Proceedings of The National Conference
On Undergraduate Research (NCUR) 2017
University of Memphis
Memphis, Tennessee
April 6-8, 2017

Computing an Algorithm to Detect Three-Dimensional Objects in Space

Ken Ryumae
Computer Science Department
Western Kentucky University
Bowling Green, Kentucky 42101 USA

Faculty Advisor: Dr. Qi Li

Abstract

Previous work in two-dimensional (2D) data has been able to theoretically prove the existence of control points and vertices of data shaped in a polygon based on the theory of ellipsis. A control point is a point used to maximize the distance to other existing control points that have already been recursively defined. An example can be seen in driving related vision recognition work, where a computer drives a car. In this work, the computer has to identify stop signs and other signs in order to drive properly. This sign identification using vision recognition is accomplished by looking at individual pixels in an image and identifying the contrast of the white stop sign border or other sign border to that of the surrounding area. By finding the border, the computer is able to identify if the picture contains a stop sign by extracting 1 and 2-piece poly-lines to create an octagon, then comparing it to actual octagons for testing[1, 2]. Other signs would follow the similar vision recognition process. In a higher dimensional space, the theoretical justification of the consistency among control points and vertices is harder, as there are not only points limited to a single plane, but also points are found in multiple planes[3]. This means that there are over six times as many possible control point locations, due to the extra planes where the points can be found on. As such, the motivation behind this research is to find a new strategy for data clustering, based on the iterative extraction of control points. In this paper, I propose to create a program for generating a large set of simulated data to experimentally justify their consistency. By testing a large set of data, the program can then be improved in order to detect objects more consistently. The research consists of two parts. The first step in this research is generating three-dimensional (3D) data for use as test cases. To do this, we generate a set of 3D points in a pyramid shape in the beginning, but we eventually expand to other 3D shapes. A pyramid is used because of its unique slant which allows for easier 3D based shape recognition from various angles. By being able to find three control points on a single plane, the use of a generalized algorithm finds the next control point. We also add some randomness to generate point sets to test the robustness of control point extraction with respect to noise. Secondly, we implement an algorithm for control point extraction, then for verifying the correctness visually on larger sets. By doing so, we are able to correctly identify shapes on a 3D scale, and expand the algorithm to compensate for multidimensional objects and shapes.

Keywords: control point extraction, simulated data, multidimensional

1. Introduction

For human beings, an object identification is usually an easy task, as we have many years of experience in learning how to differentiate one thing from another. Even younger infants can recognize objects at 9 months old[4]. However, this is very difficult for a computer. While we take almost no time in recognizing a stop sign or another sign on the road from the surrounding area, it can take the computer significantly longer to distinguish the sign. The computer needs to identify possible lines and see whether those lines could possibly form an octagon for a U.S. stop sign, or

any other known shape, and then determine whether it thinks the given image contains a stop sign, speed sign, etc. or not.

Recently, computer object recognition has been making significant progress in identifying two-dimensional shapes. For identifying a shape like a triangle, the computer looks for two control points, and theoretically finds the third point. This is theoretical because the computer is not able to check if it is completely accurate. Instead, the computer would have to have prior information to determine whether the third point is correct. In a realistic, practical environment, computers are now capable of distinguishing stop-signs and other signs from the surrounding environment by analyzing images. By looking at the contrast between the white border of the sign and the surrounding area, the computer is able to distinguish whether a stop sign exists in the picture provided.

Other research has been done in a different way instead by looking for the exact number of sides in order to identify the given shape. If a shape has more than three sides, but less than five, the program is able to identify that the shape is a quadrilateral of some sorts. This kind of research is beginning to become prevalent as self-driving cars begin to come out and be tested.

My project takes this concept a step further by developing an algorithm in which computers distinguish objects based on a set of three-dimensional data points, allowing for further potential of identifying objects in multi-dimensions. This concept is quite important because ultimately these 3D object detections can be utilized in self-driving cars, automated warehouse equipment, underwater exploration, space missions and others to prevent collisions in real world surrounding environment.

