Surface Reconstruction of 3D Point Cloud

Bachelor's Term Project-II report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Electronics and Electrical Communication Engineering

By

Soumyadeep Ghosh

(20EC39035)

Under the supervision of

Professor Ritwik Kumar Layek

Department of Electronics and Electrical Communication Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

April 29, 2024

DECLARATION

I certify that

a) The work contained in this report has been done by me under the guidance of my supervisor.
b) The work has not been submitted to any other Institute for any degree or diploma.
c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April 29, 2024 Soumyadeep Ghosh (20EC39035)

Place: Kharagpur

DEPARTMENT OF ELECTRONICS AND ELECTRICAL
COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled “Surface Reconstruction of 3D Point Cloud” submitted by Soumyadeep Ghosh (Roll No. 20EC39035) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Electronics and Electrical Communication Engineering is a record of bonafide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Date: April 29, 2024
Place: Kharagpur

Professor Ritwik Kumar Layek,
Department of Electronics and Electrical
Communication Engineering,
Indian Institute of Technology Kharagpur,
Kharagpur - 721302, India

Contents:

Introduction:

Stereo 3D reconstruction and surface reconstructions are computer vision techniques that involve creating a 3D representation of a scene or object from a pair of 2D images taken from slightly different viewpoints. We may complete the task of 3D reconstruction from a pair of stereo images without prior knowledge of the internal camera parameters (intrinsic parameters) or the relative pose of the cameras (extrinsic parameters). In a calibrated stereo system, you typically should have knowledge of these parameters, which makes the reconstruction process more straightforward. But uncalibrated stereo 3D reconstruction offers a compelling solution to this challenge by enabling the extraction of three-dimensional information from images captured by uncalibrated cameras.

Now after obtaining a 3d point cloud through the process of uncalibrated 3D reconstruction we can move further for surface reconstruction.

Surface reconstruction from point cloud data is a fundamental task in computer graphics, computer vision, and geometric modeling. It involves the process of converting a set of discrete points sampled from an object or a scene into a continuous surface representation.

One of the initial steps in surface reconstruction is often the generation of a convex hull around the point cloud. A convex hull is the smallest convex set that contains all the points in the dataset. It provides a simplified outer boundary of the object or scene, which serves as the foundation for further analysis and processing. Convex hulling is essential for removing outliers, reducing computational complexity, and defining the overall shape of the object.

Triangulation is another key technique used in surface reconstruction. It involves partitioning the space around the object into a set of triangles that collectively form a mesh. Triangulation helps in defining the surface topology and geometry more accurately, facilitating tasks such as rendering, collision detection, and physical simulation.

Among various triangulation methods, Delaunay triangulation stands out due to its desirable properties. Delaunay triangulation maximizes the minimum angle of all the triangles in the mesh, resulting in more uniform and well-conditioned triangles. This property leads to better approximation of the underlying surface and improved stability in subsequent operations.

Additionally, Delaunay triangulation exhibits optimality in terms of minimizing the maximum circumradius of the triangles, which further enhances the quality of the resulting mesh.

In this context, we aim to explore the concepts of surface reconstruction, convex hull generation, and Delaunay triangulation in detail. We will discuss the theoretical foundations, algorithmic approaches, computational complexity, and practical considerations associated with these techniques.

Mathematical and Algorithmic Introduction:

Uncalibrated 3D Reconstruction:

Corresponding Points Matching:

Matching corresponding points between two images is a fundamental task in computer vision and image processing. It involves finding the same visual feature or object in both images. The first step is to detect distinctive features or keypoints in both images. These keypoints can be corners, edges, blobs, or other salient points in the image. Common feature (keypoints and descriptors) detection algorithms including harris Corner detector, Shi-Tomasi corner detector, FAST, SIFT, SURF ,ORB detectors can be used.

In this experiment we are going to use ORB detector for detecting keypoints in the given images.

After extracting keypoints and their descriptors from both images, we can perform the actual matching step. The goal is to find the corresponding keypoints in the two images by comparing the descriptors. There are several matching techniques. In this project we have used Brute-Force Matching. This approach involves comparing each descriptor in the first image with all descriptors in the second image. It can be computationally expensive and is often used with efficient data structures like KD-trees.

After matching, we can perform geometric verification to remove incorrect matches. This step is essential in tasks like image stitching or object recognition. Common techniques include

RANSAC (Random Sample Consensus) or Hough Transform to estimate and verify geometric transformations.

Some matches may be outliers due to occlusions, noise, or other factors. Outlier rejection techniques, such as the random sample consensus (RANSAC) algorithm, can be used to eliminate incorrect matches. After applying outlier rejection, we are left with a set of valid corresponding points in both images.

Epipolar Geometry and Fundamental Matrix Computation:

The epipolar geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis (the baseline is the line joining the camera centres).

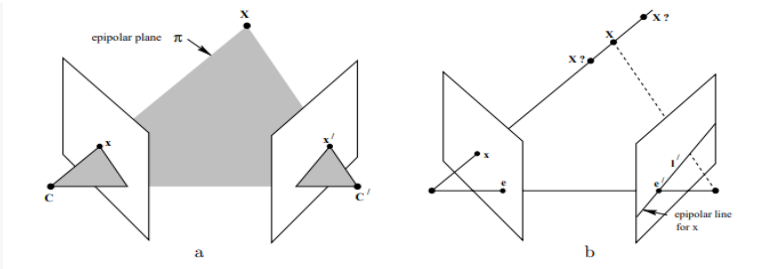


Fig: Epipolar plane and Epipolar line (Heartley and Zisserman 2003)

Suppose a point X in 3-space is imaged in two views, at x in the first, and x' in the second. As shown in figure the image points x and x', space point X, and camera centres are coplanar. Denote this plane as π. Clearly, the rays back-projected from x and x' intersect at X, and the rays are coplanar, lying in π.

Supposing now that we know only x, we may ask how the corresponding point x' is constrained. The plane π is determined by the baseline and the ray defined by x. From above we know that the ray corresponding to the (unknown) point x' lies in π, hence the point x' lies on the line of intersection l' of π with the second image plane. This line l' is the image in the second view of the ray back-projected from x. In terms of a stereo correspondence algorithm the benefit is that the search for the point corresponding to x need not cover the entire image plane but can be restricted to the line l' .

