GSoC 2021 Proposal
Boost.Geometry

Personal Details

Name	Ayush Kumar Gupta
College	Indian Institute of Technology,Roorkee
Major	Computer Science and Engineering
Program	B Tech CSE
Email	ayush.gupta02071@gmail.com
Homepage	Github:https://github.com/ayushgupta138
Mobile No	+91 9879897031
Time Zone	IST (UTC +0530)

Availability

- My summer vacations start from **10th June - 2 August** and the code coding phase extends from **7 June - 16 August**, which is almost entirely under the vacation period. So, I can easily devote 40-45 hrs per week to the project.
- My intended start and end date match with the GSoC timeline with some prior work on the topics on which my proposal is based.
- I have no prior commitments planned for my vacations. I shall keep the

community posted in case of any change of plans.

Background Information

I am currently in my sophomore year, pursuing B Tech in Computer Science and Engineering from Indian Institute of Technology, Roorkee.
Some of the courses taken which are relevant to the project are :

- **Data Structures** - Included **complexity analysis** (asymptotic complexity Big O/Theta/Omega) , linear lists, sparse tables, stacks and queues (implementing various queuing systems), **hashing** (LZW algorithm), **search trees** (AVL Trees, Red-Black Trees, **set and multiset implementations in STL**) , Multiway Trees (**B-Tree**), **Graphs** and their traversal.

- **Discrete Structures** - Included **Number theory**(Chinese Remainder theorem, Euler Totient, Prime factorization, Modular arithmetic, Euclid's gcd), **Set theory**, **Graph theory** (theoretical aspects), Logic, **Abstract algebra**(groups, rings, field), discrete logarithm.

- **Design and Analysis of Algorithms** - Included Algorithmic techniques(**Divide and Conquer**, Strassen matrix multiplication, Knapsack problem, **Dynamic programming**, Travelling salesman problem, Backtracking, **Branch and bound**), **Graph algorithms** (Breadth first search, Depth first search, Krushkal's, Prim's and Sollin's algorithm, Shortest path problems, Topological sorting, Bipartite matching), P/NP class problems, **Parallel algorithms**, Miscellaneous algorithms(Randomized algorithms, Approximation algorithms, Number theoretic algorithms).

- **Computer Architecture and Microprocessors**

- **System Software**

- **Operating Systems**

- **Object Oriented Analysis and design**

- **Software Engineering**

Programming Background

I was introduced to traditional C++ (C++ 98) programming during my high school years. My work since freshman year includes working with modern C++, Java, Matlab, Python and Dart.

I have developed some projects during my college days. Some of them include - parallelization of Computer Vision algorithms using CUDA programming on GPUs, which uses the concept of template metaprogramming, Wifi based applications, etc.

Being a competitive programmer, I have encountered various geometry algorithms and I understand the importance of template metaprogramming techniques in C++ STL and other standard libraries. Since then C++ metaprogramming techniques have intrigued me. Here is my competitive programming background
https://codeforces.com/profile/ayush_gupta .

My interests include generic programming, lambda expressions, constexpr expressions, type inference , etc. Apart from this I am also interested in the field of Deep Learning and Quantum Computing.

I was introduced to Boost C++ some months back, and the mechanism of tag dispatch interested me a lot. Since Boost is an organization which heavily uses these techniques, it was an easy choice.

The reason for specifically choosing Boost.Geometry was my competitive programming background which made me very interested in implementation of Computational Geometry algorithms through generic programming.

I plan to continue the work on computational geometry and metaprogramming techniques in C++ and continue contributing to Boost.Geometry after this Summer of Code 2021.

I am comfortable with Visual Studio and Visual Studio Code integrated with gdb. For documentation purposes I am most familiar with Doxygen. I am comfortable with both Windows and Ubuntu 20.04 operating system.

My Rating of the following are:

1. C++ 98/03 : 3
2. C++ 11/14 : 4
3. C++ Standard Library: 4.5
4. Boost C++ Libraries : 3.5
5. Git : 3.5

Some of my previous contribution to Boost.Geometry are:

Status : merged

Git : Update unit test for parse.hpp

Github: https://github.com/boostorg/geometry/pull/806

Status : merged

Git : Fix Documentation error in Area strategy

Github: https://github.com/boostorg/geometry/pull/822

Status : merged

Git : Fix #define directive convention in area_result.hpp and default_area_result.hpp

Github: https://github.com/boostorg/geometry/pull/826

Project Proposal

This project proposal is not from the Boost.Geometry's suggested projects. This proposal contains a project of my interest, on which I would love to work if selected for GSoC 2021. I propose the design and implementation of the

following 3D functions:

1) Extension of area function to 3D geometries for surface area computations.
2) Extension of distance function to 3D point sets.
3) Extension of convex hull for 3D point sets.

