



THE QUICKHULL ALGORITHM

ABOUT ME

- Carolina Herbster
- Software Developer at Instituto Atlântico
- MSc student at Universidade Federal do Ceará - UFC
- Loves Computer Graphics and crochet





TABLE OF CONTENTS

01 THE QUICKHULL ALGORITHM

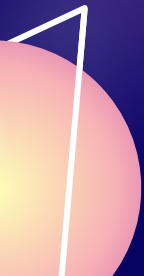

02 IMPLEMENTATION

03 RESOURCES

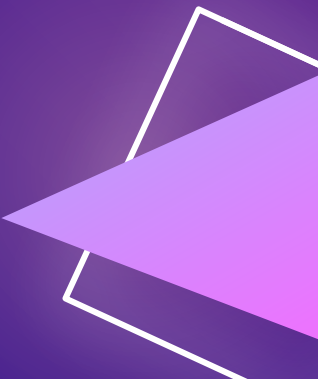

01

THE QUICKHULL ALGORITHM




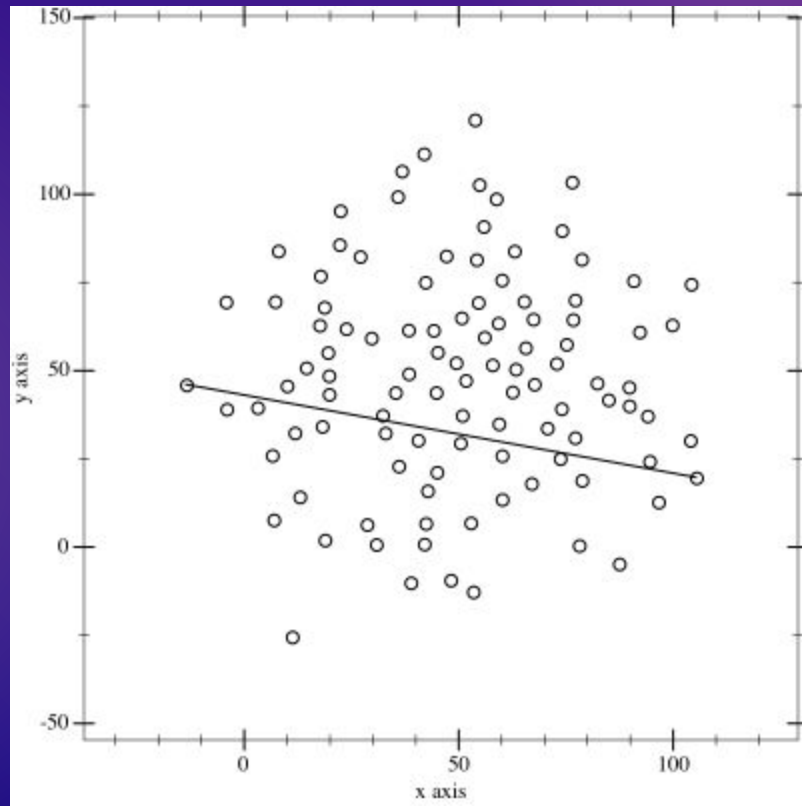


The Quickhull algorithm is a method of computing the **convex hull** of a finite set of points in n -dimensional space. It uses a divide-and-conquer approach similar to the quicksort sorting algorithm, from which its name is derived.