Given a pair of images, it was seen in the figure that to each point x in one image, there exists a corresponding epipolar line l' in the other image. Any point x' in the second image matching the point x must lie on the epipolar line l' .The epipolar line is the projection in the second image of the ray from the point x through the camera centre C of the first camera. Thus, there is a map $x \rightarrow l'$ from a point in one image to its corresponding epipolar line in the other image. It turns out that this mapping is a (singular) correlation, that is a projective mapping from points to lines, which is represented by a matrix F, the fundamental matrix.

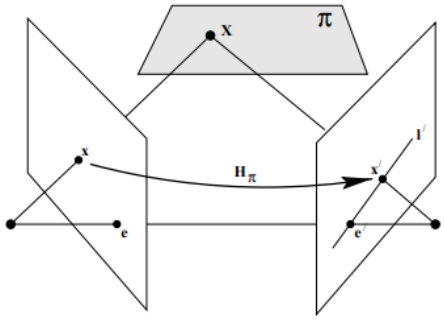


Figure: Corresponding 2D Points in Two Image Planes and Their 3D Object Point (Heartley and Zisserman 2003)

Consider a plane π in space not passing through either of the two camera centres. The ray through the first camera centre corresponding to the point x meets the plane π in a point X. This point X is then projected to a point x' in the second image. This procedure is known as transfer via the plane π. Since X lies on the ray corresponding to x, the projected point x' must lie on the epipolar line l' corresponding to the image of this ray. The points x and x' are both images of the 3D point X lying on a plane. The set of all such points x_i in the first image and the corresponding points x_i' in the second image are projectively equivalent, since they are each projectively equivalent to the planar point set X_i. Thus there is a 2D homography H_π mapping each x_i to x_i'.

Given the point x' the epipolar line l' passing through x' and the epipole e' can be written as

$l' = e' \times x' = [e']\times x'$. Since x' may be written as $x' = H_\pi x$, we have

$l' = [e']\times H_\pi x = F x$

Now as x' lies on the epipolar line $l' = F x$ corresponding to the point x. In other words

$0 = x'^T l' = x'^T F x.$

x and $x'$are homogenous representations of corresponding points in image 1 and image 2, respectively and F is the 3x3 fundamental matrix.

Using the epipolar constraint, we can represent the linear equations for multiple corresponding points in matrix form:

$$[u_1 \;\; v_1 \;\; 1]\begin{bmatrix}F_{11}\\F_{12}\\F_{13}\end{bmatrix}\begin{bmatrix}u_1'\\v_1'\\1\end{bmatrix} = 0$$

$$[u_2 \;\; v_2 \;\; 1]\begin{bmatrix}F_{21}\\F_{22}\\F_{23}\end{bmatrix}\begin{bmatrix}u_2'\\v_2'\\1\end{bmatrix} = 0$$

$$[u_n \;\; v_n \;\; 1]\begin{bmatrix}F_{21}\\F_{22}\\F_{23}\end{bmatrix}\begin{bmatrix}u_2'\\v_2'\\1\end{bmatrix} = 0$$

This can be rewritten in matrix form $A.f = 0$ where A is a matrix formed by stacking the expression of the epipolar corresponding points and f is a vector containing the nine elements of the fundamental matrix F.

To solve for f , we use the Singular Value Decomposition (SVD) of matrix A. The singular value decomposition of A is given by: A=USV^T, where U and V are orthogonal matrices S and is a diagonal matrix with singular values.

The vector f that minimizes Af subject to the constraint that $||f|| = 1$ is the right singular vector corresponding to the smallest singular value in V. This vector is the solution to the equation $A.f = 0$

The obtained fundamental matrix F is typically of rank 3. To enforce the rank-2 constraint, the SVD of F is performed. The smallest singular value is set to zero, reducing the rank to 2, and then F is reconstructed.

Finally, the obtained matrix F can be normalized if necessary.The implementation of this algorithm in code involves using matrix manipulations and the SVD method available in numerical computation libraries like NumPy, MATLAB, or other linear algebra libraries.

Geometric Error Cost Function Discussion and 3D point reconstruction Algorithm

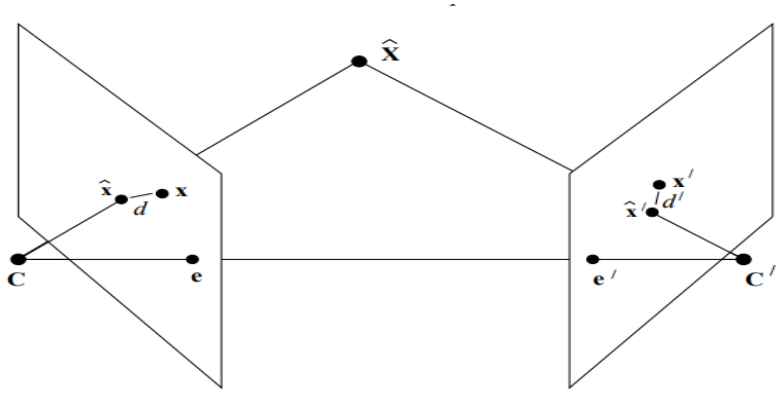


Figure: Minimization of geometric error. (Hartley and Zisserman 2003)

A typical observation consists of a noisy point correspondence $x \leftrightarrow x'$which does not in general satisfy the epipolar constraint. In reality, the correct values of the corresponding image points should be points $x^{\wedge} \leftrightarrow x^{\wedge'}$ lying close to the measured points $x \leftrightarrow x'$and satisfying the epipolar constraint $x^{\wedge'^{T}} F x^{\wedge} = 0$ exactly. We seek the points $x^{\wedge}$ and $x^{\wedge'}$that minimize the function

$$C(x, x') = d(x, x^{\wedge})^2 + d(x', x^{\wedge'})^2 \quad \text{subject to } x^{\wedge'^{T}} F x^{\wedge} = 0 \ldots..(1)$$

where $d(*,*)$ is the Euclidean distance between the points. This is equivalent to minimizing the reprojection error for a point $X^{\wedge}$which is mapped to $x^{\wedge}$ and $x^{\wedge\prime}$ by projection matrices consistent with F.