As seen in the issue https://github.com/boostorg/geometry/issues/683, output of convex_hull function was erroneous for 3D point sets. Moreover Boost.Geometry currently does not have support for polyhedral surfaces (Pull request https://github.com/boostorg/geometry/pull/789 to be reviewed and merged).

My project proposal is dependent on the following work:

Work done so far

Through the pull request https://github.com/boostorg/geometry/pull/789 on Polyhedral Surfaces, the following features have been implemented:

1. Definition of polyhedral surface geometry with the default and list based initialization and tag,point_type,ring_type,Poly_ring_type type traits.
2. Polyhedral Surface tag definition.
3. Polyhedral concept definition and corresponding concept checks.
4. Wkt read and write features.

But since, only polyhedral surfaces have been defined, some of the metafunctions are yet to be implemented.

Since my project proposal requires that the above functionality be implemented, I will try to implement these functionalities before the GSoC coding period starts.

Area Extension(ST_3DArea() from PostGIS specs)

Since the current version of Boost.geometry(1.76) has no support for calculating area and surface areas of 3D geometries, I propose the following plan for extending the area free function to 3D geometries like 3D planar rings, 3D planar polygons and surface area of polyhedral surface.

1. **Area function extension for 3D planar rings**
   For achieving this purpose, I propose a different strategy/Policy for 3D calculations.

   There are three methods(derivation can be found here [1]) to compute the area of planar rings in three dimensions:

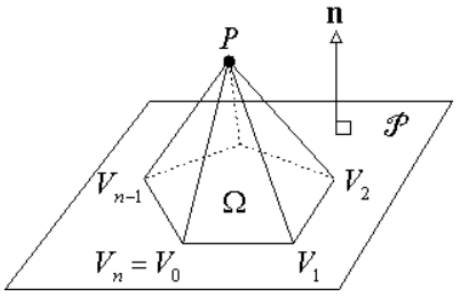


Fig 1.1 - Planar ring surface area

1) **Standard formula [Goldman,1994] -** This method includes choosing an arbitrary point P(not necessarily on the ring) and finding the area of each of the triangular faces of the pyramid. Then the projection of the area on the ring is calculated and summed up. The derivation of the formula can be found here [1].

   The formula reduces to :

   2*A(ring) = **n** * $\sum_{i=0}^{n-1} (V_i - P) \times (V_{i+1} - P)$, where **n** is the unit normal vector to the plane of the ring.
   With a choice of P as origin,

   2*A(ring) = **n** * $\sum_{i=0}^{n-1} (V_i \times V_{i+1})$

   This method computes, 6*n+3 multiplications and 4*n+2 additions and one square root operation operation.
2) **Quadrilateral Decomposition [Van Gelder, 1995]** - In this method instead of decomposing a ring into triangles, it is decomposed into quadrilaterals. Then the area of these quadrilaterals are computed using cross products and summed up. The final formula comes out to be:

$$2\,\mathrm{A}(\Omega) = \mathbf{n} \bullet \left(\sum_{i=1}^{h-1} \left(V_{2i} - V_0\right) \times \left(V_{2i+1} - V_{2i-1}\right) + \left(V_{2h} - V_0\right) \times \left(V_k - V_{2h-1}\right) \right)$$

   Where h = floor((n-1)/2) and k= 0 if n is odd and k=n-1 if n is even.

   This method computes 3*n+3 multiplications and 5*n+1 additions.

3) **2D projection [Snyder & Barr,1987]** - This is the most efficient method to compute the area of planar rings using less number of additions and multiplications as compared to the above mentioned methods.

   This method first discards one of the three dimensions and then computes the area of the projected 2D polygon using the existing boost::geometry::area() function. The choice of which coordinate to ignore is as follows:

   First, the normal vector to the plane is calculated, the dimension with the greatest absolute value is ignored.(This is done to increase the robustness of the function, as compared to the case where we ignore the dimension randomly).

   Let the normal vector to the plane be **n** = (n_x, n_y, n_z) and c = x,y,z be the coordinate that is discarded.

   Area of the planar ring is then given by:

$$\frac{\mathrm{A}\left(\mathrm{Proj}_c(\boldsymbol{\Omega})\right)}{\mathrm{A}(\boldsymbol{\Omega})} = \frac{n_c}{|\mathbf{n}|} \quad \text{where } c = x, y, \text{ or } z$$

   Where A(Ω)is the ring area and A($Proj_c(\Omega)$) is the area of the projected 2D polygon.

   The computation cost of this algorithm is : n+5 multiplications, 2*n+1 additions, 1 square root operations, and a small overhead to choose which coordinate to discard.

   I propose the **2D projection method** for area computation of 3D planar rings because it is the most efficient method and uses the previously implemented boost::geometry::area() function.

