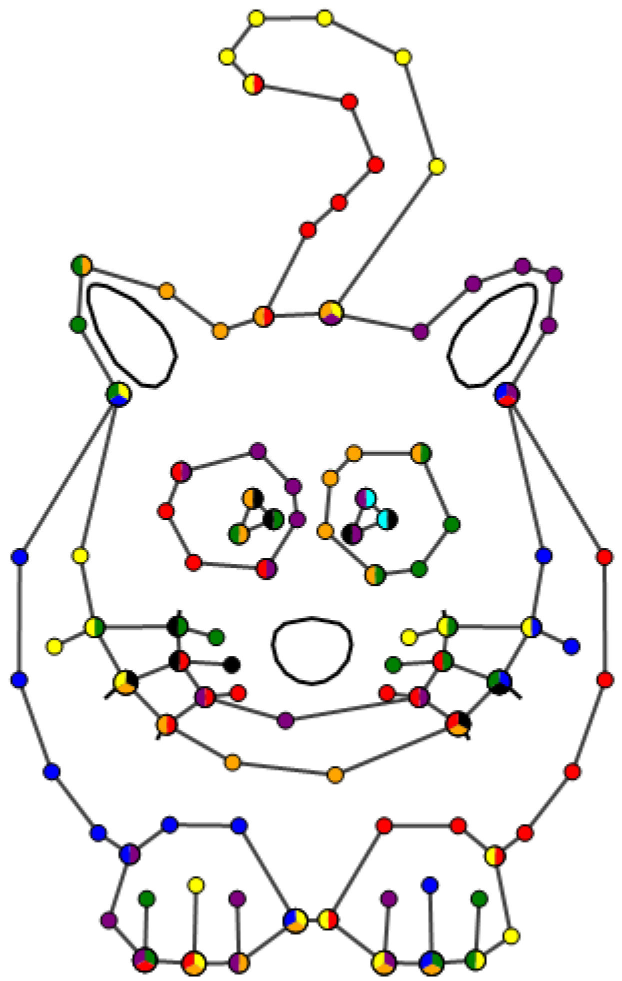
Connect The Closest Dot Puzzles

Tim van Kapel

June 23, 2014

Master's Thesis



Utrecht University
Marc van Kreveld
Maarten Löffler

Abstract

In this thesis we present a new variation of the existing *connect the dots* puzzles. The goal of the puzzle is similar to those *connect the dots* puzzles, where an illustration is revealed by connecting pairs of dots via line segments. The puzzle uses colors and the distance between dots to define which pairs of dots to connect. An algorithm is presented that allows for automatic generation of such puzzles from a planar graph in the shape of the illustration.

Contents

1 Introduction

Connect the dots or *follow the dots* puzzles are puzzles where the goal is to reveal an illustration from a set of dots. This goal is accomplished by drawing lines between pairs of dots. The indication of which pairs can differ, but usually consists of subsequent numbers. Figure 1 shows a half-solved *connect the dots* puzzle where the lines between two dots with subsequent numbers must be drawn. The result of all these lines usually creates an illustration, in this case a face. Multiple variants of this puzzle exist, such as a puzzle that uses subsequent letters instead of number, or a puzzle where two dots are connected when the distance between them is exactly some pre-determined distance [3]. In the second variant the dots are not labeled with a number or letter, instead two dots are connected when there distance from eachother is exactly x, where x is a pre-determined fixed distance. The goal of the puzzle, revealing the illustration, remains similar, however the puzzler must measure the distance between dots instead of finding the next number of letter.

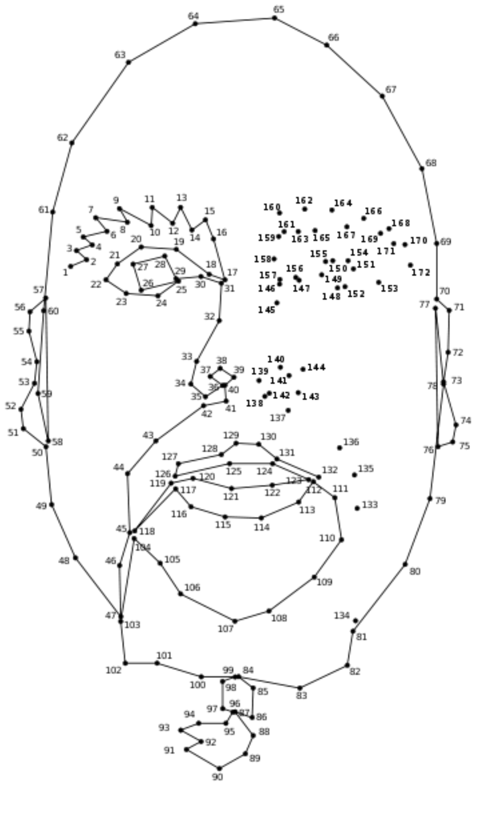


Figure 1: A half-solved *connect the dots* puzzle. Connect two subsequent dots by drawing a line segment from one to the other. The result of all line segments usually reveals an illustration.

In this thesis we will introduce a new variation on the classic *connect the dots* puzzle called *connect the closest dot* and an algorithm to automatically generate the puzzle from a line drawing. Where the classic puzzle uses numbers to indicate the dots to connect, *connect the closest dot* puzzles use colors and the distances between dots. In geometery this can be seen as a nearest neighbor graph, where the vertices are represented as the dots and the edges need to be drawn, an example is given in Figure 2. A nearest neighbor graph has the disadvantage that only graph trees, with the shortest edges all towards a single location, can be created. This restriction disallows for the puzzles to reveal an illustration when the edges are drawn. Therefore multiple colors are used, and dots only connect to the same color. This can be seen as multiple different nearest neighbor graphs, one for each color.

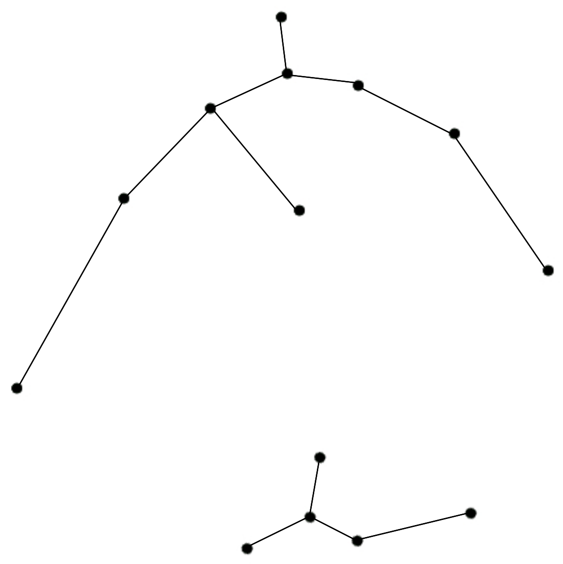


Figure 2: An example of a nearest neighbor graph. Vertices are only connected by an edge when one of the two vertices has the other as closest vertex. Longer chains of connected vertices are only possible if the distance between the vertices keeps increasing.

The goal of this thesis is to research the *connect the closest dot* puzzles and present an algorithm that can generate *connect the closest dot* based on a given illustation. The algorithm will be given an illustration from which a set of colored dots are produced. By solving the puzzle from the set of colored dots a simplified yet still recognizable representation of the original illustration should reappear.

Connect the closest dot puzzles and its rules are further explained in Section 2 where examples of the puzzle are also given. The specification for a good *connect the closest dot* puzzle are given in Section 2.1 and multiple variants of the puzzle in Section 2.2. The steps taken to automatically generate a puzzle from an illustation are described throughout section 3. Sections 3.1, 3.2 and 3.3 describe each step of the algorithm in more detail. The time complexity of the algorithm is analysed in Section 3.4. The effectiveness of the algorithms is discussed in Section 4 and the qualtity of the generated puzzles in Section 4.1. The effect of the different alogrithm parameters on the output of the algorithm is discussed in Section 4.2. This section discusses the effect of changing every parameter for the algorithm on the generated puzzle. Finally a conclusion is given, and future work is discussed in Section 5.

2 Puzzle Specifications and Variants

In this section we describe *connect the closest dot* puzzles and the rules for the creation of a puzzle. First the puzzle and the rules are described. Section 2.1 discusses what is needed for a good puzzle. It discusses what makes a puzzle better, easier or harder for the puzzler. Finally, Section 2.2 describes multiple variations of the *connect the closest dot* puzzle.

This thesis represents a new type of line puzzle as shown in Figure 3. Similar to the *connect the dots* puzzles, the *connect the closest dot* puzzle consists of a set of dots that need to be connected with line segments. However, there are no numbers or other indicators for which dots to connect. Instead each dot is represented with a color. It is possible for two dots with a different color to be at the same location. This is represented as a single dot with all different colors. The size of the dots is dependent on the number of colors for that dot. A dot with more colors will be larger than a dot with only a single color. The puzzle can also contain pre-drawn curved that are part of the solution.

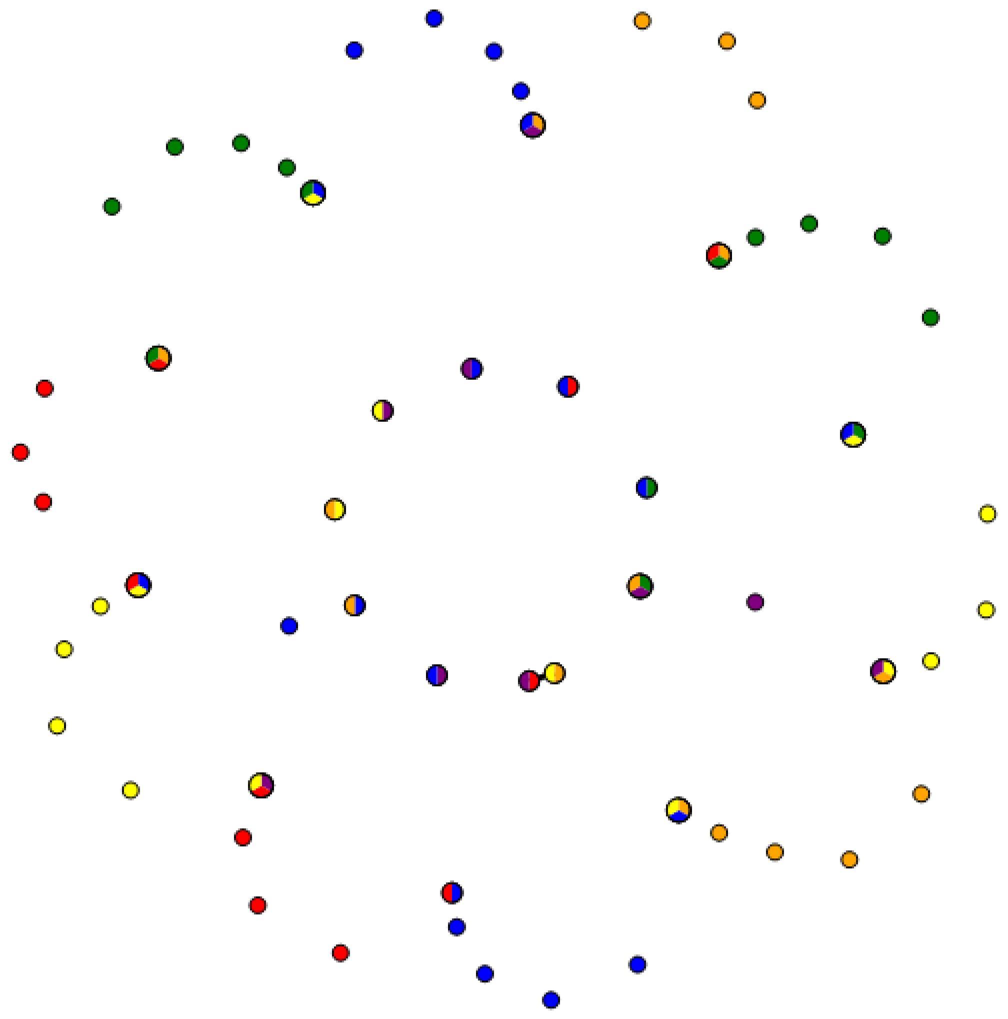


Figure 3: An example of an unsolved *connect the closest dot* puzzle.

