clouds, situations that commonly occur in point cloud data. In 3D space, such methods often detect disconnected edges making the identification of closed segments difficult without a filling or interpretation procedure (Castillo et al., 2013).

- Region Growing

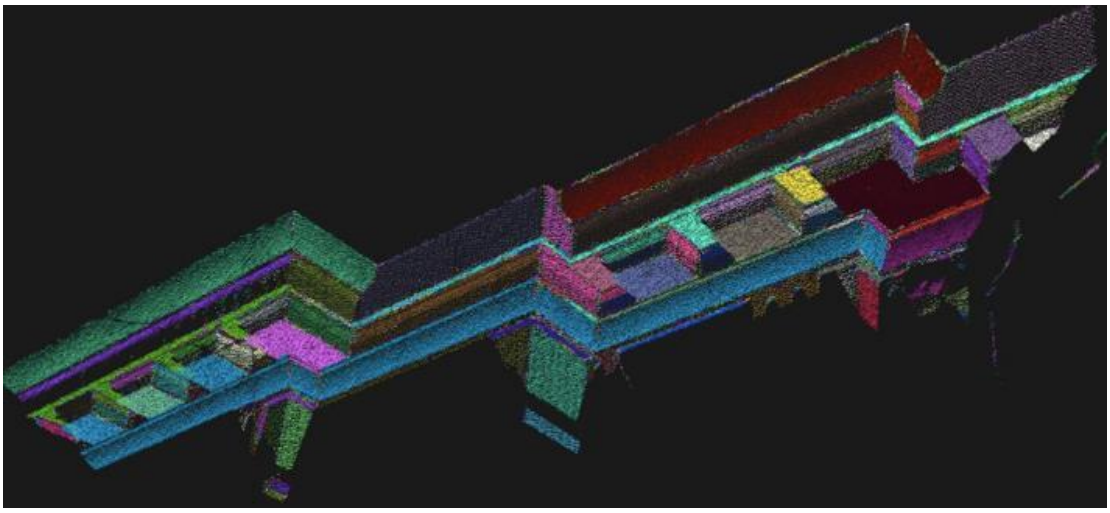
These methods start from one or more points (seed points) featuring specific characteristics and then grow around neighbouring points with similar

characteristics, such as surface orientation, curvature, etc. (Rabbani et al., 2006; Jagannathan and Miller, 2007). Region-based methods can be divided into:

Bottom-up approaches: they start from some seed points and grow the segments on the basis of given similarity criteria. Seeded region approaches are highly dependent on selected seed points. Inaccurate selection of seed points will affect the segmentation process and can cause under- or oversegmentation results.

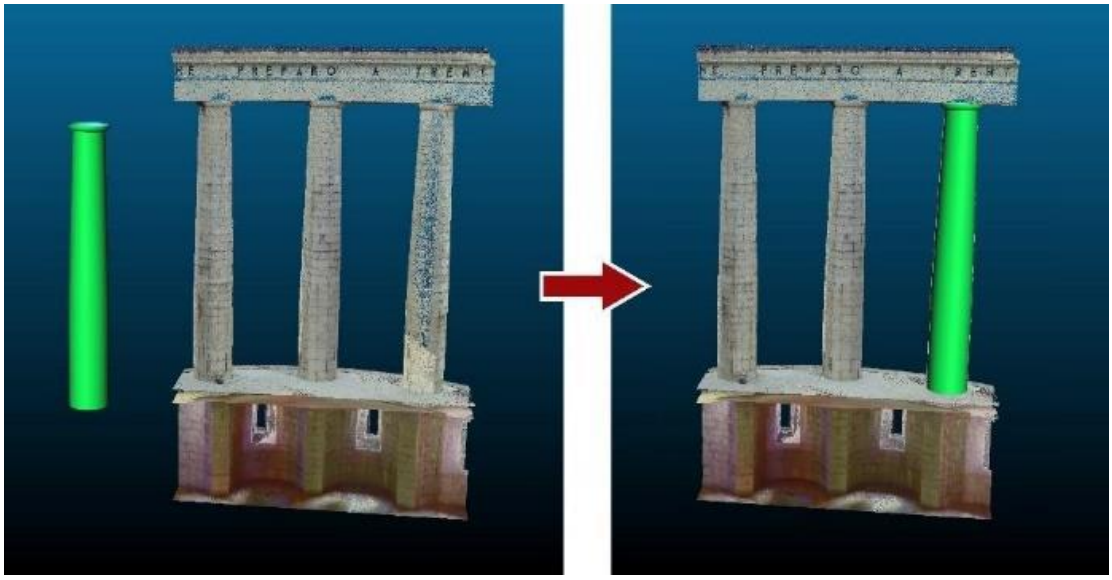
Top-down approaches: they start by assigning all points to one group and then fit a single surface to it. Where and how to subdivide unseeded-region remain the main difficulty of these methods.

