Slicing Algorithms for 3D-Printing

Fabian Schurig, *Student, B.Sc. Computer Science*
Technische Universität München
E-mail: f.schurig@tum.de

Abstract—3D printing is an upcoming trend based on a 3D modeled object which needs to be sliced and converted into instructions for the extruder. This is being done by using different algorithms in staircases.

Keywords—3D Printing, Algorithms, Slicing

1 Introduction

3D Printing became increasingly popular during the past few years. By now, 3D Printers are available in all colors and shapes. They are used for a huge field of application. The process of 3D printing started off with Stereolithography (SL), in which Liquid Resin got solidified with a laser beam. To name a few others of the most popular processes, there are printing with digital light processing (DLP), Laser Sintering, Selective Deposition Lamination (SDL) and Fused Deposition Modeling (FDM) *(3D Printing Industry, 2012 - 2015)*. This paper is focusing on FDM because it is very office- and user-friendly and the cleanest way to print an object. Another benefit of FDM is the use of eco-friendly and mechanically stable, printable material *(Stratasys Ltd. 2014)*. Currently, there is a variety of different materials such as Polylactic acid (PLA), Acrylnitril-Butadien-Styrol (ABS) for standard use and specials like flexible Nylon, XT-CF20 (Carbon), CopperFill or foods *(ColorFabb 2015; Wiggers 2015)* which are possible to be printed. Nowadays the whole process of printing is steadily being improved and therefore made easier for customers. Meanwhile big databases like Thingiverse which store 3D printing models have been newly-established. Shared printer platforms such as 3D Hubs are another possible way for people without a 3D printer to print their models.

Main features of the process are generating a digital model from a physical model and converting it into an STL file format. By the use of slicing algorithms it is possible to generate a G-code file in order to set up the properties of the printer. This machine language can be interpreted by the printer.

1.1 Computer Aided Design (CAD)

Before slicing it is important to consider the way a digital model is extracted as an STL file. There are many different possibilities to model objects, such as 3D scanning, taking measurements for a full-scale replica or building it from scratch. Furthermore by taking a video or a series of pictures all around an object it is feasible to feed those into a program like Autodesk Memento and create a 3D model. The easiest way is to download existing models from online platforms as mentioned above, i.e. GrabCAD or similar ones. As a consequence that all of these methods are making use of a computer, which leads to the fact that this entire process is called Computer Aided Design (CAD) *(Narayan 2008)*.

1.2 STL format

STL means STereoLithography and can be exported into most CAD software suites, like Autodesk Fusion 360. For this reason it has become the acronym "Standard Tessellation Language" *(Grimm 2004)*. The STL format only utilizes the three-dimensional description of the surface geometry without generating non relevant information for printing like texture or color, leading to the popularity within the community

(Chua, Leong and Lim 2003). Each triangle, which represents the surface, is characterized by three vertices and the related unit normal *(fabbers 1999)*. Due to the fact that a prototypical 3D model is closed, also called waterproofed, each vertex is part of three or even more triangles. These redundant vertices are memory expensive even when they are stored in ASCII representation. Such being the case, a more compact and common file can be produced with the equivalent binary representation *(Burns 1993)*.

1.3 G-code

Most 3D printers have three axes, an extruder, a hot end and a heated bed. Therefore it has four motors and two heated elements to control. With an STL file the printer is not yet able to control the requested moves. Hence, a G-code is introduced to transfer the favored movements to the tools of the printer. The G programming language provides every information the printer needs to know to print the desired object, including the speed and path of moving the axes. In addition it has to supply information about the temperature of the heated elements and speed of extruding the filament.

The process of slicing on the basis of a rim is demonstrated in the following.

2 Slicing Algorithms

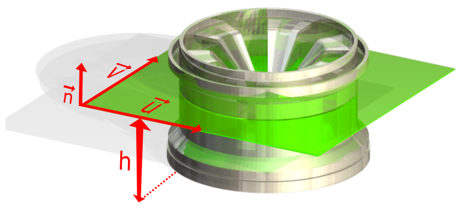

Fig. 1. A rim is cut by a green intersection-plane.