The goal of *connect the closest dot* puzzles is to reveal an illustration by connecting pairs of dots via line segments. Which pairs to connect is determined by the colors of both dots and the distance between them. Each color for each dot must be connected to the single closest dot with the same color. When a dot has multiple colors, for instance both red and blue, a line should be drawn to the closest red dot and the closest blue dot. This means that in order to solve the puzzle and reveal the illustration, for each dot, lines must be drawn to the closest dots of the same colors. Figure 4 shows the steps for solving a puzzle.

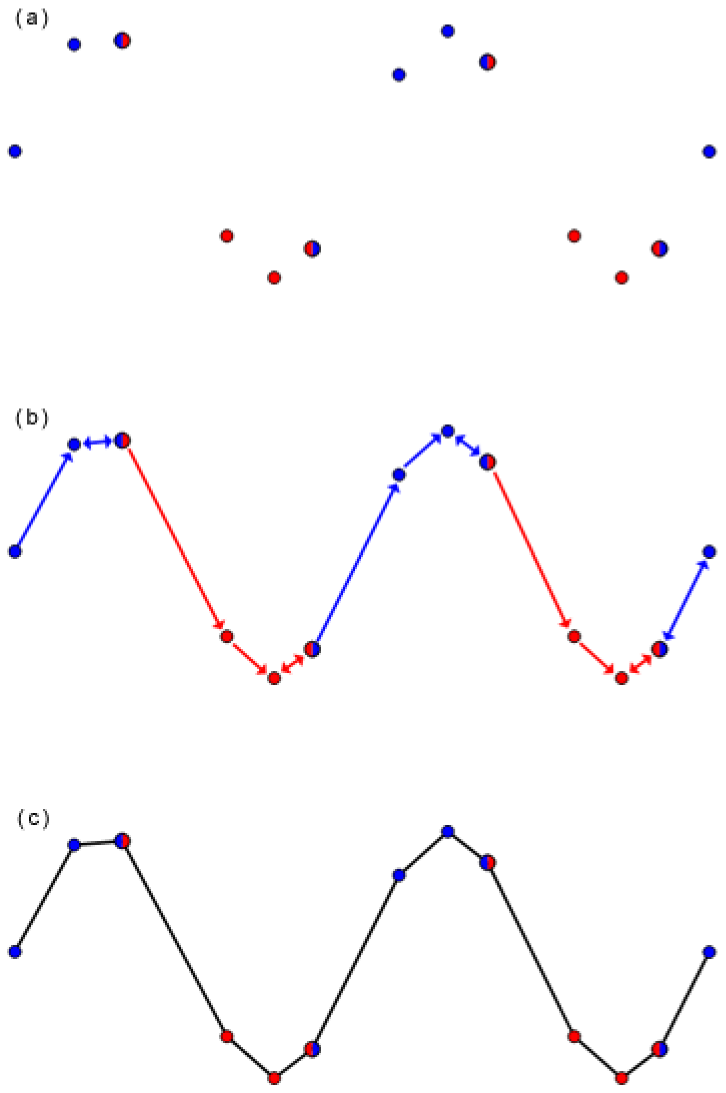


Figure 4: (a) An example of a *connect the closest dot* puzzle. (b) The solution for the puzzle, the arrows show which dot should connect to which. (c) The solved puzzle where a line is drawn between each dot and its closest dot of the same color.

2.1 Specifications

This section describes the criteria for a good puzzle. Given an illustration, multiple different *connect the closest dot* puzzles with the same resulting illustration can be created. In this Section we specify a number of of optimization that should produce a well-solvable puzzle.

One of the most important aspects for *connect the dots* puzzles is that given a seemingly random set of dots, after connecting the dots a clear illustration appears. In order for the resulting illustration to be recognizable, the result needs to stay close to the original illustration. Therefore, when a puzzle is based on an image the distance between the original and the puzzle representation needs to be minimized. A **maximum error margin** ϵ defines that the Hausdorff distance between the original polyline and its representation cannot be larger than ϵ. This means that from no point of the original or the puzzle, the distance to the other can be larger than ϵ.

Another important aspect is that the puzzler cannot recognize the illustration beforehand. Being able to see the resulting illustration based on the puzzle dots makes the puzzle less fun and less satisfying. Not only is the result no longer a surprise, but finding the dots to connect is also easier since it is already known where the lines should be drawn. For a puzzle with a lot of dots, it is easier to recognize a pattern and the resulting illustration than for a puzzle with the same illustration and fewer dots. For this reason the **total number of dots** n used in the puzzle should be **minimized**.

The task is to connect pairs of dots via a line segment. When two dots are already touching or even overlapping not only is it impossible for the puzzler to connect these dots, but visually it gives a very messy puzzle. For those reasons the **minimum distance** d_{min} between two dots must be at least more than a set threshold. The value of d_{min} should always be equal or more than the diameter of the largest dots.

For the puzzler it should be clear which dot connect to which dot. When three dots of equal color have almost the same distance from each other, it is unclear in what way the dots must be connected. Figure 5 (a) shows 3 dots at an almost equals distance, it is unclear what dots need to be connected. In this case p_0 has p_1 as its closest dot, this is denoted as $p_0 \rightarrow p_1$. For the puzzler it could also have been $p_0 \rightarrow p_2$ since the distances are almost equal. Similar to the technique used in the *unite-the-dots* puzzle [3], the distance to each dot other than the closest, should at least be a **factor** ρ larger than the distance to the closest dot. Figure 5 (b) shows that if $p_0 \rightarrow p_1$, p_2 should be outside the circle $C(p_0, |p_0p_1| * \rho)$.

Giving each pair of connecting dots a unique color also eliminates the issue where multiple dots have an equal distance. However, a puzzle with 40 different colors shows very sloppy and a lot of colors will look similar to each other. For the puzzler, rather than searching for the closest dot, it becomes a case of figuring out whether two colors are equal. A smaller number of colors is good in all cases. In order to keep the puzzle clear and to prevent colors from looking too similar the **number of colors** m should be minimized.

By minimizing the number of dots, the distance distance betwen the dots is usually larger. Figure 5 (c) shows a set of dots where the two connecting green dots lie on opposite sides of the puzzle. Due to the large distance, and it not being logical for the connecting dot to be on the opposite side of the puzzle, it will be tough for the puzzler to find the other green dot, when solving the puzzle. A **maximum distance** d_{max} between two dots that need to be connected prevents the distance being to large. Also due to the large distance between the two green dots, the color green cannot be used anywhere else in the puzzle. By reducing the distance between pairs of dots, the circles in which no dot of the same color can lay, become smaller. Therefore it is possible that fewer colors are needed.

The current puzzle uses long chains of a single color. This means that when $p_0 \rightarrow p_1$, p_1 often connects to another dot p_2 using the same color. When possible the puzzle should contain long chains of a single color, rather than short multi colored polylines. By **minimizing the number of multi colored dots** a puzzle with less changes of color is preferred.

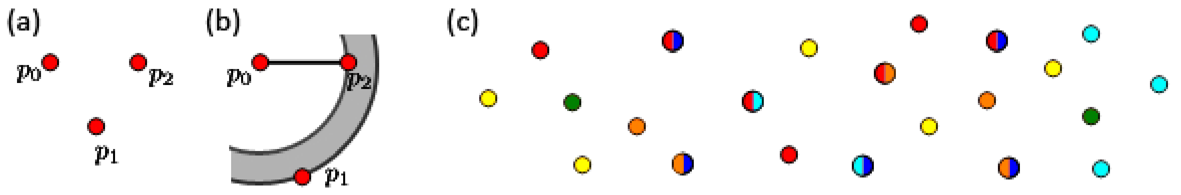


Figure 5: (a) Three dots at alost equal distance, it is unclear which dot connects with which. (b) $p_0 \rightarrow p_1$, this means that the distance betwen p_0 and p_2 must be atleast a factor ρ more than the distance between p_0 and p_1. (c) Given a puzzle, it is hard to spot that the left green dot connects with the right green dot. Also due to the large distance no more other green dots can be used.

2.2 Variants

The *connect the closest dot* puzzle as described, will be the puzzle variant that is used throughout the rest of the thesis. Using the same concept of connecting the closest equally colored dots, multiple variants of the puzzle can be made. This section describes a few variants and the difference for the puzzler.

In the *connect the closest dot* puzzles each dot connects to exactly a single closest dot. For the puzzler this creates a clear objective since it is known that the number of lines drawn from each dot equals the number of colors for that dot. However the puzzle itself suffers from this single closest rule, since it is only possible to have chains of a single color decreasing in distance between dots. This means that the length of such chain is directly limited by d_{min} and d_{max}. In order to have a long chain, the distance between dots is always larger at one end of the chain. For curved lines the distance will quickly become too large, and the allowed error to the original will become larger than ϵ.

The first variant can eliminate those issues by allowing multiple points to be at exactly the same distance. This way it is possible to reduce the increase of the distances between dots of a chain by having 2 dots at exactly the same distance to another dot. By allowing multiple closest dots, it also becomes possible to create junctions with the distances increasing towards the junction in a single color. This can potentually allow the same puzzles to be created using fewer dots and fewer colors. For the puzzler this change drastically increases the difficulty of the puzzle since the number of closest dots is now unknown. The margin ρ has to potentially be increased as well as it is even more important to know whether a dot is equally close or not. Overall this change can improve on the quality of the puzzle, but makes the puzzle harder for the puzzler. An example of such a puzzle where multiple dots can be the closest is given in Figure 6.

A second variant makes the puzzle easier for the puzzler by removing long chains of dots of a single color. When $p_0 \rightarrow p_1$ in the other puzzles it is possible that $p_1 \rightarrow p_i$. For this variant, only pairs of dots are used and no chains are created. When a dot is the closest of another dot, in this case p_1 of p_0, p_1 is forced to have p_0 as its closest. This means that for each pair of connected dots $p_i p_j$, $p_i \rightarrow p_j$ and $p_i \leftarrow p_j$ must be true. In the original puzzle the number of colors of a dot equals the number of lines drawn from that dot to others. In this variant, the number of colors equals the number of lines connecting that dot. For the puzzler this creates an easy way of checking whether a dot still requires a line or not. An example of this variant is given in Figure 7.

All of these puzzles use colored dots as a representation for the puzzle. However, colored dots might not always be the best, for instance when there are lots of different colors, colors will start looking the same. Colorblindness is also a problem. A different representation could be by using different symbols such as dots, squares and triangles. All puzzles and variants can use said representation and eliminate the described issues. A problem with this are the currently multi-colored dots. When representing these dots with symbols they will overlap, thus for using symbols the symbols need to be repositioned. A puzzle using symbols is shown in Figure 8.

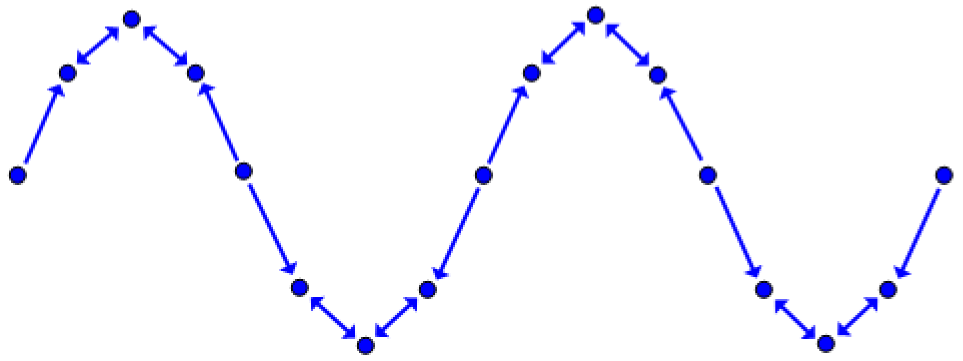


Figure 6: A different variant of the puzzle shown in Figure 4. A single dot can have multiple closest dots at exactly the same distance. This allows the puzzle to be created using only a single color.