2. **Area function extension for 3D planar polygons**
   Based on the above implementation for the area of 3D planar rings, I propose to extend the area function to 3D planar polygons in a similar way as the current implementation of 2D polygon using area function for 2D rings(Calling functions for interior and exterior rings).

3. **Area function extension for polyhedral surfaces**

Since polyhedral surfaces are represented as a collection of 3D planar rings, the surface area of the polyhedral surface can be computed using the area function described above, iterating through the polyhedral surface using boost::range::iterators.

If time permits, then after the implementation of the proposed functions, I would also define 3D boxes and implement the surface area function for the same.

Convex Hull Extension(ST_ConvexHull from PostGIS specs)

Since the current implementation of boost::geometry::convex_hull() uses **Graham-Andrew strategy** for hull calculation, the extension of the same strategy is not possible for 3D point sets. The following are some of the most efficient algorithms for convex_hull():

1) **Divide and conquer[Preparata and Shamos 1985] -** The divide and conquer algorithm is the most efficient algorithm for 3D hull computation. Having a recurrence relation:

   T(n)=2*T(n/2)+O(n)

   The algorithms achieves a complexity of $\theta(nlogn)$ for all possible inputs. The algorithm is completely described in [2] and [3].

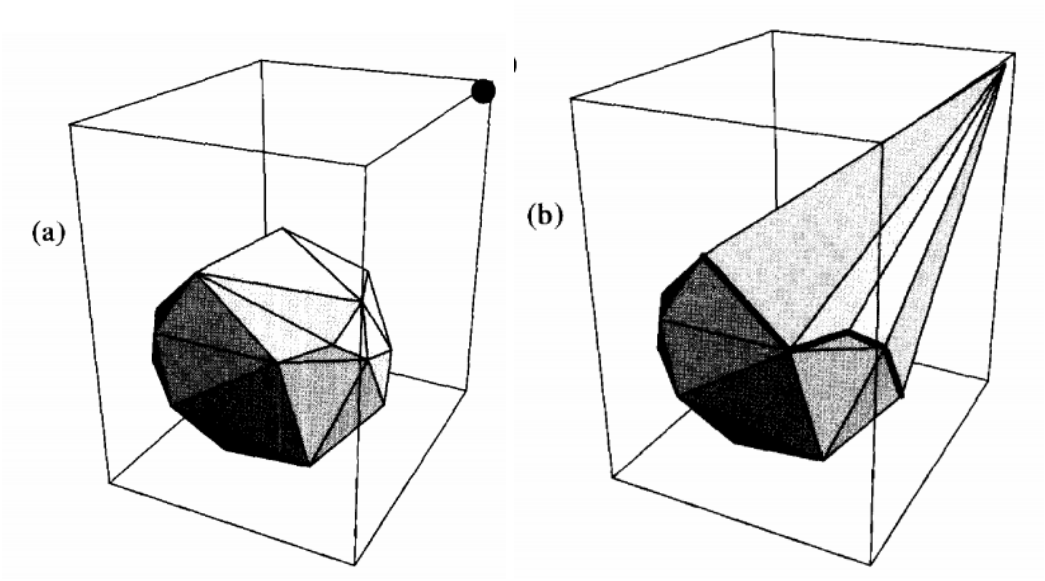
2) **Randomized Incremental Algorithm[Clarkson and Shor 1989] -** The randomized incremental algorithm is an algorithm that can be practically implemented with an **expected complexity** of **O(nlogn).** This algorithm is different from the incremental algorithm which adds a point at every iteration. As mentioned in [4], the algorithm processes the points in a random order, in order to minimize the number of points added to the hull. Moreover a conflict graph is also maintained for easy and time efficient insertion of points.

3) **QuickHull Algorithm for 3D points[Barber, Dobkin and Huhdanpaa] -** The QuickHull algorithm is one of the many variants of the original Randomized Incremental Algorithm which differs only in the choice of points to be added. At each step the point to be added is the farthest one, which minimizes the total points to be added. The performance of **QuickHull** algorithm is **better** than that of **Randomized incremental.**

Since the Divide and conquer algorithm is practically very difficult to implement

and there is no clear implementation of the same, I propose the implementation of the **QuickHull algorithm** for hull construction. The QuickHull algorithm has an expected time complexity of O(nlogn).

Moreover, the expected time complexity of the QuickHull algorithm is the same as the Divide and Conquer approach (O(nlogn)). I propose the implementation mentioned in [5] for the QuickHull algorithm.



a) Before point addition b) After point addition

Distance function Extension(ST_3DDistance from PostGIS specs)

The current implementation of boost::geometry::distance() function only works for 2D geometry objects and discards all the points other than the first two points. I propose to extend the distance function to 3D geometry objects.

