Application of Set Theory and Boolean Algebra on Inkscape's Boolean Operations

Puti Nabilla Aidira - 13521088
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*author@itb.ac.id*

***Abstract*—Inkscape is a free and open-source vector graphics editor software. It is commonly used for artistic and technical illustrations such as cartoons, clip art, logos, typography, etc. One of its powerful features is Boolean Operations which can be used to form a specific desired shape from some basic shapes. The implementation of these Boolean Operations involves the application of Set Theory and Boolean Algebra. This paper will discuss how Set Theory and Boolean Algebra help in the implementation of Inkscape's Boolean Operations. We will also take a look into the real code implementation taken from Inkscape's official repository and how it is related to Set Theory and Boolean Algebra.**

***Keywords*—Boolean Algebra, Boolean Operations, Inkscape, Set Theory, Vector Graphics.**

I. Introduction

Set theory is a branch of discrete mathematics that studies sets. Informally, a set is described as a collection of different objects called elements or members. Historically, set theory was founded in 1874 by Georg Cantor with his paper titled "On a Property of the Collection of All Real Algebraic Numbers". As of now, set theory is known as the foundation of mathematics in which all mathematical concepts are defined in terms of the primitive notions of set and membership [1].

Boolean algebra is a branch of algebra that has truth values as variables and logical operators as operators. Boolean algebra was first introduced in 1854 by George Boole in his book: "The Laws of Thought". Sets algebra, which is a set of sets closed under the set-theoretic operations is a form of Boolean algebra. That is, they have analogous basic symbols and laws.

Inkscape is a free and open-source vector graphics editor software that runs on GNU/Linux, Windows, and macOS. It is commonly used for artistic and technical illustrations including cartoons, clip art, logos, typography, diagramming, and flowcharting [2]. Inkscape's main format is the standardized SVG file format. One of Inkscape's features is Boolean Operations which works on paths. Boolean Operations available are Union, Difference, Intersection, Exclusion, Division, Cut Path, Combine, and Break Apart. These operations are widely used to form a specific desired shape from any basic shapes.

II. Theoretical Basis

A. Set Theory

1. Definition and Formal Notation

A set is a collection of different objects called elements or members. Set-builder notation for a set that is defined by a predicate looks like one of the following:

$$\begin{gathered}\{x|\Phi(x)\}\\ \{x:\Phi(x)\}\\ \{x|x\in E\ and\ \Phi(x)\}\\ \{x|x\in E\wedge\Phi(x)\}\\ \{x\in E|\Phi_1(x),\Phi_2(x)\}\end{gathered} \tag{1}$$

With $\Phi(x)$ is said to be the rule or predicate in which if holds then x belongs to the set.

Table i. Formal Symbols in Sets

Symbol	Description
P or Z^+	Set of positive integers = {1, 2, 3, …}
N	Set of natural numbers = {1, 2, 3, …}
Z	Set of integers = {…,-2, -1, 0, 1, 2, 3, …}
Q	Set of rational numbers = $\{a/b \mid a, b \in Z \text{ and } b \neq 0\}$
R	Set of real numbers.
R^+	Set of positive real numbers.
C	Set of complex number = $\{a + bi \mid a, b \in R\}$
U	Universal
Ø or { }	Empty set

2. Venn Diagram

The Venn diagram is an informal set representation that uses overlapping circles (or other shapes) to illustrate the logical relationships between them [3]. It is first used by John Venn in an 1880 paper entitled "On the Diagrammatic and Mechanical Representation of

Propositions and Reasonings". The Venn diagram is illustrated in [4, Fig. 1].

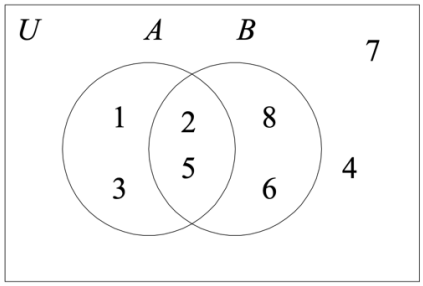

Figure 1. Venn Diagram Example

3. Intersection

The intersection is a binary set operator whereas the result set only contains elements that belong to both of the operand sets, as formally defined by (2). It is illustrated in [4, Fig. 2].