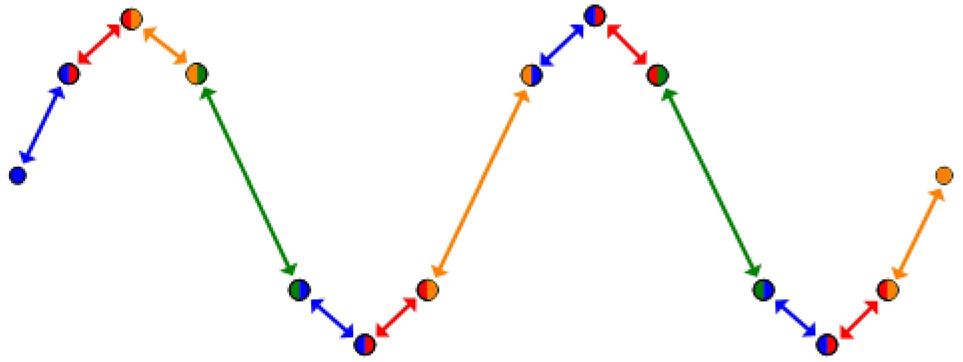


Figure 7: A different variant of the puzzle shown in Figure 4. For each pair of connected dots they must have each other as their closest neighbor. More colors are needed to create this puzzle.

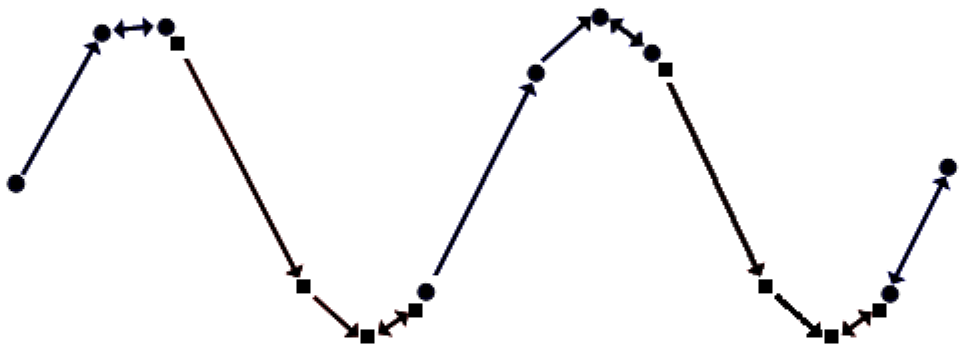


Figure 8: A different representation of the puzzle shown in Figure 4. The blue color is represented as a circle whereas the red is represented as a square.

3 Algorithm

This section discusses the algorithm to automatically generate a connect the closest dot puzzle from a polygonal drawing. The input of the algorithm is a polygonal representation of the illustration. The following parameters are also given to the algorithm: ϵ that defines the maximum allowed error, the minimum distance increase ρ, the minimum distance between two dots d_{min} and the maximum distance between two dots d_{max}. The values for these parameters can be used to change the quality of the resulting puzzle. The output of the algorithm is a set of dots in one or multiple colors such that by connecting each dot with its closest dot of the same color the original illustration is revealed. When a part of the original cannot be solved, these sections of the puzzle will be pre-drawn.

The algorithm consists of three steps: 1) splitting the input into separate polylines 2) solving each polyline separately and 3) finally assigning colors to each solved polyline to form a solvable puzzle. The output for each step is the input for the next step, at the end of the last step we will have the data needed to create the puzzle. Each step is discussed in more detail in the sections below. Finally in section 3.4 the time complexity of the algorithm is analyzed. Algorithm 1 shows an overview of the algorithm.

Algorithm 1 Global Overview

```
Input: A planar graph $G(V, E)$, maximum error $\epsilon$, margin $\rho$, minimum distance $d_{min}$ and
    maximum distance $d_{max}$
Output: A set of colored dots $V'(v'_0, ..., v'_n)$ such that by solving the puzzle, the input graph
    is revealed with a maximum error of $\epsilon$
1: $P(P_0, ..., P_n) \leftarrow$ SegmentGraph($G$)
2: for $i = 0$ to $n$ do
3:     $P' \leftarrow P' \cup \leftarrow$ SimplifySegment($P_i, \epsilon, \rho, d_{min}, d_{max}$)
4: end for
5: GraphColoring($P'$)
```

The problem of finding a set of colored points from a set of vertices and edges such that a line from each point to the closest point of the same color produces a simplified version of the input graph whose distance is $< \epsilon$, is a problem that we don't know how to solve efficiently. The problem is that each part of the graph infuences the rest, therefore the problem is simplified.

3.1 Splitting

The input of the algorithm is a planar graph $G(V, E)$ which represents the illustration. Instead of solving the problem for the whole graph, the problem is simplified by separating the graph into polylines. The problem of line simplification is a well studied problem. Splitting the graph into separate polylines also has some disadvantages such as always having fixed nodes at intersections. Not having nodes at intersections can also create artifacts when lines will be displaced during the simplification and intersections are shifted.

The splitting of a graph into separate polylines at vertices of a degree other than 2, can be done in $O(n)$ time where n is the number of edges of the graph.

Algorithm 2 SegmentGraph

```
Input: A planar graph $G(V, E)$
Output: A set of polylines $P(P_0, ..., P_n)$
 1: Set traversed $false$ for each edge in E
 2: for each vertex $v$ in V do
 3:     if the degree of $v$ not 2 then
 4:         $p \leftarrow v$
 5:         for each untraversed edge $e$ in $v$ do
 6:             while the degree of the other vertex $v'$ of $e$ is 2 do
 7:                 $p \leftarrow p \cup v'$
 8:                 $v \leftarrow v'$
 9:                 Set traversed on $e$
10:             end while
11:             Add $p$ to $P$
12:         end for
13:     end if
14: end for
        ▷ Convert loops into polylines, these are the only vertices with untraversed edges.
15: for each vertex $v$ in V do
16:     $p \leftarrow v$
17:     for each untraversed edge $e$ in $v$ do
18:         while the degree of the other vertex $v'$ of $e$ is 2 do
19:             $p \leftarrow p \cup v'$
20:             $v \leftarrow v'$
21:             Set traversed on $e$
22:         end while
23:         Add $p$ to $P$
24:     end for
25: end for
```

3.2 Polygonal Line Simplification

With the input split into separate polylines, the problem is simplified as a polyline simplification problem. Given a polyline $P(p_0, ..., p_n)$, a maximum error ϵ, a margin ρ, a minimum distance d_{min} and an optional maximum distance d_{max}, find a minimum set of points p'_0,...,p'_m such that a line segment from each point to the closest point of the same color, creates a simplified version of P. P' is a valid simplified polyline of P if the maximum distance between P and P' is as most ε, a predefined maximum Hausdorff distance. For this solution to be colored with a single color, the point set p'_0,...,p'_m is only a valid set of points when the distance between any point p'_i and any other point p'_j except the closest point to p'_i named $p'_{iclosest}$, is a factor ρ more than the distance to the closest point $p'_{iclosest}$. For any pair of two points where p'_j is not $p'_{iclosest}$ the following must be true: $d(p'_i, p'_j) > d(p'_i, p'_{closest}) * \rho$.

A line segment $\overline{p'_ip'_j}$ where $i < j$ from the simplified polyline P' is called a shortcut. This shortcut bypasses all original points between p'_i and p'_j. The shortcut $\overline{p'_ip'_j}$ is a valid shortcut if the maximum Hausdorff distance from $\overline{p'_ip'_j}$ to every point on the original line is $\leq \varepsilon$. In order for the shortcut to be a valid shortcut for the puzzle, at least one of the points should have the other point as its clear-closest point. The clear-closest point is defined when no point is closer than the distance between the closest point multiplied by the factor ρ. If the closest point to p_i is p_j and no other point is inside the circle $C(p_i, |\overline{p_ip_j}| * \rho)$, p_i has p_j as its clear closest point.

Each point can only have a single closest other point. For any point p'_i either p'_{i+1} or p'_{i-1} must be the closest. When this isn't the case, the shortcut will not be drawn when the puzzle is solved since another point is closer. When $p'_i \rightarrow p'_{i+1}$, p'_i has p'_{i+1} as its closest point to connect to, for the shortcut $\overline{p'_ip'_{i+1}}$, p'_{i-1} must have p_i as its closest point for the previous shortcut $\overline{p'_{i-1}p'_i}$. This means that for each shortcut $\overline{p'_ip'_{i+1}}$ in P', $p'_i \rightarrow p'_{i+1}$ or $p'_i \leftarrow p'_{i+1}$ must be true. Furthermore since each point only has a single closest point, when $p'_i \rightarrow p'_{i+1}$ the previous shortcut $\overline{p'_{i-1}p'_i}$, can only be created when $p'_{i-1} \rightarrow p'_i$. In Figure 9 it is shown that when $p_2 \rightarrow p_1$, the length of $\overline{p_2p_3}$ must be at least $\overline{p_1p_2} * \rho$. The polyline can therefore only be simplified when one of the following is true:

- The length of the shortcuts are **increasing** by at least factor ρ each time.
- The length of the shortcuts are **decreasing** by at least factor $1/\rho$ each time.
- The length of the shortcuts are **decreasing from both endpoints** by at leas a factor $1/\rho$ each time and connect at a single point in the middle.

The algoritm needs to consider each combination of two adjacent shortcuts, since the allowed length of a shortcut is based on the previous shortcut. Current well-accepted polygonal line simplification algorithms do not take pairs of shortcuts into account. Algorithms such a the ones presented by *Douglas and Peucker* [1] simplify the polyline just by selecting the points farthest from the polyline until the distance is smaller than ϵ. Another algorithm by *Imai and Iri* [4] generates a shortcut graph $G_{shortcut}(V, E)$, where each vertex represents a point of the polyline. Edges between the vertices are only created when the edge is a valid shortcut, thus the distance is less than ϵ. Since each edge is a guaranteed valid shortcut

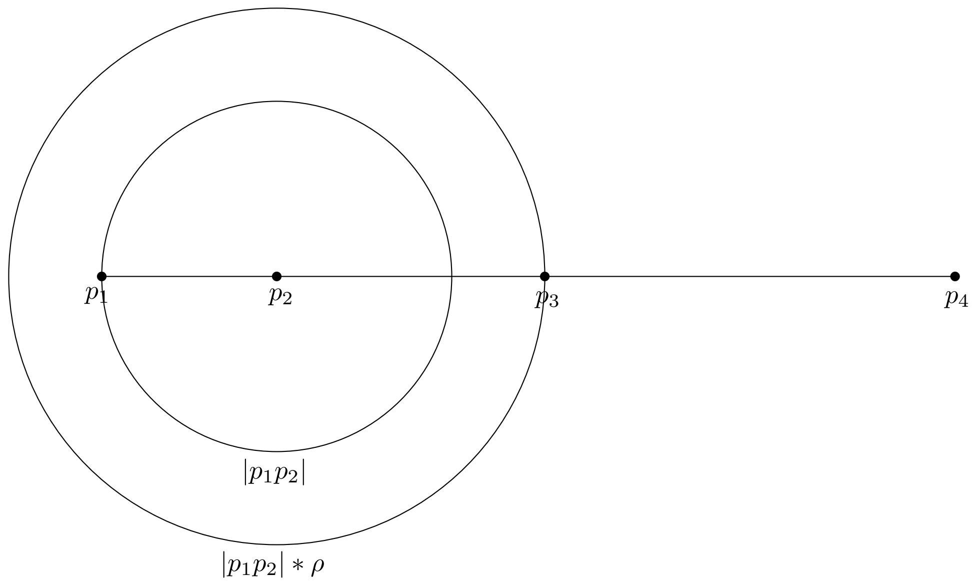


Figure 9: A possible solution with an increasing shortcut distance of at least factor ρ. In order for the shortcut $\overline{p_1p_2}$ where p_2 has p_1 as its closest point, to exists. No point can be placed inside the circle $C(p_2, |\overline{p_1p_2}| * \rho)$. When no other point is inside C we say that p_2 has p_1 as its clear-closest point, $p_2 \to p_1$.

between these two points, the problem of finding a simplified polyline with a minimum number of points is as simple as running a shortest path algorithm, such as the one by *Dijkstra* [2].