Since there are large combinations of geometry objects for which distance function needs to be implemented, I propose the implementation of a subset of the complete combinational implementation of distance function.

a) Distance calculation between a 3D point and a planar ring.
b) Distance calculation between a 3D point and a planar polygon.
c) Distance calculation between a 3D point and a polyhedral surface.

Here by distance I mean shortest distance between two geometries.

1. **Point and Planar Rings**

   I propose the projection method for shortest distance calculation between a 3D point and the planar ring. This is the most efficient method for distance calculation as it uses the previously optimized boost::geometry::distance() function. The rough method is as follows:
   First we find the projection of the 3D point onto the plane of the ring and then find the shortest distance in 2D realm. The final shortest distance would be:

   $$d_{min} = \sqrt{d^2 + r^2}$$

   Where,

   d is the perpendicular distance between the point and the plane of the ring.

   r is the minimum 2D distance between projected point and planar ring.

   The formula for finding d is given here [6].

2. **Point and planar polygon**

   Similar to the current Boost.Geometry implementation for distance between point and polygon, I propose the implementation of distance function for 3D planar rings using the above mentioned function.

3. **Point and polyhedral surface**

   There are two ways to find the distance between a 3D point and a polyhedron.

   a) **Distance between two convex point sets[Gilbert, Johnson and Keerthi 1988 [7]] -** The GJK algorithm is a general purpose algorithm for calculating shortest distance between two convex sets in any arbitrary dimension in **O(mlogm)** time,where m is the total number of points in two convex point sets. Since a point is by default a convex set, the algorithm can be used to compute distance between points and convex polyhedron and distance between two convex polyhedrons.

b) **Using visible face method -** In this method, similar to the one used in QuickHull and Randomized Incremental Algorithm, we first identify all the faces visible from a given point and compute only the distance of those planes to the given point.

   The distance would be the minimum of the distance calculated.

Since both the methods have some pros and cons, the algorithm I propose both the algorithms for distance calculation. My implementation would depend on the suggestions of the mentors and the demands of the community. I am open to implementation of both the mentioned algorithms.

Proposed Milestones and Schedule

Community bonding phase(May 17 - June 7)

- Familiarize with the community.
- Implement some meta functions for polyhedral surfaces.
- Discuss and finalize specific algorithms to be implemented.
- Finalize algorithms and start implementation of area extension for planar rings

Phase 1 (June 7 - June 20)

- Finalize area extension for planar rings and planar polygon
- Complete area extension for polyhedral surfaces.
- Unit testing of the area function extension and documentation.

Phase 2 (June 21 - July 6)

- Implement helper functions and decide optimal data structures for 3D convex hull.
- Implement QuickHull algorithm for convex hulls.
- Finalize convex hull, unit testing and documentation.

Phase 3 (July 7 - July 20)

- Implement distance function extension for planar rings and polygons using the above mentioned algorithms.

- Implement distance function for a point and polyhedral surface.
- Finalize distance function for two convex sets (Depends on which algorithm is chosen during the community bonding phase).
- Unit testing and documentation.

Phase 4 (July 21 - August 7)

- Final code review and potential bug fixes.
- Finalize all proposed functions.
- Finalize documentation and statistical testing.

August 8 - August 16

Buffer period for unexpected delays.

Plans beyond Summer of Code time frame

After the implementation of all the proposed functions, I plan to further extend distance function for other 3D geometry combinations. I also plan to implement the 3D box geometry and implement the proposed functions for the same.

Competency Test

I have implemented a prototype of a **Randomized incremental algorithm** for 3D convex hulls. Since I propose to implement the QuickHull algorithm, it only requires a small modification in the Randomized Incremental Algorithm to arrive at QuickHull algorithm. The prototype implementation constructs the hull only for boost::model::multipoint.

Link to my competency test is:

(will add in final proposal)
The competency test contains the code as well as some results produced by the code on multipoints.

Acknowledgement

(will add in final proposal)

References

[1] - http://geomalgorithms.com/a01-_area.html#3D%20Polygons

[2] - Section 3.4.3 in Computational Geometry - An Introduction by Franco P. Preparata and Michael Ian Shamos
https://book4you.org/book/953554/9475f0

[3] - Section 4.2.2 in Computational Geometry in C by Joseph O'Rourke
https://book4you.org/book/437694/e8f3fe

[4] - https://link.springer.com/content/pdf/10.1007/BF02187740.pdf

[5]-https://www.researchgate.net/publication/2641780_The_QuickHull_Algorithm_for_Convex_Hulls

[6] - https://mathinsight.org/distance_point_plane

[7] - https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf

[8] - https://www.medien.ifi.lmu.de/lehre/ss10/ps/Ausarbeitung_Beispiel.pdf