$$A \cap B = \{ x \mid x \in A \text{ and } x \in B \} \quad (2)$$

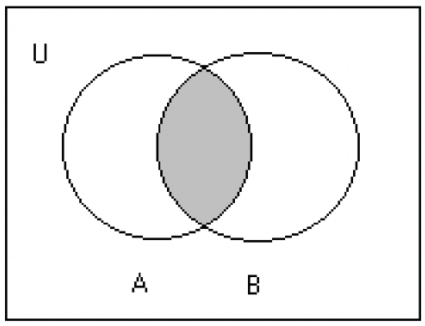

Figure 2. Intersection Illustrated in Venn Diagram

4. Union

The union is a binary set operator whereas the result set contains all elements that belong to either operand set, as formally defined by (3). It is illustrated in [4, Fig. 3].

$$A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \quad (3)$$

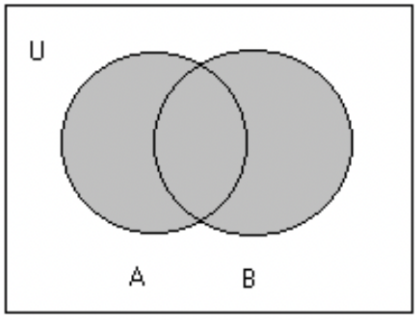

Figure 3. Union Illustrated in Venn Diagram

5. Complement

The complement is a unary set operator whereas the result set contains all elements in the universe that do not belong to the operand set, as formally defined by (4). It is illustrated in [4, Fig. 4].

$$\bar{A} = \{ x \mid x \in U, x \notin A \} \quad (4)$$

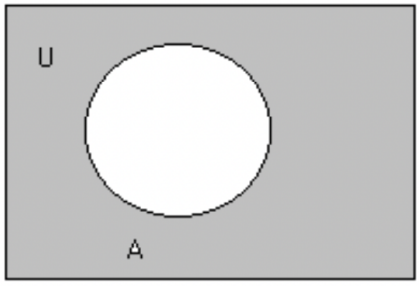

Figure 4. Complement Illustrated in Venn Diagram

6. Difference

The difference is a binary set operator whereas the result set contains all elements that belong to the first operand set but do not belong to the second operand set, as formally defined by (5). It is illustrated in [4, Fig. 5].

$$A - B = \{ x \mid x \in A \text{ and } x \notin B \} = A \cap \bar{B} \quad (5)$$

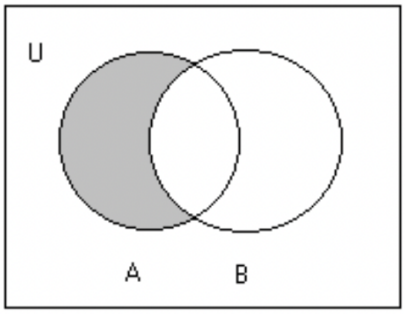

Figure 5. Difference Illustrated in Venn Diagram

7. Symmetric Difference

The symmetric difference is a binary set operator whereas the result set contains all elements that do not belong to the intersection of the operand sets, as formally defined by (6). It is illustrated in [4, Fig. 6].

$$A \oplus B = (A \cup B) - (A \cap B) = (A - B) \cup (B - A) \quad (6)$$

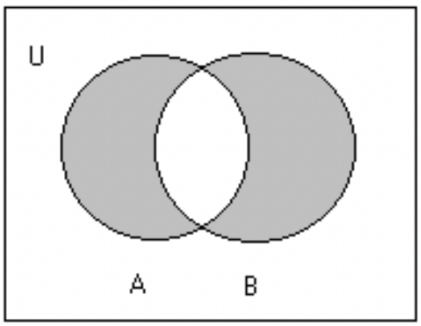

Figure 6. Symmetric Difference Illustrated in Venn Diagram

8. *Inclusion-Exclusion Principle*

The Inclusion-Exclusion Principle describes the cardinality, the number of different elements in a set, of a union set, as in (7).

$$|A \cup B| = |A| + |B| - |A \cap B| \tag{7}$$

With, |A| defined the cardinality of set A.