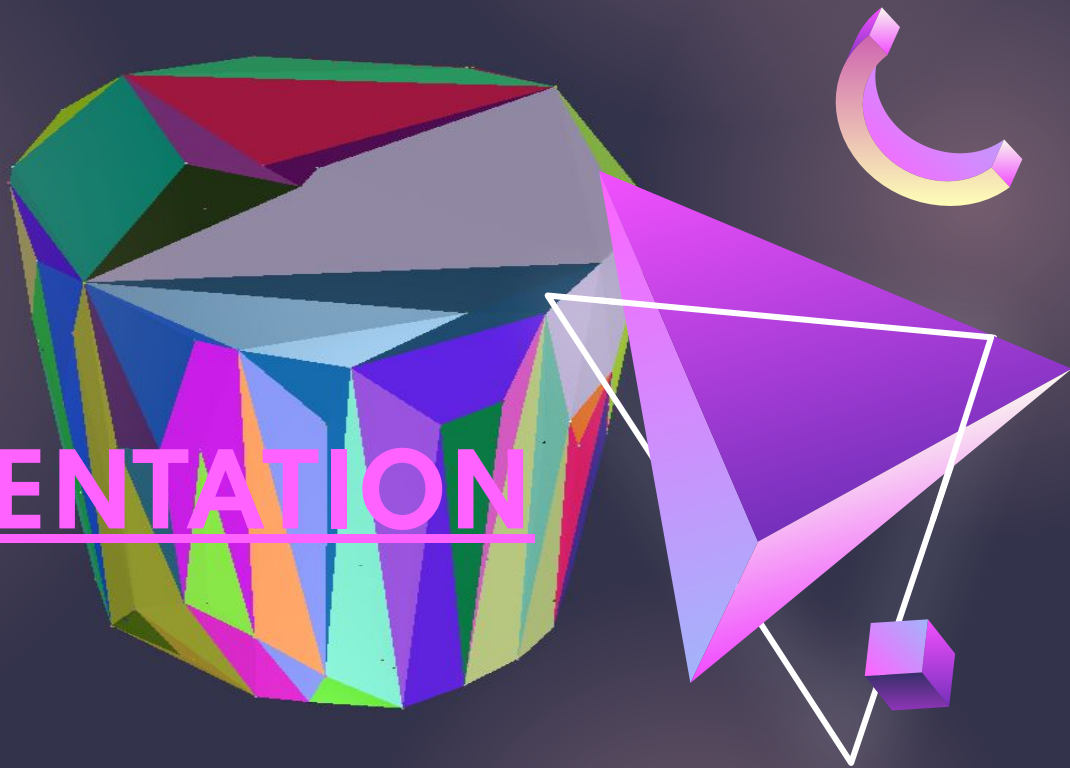
The worst case complexity for quickhull is $O(n \log(r))$, where n is the number of input points, and r is the number of processed points.







02

IMPLEMENTATION





The language used for the project was JavaScript. For the 3D rendering, the chosen library was Babylon.js. The code followed an OOP approach, with classes for computing the convex hull, and classes for the Face and Half-Edge data structure. After computing all the faces of the convex hull, the buildRenderableMesh method constructs a Babylon.js mesh from the list of faces, and builds an animation by fading in the face's material transparency at the step it was created, and fading out at the step it was deleted. The code is hosted on Github, and the page can be seen at any moment in Github Pages.



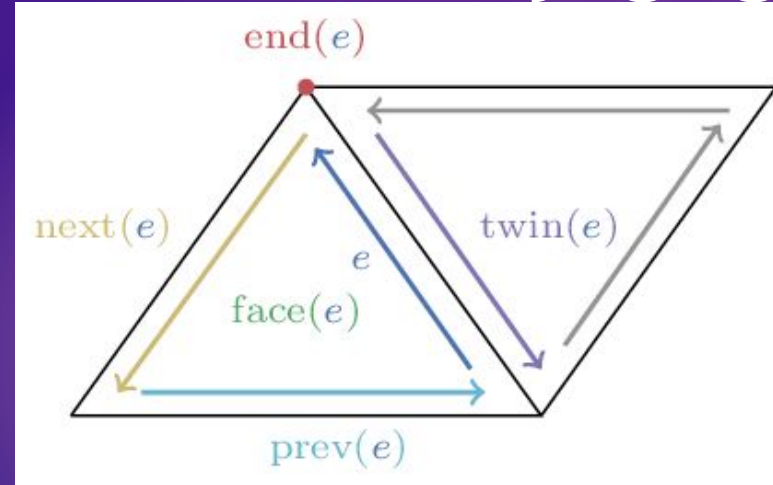
```
✓ Quickhull3D
  > DIM_TO_AXIS
    constructor
  > buildInitialSimplex
    addPointToFace
  > nextPointToAdd
    addPointToHull
    oppositeFaceDistance
  > resolveUnclaimedPoints
  > addNewFaces
  > addAdjoiningFace
    removePointFromFace
  > removeAllPointsFromFace
  > calculateHorizon
  > deleteFacePoints
  > build
  > buildRenderableMesh
```



babylon.js



The chosen data structure to represent face adjacency is the Half-Edge, which contains the face to which the edge belongs, the next and prev edges in CCW order, and the “twin” or “opposite” faces. Each Face was comprised of three half-edges.