2. Question

The purpose of this research was to see if the creation of an algorithm allowed a computer to identify 3D objects when given a set of data points. By having the computer able to identify 3D objects, technologies like self-driving cars and remote control mechanisms can become more aware of their surroundings, allowing them to be safer and more reliable.

3. Rationale

To begin, I created a pyramid with three vertices on the x, y, and z-axes. I used a triangular pyramid, tetrahedron, in these cases because objects like spheres required many data points in order to be accurate. Additionally, cubes were too basic of shapes, as they could be easily identified with simple point calculations. As such, a tetrahedron was used because of its unique slant where it was a better base case for testing, especially with visual inspection. By being able to find three control points on a single plane, a generalized algorithm was made to determine the next control point. I also added “noise” to the points to simulate more real life scenarios. The noise was added because real data would usually never be exact integers. This was caused by small variables when collecting data. Instead, the data was much more precise, which the noise was trying to replicate.

Secondly, I implemented an algorithm for control point extraction, and then verified the correctness on larger sets to visually verify the correctness by utilizing Wolfram Mathematica. By visualizing it in Wolfram Mathematica, a user would know that the points being generated and tested correctly created the objects that were being tested. This was also shown when no errors were produced when creating the planes for each side. Errors that might come up could alter the plane, morphing the intended shape into a completely different object. In doing this, I was able to correctly identify shapes on a 3D scale, and then expanded the algorithm to compensate for multidimensional objects and shapes.

4. Materials/Methods

As a base case, I first created a program in Java through Eclipse IDE to generate a list of points representing a pyramid with the tip at the origin and each of the vertices located on the three axes. Java and Eclipse were used because of my familiarity with both. Java is also platform-independent, meaning a program that was made on one computer could be easily transferred to another. Java programs are compiled and fast so that if I needed to test with much larger test data sets, it would be scalable. Eclipse IDE is popular and open-source, allowing it to be constantly

updated with the newest software. Its debugging function is user friendly and easy to use when trying to determine what a variable holds at any given time of execution. In my program, by using the vector equation of the plane, I found which plane was the base of the pyramid[5, 6, 7, 8]. With this information, I created the boundaries that were used in the for-loops, which looped the program repeatedly until the loop's exit condition was met. Then, by using three nested for-loops, I generated double type points that were within the pyramid ranging from zero to a varied maximum value, with a difference of one (Figure 1).

```
//Now we need to take the cross product of the two vectors.
double icoord = verty * vertz;
double jcoord = vertx * vertz;
double kcoord = vertx * verty;

int count = 1;

//Now, the equation of the plane can be formed by the expression:
//icoord(X-vertx) + jcoord * Y + kcoord * Z = 0
//This equation will be used, although X and Z will have points plugged in for it.
System.out.println("The equation of the plane with the three vertices is " + icoord + "(X - (" + vertx + ")) + " + jcoord + "Y + " + kcoord + "Z = 0");

//Next, we need to output the individual points to a new file.
for(int i = 0; i <= verty; i++){//for the y axis
    for(int j = 0; j <= vertx; j++){//for the x axis
        for(int k = 0; k <= (-icoord * (j - vertx) - jcoord * i)/kcoord; k++){//this will give us the z coord
            System.out.println("("+j+","+i+","+k+")");
            System.out.println(count++);
        }
    }
}
```

Figure 1. Java Code implementing the plane formula to find the coordinates of the points.

The points being generated utilize the vector equation of the plane for the boundaries of the point generation. It uses 3 for- loops to simulate 3D, x, y, and z plane coordinates[9].