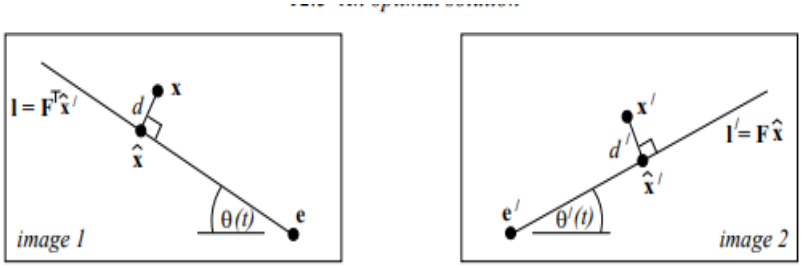


Figure : The projections $x^{\wedge}$ and $x^{\wedge\prime}$ of an estimated 3D point $X^{\wedge}$ lie on a pair of corresponding epipolar lines. (Hartley and Zisserman 2003)

Any pair of points satisfying epipolar constraint must lie on a pair of corresponding epipolar lines in the two images. Thus, in particular, the optimum point $x^{\wedge}$ lies on an epipolar line l and $x^{\wedge'}$ lies on the corresponding epipolar line l'. On the other hand, any other pair of points lying on the lines l and l' will also satisfy the epipolar constraint. This is true in particular for the point $x_{\perp}$ on l lying closest to the measured point x, and the correspondingly defined point $x_{\perp}^{'}$ on l'. Of all pairs of points on the lines l and l', the points $x_{\perp}$ and $x_{\perp}^{'}$ minimize the squared distance sum of equation 1. It follows that $x^{\wedge'} = x_{\perp}^{'}$ and $x^{\wedge} = x_{\perp}$, where $x_{\perp}$ and $x_{\perp}^{'}$ are defined with respect to a pair of matching epipolar lines l and l' . Consequently, we may write $d(x, x^{\wedge}) = d(x, l)$, where $d(x, l)$ represents the perpendicular distance from the point x to the line l. A similar expression holds for d($x^{'}$, $x^{\wedge'}$). In view of the previous paragraph, we may formulate the minimization problem differently as follows. We seek to minimize

$d(x, l)^2 + d(x^{'}, l^{'})^2$ --- (2)

where l and l' range over all choices of corresponding epipolar lines. The point $x^{\wedge}$ is then the closest point on the line l to the point x and the point $x^{\wedge'}$ is similarly defined. Our strategy for minimizing equation 2 is as follows:

(1) Parametrize the pencil of epipolar lines in the first image by a parameter t. Thus an epipolar line in the first image may be written as $l(t)$.

(2) Using the fundamental matrix F, compute the corresponding epipolar line l' (t) in the second image.

(3) Express the distance function $d(x, l(t))^2 + d(x', l'(t))^2$ explicitly as a function of t.

(4) Find the value of t that minimizes this function. In this way, the problem is reduced to that of finding the minimum of a function of a single variable t, i.e.

$$\min_{x^{\wedge}} C = d(x, x^{\wedge})^2 + d(x^{'}, x^{\wedge'})^2 = \min_{t} C = d(x, l(t))^2 + d(x^{'}, l'(t))^2$$

Now, we may simplify the analysis by applying a rigid transformation to each image in order to place both points x and $x^{'}$ at the origin, $(0,0,1)^T$ in homogeneous coordinates. Furthermore, the epipoles may be placed on the x-axis at points $(1,0,f)^T$ and $(1,0,f^{'})^T$ respectively. A value f equal to 0 means that the epipole is at infinity. Applying these two rigid transforms has no effect

on the sum-of-squares distance function in equation 1, and hence does not change the minimization problem.

Thus, in future we assume that in homogeneous coordinates, $x = x' = (0,0,1)^T$ and that the two epipoles are at points $(1,0,f)^T$ and $(1,0,f')^T$. In this case, since $F(1,0,f)^T = (1,0,f')F = 0$, the fundamental matrix has a special form

$$F = \begin{matrix} ff'd & -f'c & -f'd \\ -fb & a & b \\ -fd & c & d \end{matrix} \quad \ldots\ldots(3)$$

Consider an epipolar line in the first image passing through the point $(0,t,1)^T$ (still in homogeneous coordinates) and the epipole $(1,0,f)^T$. We denote this epipolar line by $l(t)$. The vector representing this line is given by the cross product $(0,t,1) \times (1,0,f) = (tf,1,-t)$, so the squared distance from the line to the origin is

$$d(x,l(t))^2 = \frac{t^2}{1+(tf)^2}$$

Using the fundamental matrix to find the corresponding epipolar line in the other image, we see that

$l'(t) = F(0,t,1)^T = (-f'(ct+d), at+b, ct+d)^T \ldots(4)$

This is the representation of the line $l'(t)$ as a homogeneous vector. The squared distance of this line from the origin is equal to

$$d(x', l'(t))^2 = \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2}$$

The total squared distance is therefore given by

$$s(t) = \frac{t^2}{1+(tf)^2} + \frac{(ct+d)^2}{(at+b)^2+f'^2(ct+d)^2} \qquad \ldots(5)$$

Our task is to find the minimum of this function. We may find the minimum using techniques of elementary calculus, as follows. We compute the derivative

$$s'(t) = \frac{2t}{(1+(f^2t^2))^2} + \frac{2(ad-bc)(at+b)(ct+d)}{((at+b)^2+f'^2(ct+d)^2)^2} \qquad \ldots(6)$$

Maxima and minima of $s(t)$ will occur when $s'(t)$ =0. Collecting the two terms in s' (t) over a common denominator and equating the numerator to 0 gives a condition

$$g(t) = t((at+b)^2 + f'^2(ct+d)^2)^2 - (\mathrm{ad} - \mathrm{bc})(1+(f^2t^2))^2\,(\mathrm{at}+\mathrm{b})(\mathrm{ct}+\mathrm{d}) = 0 \ldots.(7)$$

Now Given a measured point correspondence $x \leftrightarrow x'$, and a fundamental matrix F, in order to compute the corrected correspondences $\hat{x} \leftrightarrow \hat{x}'$ that minimize the geometric error subject to the epipolar constraint $\hat{x}'^T F \hat{x} = 0$ we will use the following algorithm .