By name, the base algorithm is all about cutting triangles with planes. In FDM, the STL file gives out the required triangles in order to apply the melted filament layer by layer onto the printing area. For that reason there is a plethora of cutting planes through the object parallel to the ground plane with varying z values (height). The intersection of such a plane with the object is the resulting layer to print. Plenty of layers printed from the bottom to the top conclude in the item.

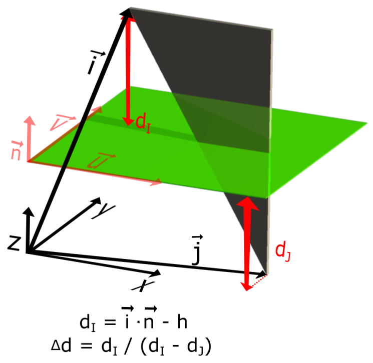
$\vec{i}$
$\vec{v}$
$\vec{n}$
d_I
$\vec{u}$
y
z
$\vec{j}$
d_J
x
$d_I = \vec{i} \cdot \vec{n} - h$
$\Delta d = d_I / (d_I - d_J)$

Fig. 2. A triangle cut by an intersection plane (green) with its position vectors i and j

Each intersection plane is spanned by the vectors u and v and cuts various triangles. Supplementary the height h of a plane from the bottom and the normal vector n is known. With the help of the given vertices of the triangles it is possible to calculate each distance from the vertices to the cutting plane. The edge vector e can be defined with respectively two of the three position vectors. Implementing the hessian normal form and basic vector operations, it is possible to calculate the intersections for every edge based on the previous position vectors and filed distances to the plane with the Intercept theorem.

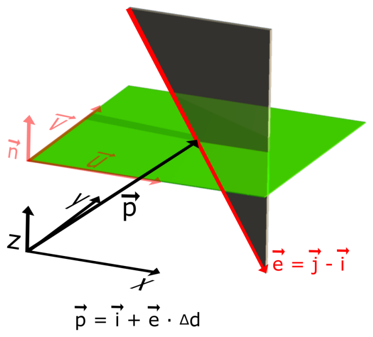
$\vec{v}$
$\vec{n}$
$\vec{u}$
y
$\vec{p}$
z
x
$\vec{e} = \vec{j} - \vec{i}$
$\vec{p} = \vec{i} + \vec{e} \cdot \Delta d$

Fig. 3. Generating an edge vector e and a new position vector p on the slice.

By doing so for all three edges of a triangle the result will look like the following. Firstly there could be a line defined by two new position vectors p and q, or the plane is cutting the triangle exactly in one vertex. Finally there is the possibility that the plane does not cut the triangle at all. There are also some special cases that the edge can slice the plane i.e. it is parallel to the normal vector n. Such can be calculated much faster. Thus, each intersection for every triangle of the object with all planes has to be reckoned. This can be done with the following slicing algorithms.

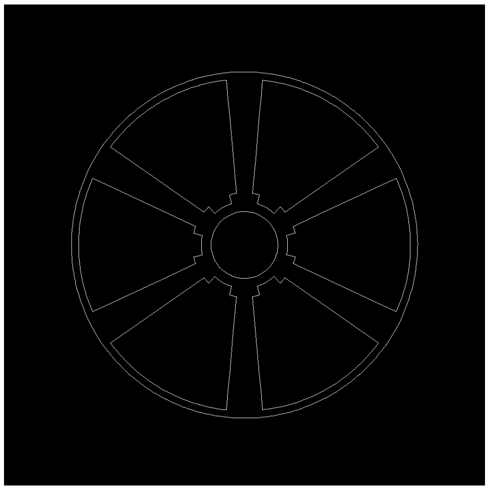

Fig. 4. The sliced rim from the green intersection layer above.