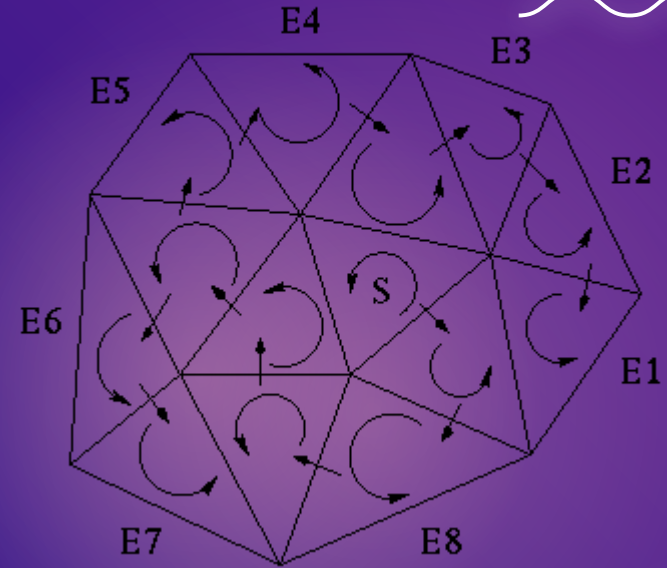





The algorithm execution can be resumed in three steps:

1. Find initial polyhedra/simplex
2. Find next point to be added to the hull
3. While there are still points to be added:
 - a. Find the “horizon”, or the boundary between visible and not visible faces
 - b. Delete all the faces inside the boundary and create new faces from that new point to the boundary edges
 - c. Repeat step 2