B. *Boolean Algebra*

1. *Definition*

Boolean Algebra is a mathematical structure formed by basic logical laws and properties described by George Boole in his book "The Laws of Thought" [4]. The formal definition of Boolean Algebra is for B, a set defined by two binary operators, + and ·, and a unary operator '. Let 0 and 1 be two elements different from B. Then, the tuple (8) called Boolean Algebra if for a, b, c ∈ B applies axioms (9).

$$< B, +, \cdot, ', 0, 1 > \tag{8}$$

1. Identity:

(i) a + 0 = a

(ii) a · 1 = a

2. Commutative:

(i) a + b = b + a

(ii) a · b = b . a

3. Distributive:

(i) a · (b + c) = (a · b) + (a · c)

(ii) a + (b · c) = (a + b) · (a + c)

4. Complement

For every a ∈ B there exist a unique element a' ∈ B such that:

(i) a + a' = 1

(ii) a · a' = 0

(9)

2. *Operators*

Boolean Algebra operators include +, · , ', and the derivated, derivation as in (10), XOR (⊕) operators. The truth table of these operators is shown in Table. ii.

Table ii. Truth Table of Boolean Algebra Operations

a	b	a + b	a · b	a ⊕ b
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

$$a \oplus b = (a + b) \cdot (a' + b') \tag{10}$$

3. *Sets Algebra as Boolean Algebra*

Set algebra is defined as a set of sets closed under the set-theoretic operation. Set algebra is a form of Boolean Algebra as they have analogous operators, laws, and properties. The analogous operators showed in Table. iii. Set algebra tuple showed in (11).

Table iii. Analogous Operators between Boolean Algebra and Sets Algebra

Boolean Algebra	Sets Algebra
1	U
0	Ø
$p \cdot q$	$A \cap B$
$p + q$	$A \cup B$
p'	$\bar{A}$

$$< B, \cap, \cup, -, \emptyset, U > \tag{11}$$

C. *Vector Graphics*

Vector graphics is a form of computer graphics that uses vector data models. Vector data models is a data models that use points and their associated (x, y) coordinate pairs to represent the vertices of spatial features. In contrast with the raster data models, in which spatial information is quantized into discrete grid cells, vector data models have their spatial information linked via a simple identification number given to each feature in a map [5]. The comparison between raster and vector graphics is shown in [6, Fig. 7].

The fundamental elements of vector graphics are point, line, and polygon, as in [5, Fig. 8]. Points are zero-dimensional objects containing only a single coordinate pair. Lines are one-dimensional features composed of multiple, explicitly connected points. Linea is also often referred to as paths. Polygons are two-dimensional features created by multiple lines that loop back to create a closed feature [5]. Other more complex elements of vector graphics also include circular arcs, cubic spines, Bézier curves, circles, ellipses, spheres, polygon mesh, fractals, etc.

Vector graphics can be edited with a vector graphic editor. Modifications provided by vector graphic editors typically allow translation, rotation, mirroring, stretching, skewing, affine transformations, as well as changing of z-order [7]. Other than that, boolean/set operations such as union, intersection, difference, etc. are also commonly provided in vector graphic editors software [8].

The World Wide Web Consortium (W3C) standard for vector graphics is Scalable Vector Graphics (SVG). Meanwhile, other file formats such as WMF, EPS, PDF, CDF, and AI, are also commonly used to represent vector graphics.

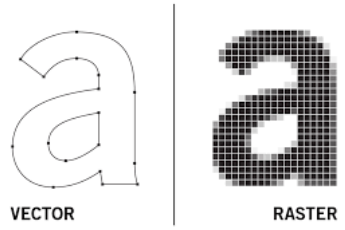

Figure 7. Vector Graphic and Raster Graphic Comparison

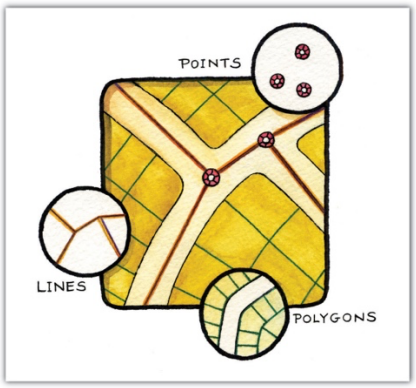

Figure 8. Points, Lines, and Polygons as Vector Graphics Fundamental Elements

III. INKSCAPE'S BOOLEAN OPERATIONS IMPLEMENTATION

Inkscape is a free and open-source vector graphics editor software that runs on GNU/Linux, Windows, and macOS. Inkscape is widely used for artistic and technical illustrations including cartoons, clip art, logos, typography, diagramming, and flowcharting [2]. One of its features is Boolean Operations which are operations between two or more paths (or objects that are automatically converted into paths) [9]. These Boolean Operations include Union, Difference, Intersections, Exclusions, Division, Cut Path, Combine, and Break Apart. For the sake of simplicity and to preserve relevancy with the topic, only Union, Difference, Intersections, and Exclusion will be discussed.