Algorithm 1 Trivial Algorithm

```
1:  for each plane p in cutting planes do
2:     init slice of p
3:     for each triangle t in triangle mesh do
4:        if t intersects p then
5:           calculate intersection points
6:        end if
7:        if intersection points == 2 then
8:           add new line to slice of p
9:        end if
10:    end for
11: end for
```

2.1 Trivial Slicing

A trivial method is traversing through all triangles in the given triangle mesh and calculating the intersections for every plane. The slice of every layer has to be filed. Checking and calculating intersection points for every triangle even if those are not subtended by the cutting-plane is a very slow method especially for files with lashings of triangles. Some more efficient algorithms are described in the following.

2.2 Sweep Plane Slicing

A different way of slicing has been introduced by McMains and Sequin *(McMains and Sequin 1999)*. Before operating their algorithm a circular linked list with pointers to the edges is kept within a standard status structure. With this it is possible to store topological data. ″Edge uses″ describes a structure which represents triangle boundaries. In this algorithm it is just a ″virtual″ plane sweeping from bottom to top. An event is triggered each time the plane reaches a vertex or a z-coordinate of a slice. The event could be a rearrangement of the status data structure if a vertex is reached or a snapshot of all cut edges. *(Gregori, Volpato, Minetto and da Silva 2014)*

2.3 Triangle Grouping

Another algorithm is based on grouping triangles by z-coordinates. *(Tata, Fadel, Bagchi and Aziz 1998)* In this method triangles are grouped by their minimum z_{min} and maximum z_{max} values. Triangles with the same z_{min} value can be clustered together in buckets. Each bucket can have sub-buckets grouped by z_{max} values of the triangles. In result one gets a bucket for each z_{min} value and this bucket contains sub-buckets with different z_{max} values. Before processing the triangles another problem arises which is sorting them in optimal time. For this reason all triangles can be filed in a binary search tree or with bucket sort. The triangles are processed by a key characteristic identifier. The real implementation details are not shared. By using a structure like a binary search tree and knowing the z-coordinates from each intersection plane it is possible to get the result of the triangles which have been cut from a plane even faster. To calculate the slices it is feasible to use the same method as mentioned above in the trivial algorithm. Due to the fact that the triangles have to be sorted just once it is slower but also has the advantage of a faster contour assembly and slicing the item faster a second time with another different distance of z-coordinates. *(Gregori, Volpato, Minetto and da Silva, 2014)*

2.4 Incremental Slicing

Incremental slicing is an asymptotically optimal algorithm in which each triangle is represented by an interval from its $z_{\min}$ to $z_{\max}$ values. Instead of having a triangle mesh one uses an interval mesh in this case. The intervals can also be stored in an interval tree *(Berg, Cheong, Kreveld and Overmars 2008)*. The z-coordinates of each plane from bottom to top are filed in an array. The triangle intersections can be solved with the stabbing problem by using the intervals as input. *(Edelsbrunner 1980)* When performing a cut it is possible to get an exact interval of all cut triangles. *(Gregori, Volpato, Minetto and da Silva 2014)*

3 Infills

Unfortunately only the outline of the object is available and needs to be filled for stability reasons. This can be done with the Vattis clipping algorithm and the polygon offsetting. For a fast solution to this problem Clipper and Boost.Geometry are the commonly used libraries. There are also many different kinds of infills. Each one has its advantages and disadvantages based on stability, flexibility and efficiency just to name a small list. Commonly used infills are *Honeycomb* for items which need to be stable and *Rectlinear* for a faster print without losing a lot of stability. Another fact is that typically each infill is rotated 90 degrees per layer to gain more solidity. For the very bottom and top layers a three layer solid infill is recommended just as three perimeters for the outline.

4 Support Material

Overhangs such as in bridges are another difficulty which have to be considered when printing with FDM. The support material generation depends on the printer and layer resolution. The algorithm superimposes the layers in order to detect the top view outline and hidden edges of the object. If there is a layer exceeding the previous layer the algorithm constructs a projection of the new outline underneath and generates a defined support structure.