(i) Define transformation matrices $T = \begin{matrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{matrix}$ and $T' = \begin{matrix} 1 & 0 & -x' \\ 0 & 1 & -y' \\ 0 & 0 & 1 \end{matrix}$.

These are the translations that take x = $(x, y, 1)^T$ and x' = $(x', y', 1)^T$ to the origin.

(ii) Replace F by $T^{-T} F\, T^{-1}$, The new F corresponds to translated coordinates.

(iii) Compute the right and left epipoles e =$(e1, e2, e3)^T$ and e' = $(e1', e2', e3')^T$such that $e^T F = 0$ and Fe = 0. Normalize (multiply by a scale) e such that $e1^2 + e2^2 = 1$ and do the same to e' .

(iv) Form matrices $R = \begin{matrix} e1 & e2 & 0 \\ -e2 & e1 & 0 \\ 0 & 0 & 1 \end{matrix}$ and $R' = \begin{matrix} e1' & e2' & 0 \\ -e2' & e1' & 0 \\ 0 & 0 & 1 \end{matrix}$.

and observe that R and R' are rotation matrices, and Re = $(1,0,e3)^T$and $R'e'$ = $(1,0,e3')^T$.

(v) Replace F by $R'FR^T$. (The resulting F must have the form equation 3).

(vi) Set $f = e3, f = e3', a = F22, b = F23, c = F32\ and\ d = F33.$

(vii) Form the polynomial $g(t)$ as a polynomial in t according to eqn. Solve for t to get 6 roots.

(viii) Evaluate the cost function equation 5 at the real part of each of the roots of $g(t)$ (alternatively evaluate at only the real roots of $g(t)$. Also, find the asymptotic value of equation 1 for t = ∞, namely $1/f^2 + c^2/(a^2 + f^2c^2)$. Select the value t_{min} of t that gives the smallest value of the cost function.

(ix) Evaluate the two lines $l = (tf, 1, -t)$ and l' given by equation 4 at t_{min} and find $x\hat{}$ and $x\hat{}'$ as the closest points on these lines to the origin. For a general line (λ, μ, ν), the formula for the closest point on the line to the origin is $(-\lambda\nu, -\mu\nu, \lambda^2 + \mu^2)$.

(x) Transfer back to the original coordinates by replacing $x\hat{}$ by $T^{-1}R^T x\hat{}$ and $x\hat{}'$ by $T'^{-1}R'^T x\hat{}'$.

(xi) The 3-space point $X\hat{}$ may then be obtained by the homogeneous method of linear triangulation.

Surface Reconstruction

Surface reconstruction is a problem in the field of computational geometry that is concerned with recreating a surface from scattered data points sampled from an unknown surface. Surface reconstruction algorithms have mostly been used in computer graphics applications thus far, where physical models are digitally scanned in three dimensions using mechanical digitising probes or laser range scanners. Surface reconstruction algorithms are used to convert the set of digitized points into a wire frame mesh model, which can be coloured, textured, shaded, and placed into a 3D scene

The surface reconstruction problem naturally arises in computer aided geometric design where it is often referred to as reverse engineering. Typically, the surface of some solid, e.g., a clay mock-up of a new car, has to be turned into a computer model. This modeling stage consists of (i) acquiring data points on the surface of the solid using a scanner (ii) reconstructing the surface from these points. Notice that the previous step is usually decomposed into two stages. First a piecewise linear surface is reconstructed, and second, piecewise-smooth surface is built upon the mesh. Surface reconstruction is also ubiquitous in medical applications and natural sciences, e.g., geology. In most of these applications the embedding space of the original surface is R3.

Convex Hull :

Surface reconstruction refers to the process of creating a continuous surface representation from a set of discrete points or a point cloud. One method for surface reconstruction is using the convex hull.

A convex hull is a fundamental concept in geometry, particularly in computational geometry, which refers to the smallest convex shape (typically a polygon or polyhedron) that encloses a given set of points or a set of objects in a Euclidean space. The convex hull of a set of points is the smallest convex set that contains the points.

An incremental algorithm for the convex hull repeatedly adds a point to the convex hull of the previously processed points. Of particular interest is the Beneath-Beyond Algorithm [Gru¨ nbaum 1961; Kallay 1981; Preparata and Shamos 1985]. A new point is processed in three steps. First, locate the visible facets for the point. The boundary of the visible facets is the set of horizon ridges for the point. A facet is visible if the point is above the facet. Second, construct a cone of new facets from the point to its horizon ridges. Third, delete the visible facets, thus forming the convex hull of the new point and the previously processed points.

The original randomized incremental algorithm upon which we build was proposed by Clarkson and Shor [1989]. They work in the dual space of halfspace intersections. Their algorithm adds a halfspace by intersecting it with the polytope of the previous intersections. They randomly select a halfspace to add to the polytope. For each unprocessed halfspace, they maintain the list of polytope edges that intersect the halfspace. The conflict graph is the set of all such lists. When a halfspace is processed, the conflict graph identifies the modified edges. To reduce memory requirements to O(n), they propose storing a single modified edge for each unprocessed halfspace. A simple search of adjacent edges identifies the remaining modified edges.

Our Quickhull Algorithm is a variation of Clarkson and Shor's algorithm. As with most of the variations, we work in the space of points and convex hulls instead of the dual space of halfspaces and polytopes. This turns the conflict graph into an outside set for each facet. A point is in a facet's outside set only if it is above the facet. As in Clarkson and Shor's algorithm, an unprocessed point is in exactly one outside set. Our variation is to process the furthest point of an outside set instead of a random point. In $\boldsymbol{R}^2$, this is the well-known Quickhull Algorithm

[Bykat 1978; Eddy 1977; Green and Silverman 1979; Preparata and Shamos 1985]. We can compare Quickhull with the randomized incremental algorithms by changing the selection step of Quickhull. If Quickhull selects a random point instead of a furthest point, it is a randomized incremental algorithm. In our empirical tests, Quickhull runs faster than the randomized algorithms because it processes fewer interior points. Also, Quickhull reuses the memory occupied by old facets.