After having a list of points generated, I wanted to simulate real data by adding random variability. The noise was added to simulate real data because in reality, the input data would not be exact integers, instead more random double values. The given input data would most likely be taken from a 3D software, where the data were the surface points of an object. To do this, I found a pseudorandom value through Java's Random class between -0.5 and 0.5. Java's Random class generated the pseudorandom values by finding a seed value based on the current time and running it through a simple formula to get the new random value within the given constraints. I made this modification to each point in the list to prevent any rounding from occurring (Figure 2). By having the list of points, I exported the list to a new text file which could then be put into Wolfram Mathematica for visualization[10].

```
{1.00000562992722,2.2587688046673224E-6,2.00000381373651},
{1.0000024744394385,7.1539073766884335E-6,3.0000029660288887},
{1.0000073202283242,8.397193068983692E-6,4.000006106588511},
{2.0000003612315824,4.2505614270322766E-6,4.1193821438873695E-6},
{2.0000032971142456,6.699282944425849E-6,1.0000010820729834},
{2.000009945558699,3.222692577294936E-6,2.0000018575022045},
{2.000008617068088,9.675430386435461E-6,3.000004063447407},
{3.0000066951282753,2.5381291663463217E-6,7.8307997603161936E-6},
{3.0000053597156175,7.706531496031072E-7,1.0000074888459578},
{3.0000025570665025,7.762990744445561E-6,2.0000060831002573},
{4.0000052625619675,5.962856964044252E-6,5.6966706669562436E-6},
{4.000005261510153,7.251457068245327E-6,1.000008544609132},
{5.000006457813032,5.634522843053002E-6,7.1290756389861945E-6},
{5.475769486125052E-6,1.000008237206831,9.86942820994599E-6},
{1.8092982732368446E-6,1.0000099226362522,1.0000047294154188},
{9.378704574205895E-6,1.000007045515918,2.0000021825845673},
```

Figure 2: Example of points after randomization occurs.

Once I started my implementation for graphs, I took the randomized points, rounded them, and then found the exact points, which I then graphed on Wolfram Mathematica. Some numbers shown have an E, meaning exponent. In many cases, they are E-6, showing that the number must be multiplied by 10^-6 in order to get the actual number.

After this, I imported the list into Wolfram Mathematica, where I did my visualization. I chose Wolfram Mathematica for data visualization due to its wide range of functionality for further analysis. I plotted each individual point in Wolfram Mathematica to generate a 3D list plot (Figure 3). By plotting the points in Wolfram Mathematica, the program successfully created each plane, which would have been unsuccessful if the points were not correctly generated.