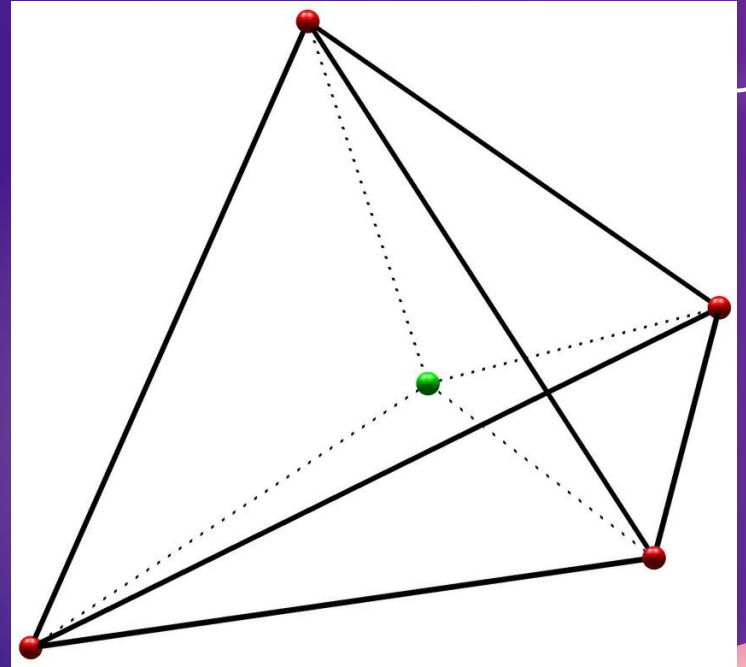
Horizon Contour



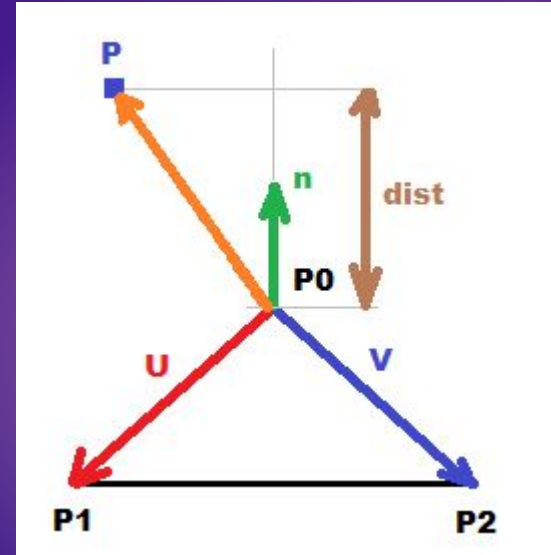


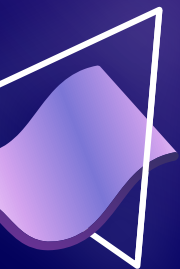
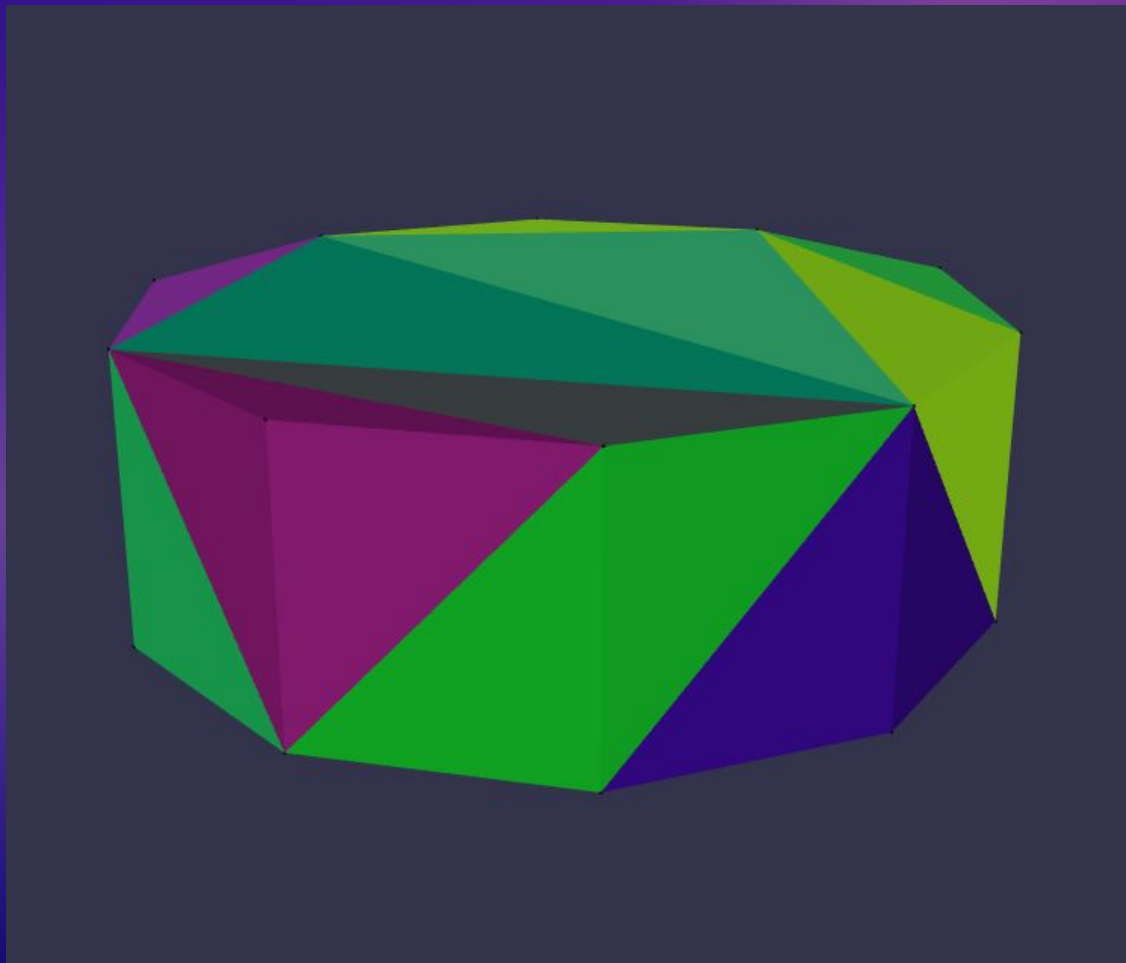
For the initial simplex stage, its execution can be resumed in the following steps:

1. Find the two points with the highest one-dimensional distance between them and form a line
2. Find the point with the highest distance to that line and form a plane with the previous two points
3. Find the point with the highest unsigned distance to the plane and form a tetrahedron with the four points



For finding the next point to add, you can iterate through every point that's not inside the hull, and chose the point with the highest distance to it.





	125 surf/250 vol points	1250 surf / 2500 vol points	12500 surf / 25000 vol points
CONSTRUCT HULL	16.69 ms	147.10 ms	8.43 s
BUILD RENDERABLE MESH	18.39 ms	150.38 ms	83.13 ms

Obs: Most of the time spent while building the mesh is creating separate submeshes so that each face's transparency can be separately controlled during the animation

RESOURCES

PRESENTATIONS

- Implementing Quickhull, Dirk Gregorius:
http://media.steampowered.com/apps/valve/2014/DirkGregorius_ImplementingQuickHull.pdf

SITES

- Quickhull3D:
<http://algorist.ru/maths/geom/convhull/qhull3d.php>
- How to generate equidistributed points on the surface of a sphere:
https://www.cmu.edu/biolphys/deserno/pdf/sphere_equi.pdf
- Evenly distributing n points on a sphere:
<https://stackoverflow.com/questions/9600801/evenly-distributing-n-points-on-a-sphere>