We assume that the input points are in general position (i.e., no set of d + 1 points defines a (d - 1)-flat), so that their convex hull is a simplicial complex [Preparata and Shamos 1985]. We represent a d-dimensional convex hull by its vertices and (d - 1)-dimensional faces (the facets). A point is extreme if it is a vertex of the convex hull. Each facet includes a set of vertices, a set of neighboring facets, and a hyperplane equation. The (d - 2)-dimensional faces are the ridges of the convex hull. Each ridge is the intersection of the vertices of two neighbouring facets.

For processing a point we use a simplification of Gru¨nabaum's Beneath Beyond Theorem [Gru¨nbaum 1961, Theor. 5.2.1]. The randomized incremental algorithms are based on this theorem.

(SIMPLIFIED BENEATH-BEYOND ALGORITHM): Let H be a convex hull in $\boldsymbol{R}^d$, and let p be a point in $\boldsymbol{R}^d$- H. Then F is a facet of conv(p U H) if and only if
(1) F is a facet of H, and p is below F; or
(2) F is not a facet of H, and its vertices are p and the vertices of a ridge of H with one incident facet below p and the other incident facet above p.
The central problem of Beneath-Beyond is determining the visible facets efficiently. Since a facet is linked to its neighbours, locating one visible facet allows the rest to be located quickly. Most of the randomized algorithms use the previously constructed facets to locate the first visible facet for a point. Our solution is simpler. After initialization, Quickhull assigns each unprocessed point to an outside set. By definition, the corresponding facet is visible from the point. When Quickhull creates a cone of new facets, it builds new outside sets from the outside sets of the visible facets. This process, called partitioning, locates a visible new facet for each point. If a point is above multiple new facets, one of the new facets is selected. If it is below all of the new facets, the point is inside the convex hull and can be discarded. Partitioning also records the furthest point of each outside set.

QuickHull algorithm

```
create a simplex of d+1 points
for each facet F
    for each unassigned point p
        if p is above F
            assign p to F's outside set
for each facet F with a non-empty outside set
    select the furthest point p of F's outside set
    initialize the visible set V to F
    for all unvisited neighbours N of facets in V
        if p is above N
            add N to V
    the boundary of V is the set of horizon ridges H
    for each ridge R in H
        create a new facet from R and p
        link the new facet to its neighbour
    for each new facet F'
        for each unassigned point q in the outside set of a facet in V
            if q is above F'
                assign q to F' 's outside set
    delete the facets in V
```

Quickhull selects a nondegenerate set of points for the initial simplex. If possible, it selects points with either a maximum or minimum coordinate.

To prove the correctness of Quickhull, we first prove that a point can be partitioned into any legal outside set. If so, extreme points of the input will always be vertices of the output. Assume the contrary, and consider an extreme point p that is not assigned to an outside set and hence never added to the convex hull. Since p is an extreme point, it must have been outside at least one facet of the initial simplex. By assumption, there is a point q with p in its visible outside sets but not in its new outside sets. So p is above a visible facet and below all new facets for q. This implies that p is inside the convex hull and hence not an extreme point.

Quickhull and the randomized algorithms perform essentially the same steps, but we prefer Quickhull. Quickhull uses less space than most of the randomized incremental algorithms and runs faster for distributions with non extreme points. The main costs for these algorithms are creating facets and computing distances. To isolate the effect of randomization on time efficiency, we can change the processing order of Quickhull. Instead of selecting the furthest point of an outside set, we can select a random point from all outside sets.

We conjecture that Quickhull iterations are average (i.e., each iteration creates an average number of new facets whose outside sets are average size). This defines two balance conditions. Let d be the dimension, n the number of input points, r the number of processed points, and f_r the maximum number of facets of r vertices (f_r = O($r^{\lfloor \frac{d}{2} \rfloor}$/⌊d/2⌋!)

There are two costs to Quickhull: adding a point to the hull and partitioning. The dominant cost for adding a point is O(d^3) work to create hyperplanes. The dominant cost for partitioning is O(d) work to compute distances. The total cost for adding points is proportional to the total number of new facets created, O($d^3 \sum_{j=1}^{r} df_i / j$). This simplifies to O($f_r$) (each term is less than df_r/r, and the ⌊d/2⌋! denominator of f_r subsumes d^4). The cost of partitioning one point in one iteration is proportional to the number of new facets. The total cost is O (d $\sum_{j=1}^{r} d^2(n-j)f_j / j^2$). Expanding f_j yields O($d^3\, n \sum_{j=1}^{r} j^{\lfloor \frac{d}{2} \rfloor - 2}$ /⌊d/2⌋!). If d ≤ 3, the sum is O(n log r). Otherwise, each term is less than f_r/r^2 , and the sum is O(n f_r/r).

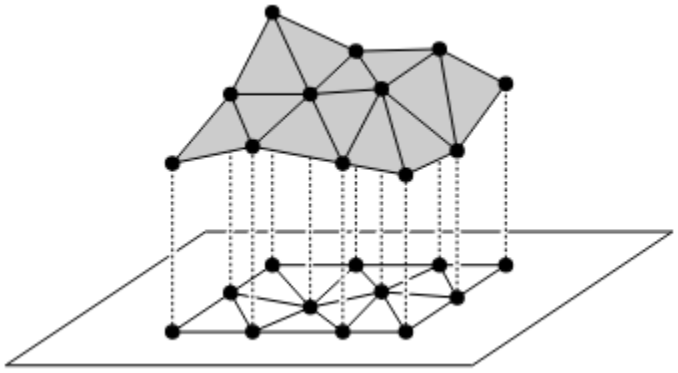
Our goal is a practical algorithm for general-dimension convex hulls. We have shown empirical evidence that the algorithm satisfies its balance conditions and that it performs like a randomized incremental algorithm that is output sensitive to the number of vertices. The Quickhull Algorithm uses less space than most of the randomized incremental algorithms and executes faster for inputs with non extreme points.