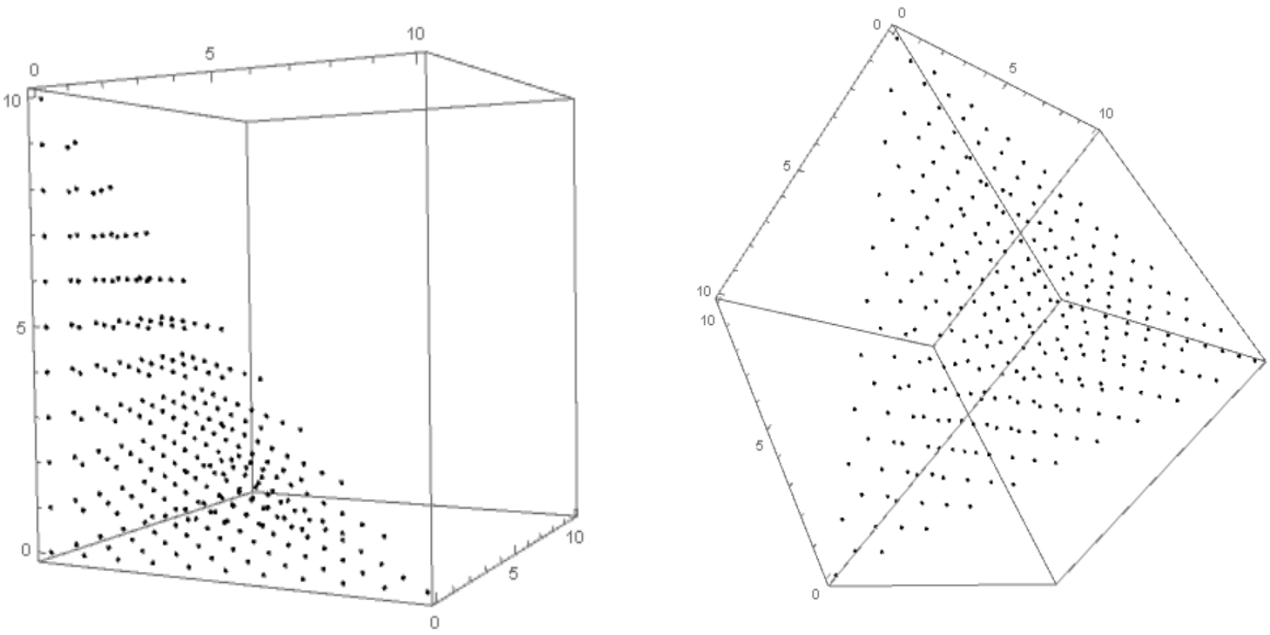

Figure 3. Graphs of points when visualized in Wolfram Mathematica in two different angles.

The points being graphed simulated a 10 by 10 by 10 tetrahedron. This plot displayed each individual point within a cube, and allowed the user to interactively change the orientation of the cube. I then utilized a Wolfram

Mathematica function, `Mesh -> all` to convert the points into solid planes for easier identification (Figure 4).

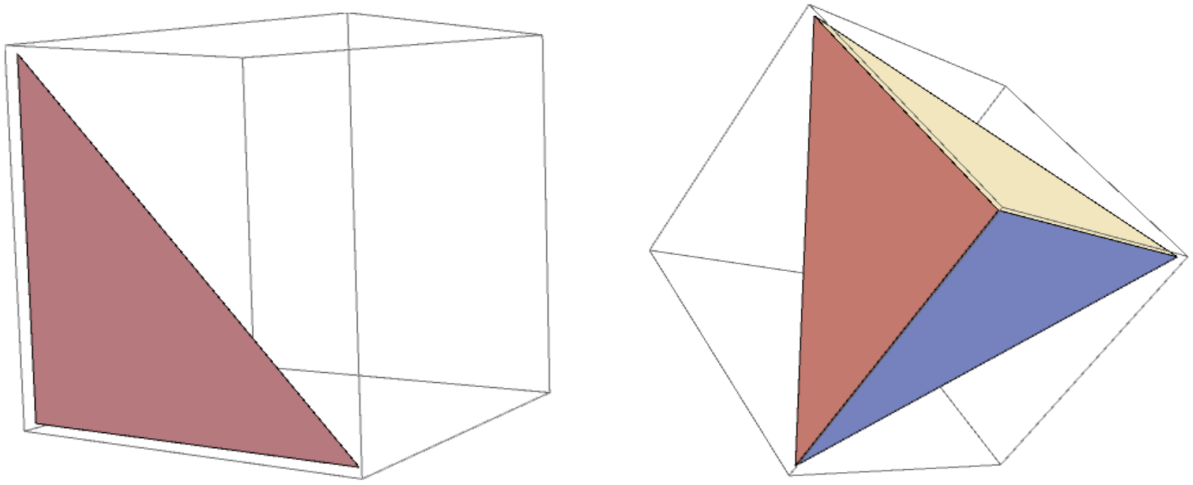

Figure 4. The planes of the points after being plotted in Wolfram Mathematica from two different angles.

Each x, y and z axis is colored differently, and it is clear visually to see the figure is in fact a pyramid shape. In addition to this, I tested to see how big of the noise I could make without breaking the program. I found that noise values that exceeded .5 would begin to cause errors within the program because then the program would begin to detect overlapping points, or points that did not necessarily exist.

5. Results

Only "accuracy" was tested for results. Originally, time was also going to be tested, but with the small number of data points, there was no need for time to be tested, as the time taken to process all information was all about the same, differing only by milliseconds. In regards to accuracy, I tested for the accuracy of the points, by checking the points along with the actual formula of the plane, which differed from test to test (Table 1). When testing with negative numbers, the program produced an error because Java did not like negative numbers when looking at element positions, and as such, it returned an out of bounds exception error. To correct this, the user would have to change the location of the origin in respect to the object being analyzed.

Table 1. Results after testing the point generating program.