In order to ensure the increase of lengths of two adjacent shortcuts, the *Imai-Iri* algorithm is adjusted to allow unique pairs of shortcuts to be marked as valid or invalid. Instead of running a shortest path algorithm on $G_{shortcut}$, it will be applied to an edge graph G_{edge} of $G_{shortcut}$. Each vertex of G_{edge} represents a unique shortcut, an edge from $G_{shortcut}$. An edge between two vertices in G_{edge} represents a valid combination of two shortcuts, this way the increase or decrease of shortcut the lengths can be ensured. An edge between two vertices is added when 1) the two shortcuts are ajacent i.e. they have one point in common and 2) the length of the shortcuts is increasing or decreasing with at least factor ρ.

A shortcut is valid when one of the two points of the shortcut has the other point as its clear-closest point. This means that no other point can be within a certain circle around this point. Also every point p'_i of P' must have either p'_{i-1} or p'_{i+1} as its clear-closest. In figure 10 it is shown that only increasing the shortcut length does not always ensure that all shortcuts are valid for the puzzle. It is shown that despite the increasing shortcut lengths, $p'_3 \to p'_2$ is not valid. Due to the shape of the polyline, p'_0 is closer to p'_3 than p'_2.

Given the polyline $P(p_0, ...p_n)$ and a possible shortcut $\overline{p_ip_j}$ where $p_i \to p_j$, this is not a valid shortcut when another point lies within the circle $C(p_i, |\overline{p_ip_j}| * \rho)$. The radius of the circle is $distance(p_i, p_j) * \rho$. To ensure that the shortest path algorithm cannot select a point that lies within C, any shortcut where a point of P can be chosen inside the cirle is invalid. The shortcut $\overline{p_ip_j}$ is therefore only a valid shortcut, and thus added to G_{edge}, when no point

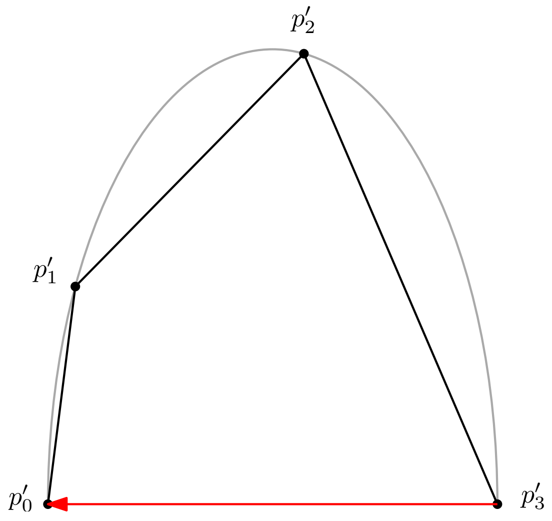


Figure 10: A possible solution with an increasing shortcut length. The solution however is still not valid since the line segment $\overline{p'_2p'_3}$ does not exists when these points would be chosen for the puzzle due to p'_0 being closer to p'_3 than p'_2.

inside C can be chosen. Since there is already an increasing or decreasing length contraint, the next point p_k of shortcut $\overline{p_jp_k}$ will always lie outside of C. However any point after the next shortcut can lie inside C. Therefore the shortcut $\overline{p_ip_j}$ is invalid when the polyline re-enters C again. Each side of the polyline $P_1(p_0, ..., p_i)$ and $P_2(p_i, ..., p_n)$ can only intersect with C once. A vertex will only be added to G_{edge} when 1) the distance of the shortcut to the original is no more than ϵ, 2) the length of the shortcut is longer than d_{min} and shorter than d_{max} and 3) the polylines P_1 and P_2 intsect C at most once. An edge will only be added if 1) the two shortcuts are ajacent, 2) the length of the shortcuts is increasing or decreasing with atleast factor ρ.

Any shortcut in G_{edge} is a valid shortcut and cannot be invalidated by any other shortcut. Any shortest path algorithm can be used to find a path from p_0 to p_n using with smallest number of shortcuts. The simplified polyline can either consist of increasing shortcut lengths or decreasing shortcut lengths. The algorithm is therefore run twice, once on the increasing shortcut lenghts graph G_{edge_inc} and once on the decreasing G_{edge_dec}. It is also possible to decrease the shortcut length from the endpoint to a single point that both results have in common. For this case a bi-directional Dijktra is run on both graphs until a single point is found. All three of these solutions try to find an optimal solution using a single color. It is possible that no solution using only a single color can be used to simplify the polyline. A polyline in the shape of a circle for instance will never be able to be solved using a single color. In this case the algorithm takes the solution that reaches the furthest node and tries to solve the remaining polyline using the three possible options. Each of these colors will be handles as separate polylines and thus be given a different color. If the remaining is unsolvable again the furthest solution is again selected. When even no solution using multiple colors can be found, the polyline will be pre-rendered in the puzzle.This can be the case when the parameters do not allow a large shortcut due to a too small ϵ and short shortcuts are invalid due to a d_{min}.

Algorithm 3 SimplifyPolyline

```
Input: A polyline $P(p_0, ..., p_n)$, maximum error $\epsilon$, margin $\rho$, minimum distance $d_{min}$ and
    maximum distance $d_{max}$
Output: A set of simplified polylines $P'$.
 1: $G_{edge\_inc} \leftarrow$ BuildEdgeGraph($P$, $\varepsilon$, $\rho$, $d_{min}$, $d_{max}$)
 2: $G_{edge\_dec} \leftarrow$ BuildEdgeGraph(reverse of $P$, $\varepsilon$, $\rho$, $d_{min}$, $d_{max}$)
 3: $P' \leftarrow P' \cup$ Dijkstra($G_{edge\_inc}$, 0, $n$)
 4: $P' \leftarrow P' \cup$ Dijkstra($G_{edge\_dec}$, 0, $n$)
 5: $P' \leftarrow P' \cup$ BidirectionalDijkstra($G_{edge\_inc}$, $G_{edge\_dec}$, 0, $n$)
 6: if $P'$ contains no solution then
 7:     $x \leftarrow 0$
 8:     $y \leftarrow$ false
 9:     while $x \neq n$ do
10:         $P'_{inc} \leftarrow$ Dijkstra($G_{edge\_inc}$, $x$, $n$)
11:         $P'_{dec} \leftarrow$ Dijkstra($G_{edge\_dec}$, $x$, $n$)
12:         $P'_{bi} \leftarrow$ BidirectionalDijkstra($G_{edge\_inc}$, $G_{edge\_dec}$, $x$, $n$)
13:         if any of $P'_{inc}$, $P'_{dec}$ or $P'_{bi}$ has a solution then
14:             Add the solutions to $P'$
15:             $y \leftarrow$ true
16:         else
17:             Add the furtest solution to $P'$
18:             $x \leftarrow$ furthest node of the solution
19:         end if
20:     end while
21:     if $y =$ false then
22:         Mark the polyline to be pre-rendered
23:     end if
24: end if
```

3.3 Graph Coloring

Now that each polyline has one or more simplified solutions and each solution can consist of a single color, the goal is to assign a color to each polyline such that a minimum number of different colors is used. Each polyline can have multiple solution:, an increasing, decreasing and bi-directional one. A single solution must be chosen. Each simplified polyline has an area in which no other point of the same color can be placed, the uniform of all circles C of all points in the polyline. The circles are used to indicate that no other point could be placed inside them, or the shortcut would be invalid. The same is true for coloring multiple polylines. No point of a single polyline can be placed inside the uniform of all cirles of another polyline when both polylines have the same color.

The graph-coloring problem is a well known problem. Given a graph $G(V, E)$, assign a color to each vertex such that 1) no two vertices connected by an edge have the same color and 2) a minimum number of colors is used. Computing the exact optimal solution is known to be a NP-hard problem. Several heuristics exist for solving the graph coloring problem in more efficient time. A greedy or *First Fit* method is to assign vertices the first available color. The result of the number of used colors depends on the order of the vertices in which they are assigned a color. The *First Fit* algorithm can be run in $O(n)$ time where n is the number of vertices. Since the quality of *First Fit* is based on the ordering of the vertices, other heuristics try to order the vertices for an optimal *First Fit* approach. *Largest degree ordering* or *LDO* orders the vertices based on their degree, coloring the vertex with the most attached edges first. *LDO* has an upper bound of using at most one more color than the largest degree. Several other heuristics such as *saturation degree ordering* (*SDO*) try to order the vertices optimally for the *First Fit* coloring. The algorithm of this thesis uses a combination of both *LDO* and *SDO*. Al Omari [5] shows that the running time of the combination of *LDO* and *SDO* is equal to that of *SDO* but yields better results in general.

The coloring of the polylines can be converted to the graph coloring problem. Let every polyline that can be colored using a single color, be a vertex in G. Add an edge between two polylines P_i and P_j, when either a node from P_i lies within the area of P_j or vice versa. The simplification of the polylines can however have multiple solutions for a single polyline. The chosen solution for the polyline based on minimizing 1) the number of dots the solution contains and 2) the number of coloring edges that will connect to this vertex. When two polylines are adjacent they can have the same color as long as the shortcut distance is increased with at least factor ρ. When a polyline has multiple adjacent polylines who can have a similar color, not all of those can have the same color. Given three polylines P_0, P_1 and P_2 where P_0 is adjacent to P_1 and P_1 is adjacent to P_2. When the shortcut distance of P_1 is larger than that of $P_0 \times \rho$, P_0 and P_1 can have the same color. The same case is true for P_1 and P_2. Also since the distance between P_0 and P_2 is large enough they can also be of equal color. This situation is shown in Figure 11 (a). The result of this for the graph coloring is a graph without edges between P_0, P_1 and P_2 as in Figure 11 (b). The graph coloring algorithm can color all three polylines with a single color since there are no edges between them. Where each pair of polylines could have the same color the combination of all three cannot. In Figure 11 (c) it is shown that the combination of all three using a single color will cause P_1 to not be drawn in the puzzle. This problem shows that even though each individual pair of polylines can have the same color, it does not grantee that the combination of all those pairs results in a valid coloring of the dots for the puzzle.

The solution to this is to consider adjacent polylines that can have the same color, as a single vertex in the graph coloring graph G. This means that two vertices are merged into a single vertex representing both polylines. Two polylines are merged when 1) the two polylines are adjacent and 2) the polylines can have the same color. Given the example of Figure 11 the polylines can be merged as $p_0 - p_1$, $p_1 - p_2$ or $p_2 - p_3$. Wich polylines to merge can have a huge impact on the amount of colors needed to color the other polylines. The heuristic for graph coloring orders the vertices on the degree, where the highest degrees are seen to be harder to color. Therefore the two polylines that result in a merged vertex with the least degree, are merged first.

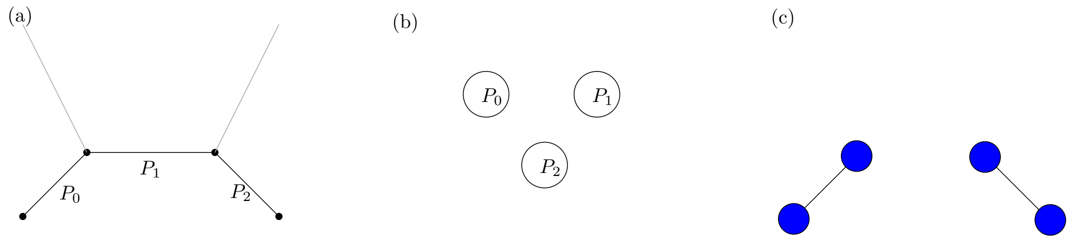


Figure 11: (a) Given the simplified polylines P_0, P_1 and P_2 each polyline can have the same color with each other. (b) This generates a graph with three vertices for each polyline without any edges since they can have the same color. (c) Applying the single color to the puzzle dots results in a solution where only polylines P_0 and P_2 are drawn.

3.4 Time Complexity

This section discusses the time complexity of each step in the algorithm. Given the input graph with n vertices and m edges the first step of splitting the input into separate polylines can be can be executed in $O(m)$ time. The algorithm for splitting into polylines ensures that each edge is only visited once.