Triangulations

Height Interpolation and Terrain Modelling:

We can model a piece of the earth's surface as a terrain. A terrain is a 2-dimensional surface in 3-dimensional space with a special property: every vertical line intersects it in a point, if it intersects it at all. In other words, it is the graph of a function $f : A \subset R2 \rightarrow R$ that assigns a height f(p) to every point p in the domain, A, of the terrain. We don't know the height of every point on earth; we only know it where we've measured it. This means that when we talk about some terrain, we only know the value of the function f at a finite set $P \subset A$ of sample points. From the height of the sample points we somehow have to approximate the height at the other points in the domain.

Our approach for approximating a terrain is as follows. We first determine a triangulation of P: a planar subdivision whose bounded faces are triangles and whose vertices are the points of P. (We assume that the sample points are such that we can make the triangles cover the domain of the terrain.) We then lift each sample point to its correct height, thereby mapping every triangle in the triangulation to a triangle in 3-space. The figure below illustrates this. What we get is a polyhedral terrain, the graph of a continuous function that is piecewise linear. We can use the polyhedral terrain as an approximation of the original terrain.



Triangulations of Planar Point Sets

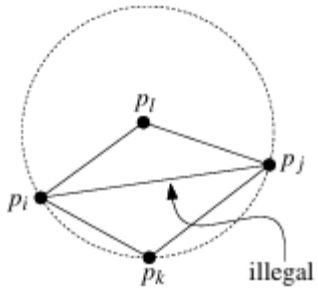
Let P :={p1,p2,...,pn} be a set of points in the plane. To be able to formally define a triangulation of P, we first define a maximal planar subdivision as a subdivision S such that no edge connecting two vertices can be added to S without destroying its planarity. In other words, any edge that is not in S intersects one of the existing edges. A triangulation of P is now defined as a maximal planar subdivision whose vertex set is P.

It is not difficult to see that any segment connecting two consecutive points on the boundary of the convex hull of P is an edge in any triangulation T. This implies that the union of the bounded faces of T is always the convex hull of P, and that the unbounded face is always the complement of the convex hull. The number of triangles is the same in any triangulation of P. This also holds for the number of edges. The exact numbers depend on the number of points in P that are on the boundary of the convex hull of P. Let P be a set of n points in the plane, not all collinear, and let k denote the number of points in P that lie on the boundary of the convex hull of P. Then any triangulation of P has 2n−2−k triangles and 3n−3−k edges.

Let C be a circle, a line intersecting C in points a and b, and p, q, r, and s points lying on the same side of . Suppose that p and q lie on C, that r lies inside C, and that s lies outside C. Then

$$\measuredangle arb > \measuredangle apb = \measuredangle aqb > \measuredangle asb.$$

Also if edge pipj be incident to triangles pipjpk and pipjpl, and let C be the circle through pi, pj, and pk. The edge pipj is illegal if and only if the point pl lies in the interior of C. Furthermore, if the points pi, pj, pk, pl form a convex quadrilateral and do not lie on a common circle, then exactly one of pipj and pkpl is an illegal edge. Observe that the criterion is symmetric in pk and pl: pl lies inside the circle through pi, pj, pk if and only if pk lies inside the circle through pi, pj, pl. When all four points lie on a circle, both pipj and pkpl are legal. Note that the two triangles incident to an illegal edge must form a convex quadrilateral, so that it is always possible to flip an illegal edge. We define a legal triangulation to be a triangulation that does not contain any illegal edge.



Algorithm: LEGALTRIANGULATION(T)
Input. Some triangulation T of a point set P.
Output. A legal triangulation of P.
while T contains an illegal edge pipj
do (* Flip pipj *)
Let pipjpk and pipjpl be the two triangles adjacent to pipj
Remove pipj from T, and add pkpl instead.
return T

Delaunay Triangulation

In computational geometry, a Delaunay triangulation or Delone triangulation of a set of points in the plane subdivides their convex hull into triangles whose circumcircles do not contain any of the points. This maximizes the size of the smallest angle in any of the triangles, and tends to avoid silver triangles.

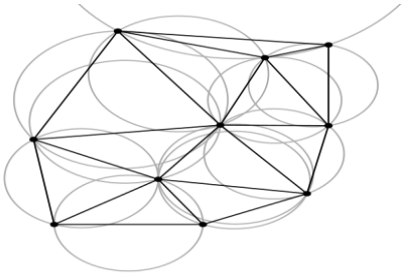


Fig: A Delaunay triangulation in the plane with circumcircles shown (Fig from Wikipedia)

The Delaunay triangulation of a discrete point set P in general position corresponds to the dual graph of the Voronoi diagram for P. The circumcenters of Delaunay triangles are the vertices of

the Voronoi diagram. In the 2D case, the Voronoi vertices are connected via edges, that can be derived from adjacency-relationships of the Delaunay triangles: If two triangles share an edge in the Delaunay triangulation, their circumcenters are to be connected with an edge in the Voronoi tessellation.

Let P be a set of n points. The Voronoi diagram of P is the subdivision of the plane into n regions, one for each site in P, such that the region of a site $p \in P$ contains all points in the plane for which p is the closest site. The Voronoi diagram of P is denoted by Vor(P). The region of a site p is called the Voronoi cell of p; it is denoted by V(p). Graph G has a node for every Voronoi cell—equivalently, for every site—and it has an arc between two nodes if the corresponding cells share an edge. Note that this means that G has an arc for every edge of Vor(P). As you can see in Figure below, there is a one-to-one correspondence between the bounded faces of G and the vertices of Vor(P).

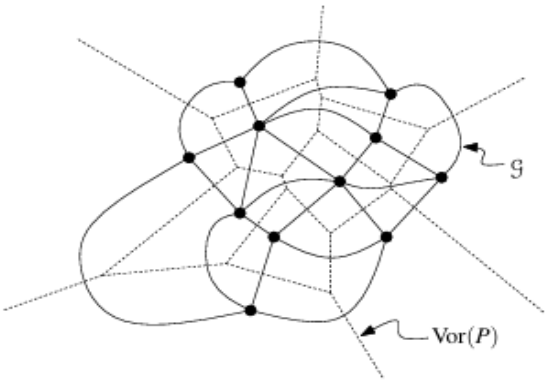


Fig: The dual graph of Vor(P)

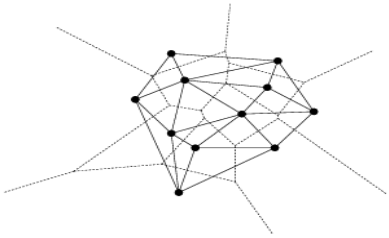


Fig: The Delaunay graph DG(P)

Consider the straight-line embedding of G, where the node corresponding to the Voronoi cell V(p) is the point p, and the arc connecting the nodes of V(p) and V(q) is the segment pq. We call this embedding the Delaunay graph of P, and we denote it by DG(P).