After the contour of each layer has been computed, the G-code can be generated.

5 Conclusion

With todays possibility of 3D printing many useful things like hand prostheses can be created and prototyped faster with cheaper production processes. For this process many staircases must be overcome. Starting with generation of a computer from a physical model, getting over to a STL file which can be processed by algorithms to get a G-code for the printers' instructions.

Free widespread open source slicing programms are Slic3r and Cura which are running with modified algorithms described above. Such algorithms can be trivial and slow or more efficient but complex. Because of the diversity of settings in such programs and their dependend printing results, staircases are difficult to be skipped. A simple documentated Java implementation for a trivial slicing algorithm with use of the hessian normal form is in the appendix of this paper.

In future there will be printers with a webfrontend and preconfigured slicing settings. These printers will be similar utilized as paper printers but instead of uploading a Document-File (PDF) they need a STL file. Also thinkable are printers with integrated laser scanners to copy 3D objects.

References

[1] H. Edelsbrunner, *Dynamic Data Structures for orthogonal intersection queries. [Mit Fig.]* , Inst. f. Informationsverarbeitung, TU Graz, 1980

[2] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[3] K. Tata, G. Fadel, A. Bagchi, and N. Aziz, *Efficient slicing for layered manufacturing*, Rapid Prototyping Journal ,vol. 4, no. 4, pp. 151167, 1998

[4] S. McMains and C. Sequin, *A coherent sweep plane slicer for layered manufacturing, in Proceedings of the fifth ACM symposium on Solid modeling and applications* - SMA 99. New York, New York, USA: ACM Press, 1999, pp. 285295.

[5] Rodrigo Gregori, Neri Volpato, Rodrigo Minetto and Murilo da Silva (2014). *Slicing Triangle Meshes: An Asymptotically Optimal Algorithm. p. 2 Available from: http://www.dainf.ct.utfpr.edu.br/ murilo/public/slicing.pdf* [Accessed 2015]

[6] Stratasys Ltd., 2014. *FDM Technology. , About Fused Deposition Modeling. Available from: http://www.stratasys.com/3d-printers/technologies/fdm-technology* [Accessed 2015].

[7] 3D Printing Industry, 2012 - 2015. *3D Printing Processes: Free Beginner's Guide 3D Printing Industry. 3D Printing Industry. Available from: http://3dprintingindustry.com/3d-printing-basics-free-beginners-guide/processes/* [Accessed 2015].

[8] Narayan, K. Lalit (2008). *Computer Aided Design and Manufacturing. New Delhi: Prentice Hall of India. p. 3. ISBN 812033342X.*

[9] Gonen, R., 2013. *Stereolithography (3D Printing) Algorithms and Thoughts. Raveh Gonens Blog. Available from: https://ravehgonen.wordpress.com/2013/02/19/stereolithography-3d-printing-algorithms-and-thoughts/* [Accessed 2015].

[10] ColorFabb, 2015. *SPECIALS FILAMENTS. Available from: http://colorfabb.com/specials* [Accessed 2015].

[11] Wiggers, K., 2015. *Why 3D food printing is more than just a novelty; it's the future of food. Digital Trends. Available from: http://www.digitaltrends.com/cool-tech/3d-food-printers-how-they-could-change-what-you-eat/* [Accessed 2015].

[12] Chua, C. K; Leong, K. F.; Lim, C. S. (2003), *Rapid Prototyping: Principles and Applications (2nd ed.), World Scientific Publishing Co, ISBN 981-238-117-1 Chapter 6, Rapid Prototyping Formats. Page 237, "The STL (STeroLithography) file, as the de facto standard, has been used in many, if not all, rapid prototyping systems." Section 6.2 STL File Problems. Section 6.4 STL File Repair.*

[13] fabbers, 1999. *The StL Format. Available from: http://www.fabbers.com/tech/stl_format* [Accessed 2015].