The second step involves building an edge graph and running a Dijkstra shortest path on the edge graph. Given a polyline with k vertices the edge graph can contain at most k^2 vertices and k^4 edges. Since a polyline can have n vertices, k is equal to n. Dijkstra's shortest path algorithm runs in $O(m + n \log n)$ time on a graph with n vertices and m edges. This means that the shortest path algorithm on the edge graph runs in $O(n^4 + n^2 \log n^2)$ time. The edge graph is build in $O(n^4)$ time. This makes the polyline simplification take $O(n^4)$ time when all polylines can be solved using a single color. When multiple colors are used, the shortest path algoritm will be run multiple times per polyline. This can happen at most the number of times of the number vertices of the polyline, when each time only a single shortcut to the next point is used. This can therefore happen at most n times, thus the maximum running time is $O(n^5)$ time.

The graph coloring heuristic takes $O(l^3)$ time, where l is the number of vertices of the coloring graph. At most, each edge of the input can be a vertex in graph coloring, thus $O(m^3)$ time where m is the number of edges of the original input illustration. Computing whether two polylines can be the same color, is the same as computing whether any point lies inside a set of circles. This can be computed in $O(n \log n)$ time. This is computed for each pair of polylines thus $O(m^2)$ times. The total running time of the graph coloring takes $O(m^2 n \log n + m^3)$ time.

Splitting into polylines	$O(m)$
Polyline simplification	$O(n^5)$
Graph coloring	$O(m^2 n \log n)$
Total	$O(n^5)$

Table 1: The running times for each step of the algorithm where n is the number of nodes and m the number of edges of the input graph.

4 Results

This section shows the results of the algorithm and the effect of the different parameters. The input for each puzzle was a planar graph scaled to fit into bounds of 15 by 15 cm. Unless specified, the values of the parameter are $\epsilon = 3$mm, $\rho = 25\%$, $d_{min} = 4.5$ mm and $d_{max} = \infty$. The diameters of the dots are 2.4 mm of one color, 3 mm two 2 colors, 3.6 mm for three colors and 4.2 mm for four colors. The test results are an average over 18 different puzzles. The resulting puzzles using these default parameters can be found in Appendix A.

4.1 Puzzle Quality

Automatically generated puzzles do not always generate ideal results. In this section the quality of the generated puzzles is described. The quality of the puzzle is measured by the criteria specified in Section 2.1.

Figure 3 and Figure 12 show two automatically generated puzzles using the default parameters. Both puzzles show that the algorithm is capable of producing a well-solvable puzzle. Almost every part of the initial illustrations is converted into the puzzle and only 0.4% and 11% of the original illustration is pre-rendered. The parts that are pre-rendered are either very short (thus shorther than d_{min}) or have a high curvature (thus ϵ is too large for d_{min}). In both cases the short pre-rendered parts are due to the puzzle having multiple intersections close to each other. Since the algorithm splits the input into polylines, node will always be placed at the intersections. If the distance between the intersection nodes is smaller than d_{min} this will always be pre-rendered.

The specifications state that in a good puzzle no two dots can be too close or overlap. The algorithm uses the d_{min} parameter to ensure the distance during the polyline simplification. However, as can be seen in Figure 12, two dots can still be closer to each other than d_{min}. This is the case when two different polylines are close to each other. Each polyline is simplified separately and combining the result of both simplifications can cause this conflict. Since these dots represent two different polylines, the dots will never connect to each other. This issue will never happen when the original illustration does not contain multiple polylines close to each other. Another case where two dots can overlap is when two intersections are close to each other. When not all polylines are pre-drawn, a node will always be placed at the intersection. When two intersections are close, two dots will be close aswell. Again since the distance is less than d_{min} the two dots will never be connected via a line segment. However, it is possible that one dot obscures the other one such that for the puzzler the puzzle becomes unsolvable. When the input of the algorithm does not contain close polylines or close intersections this wil not hapen.

For the puzzler the generated puzzles are well-solvable. The distance buffer of 25% is big enough to connect the closest dot without any issue.The number of used colors on averge is also acceptable. An averge number of 7.33 different colors are needed with an maximum of 11. As seen in both puzzles the different colors are still well distinguishable. By optimizing the parameters for each puzzle the number of dots and/or the number of colors can be reduced to produce a better puzzle.

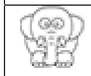
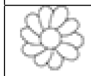
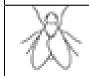
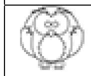
Image	Nodes	Edges	Distance (cm)	Points	Colors	Time (s)	Pre-Drawn (%)
	668	677	115.9	76	9	49.6	7.4
	880	895	91.2	60	10	5.7	21.7
	1287	1296	134.8	98	8	24.5	11.0
	828	828	82.9	47	4	1951.2	9.3
	1371	1382	144.9	97	9	16.9	4.0
	1029	1030	97.5	63	8	455.4	10.1
	729	732	75.1	46	7	32.2	1.0
	1027	1037	111.8	58	6	4.0	0.4
	916	923	93.9	49	8	32.1	0.3
	1075	1079	104.8	78	8	81.2	3.9
	636	641	64.7	25	6	29.3	3.5
	849	856	88.9	49	9	215.2	0.0
	855	858	86.4	63	7	158.3	4.8
	561	563	59.7	30	5	421.6	0.0
	1718	1732	164.5	98	8	53.9	15.6
	2917	2924	190.6	137	11	63.5	21.8
	704	704	73.0	51	5	751.2	1.7
	751	753	77.1	48	4	142.3	4.0

Table 2: A table showing the input and output for each puzzle. The first colomn shows a small representation of the image. Nodes and Edges show the number of nodes and edges for input graph for this illustration, Distance is the sum of the length of all edges from the input. Points is the number of puzzle points in the generated puzzle using the default parameters and Colors is the number of different colors. Time defines the time taken to compute the puzzle and finally the percentage distance of edges that is pre-drawn, is given.

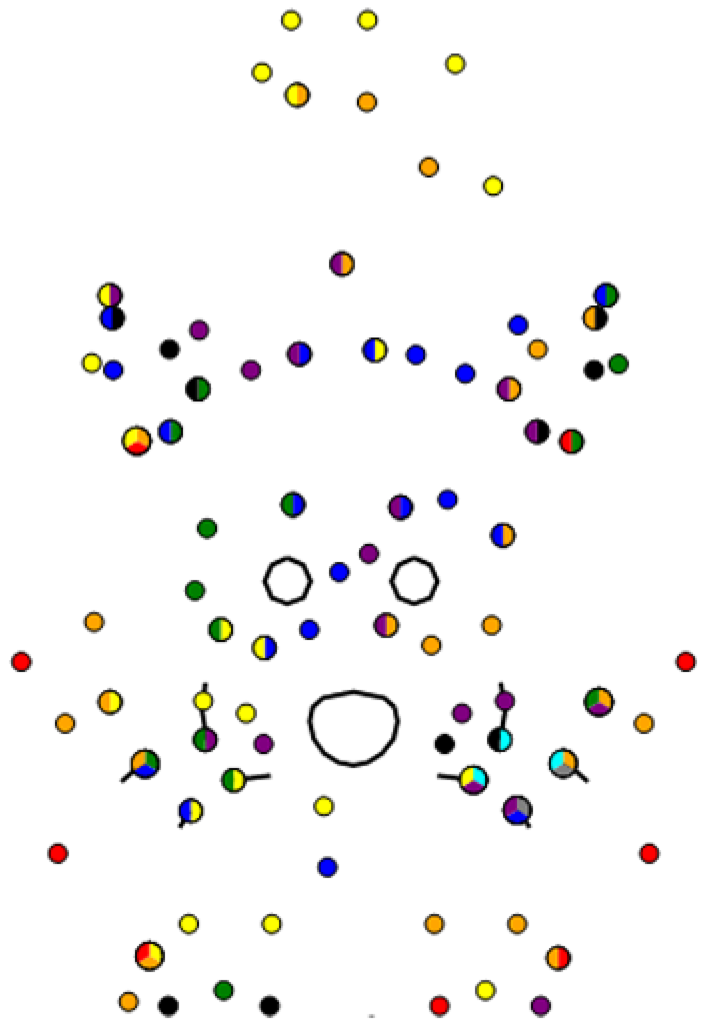


Figure 12: An example of a connect the closest dot puzzle.

4.2 Parameter Results

When automatically generating the puzzles the outcome can be influenced by changing the input parameters. In this section we change each parameter over a range of different values. For each parameter, the results are measured by number of points, number of colors, number of points with 2 or more colors, maximum number of colors for a single point, number of segments that cannot be solved with a single color and the percentage of the puzzle that is unsolvable (pre-rendered). The most important and interesting results are described in this section, the full results can be found in Appendix B.

4.2.1 Minimum Distance

The minimum distance d_{min} is used to set a minimum distance between two dots. The algorithm only allows two dots whose distance from each other is more than d_{min} to be part of the simplified polyline. As shown before this does not affect dots of multiple polylines.

The value of d_{min} is tested over a range from 0, no restriction on the minimum distance, to 15, two connecting dots need a minimum distance of 15 mm. Figure 13 shows the average percentage of the illustration that is unsolvable and will be pre-rendered. As shown, a larger d_{min} results in a longer distance of the puzzle to be pre-rendered. A puzzle with lots of pre-rendered polylines lowers the quality of the puzzle, but dots can also not be too close to each other. Polylines become unsolvable due to either 1) the polyline is to short and distance between the endpoints is smaller than d_{min} or 2) the curvature of the polyline is too large, meaning that a shortcut with a minimum length of d_{min} can no longer sustain an error margin less than ϵ.

4.2.2 Maximum Distance

The maximum distance d_{max} is used to limit two connecting dots to have a maximum distance from each other. Figures 14 and 15 show the change in the number of points and colors for values of d_{max} from 9 mm to 45 mm. The graphs show that a smaller d_{max} requires more dots since dots shortcuts have a short maximum length. Interestingly having more dots and shorter shortcuts reduces the number of colors. This shows that d_{max} can be used to reduce the number of colors at the expense of a more dense set of dots. The full results as shown in Appendix B also show that a too small d_{max} can lead to more unsolvable polylines. The d_{max} of 9 mm shows a 30% rate of unsolvable segments, this is logical considering the options for shortcuts are limited to only a length between 4.5 mm and 9 mm. This can be solved by reducing the minimum shortcut distance to allow for more options.

4.2.3 Distance Buffer

The factor between the closest dot and any other dot is denoted as ρ. A larger factor of ρ means a larger area for each shortcut in which no dot of the same color can be placed.

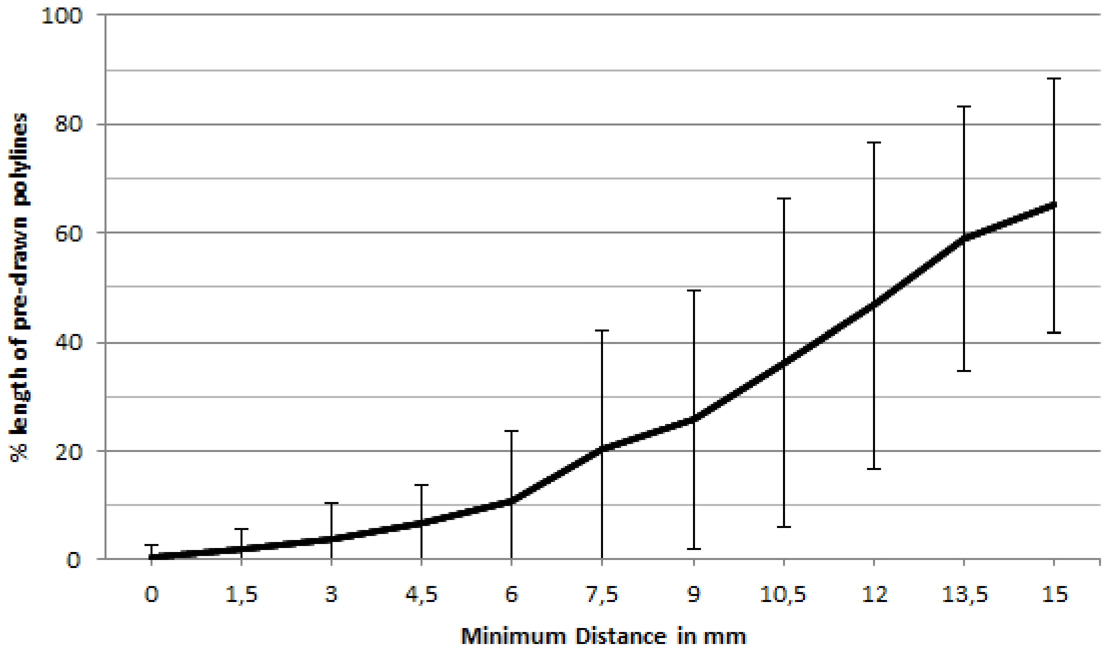


Figure 13: The effect of changing the d_{min} parameter on the percentage distance of the input that will be pre-rendered. It is shown that a higher d_{min} results in a puzzle with longer pre-rendered lines.