A. Union

Union Operation is an operation between two paths that keep the common outline the common outline of all selected paths [9]. The illustration of how the operation works is shown in Fig. 9. Looking at its definition, it is clear that the Union Operation uses the same logic as the union in sets algebra. Inkscape implementation, taken from the official Inkscape repository, written in C++ is shown in [10, Fig. 10].

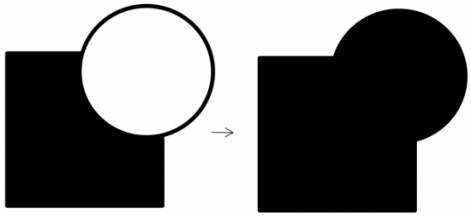

Figure 9. Union Operation

```
PathVector PathIntersectionGraph::getUnion()
{
  PathVector result = _getResult(false,
                                 false);
  _handleNonintersectingPaths(result, 0,
                                 false);
  _handleNonintersectingPaths(result, 1,
                                 false);
  return result;
}
```

Figure 10. Code Snippet of Inkscape's Union Operation Implementation

The code uses the _getResult function that takes two boolean arguments. The _getResult function is mainly a function to determine which 'direction' to go. The first argument is for the first path operand, supposedly called A. If the first argument's value is 'false' then it is supposed to 'go outside' in path A. Otherwise, if the first argument's value is 'true' then it is supposed to 'go inside' in path A. Similarly, the second argument applies the same rules for the second path operand, supposedly called B. To make it clear, the term 'direction', 'go inside', and 'go outside' are terms used in the source code comment of this function. The code snippet of how the function gets implemented fully is shown in [10, Fig. 11]. However, to preserve relevancy and simplicity, the details of how the function gets implemented won't be further discussed. The _handleNonintersectingPaths function which handles path operands that do not intersect also won't be further discussed.

Nonetheless, note that we can define the term 'go outside' as including points from the current path operand that do not intersect (in terms of area) with the other path operand, or intuitively means the points that lie outside the intersection area. In another word, 'go outside' in A means including points in A that does not intersect (in term of area) with B, and vice versa. On the contrary, we can define the term 'go inside' as including points from the current path operand that exclusively intersect (in terms of area) with the other path operand, or intuitively means the points that lie inside the intersection. In another word, 'go inside' in A means to only include points in A that intersect with B, and vice versa.

With this definition in mind, it can be concluded that this

implementation of Union Operation means that the resulting path will contain all of the points from both path operands, except the points inside the intersection area. It is because points inside the intersection area will only count once, that is as an area inside the union path. Which is, reciprocal with the inclusion-exclusion principle defined in (7).

```
PathVector PathIntersectionGraph::_getResult(bool enter_a, bool
enter_b){
        PathVector result;
        if (_xs.empty()) return result;
        // reset processed status
        _ulist.clear();
        for (auto & _component : _components) {
            for (auto & li : _component) {
                for (auto & k : li.xlist) {
                    _ulist.push_back(k);}}}
        unsigned n_processed = 0;
        while (true) {
            // get unprocessed intersection
            if (_ulist.empty()) break;
            IntersectionVertex &iv = _ulist.front();
            unsigned w = iv.which;
            ILIter i =
            _components[w][iv.pos.path_index].xlist.iterator_to(iv);
            result.push_back(Path(i->p));
            result.back().setStitching(true);
            bool reverse = false;
            while (i->_proc_hook.is_linked()) {
                ILIter prev = i;
                std::size_t pi = i->pos.path_index;
                // determine which direction to go
                // union: always go outside
                // intersection: always go inside
                // a minus b: go inside in b, outside in a
                // b minus a: go inside in a, outside in b
                reverse = false;
                if (w == 0) {
                    reverse = (i->next_edge == INSIDE) ^ enter_a;
                } else {
                    reverse = (i->next_edge == INSIDE) ^ enter_b;}
                // get next intersection
                if (reverse) {
                 i = cyclic_prior(i, _components[w][pi].xlist);
             } else {
                 i = cyclic_next(i, _components[w][pi].xlist);}
             // append portion of path
             PathInterval ival = PathInterval::from_direction(
                 prev->pos.asPathTime(), i->pos.asPathTime(),
                 reverse, _pv[i->which][pi].size());
             _pv[i->which][pi].appendPortionTo(result.back(), ival,
                                               prev->p, i->p);
             // mark both vertices as processed
             //prev->processed = true;
             //i->processed = true;
             n_processed += 2;
             if (prev->_proc_hook.is_linked()) {
                 _ulist.erase(_ulist.iterator_to(*prev));}
             if (i->_proc_hook.is_linked()) {
                 _ulist.erase(_ulist.iterator_to(*i));}
             // switch to the other path
             i = _getNeighbor(i);
             w = i->which}
         result.back().close(true);
         if (reverse){
             result.back() = result.back().reversed();
         }
         if (result.back().empty()) {
             // std::cerr << "Path is empty" << std::endl;
             throw GEOM_ERR_INTERSECGRAPH;
         }
     }
     if (n_processed != size() * 2) {
         // std::cerr << "Processed " << n_processed << "
         intersections, expected " << (size() * 2) << std::endl;
         throw GEOM_ERR_INTERSECGRAPH;
     }
     return result;
 }
```