[14] Grimm, Todd (2004), *User's Guide to Rapid Prototyping, Society of Manufacturing Engineers, p. 55, ISBN 0-87263-697-6. Many names are used for the format: for example, "standard triangle language", "stereolithography language", and "stereolithography tesselation language". Page 55 states, "Chuck Hull, the inventor of stereolithography and 3D Systems' founder, reports that the file extension is for stereolithography."*

[15] Burns, Marshall (1993). *Automated Fabrication.* Prentice Hall. ISBN 978-0-13-119462-5.

[16] Daniel Norée (2013) *OpenRC Truggy - 3D Model. Available from: http://www.thingiverse.com/thing:42198* [Accessed 2015].

[17] Fabian Schurig (2015) *Designs of 3D models and parts for the OpenRC Truggy. Available from: http://www.thingiverse.com/Bitfrost/designs* [Accessed 2015].

Appendix

Listing 1. Main.java

```java
package slicer;

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;

public class Main {

  public static void main(String[] args) {
    // Import Ascii stl file, src= http://www.thingiverse.com/thing:39751
    ImportSTL stl = new ImportSTL("OpenRC_Truggy_Rim.stl");
    try {
      stl.readFile();
    } catch (IOException e) {
      e.printStackTrace();
    }

    Plane p = new Plane(new Vector(0, 0, 1)); // z axis
    float objHeigth = 50; // in mm
    int counter = 0;

    // Trivial Algorithm
    for (float cutter = 0; cutter < objHeigth; cutter += 0.2f) {
      counter++;
      p.setDistance(cutter);

      ArrayList<Vector> intersectPoints = new ArrayList<Vector>();

      // Position Vector of the plane
      Vector e = p.normal.mul(p.height);

      // Draw new Image
      BufferedImage img = new BufferedImage(1000, 1000,
          BufferedImage.TYPE_INT_RGB);
      Graphics2D g2d = img.createGraphics();

      for (int i = 0; i < stl.triangleMesh.size(); i++) {

        ArrayList<Vector> ints = stl.triangleMesh.get(i)
            .intersectPlane(p);

        float[] x = new float[ints.size()];
        float[] y = new float[ints.size()];

        for (int j = 0; j < ints.size(); j++) {
          // calc position vector of poins in plane via hessian
          Vector s = e.sub(ints.get(j));
          x[j] = p.getV().scalarProduct(s);
          y[j] = p.getU().scalarProduct(s);
        }

        // draw resulting line in image
        if (x.length > 1) {
          g2d.drawLine((int) (x[0] * 10 + img.getWidth() / 2),
              (int) (y[0] * 10 + img.getHeight() / 2),
              (int) (x[1] * 10 + img.getWidth() / 2),
              (int) (y[1] * 10 + img.getHeight() / 2));
        }
```

```
            intersectPoints.addAll(ints);
         }
         System.out.println("#Intersects␣=␣" + intersectPoints.size());
         g2d.dispose();
         // write output image in file
         try {
            ImageIO.write(img, "png",
                  new File("render/" + counter + ".png"));
         } catch (IOException ex) {
            ex.printStackTrace();
         }
      }
   }
}
```