Region-based algorithms includes two steps: identification of the seed points based on the curvature of each point and growing them based on predefined criteria such as proximity of points and planarity of surfaces. The initial algorithm was introduced by Besl et al. (1988) and then several variations were presented in the literature. The region growing method proposed by Vosselman et al. (2004) has introduced the use of color properties beside geometrical criteria. Surface normal and curvatures constraints were widely used to find the smoothly connected areas (Klasing et al., 2009; Belton and Lichti, 2006) whereas Xiao et al. (2013) proposed to use sub window as the growth unit. Ackermann and Troisi (2010) used a region growing approach to segment planar pitched roofs in 3D point clouds for automatic 3D modelling of buildings. Anh-Vu Vo et al. (2015) presented an octree-based region growing approach for a fast surface patch segmentation of urban environment 3D point clouds.



- Segmentation by model fitting

This approach is based on the observation that many man-made objects can be decomposed into geometric primitives like planes, cylinders and spheres (Fig. 3). Therefore, primitive shapes are fitted onto point cloud data and the points that conform to the mathematical representation of the primitive shape are labelled as one segment. As part of the model fitting-based category, two widely employed algorithms are the Hough Transform (HT) (Ballard, 1981) and the Random Sample Consensus (RANSAC) approach (Fischer and Bolles, 1981).



Algorithm of Demo

The demo we made is an application with real-time normal estimation and navigation. To achieve this, we need a fast and robust algorithm to process point clouds we get.

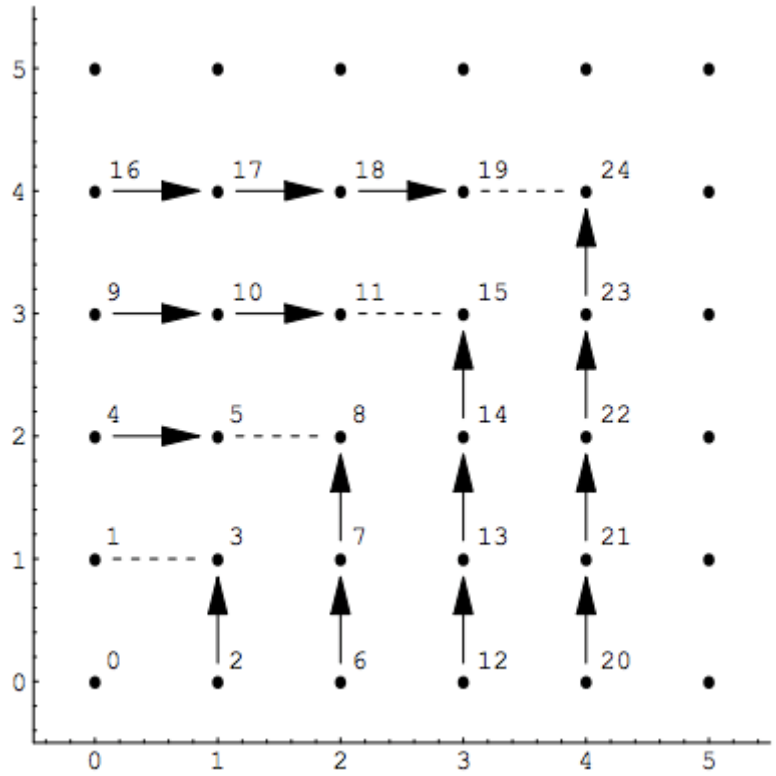
1. Voxelize

   First, we will set start point as the origin. Then for each point in point cloud, we will calculate the voxel it belongs to, and store the information of point in the voxel structure. We will store all non-empty voxels in hashset.