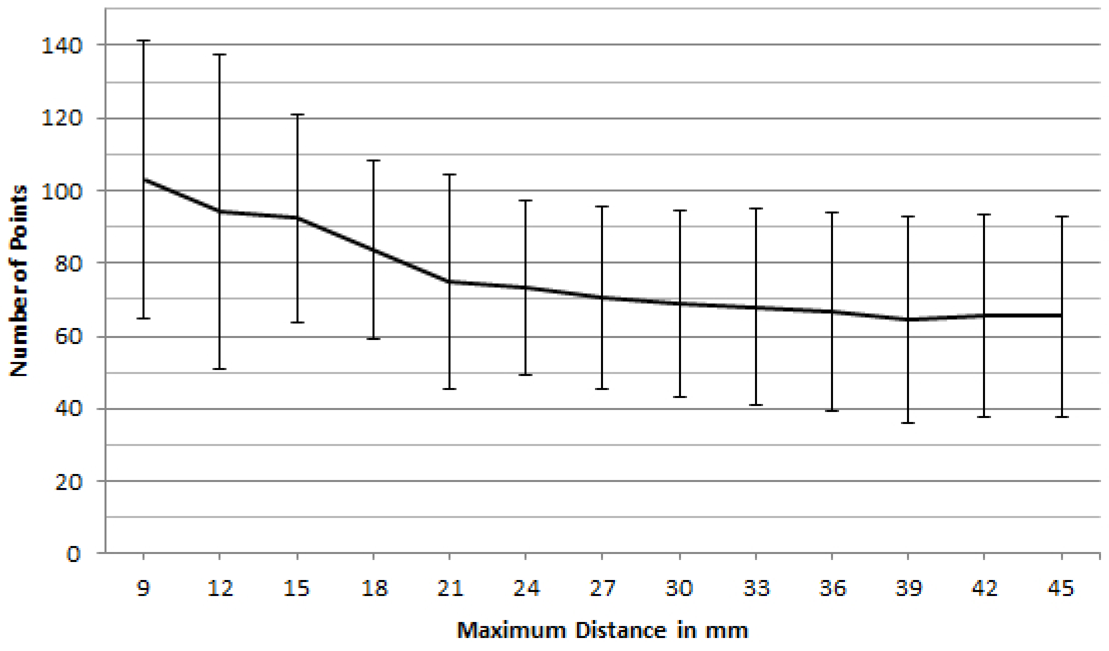


Figure 14: The effect of changing the d_{max} parameter on the number of dots. It shows that a smaller d_{max} requires more dots.

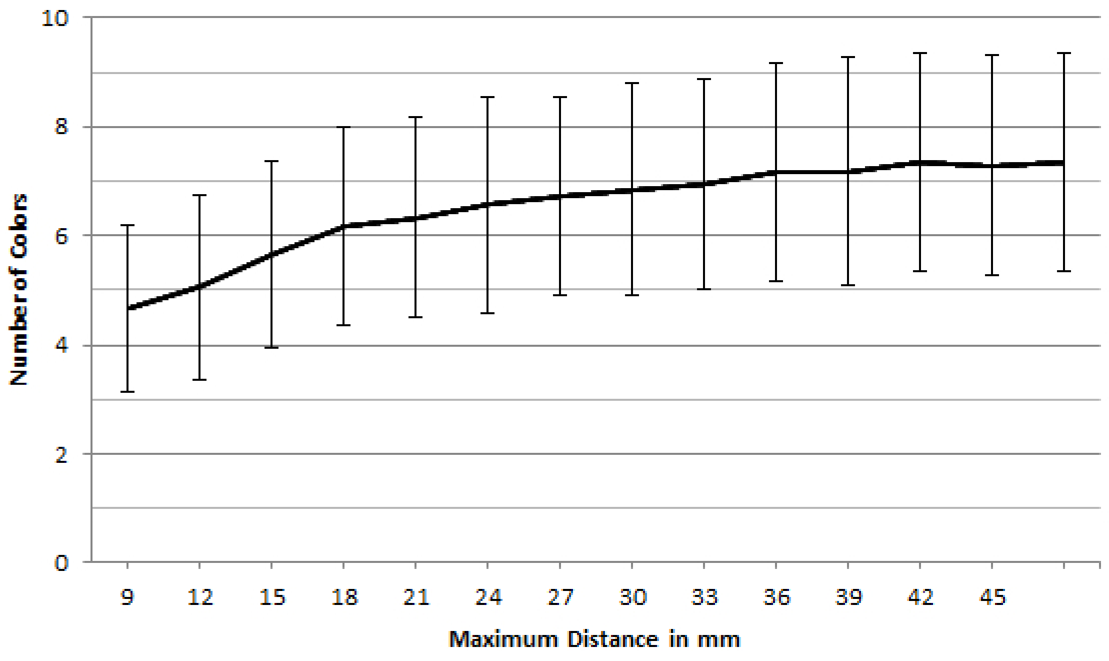


Figure 15: The effect of changing the d_{max} parameter on the number of different colors. It shows that a smaller limit to the maximum shortcut length reduces the number of colors.

Figure 16 shows the number of colors used for ρ ranging from 0% to 100%. It is clearly visible that a larger margin drastically increases the number of colors needed to generate a puzzle. This result is as expected considering that the area in which a color cannot be reused becomes larger for a larger ρ. Furthermore, the results show a large decrease of the number of points and an increase of the number of segments with multiple colors for an increase of ρ. This is due to the fact that a larger ρ means that for the same initial shortcut a second shortcut with the same color must be longer than for a shorter ρ. Therefore the minimum length for a shortcut increases faster and the algorithm has less options to simplify a polyline using only a single color.

4.2.4 Error Margin

The error margin ϵ ensures a certain level of quality of the resulting image. A larger ϵ allows for a larger distance to the original image. Figure 17 shows that a smaller ϵ requires many more points. The ϵ values of 0.6 mm and 1.2 mm result in fewer points since this error margin is too small and results in 40% and 23% of the puzzle to be pre-rendered. As already shown before with d_{max}, a shorter shortcut length results in fewer different colors. Figure 18 shows the increase of colors is not as large, values 0.6 mm and 1.2 mm excluded due to the large pre-rendered parts. While more points are needed for a smaller ϵ this is only the case on the curved polylines. Straight polylines will result in long shortcuts no matter the ϵ and thus increase the number of colors. The results show that the number of colors are mostly influenced by the values of $\epsilon < 3$ mm.

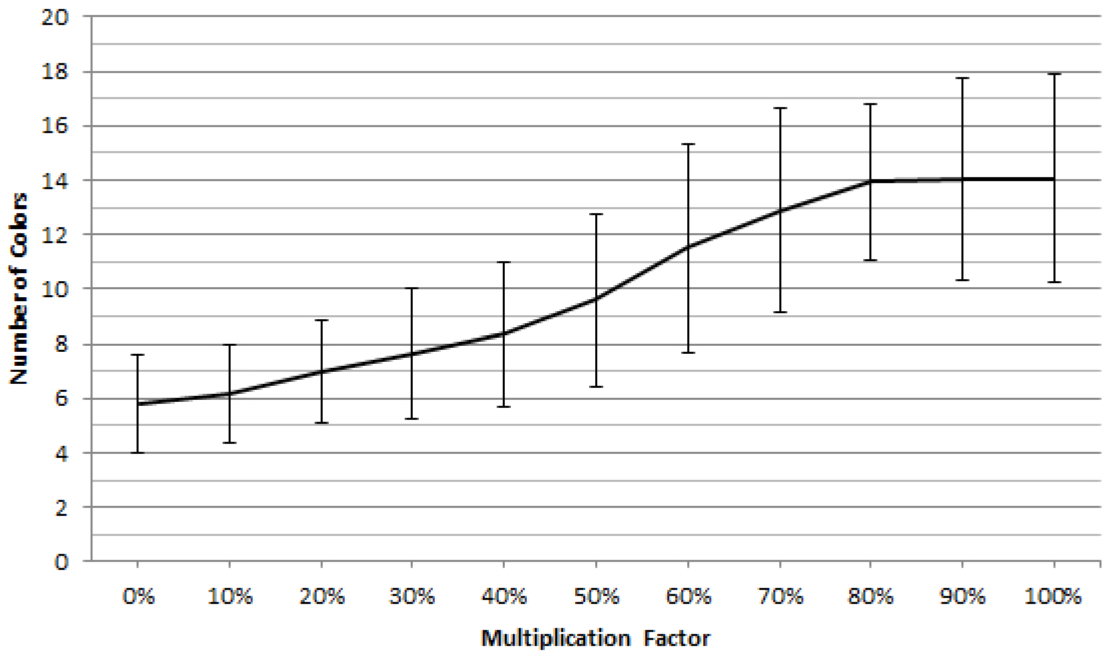


Figure 16: The effect of changing the ρ parameter on the number of colors. It shows that a smaller ρ requires fewer colors.

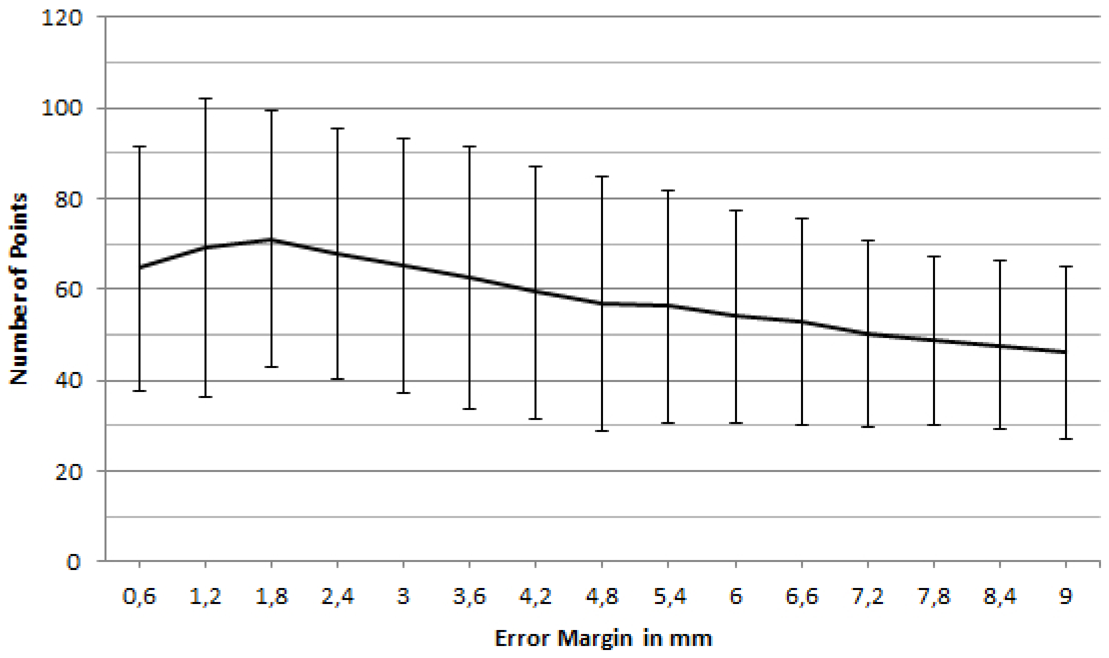


Figure 17: The effect of changing ϵ on the number of points. It shows that a smaller ϵ requires more points.