Listing 2. ImportSTL.java

```
package slicer;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class ImportSTL {
   public ArrayList<Triangle> triangleMesh;
   public String fileName;
   public String solid;
   private Scanner sc;
   private Triangle tri;

   public ImportSTL(String fileName) {
      super();
      this.fileName = fileName;
      triangleMesh = new ArrayList<Triangle>();
   }

   /**
    * Reads an ASCII STL File
    * @throws IOException
    */
   public void readFile() throws IOException {
      sc = new Scanner(new FileReader(fileName));
      tri = null;
      Vector normal = null;

      while (sc.hasNext()) {
         String next = sc.next();
         // System.out.println(next);
         if (next.equals("solid") && sc.hasNext()) {
            solid = next;
         }
         if (next.equals("facet")) {
            tri = null;
         }
         if (next.equals("normal")) {
            float x = readNumber(sc.next());
            float y = readNumber(sc.next());
            float z = readNumber(sc.next());
            normal = new Vector(x, y, z);
            // System.out.println(normal.toString());
         }
         if (next.equals("vertex")) {
            float x = readNumber(sc.next());
            float y = readNumber(sc.next());
            float z = readNumber(sc.next());
            Vector vectorX = new Vector(x, y, z);
            // System.out.println(vectorX.toString());
            sc.next();
            x = readNumber(sc.next());
            y = readNumber(sc.next());
            z = readNumber(sc.next());
            Vector vectorY = new Vector(x, y, z);
            // System.out.println(vectorY.toString());
            sc.next();
            x = readNumber(sc.next());
            y = readNumber(sc.next());
            z = readNumber(sc.next());
            Vector vectorZ = new Vector(x, y, z);
            // System.out.println(vectorZ.toString());
            triangleMesh
                .add(new Triangle(vectorX, vectorY, vectorZ, normal));
            // System.out.println(triangleMesh.toString());
        }
        if (next.equals("endfacet")) {

        }

    }

    System.out.println(triangleMesh.toString());

  }

  /**
   * reads the values of a string
   * @param s string to get valueOf
   * @return
   */
  public float readNumber(String s) {
    return (float) (Double.valueOf(s)).doubleValue();
  }

}
```

Listing 3. Line.java

```
package slicer;

public class Line {
  public Vector[] vec = new Vector[2];

  /**
   * Intersection Line of Triangle with a Plane
   */
  public Line(Vector start, Vector end) {
    super();
    this.vec[0] = start;
    this.vec[1] = end;
  }

}
```

Listing 4. Plane.java

```
package slicer;

public class Plane {
  public float height; // height to origin
  public Vector normal;

  /**
```

```
     * Represents a Cutting Plane
     * @param normal Normal of the plane. Most times in z direction.
     */
    public Plane(Vector normal) {
      super();
      this.height = 0;
      this.normal = normal;
    }

    /**
     * returns a support vector u of the plane
     * @return
     */
    public Vector getU() {
      Vector u = new Vector(−normal.y, normal.x, 0);
      if (u.isNull()) {
        u = new Vector(−normal.z, 0, normal.x);
      }
      return u;
    }

    /**
     * returns a support vector v of the plane
     * @return
     */
    public Vector getV() {
      Vector v = new Vector(0, −normal.z, normal.y);
      if (v.isNull()) {
        v = new Vector(−normal.z, 0, normal.x);
      }
      return v;
    }

    public float getDistance() {
      return height;
    }

    public void setDistance(float distance) {
      this.height = distance;
    }

    public Vector getNormal() {
      return normal;
    }

    public void setNormal(Vector normal) {
      this.normal = normal;
    }

    /**
     * returns the distance of a vertex to the plane
     * @param vertex
     * @return distance
     */
    public float distanceToPoint(Vector vertex) {
      return vertex.scalarProduct(normal) − height;
    }

}
```

Listing 5. Triangle.java

```
package slicer;

import java.util.ArrayList;
import java.util.Arrays;
```

```

public class Triangle {
  public Vector[] corner = new Vector[3];
  public Vector normal;

  /**
   * Represents a triangle.
   * @param one first edge
   * @param two second edge
   * @param three third edge
   * @param normal normal of the triangle
   */
  public Triangle(Vector one, Vector two, Vector three, Vector normal) {
    super();
    this.corner[0] = one;
    this.corner[1] = two;
    this.corner[2] = three;
    this.normal = normal;
  }

  /**
   * Subtracts a vector s from the triangles corners.
   * @param s vector which will be subtracted
   * @return new Triangle
   */
  public Triangle sub(Vector s) {
    corner[0] = corner[0].sub(s);
    corner[1] = corner[1].sub(s);
    corner[2] = corner[2].sub(s);
    return this;
  }

  public boolean checkVector(Vector v) {
    if (v.z >= 0 && v.z <= 1) {
      return true;
    }
    return false;

  }

  /**
   * Returns a new position vector of the cutting point with the plane.
   * @param p slicing plane
   * @param i corner of a triangle
   * @param j corner of a triangle
   * @return position vector of the cutting point on the plane
   */
  public Vector combination(Plane p, int i, int j) {

    float distanceI = p.distanceToPoint(corner[i]);
    float distanceJ = p.distanceToPoint(corner[j]);

    if (distanceI * distanceJ < 0) {
      float factor = distanceI / (distanceI - distanceJ);
      //System.out.println(factor);
      Vector edge = Vector.sub(corner[j], corner[i]);
      return corner[i].add(edge.mul(factor));
    } else if (distanceI == 0) {
      return corner[i];
    } else if (distanceJ == 0) {
      return corner[j];
    }
    return null;
  }