2. Hash

   We needed to get a deterministic number as hash value from three ordered numbers x,y,z for the hashset mentioned above. We adopt the method proposed by David Mauro(2014).

   There’s a pairing function by Matthew Szudzik that assigns numbers along the edges of a square instead of the traditional Cantor method of assigning diagonally.



When applied to 3D space, the formula should be:

```
function(x, y, z) {
  max = MAX(x, y, z)
  hash = max^3 + (2 * max * z) + z
  if (max == z)
    hash += MAX(x, y)^2
  if (y >= x)
    hash += x + y
  else
    hash += y
  return hash
}
```

if we want to be able to use negative coordinates, we can simply add this to the top of our function:

x = if x >= 0 then 2 * x else -2 * x - 1

y = if y >= 0 then 2 * y else -2 * y - 1

z = if z >= 0 then 2 * z else -2 * z – 1

3. Calculate Normal Vector and Plane Fitted for Each Voxel

We want to achieve this goal in real time instead postprocessing, after looking up a lot of paper, we come up with a method based on Alexandre Boulch and Renaud Marlet's work. (2012)

Before describing our algorithm, let us consider the following simple situation. Let P be a point on a piecewise planar surface and let N_P be a neighborhood of P on this surface (a subsurface). There are two basic cases:

• If P lies far from any edge or sharp feature, then picking three points in N_P defines the planar patch that P lies on, and thus the normal (if the points are not collinear).

• If P lies near an edge partitioning the neighborhood N_P into $N_{1,P} \cup N_{2,P}$ with $P \in N_{1,P}$, then picking three points in N_P does not necessarily determine the right normal. It defines either the correct normal (if all points lie in $N_{1,P}$), or the normal associated with the plane on the opposite side of the edge (if all points lie in $N_{2,P}$), or a "random" plane (if the points are not on the same side of the edge).

In the second case, as $N_{1,P}$ is likely to be larger than $N_{2,P}$ since $P \in N_{1,P}$ is not exactly on the edge, the probability of picking the dominant plane and thus the correct normal is higher than the probability of picking the normal on the opposite edge side. When P is very close to the edge, drawn triples are likely to lie on both sides of the edge. However, as it leads to a "random" normal, the correct normal

still is the one with the highest probability. This generalizes to situations where P is close to several edges, including coincident edges. This idea applies as well to a point cloud C in which neighboring points N_P are defined for any point $P \in C$. Our method is a robust variant of this simple principle, to handle noise and outliers. For this, we sample as many planes as necessary to gain enough confidence that we can identify the actual maximum of the probability density. In practice, we discretize the problem and fill a Hough accumulator until a normal can be confidently chosen based on the most voted bin, with some refinements.

The basic algorithm can be expressed through the following steps:

Given voxel set V, number of planes to pick T

for each voxel v in V:

    Reset(accumulator)

    P is the points in v

    i = 0

    while(i < T):

        p1, p2, p3 = RandomPick3Points(P)

        normal = GetNormalVector(p1,p2,p3)

        Accumulate (normal, accumulator)

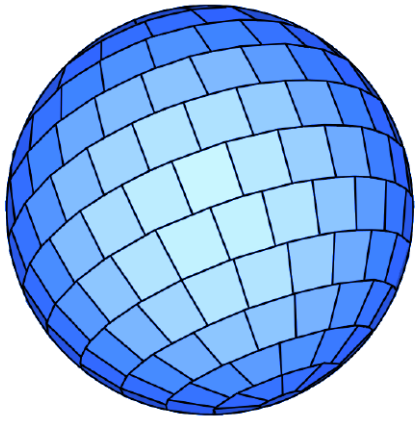
    bestBin = GetBinWithMostVotes(accumulator)

    normals[v] = GetAverageNormalInBin(bestBin)