Figure 11. Code Snippet of Inkscape's _getResult Function Implementation

B. Difference

Difference Operation is an operation between two paths that subtract one path from another one [9]. In the difference operation, the stacking order should be considered. The rule is that the bottom path will be subtracted by the top path, and vice versa. The illustration of how the operation works is shown in Fig. 12. The Difference Operation is analogous to the difference operator, A – B and B – A, in sets algebra. Inkscape implementation of Difference Operation, written in C++, is shown in [10, Fig. 15].

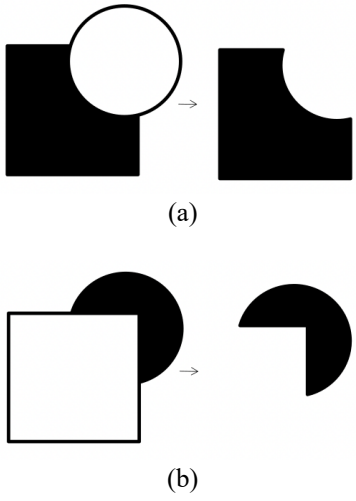

Figure 12. (a) Difference Operation with Square as The Bottom Path, (b) Difference Operation with Circle as The Bottom Path

```
PathVector
PathIntersectionGraph::getAminusB()
{
  PathVector result = _getResult(false,
                                 true);
  _handleNonintersectingPaths(result, 0,
                              false);
  _handleNonintersectingPaths(result, 1,
                              true);
  return result;
}

PathVector
PathIntersectionGraph::getBminusA()
{
  PathVector result = _getResult(true,
                                 false);
  _handleNonintersectingPaths(result, 1,
                              false);
  _handleNonintersectingPaths(result, 0,
                              true);
  return result;
}
```

Figure 13. Code Snippet of Inkscape's Difference Operation Implementation

The implementation divides into two functions, getAminusB and getBminusA. getAminusB is analogous to A – B in set algebra, whereas A is the bottom path and B is the top path. It can be seen that, with the interpretation and definition discussed in the Union section, getAminusB() will 'go outside' in A and 'go inside' in B. That is, the resulting path will contain all the points on path A that lie outside the intersection area and all the points on path B that lie inside the intersection area. Similarly, the opposite applies to getBminusA().

C. Intersection

Intersection Operation is an operation between two paths that only keep those parts covered by all selected paths [9]. The illustration of how the operation works is shown in Fig. 14. By its definition, it is clear that Intersection Operation is analogous to the intersection in sets algebra. Inkscape implementation of Intersection Operation, written in C++, is shown in [10, Fig. 15].

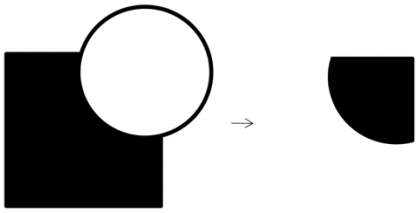

Figure 14. Intersection Operation

```
PathVector
PathIntersectionGraph::getIntersection()
{
  PathVector result = _getResult(true,
                                 true);
  _handleNonintersectingPaths(result, 0,
                              true);
  _handleNonintersectingPaths(result, 1,
                              true);
  return result;
}
```

Figure 15. Code Snippet of Inkscape's Intersection Operation Implementation

With definitions and interpretations discussed in the Union section, it can be concluded that getIntersection() will 'go inside' in both A and B. In other words, the resulting path will include only the points that lie inside the intersection area between paths A and B.