(x vertices, y vertices, z vertices)	Actual Number of Points	Program's Findings	Difference	% Accuracy
(1, 1, 1)	4	4	0	100
(5, 5, 5)	56	56	0	100
(10, 10, 10)	286	286	0	100
(15, 15, 15)	816	816	0	100

The results showed that the program was able to correctly identify the number of points that were contained within the tetrahedron. We used these points when looking at the visualization of the tetrahedrons. The points represented the vertices, and in the example of (1, 1, 1), the actual points being analyzed were the vertices at (0, 0, 0), (1, 0, 0), (0, 1, 0), and (0, 0, 1). These unit points contributed to create the tetrahedron, with each vertex acting as one of the tips of the tetrahedron. For the (5, 5, 5), there were 56 unit points that contributed to create the the entire shape of the tetrahedron. In this case, a unit point was a point that only contained integer coordinates.

6. Conclusion

When developing the equations to find the points that would contain a pyramid, the equation was extremely accurate, finding the correct values 100% of the time. This was checked alongside a verified program which was able to output the number of unit points within the given vertices. When testing in the beginning, time was to be tested alongside accuracy; however, the time differences produced from working with the data sets were so insignificant that there was no need to test for the speed. As such, only the accuracy was tested. After plotting the points in Wolfram Mathematica, I was able to visually verify that the points being plotted were getting the noise taken out, thus allowing for the unit points to be plotted in their specific shape.

7. Future Directions

In the future, I plan to improve the algorithm so that given a set of data points, the algorithm will be able to return what shape the object has. Furthermore, I plan to modularize the program so that it can be used in multi-dimensions (3D+). One possible application can be seen with self-driving cars, allowing the car itself to get a better idea of the environment that it is driving in as well as mapping the surrounding environment more efficiently in real time to prevent collisions. This method could also be used in outer space, or in deep-sea exploration, where we do not have a good idea of the terrain we are entering. Utilizing something like this would allow for better terrain-mapping for human visualization and could become triggers for subsequent actions to take to intelligently adapt to surrounding environment.

8. Acknowledgements

The author wishes to express his appreciation to Western Kentucky University for allowing him to do this research on their campus, as well as Dr. Qi Li for being his mentor throughout this process.

9. References

1. A geometric framework for stop sign detection - IEEE Xplore Document. Accessed August 01, 2017.http://ieeexplore.ieee.org/document/7230403/.
2. Real-time recognition of U.S. speed signs - IEEE Xplore Document. Accessed August 01, 2017. http://ieeexplore.ieee.org/document/4621282/.
3. Zhang, Zhengyou, and Olivier D. Faugeras. "Finding clusters and planes from 3D line segments with application to 3D motion determination." SpringerLink. May 19, 1992. Accessed August 01, 2017. https://link.springer.com/chapter/10.1007/3-540-55426-2_26.
4. "Babies recognize real-life objects from pictures as early as nine months, psychologists discover" ScienceDaily. Accessed August 01, 2017. https://www.sciencedaily.com/releases/2014/04/140429205733.htm
5. "Calculate distance in 3D space." Mathematics Stack Exchange. Accessed August 01, 2017. https://math.stackexchange.com/questions/42640/calculate-distance-in-3d-space.
6. "Cross Products." Calculus II - Cross Product. Accessed August 01, 2017. http://tutorial.math.lamar.edu/Classes/CalcII/CrossProduct.aspx.
7. "Equations of Planes." Calculus III - Equations of Planes. Accessed August 01, 2017. http://tutorial.math.lamar.edu/Classes/CalcIII/EqnsOfPlanes.aspx.
8. "Plane." From Wolfram MathWorld. Accessed August 01, 2017. http://mathworld.wolfram.com/Plane.html.
9. "Creating a triangle with for loops." Java - Creating a triangle with for loops - Stack Overflow. Accessed August 01, 2017. https://stackoverflow.com/questions/11409621/creating-a-triangle-with-for-loops.
10. "How do I create a file and write to it in Java?" How do I create a file and write to it in Java? - Stack Overflow. Accessed August 01, 2017. https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java.