```

```java
  /**
   * Cuts a triangle with a plane.
   * @param p slicing plane
   * @return intersection points with plane
   */
  public ArrayList<Vector> intersectPlane(Plane p) {

    ArrayList<Vector> result = new ArrayList<Vector>();

    Vector[] tmp = new Vector[3];

    tmp[0] = combination(p, 0, 1);
    tmp[1] = combination(p, 0, 2);
    tmp[2] = combination(p, 1, 2);

    for (int i = 0; i < 3; i++) {
      if (tmp[i] != null) {
        result.add(tmp[i]);
      }
    }
    return result;
  }

  @Override
  public String toString() {
    return "Triangle [corner=" + Arrays.toString(corner) + ", normal="
        + normal + "]";
  }

}
```

Listing 6. Vector.java

```java
package slicer;

public class Vector {
  public float x, y, z;

  /**
   * Represents a 3 dimensional Vector.
   * @param x coordinate
   * @param y coordinate
   * @param z coordinate
   */
  public Vector(float x, float y, float z) {
    super();
    this.x = x;
    this.y = y;
    this.z = z;
  }

  /**
   * Calculates the scalar with a vector p
   * @param p
   * @return scalarProduct
   */
  public float scalarProduct(Vector p) {
    return this.x * p.x + this.y * p.y + this.z * p.z;
  }

  /**
   * length of a vector
   * @return
   */
  public float length() {
    return (float) Math.sqrt(x * x + y * y + z * z);
}

/**
 * checks if it is a null vector
 * @return
 */
public boolean isNull() {
  if (this.x == 0 && this.y == 0 && this.z == 0) {
    return true;
  }
  return false;
}

/**
 * vector multiplication with a factor
 * @param factor
 * @return
 */
public Vector mul(float factor) {
  return new Vector(this.x * factor, this.y * factor, this.z * factor);
}

/**
 * vector division with a divisor
 * @param divisor
 * @return
 */
public Vector div(float divisor) {
  return new Vector(this.x / divisor, this.y / divisor, this.z / divisor);
}

/**
 * vector addition with another vector
 * @param v
 * @return
 */
public Vector add(Vector v) {
  return new Vector(this.x + v.x, this.y + v.y, this.z + v.z);
}

/**
 * vector subtraction with another vector
 * @param v
 * @return
 */
public Vector sub(Vector v) {
  return new Vector(this.x - v.x, this.y - v.y, this.z - v.z);
}

/**
 * static vector addition of two vectors a and b
 * @param a vector
 * @param b vector
 * @return new vector
 */
public static Vector add(Vector a, Vector b) {
  return new Vector(a.x + b.x, a.y + b.y, a.z + b.z);
}

/**
 * static vector subtraction of two vectors a and b
 * @param a vector
 * @param b vector
 * @return
 */
```

```
  public static Vector sub(Vector a, Vector b) {
    return new Vector(a.x − b.x, a.y − b.y, a.z − b.z);
  }

  @Override
  public String toString() {
    return "Vector␣[x=" + x + ",␣y=" + y + ",␣z=" + z + "]";
  }

}
```