There’re several things to be noticed:

- Accumulator shape

As we only estimate the normal direction, not the orientation, picking a triple of points defines a normal described by two angles (q,f), modulo p. It votes into an accumulator that partitions half the unit sphere into similar bins. We use the spherical accumulator of Borrmann et al. (2011) that provides bins with similar area and allows easy and fast computation of bin indexes, given the normal angles. This accumulator first divides the sphere according to the parallels, with same angle size. Then the slice (between two parallels) are divided into bins of nearly the same area. We have chosen this accumulator after also testing a geodesic sphere accumulator, that is more isotropic but considerably slower for a limited precision improvement.



n_ϕ	M
5	23
10	82
15	171
20	290
25	441

Accumulator shape & size

- Discretization Issues

Using a discrete accumulator leads to some issues:

First, there is a binning effect. If the main peak of the distribution of normals lies near a bin boundary, votes will be almost equally distributed between two (or more) adjacent bins. After a limited number of plane pickings, the actual peak may not be in the most voted bin. The effect is stronger for noisy data as the distribution is flatter and votes are distributed into more bins, that receive less votes on average.

Second, bins are not isotropic. Although the above accumulator guarantees the nearly equal area for bins, their shape is different. For instance, polar bins are disks (caps) whereas equatorial bins are squares. A classic solution to bin discretization would be for a normal to share parts of its vote in neighboring bins, but it does not answer the second problem. Both issues can be addressed by randomly rotating the accumulator and running the algorithm several times. The more rotations, the more precise the estimation. From the implementation point of view, it is more efficient to rotate the points, rather than rotating the accumulator itself.

- Complexity

  O(k) for each voxel, where k is the sampling size and k << n, n is the number of points in each voxel. So it's very fast and can be run in real time.

  Compared to RANSAC for same purpose: its complexity is O(kn)

4. Navigation

   For each plane that's walkable (normal vector is nearly vertical) extracted from each voxel, we take them as a node, and we use A* algorithm to do pathfinding. Still, there're some things to be considered: collision avoidance and enough area to walk.

Application

1. Indoor Navigation

As the indoor location is quite inaccurate, we can use the stored area description matching to recognize the accurate location of the user. And plus we have the navigation across different floors in a quite large scale, so we can make a reliable indoor navigation system.

For example, in a large mall or plaza, the owner can create his own navigation app based on our application. Firstly, walk through the whole mall to remember the entire area description, and Secondly mark the location of different shops and other facilities like the counters, the toilets and the exits. Then the app can firstly use the partial area description nearby to recognize where the device is, and secondly search the route to the desired destination for the user, and finally show the route to lead the user get to his destination.

2. Activity

We can combine other games with our AR app.

For example, some special venue like the Buzz Lightyear Astro Blasters in Hong Kong Disneyland can create its own AR game based on our system. Firstly, as usual, scan the whole area to remember the entire area description. Secondly make the enemy appearances in some locations. Thirdly implement a First-Person-Shooting game with

UI. So that the player can play the game through their phone.

For another example, in the Orientation Day of CUHK, the host can create an app based on our app to implement different immersive AR games that help the player understand the faculty and the programmes in the corresponding booths.

3. Exhibition or Museum

In an exhibition or a museum, our app can be implemented to be a powerful navigation as well. And the more important is, the host can prepare more active 3D animations with the exhibits.

For example, besides an ancient china bowl, the app recognizes the area and then shows the corresponding 3D animation about how the people discovered this bowl and how the ancient people create this bowl for better understanding. Also, besides a famous Chinese calligraphy, the app recognizes where the device is and which exhibit is nearby, then the app shows the image the poem describes and the scene where and how the poet wrote it, the historical background of it.