Computing the Delaunay Triangulation

The algorithm is randomized incrementally, so it adds the points in random order and it maintains a Delaunay triangulation of the current point set. Consider the addition of a point pr. We first find the triangle of the current triangulation that contains pr —and we add edges from pr to the vertices of this triangle. If pr happens to fall on an edge e of the triangulation, we have to add edges from pr to the opposite vertices in the triangles sharing e. We now have a triangulation again, but not necessarily a Delaunay triangulation. This is because the addition of pr can make some of the existing edges illegal. To remedy this, we call a procedure LEGALIZEEDGE with each potentially illegal edge. This procedure replaces illegal edges by legal ones through edge flips.

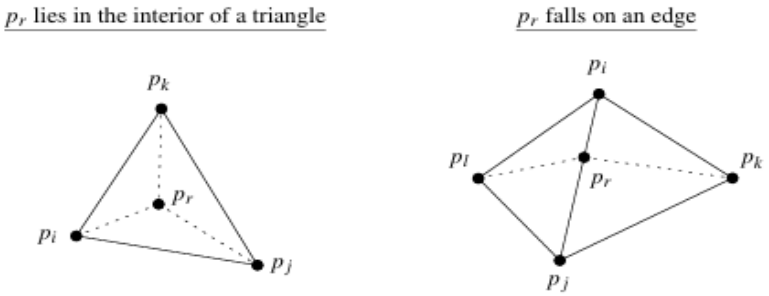


Fig: The two cases when adding a point pr

Algorithm: DELAUNAYTRIANGULATION(P)

Input. A set P of n+1 points in the plane.

Output. A Delaunay triangulation of P.

Let p0 be the lexicographically highest point of P, that is, the rightmost among the points with largest y-coordinate.

Let p−1 and p−2 be two points in R2 sufficiently far away and such that P is contained in the triangle p0 p−1 p−2.

Initialize T as the triangulation consisting of the single triangle p0p−1p−2.

Compute a random permutation p1,p2,...,pn of P\{p0}.

for r ←1to n

do (* Insert pr into T: *)

Find a triangle pipjpk ∈ T containing pr.

```
    if pr lies in the interior of the triangle pi pj pk
        then Add edges from pr to the three vertices of pipjpk, thereby
            splitting pipjpk into three triangles.
            LEGALIZEEDGE(pr,pipj,T)
            LEGALIZEEDGE(pr,pjpk,T)
            LEGALIZEEDGE(pr,pkpi,T)
        else (* pr lies on an edge of pipjpk, say the edge pipj *)
            Add edges from pr to pk and to the third vertex pl of the other
            triangle that is incident to pipj, thereby splitting the two triangles
            incident to pipj into four triangles.
            LEGALIZEEDGE(pr,pipl,T)
            LEGALIZEEDGE(pr,plpj,T)
            LEGALIZEEDGE(pr,pjpk,T)
            LEGALIZEEDGE(pr,pkpi,T)
Discard p−1 and p−2 with all their incident edges from T.
return T
```

A triangulation is a Delaunay triangulation if all its edges are legal. In the spirit of algorithm LEGALTRIANGULATION, we therefore flip illegal edges until the triangulation is legal again. The question that remains is which edges may become illegal due to the insertion of pr. Observe that an edge pipj that was legal before can only become illegal if one of the triangles incident to it has changed. So only the edges of the new triangles need to be checked. This is done using the subroutine LEGALIZEEDGE, which tests and possibly flips an edge. If LEGALIZEEDGE flips an edge, other edges may become illegal. Therefore LEGALIZEEDGE calls itself recursively with such potentially illegal edges.

Algorithm: LEGALIZEEDGE(pr, pipj,T)

```
(* The point being inserted is pr, and to be flipped. *)
if pipj is illegal
    then Let pipjpk be the triangle adjacent to prpipj along pipj.
        (* Flip pipj: *) Replace pipj with prpk.
        LEGALIZEEDGE(pr,pipk,T)
        LEGALIZEEDGE(pr,pkpj,T)
```

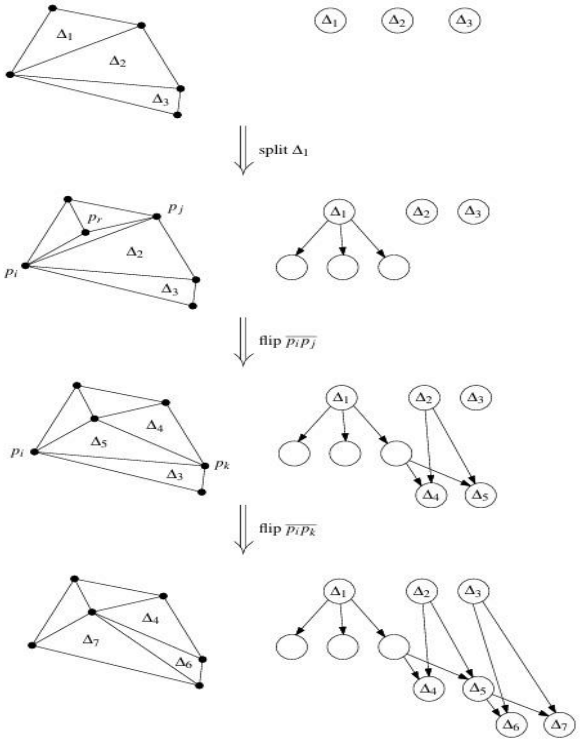
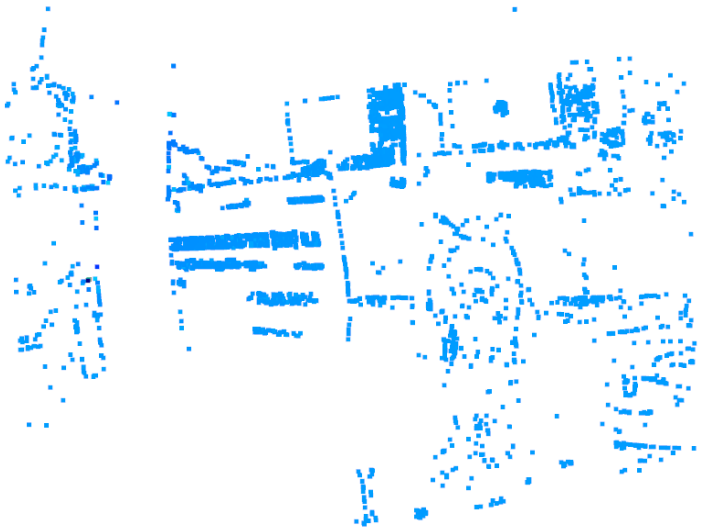