D. Exclusion

Exclusion Operation is an operation between two paths that keeps those parts covered by an odd number of paths [9]. In the case of operation between two paths, the result is where the paths do not overlap. The illustration of how the operation works is shown in Fig. 16. From its definition, Exclusion Operation can be seen as analogous to the symmetric difference in sets algebra. Inkscape implementation of Intersection Operation, written in C++, is shown in [10, Fig. 17].

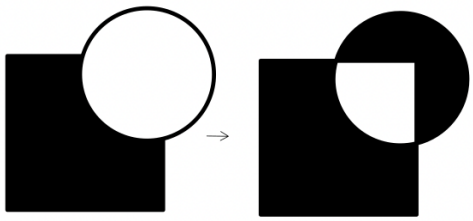

Figure 16. Exclusion Operation

```
PathVector PathIntersectionGraph::getXOR()
{
  PathVector r1, r2;
  r1 = getAminusB();
  r2 = getBminusA();
  std::copy(r2.begin(), r2.end(),
            std::back_inserter(r1));
  return r1;
}
```

Figure 17. Code Snippet of Inkscape's Exxlusion Operation Implementation

Although the implementation of copy and back_inserter functions won't be further discussed, it can be interpreted that getXOR() takes two functions: getAminusB and getBminusA, then sort of 'add' them together. It is reciprocal with the symmetric difference definition in (6), where symmetric difference can be defined as the union of A – B and B – A.

IV. Conclusion

Set theory and set algebra, as a form of Boolean Algebra, is a fundamental and useful concepts. It is used not only in the field of Math and Science but also in Art and Creativity. Particularly, it is used in the implementation of Boolean Operations provided in Inkscape. Thus, having knowledge of set theory and set algebra enables us to create complex shapes, as complex as our imagination can be.

V. Acknowledgment

First and foremost, I would like to thank Allah Swt. as without his blessing I wouldn't be able to finish this paper. Second, I would also like to thank Dr. Nur Ulfa Maulidevi, S.T., M.Sc. as my Discrete Mathematics lecturer for her lectures that inspire me to write this paper. Not to forget, I wish to show my appreciation for Dr. Ir. Rinaldi Munir, MT. as his lectures material help me to compose this paper. Lastly, I also want to thank all of my friends and colleagues who always support me.

References

[1] Kunen, Kenneth (1980), Set Theory: An Introduction to Independence Proofs, North-Holland, ISBN 0-444-85401-0.

[2] https://inkscape.org/about/, accessed 10/12/2022.

[3] "What is a Venn Diagram". Lucidchart, https://www.lucidchart.com/pages/tutorial/venn-diagram#:~:text=A%20Venn%20diagram%20uses%20overlapping,items%20are%20similar%20and%20different, accessed 10/12/2022.

[4] Munir, Rinaldi. “Himpunan (Bag.1 – Update 2022”. Program Studi Teknik Informatika STEI ITB: 2022, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2022-2023/Himpunan(2022)-1.pdf, accessed 10/12/2022.
[5] “Essentials of Geographic Information Systems v. 1.0”. Saylor Academy: 2012, https://saylordotorg.github.io/text_essentials-of-geographic-information-systems/s08-02-vector-data-models.html, accessed 10/12/2022.
[6] https://blog.fileformat.com/2021/08/25/raster-vs-vector-images-a-brief-comparison/, accessed 10/12/2022.
[7] Nigel Chapman; Jenny Chapman (2002) [2000]. Digital Multimedia. Wiley. p. 70. ISBN 0-471-98386-1.
[8] Barr, Alan H. (July 1984). "Global and Local Deformations of Solid Primitives" (PDF). SIGGRAPH. 18 (3): 21–30. CiteSeerX 10.1.1.67.6046. doi:10.1145/800031.808573. ISBN 0897911385. S2CID 16162806.
[9] https://inkscape-manuals.readthedocs.io/en/latest/boolean-operations.html , accessed 10/12/2022.
[10] https://inkscape.gitlab.io/inkscape/doxygen/intersection-graph_8cpp_source.html#l00338, accessed 10/12/2022.

STATEMENT

I hereby declare that the paper I am writing is my own writing, not an adaptation or translation of someone else's paper, nor plagiarism.

Bandung, 12 December 2022

Puti Nabilla Aidira 13521088