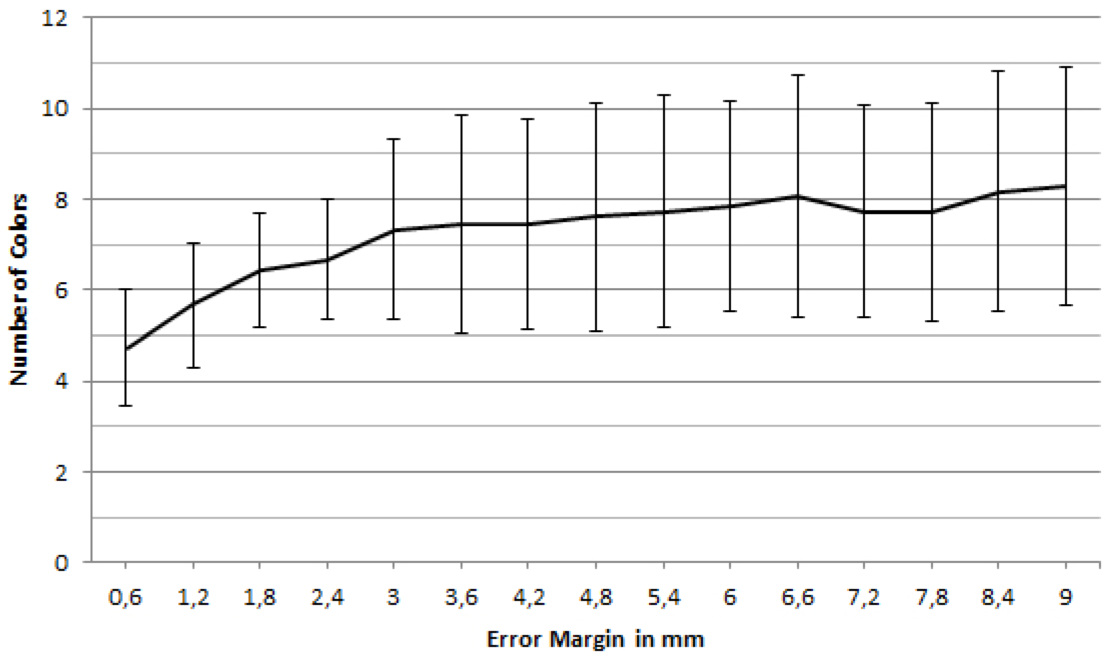


Figure 18: The effect of changing ϵ on the number of colors. It shows that the number of colors is mostly influenced at the lower ϵ values.

5 Conclusion and Future Work

In this thesis we presented a new type of line puzzle. Based on the idea of connect the dots puzzles, pairs of dots need to be connected by line segments to reveal an illustration. Where traditional connect the dots puzzles use number or letters to indicate which dots to connect, our puzzle connects dots based on the color and distance between the dots. From each dot a line segment must be drawn to the closest dot of the same color. When a dot has multiple colors, a line must be drawn to the closest dot for each color. Furthermore, a few variants of the puzzle are described. Each variant makes the puzzle easier or harder to solve.

In Section 3 we presented an algorithm that generates a puzzle based on a line drawing. The proposed algorithm does not solve the problem as a whole but instead solves the problem in smaller stages. The input, a line drawing in graph form, is split into separate polylines. Each polyline is simplified to a small set of dots that can be used for the puzzle. This simplification is done by the creation of an shortcut edge graph. This graph ensures that the result of any shortest path in this graph consists of a valid set of points that can be used for the puzzle. After each polyline is solved individually a color is assigned to each of the polylines using a graph coloring heuristic.

The provided algorith is capable of generating a well-solvable puzzle from a given line drawing in reasonable time. The results show that the algorithm is capable of generating a puzzle from most illustrations using the same parameters. By changing the parameters for each puzzle the results can be optimized to procuce an even better puzzle.

5.1 Future Work

One of the most important aspects of a new type of puzzle is whether it is actually a fun puzzle. While this thesis describes the new puzzle and presents an algorithm for it, the most important questing is whether people like the new puzzle. Following research could conduct a user study on these puzzles. Whether the puzzles are any fun and also whether the automatically generated puzzles are equally fun compared to the hand-made puzzles.

While the algorithm prevents two nodes from being too close to each other when they should connect, two nodes can still overlap when they are part of different polylines. This is due to the algorithm solving each polyline individually. While this is not a problem for most illustrations, these artifacts can prevent the algorithm from generating good puzzles for some illustrations. Future reseach might look at solving the problem where no two points can be closer than d_{min}. Also, since the graph coloring is done after the puzzle points are chosen, sometimes more colors are used than needed. It is possible that the number of colors for the puzzle can be reduced by selecting a different set of points. Currently the algorithm is optimized for using the least amount of points first. It would be interesting if the algorithm can consider the effect on the number of colors when the points are being selected.

Finally we describe multiple variants of connect the closest dot puzzles. It would be interesting whether the current algorithm can generate these puzzles as well or what the differences are. It is also possible that other variants are more fun for the puzzler. Future work can look at all these aspects for each variant of the puzzle.

6 References

References

[1] Thomas K. Peucker David H. Douglas. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature bibtex. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, Oct 1973.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1):269–271, 1959.

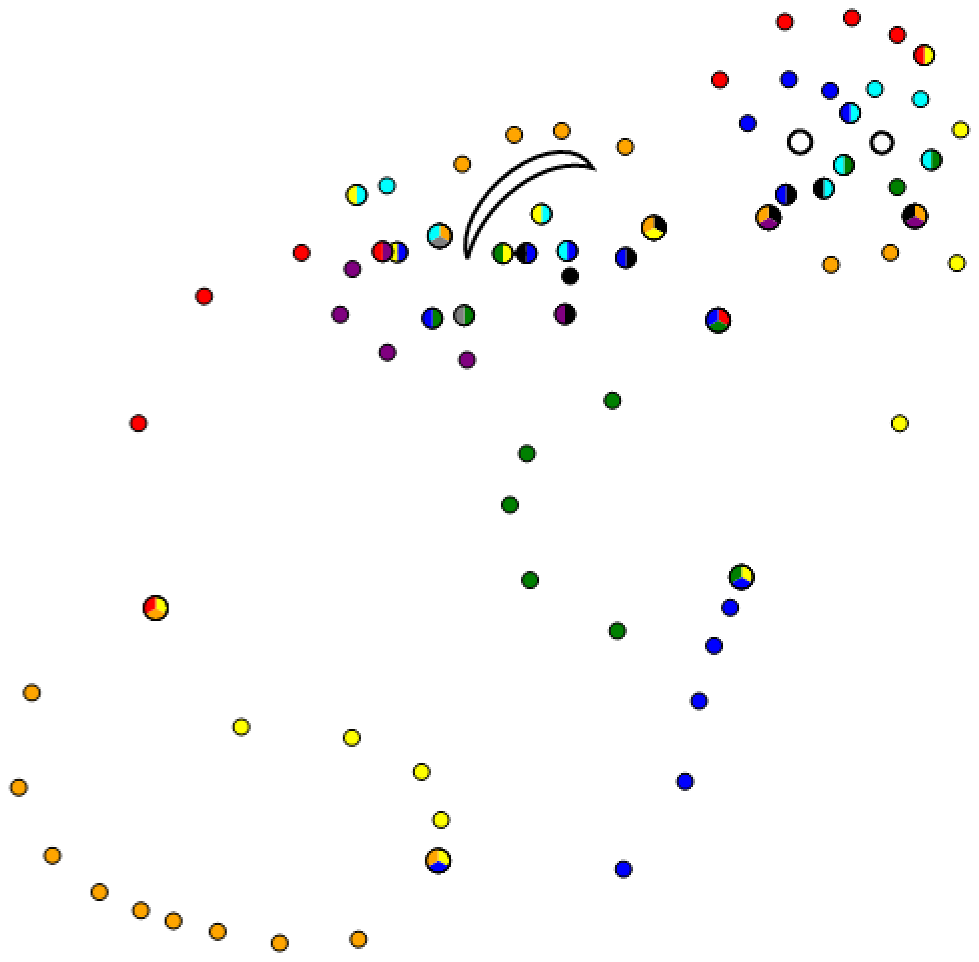
[3] Marc van Kreveld Frank Staals, Maarten Löffler. Clear unit-distance graphs. *EuroCG*, 29:213–216, March 2013.

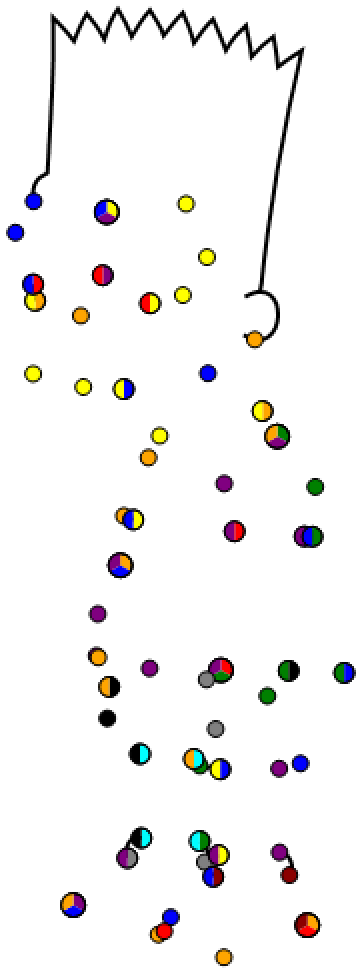
[4] Masao Iri Hiroshi Imai. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31–41, 1986.

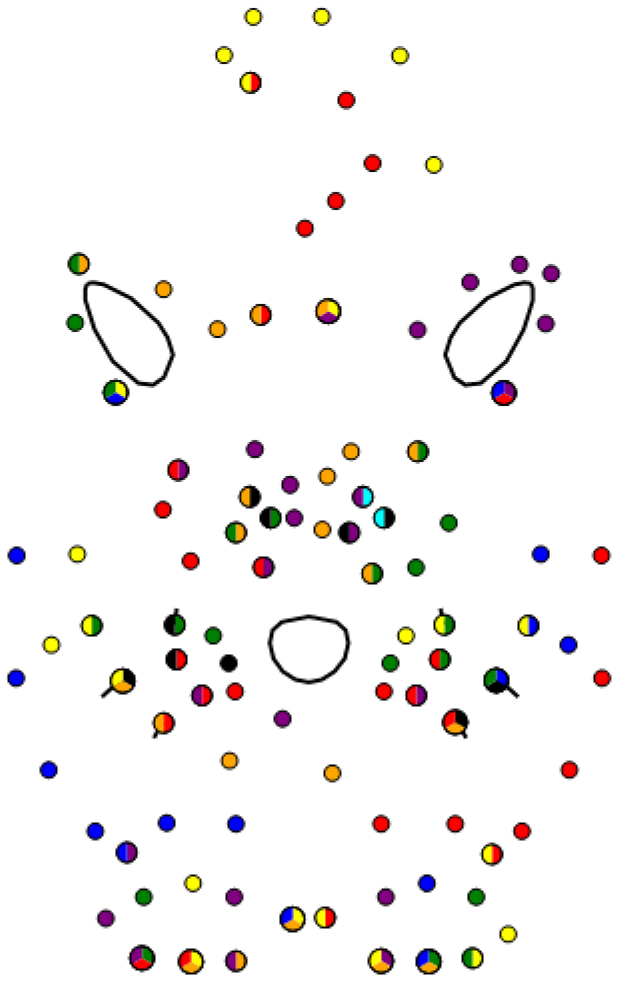
[5] Khair Eddin Sabri Hussein Al-Omari. New graph coloring algorithms. *American Journal of Mathematics and Statistics*, 2(4):739–741, 2006.