Fig: The effect of inserting point pr into triangle ∆1 on the data structure D (the part of D that does not change is omitted in the figure)

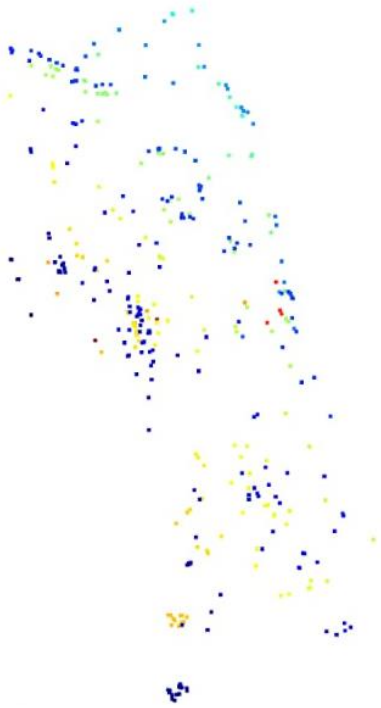
Experimental Set up and Results:

We used a pair of cameras and captured images of the same scene at the same time. Now running the python code of the algorithm discussed above we can get the reconstructed 3D points of the scene back and plot them using open3D library and can obtain a 3D point cloud. And moving further we used the algorithm for obtaining the convex hull and the Delaunay triangulation and ran a python code and obtain the surface reconstructed figures.

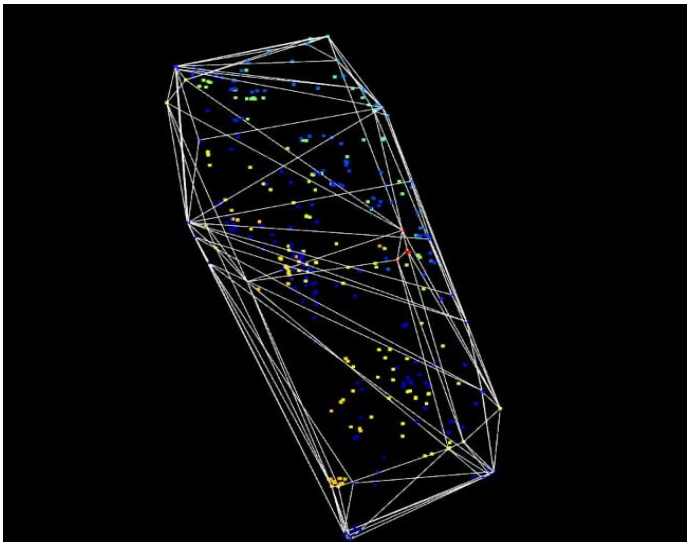
Some of the images and reconstructed 3D point clouds are shown below .



3D point cloud(6996 Matching Points)



3D point cloud of the statue



Convex Hull of the 3d point cloud of the statue

Triangulated 3D point cloud of the statue

Future Scope and Conclusion :

In this experiment we mainly focused on surface reconstruction of the 3D point cloud obtained previously from uncalibrated 3d reconstruction and we obtain the desired convex hull and the Delaunay triangulated figure of that point cloud .

Now we can implement more sophisticated algorithms for smoothing the point cloud data. This could involve techniques like Gaussian smoothing, bilateral filtering, or variational methods to preserve important features while reducing noise. We can develop methods to accurately map textures onto the smoothed surface. This could involve techniques such as UV mapping, texture splatting, or even deep learning-based approaches for texture synthesis and analysis. Apart from this, We can extend the analysis beyond color to semantic segmentation, where different parts of the object are identified and labeled based on their function or category. This could be useful for tasks like object recognition, scene understanding, or virtual reality applications.

The combination of convex hulling, triangulation, smoothing, and texture analysis opens up a wide range of possibilities for analyzing and visualizing 3D point cloud data. By further refining these techniques and integrating them with advanced algorithms and machine learning models, we can unlock even more insights from these rich datasets. Whether it's for improving object recognition, enhancing virtual environments, or aiding in scientific research, the future looks promising for the field of 3D point cloud analysis and visualization.

References:

- R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision. Cambridge, U.K.: Cambridge Univ. Press, 2003.
- C. BRADFORD BARBER , DAVID P. DOBKIN and HANNU HUHDANPAA ,The Quickhull Algorithm for Convex Hulls,1996
- http://www.qhull.org Qhull Scipy Documentation
- "Computational Geometry: Algorithms and Applications" by Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Springer.
- Claire Bourgeois-République, Albert Dipanda, Alain Koch. “A structured light system encoding for an uncalibrated 3D reconstruction based on Evolutionary Algorithms”. 2013 International Conference on Signal-Image Technology & Internet-Based Systems.
- Wei Yin , Jianming Zhang , Oliver Wang , Simon Niklaus ,Simon Chen, Yifan Liu , and Chunhua Shen. “Towards Accurate Reconstruction of 3D Scene Shape From A Single Monocular Image“. IEEE Trans. on Pattern Analysis And Machine Intelligence, vol. 45, no. 5, May 2023.
- A.Rajeswari ,B.Bhuvaneshwari ,V.Gowri Priyaa. “Depth Measurement and 3D Reconstruction of Stereo Images”. 2012 International Conference on Communication Systems and Network Technologies.