Rodenberg, O. B. P. M., Verbree, E., & Zlatanova, S. (2016). Indoor A* pathfinding through an octree representation of a point cloud. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*, 249.

Broersen T, Fichtner F, De Liefde I. Pathfinding through Identified Empty Space in Point Clouds[J]. GIM INTERNATIONAL-THE WORLDWIDE MAGAZINE FOR GEOMATICS, 2016, 30(4): 21-+.

Boulch A, Marlet R. Fast and robust normal estimation for point clouds with sharp features[C]//Computer graphics forum. Blackwell Publishing Ltd, 2012, 31(5): 1765-1774.

Borrmann D, Elseberg J, Lingemann K, et al. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design[J]. 3D Research, 2011, 2(2): 3.

Vo A V, Truong-Hong L, Laefer D F, et al. Octree-based region growing for point cloud segmentation[J]. ISPRS Journal of Photogrammetry and Remote Sensing, 2015, 104: 88-100.

David Mauro, 2014, A Hashing Function for X, Y, Z Coordinates. http://dmauro.com/post/77011214305/a-hashing-function-for-x-y-z-coordinates

Grilli E, Menna F, Remondino F. A review of point clouds segmentation and classification algorithms[J]. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci, 2017, 42(2): W3.

Rabbani, T., Van Den Heuvel, F., & Vosselmann, G., 2006. Segmentation of point clouds using smoothness constraint. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 36(5), pp. 248-253.

Bhanu, B., Lee, S., Ho, C. C. and Henderson, T., 1986. Range data processing: representation of surfaces by edges. Proc. 8th International Conference on Pattern Recognition, pp. 236-238.

Sappa, A. D., & Devy, M. , 2001. Fast range image segmentation by an edge detection strategy. Proc. IEEE 3rd 3-D Digital Imaging and Modeling, pp. 292-299

Wani, M. A., & Arabnia, H. R. , 2003. Parallel edge-region-based segmentation algorithm targeted at reconfigurable multiring network. The Journal of Supercomputing, Vol. 25(1), pp. 43-62.

Castillo, E., Liang, J., & Zhao, H., 2013. Point cloud segmentation and denoising via constrained nonlinear least squares normal estimates. In Innovations for Shape Analysis (pp. 283-299). Springer Berlin Heidelberg.

Wikipedia, Random sample consensus, 2018. https://en.wikipedia.org/wiki/Random_sample_consensus

Jagannathan, A. and Miller, E. L., 2007. Three-dimensional surface mesh segmentation using curvedness-based region growing approach. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 29(12), pp. 2195-2204.

Besl P.J., Jain R.C., 1988. Segmentation through variable order surface fitting. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 10.

Sithole, G. and Vosselman, G., 2004. Experimental comparison of filter algorithms for bare-Earth extraction from airborne laser scanning point clouds. ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 59(1), pp.85-101.

Klasing, K., Althoff, D., Wollherr, D. and Buss, M., 2009. Comparison of surface normal estimation methods for range sensing applications. Proc. IEEE International Conference on Robotics and Automation, pp. 3206–3211.

Belton, D. and Lichti, D. D., 2006. Classification and segmentation of terrestrial laser scanner point clouds using local variance information. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 36(5), pp. 44-49.

Ackermann S, Troisi S, 2010. Una procedura di modellazione automatica degli edifici con dati LIDAR. Bollettino SIFET, Vol. 2, pp. 9-25.

Xiao, J., Zhang, J., Adler, B., Zhang, H. and Zhang, J., 2013. Three-dimensional point cloud plane segmentation in both structured and unstructured environments. Robotics and Autonomous Systems 61, pp. 1641-1652.

Ballard, D. H., 1991. Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognition, Vol. 13(2), pp. 183-194.

Fischler, M. A., & Bolles, R. C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, Vol. 24(6), pp. 381-395.

Chen, D., Zhang, L., Mathiopoulos, P. T., & Huang, X., 2014. A methodology for automated segmentation and reconstruction of urban 3-D buildings from ALS point clouds. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 7(10), pp. 4199-4217.