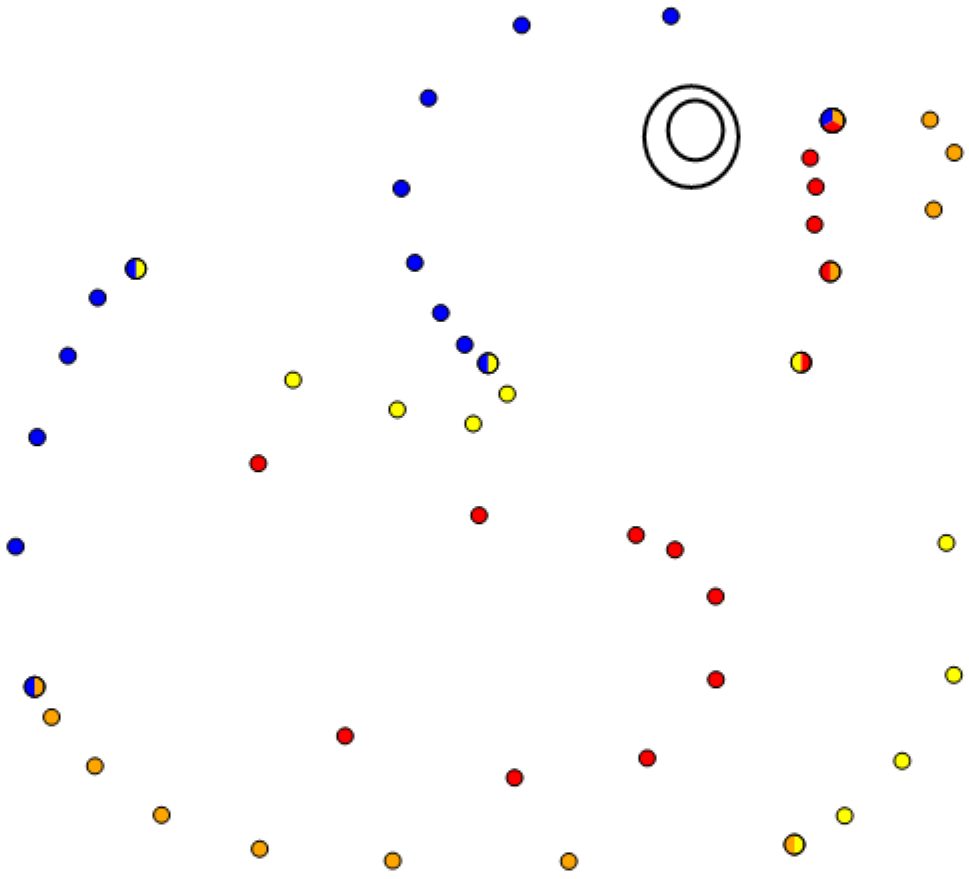
7 Appendices

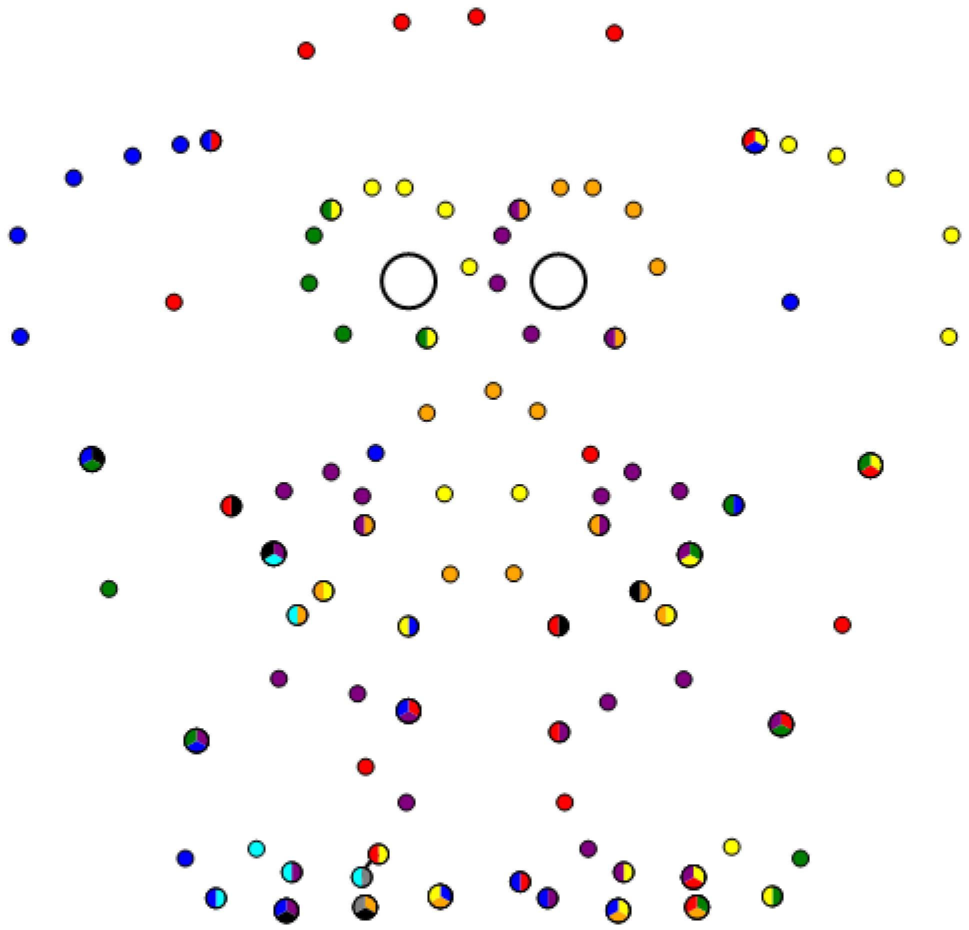
A Puzzles

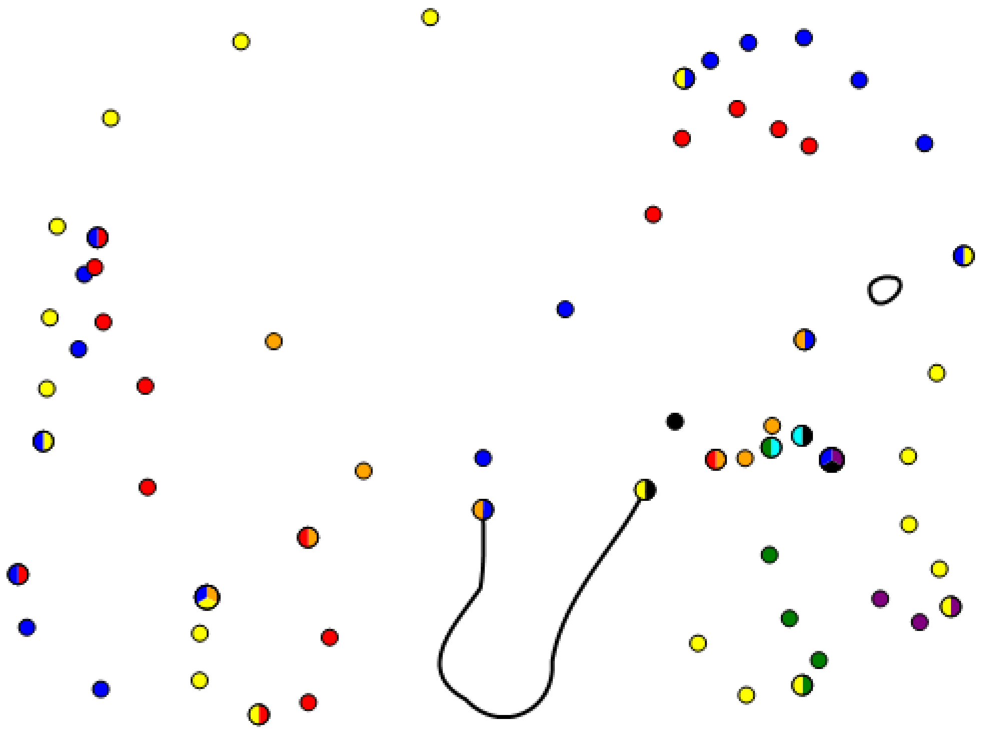


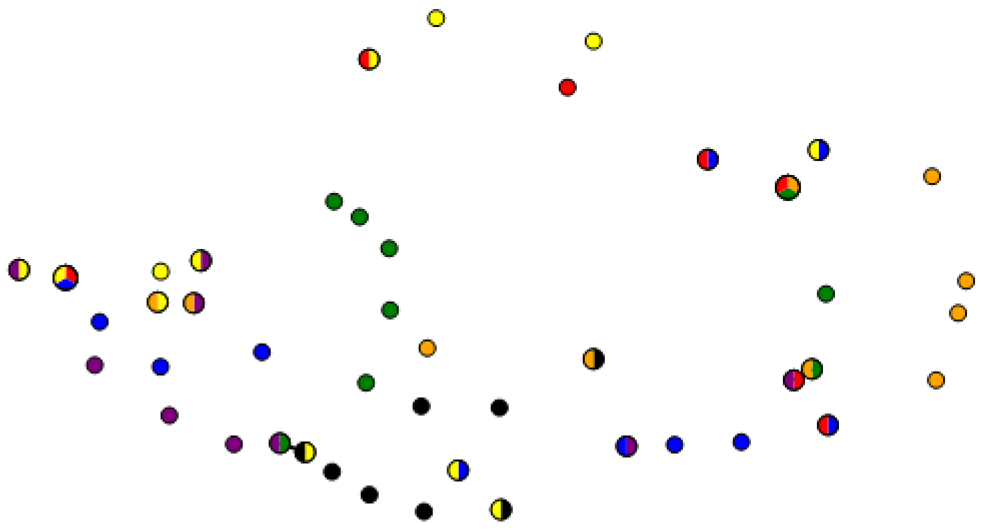


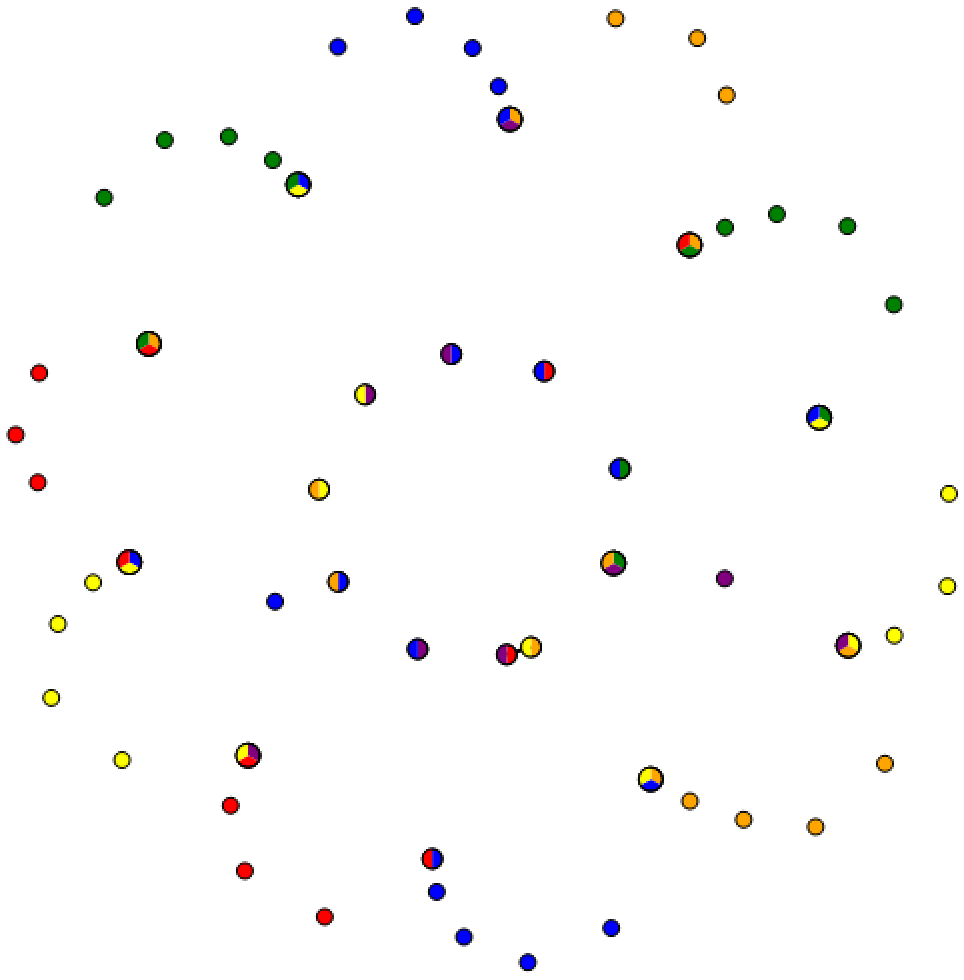


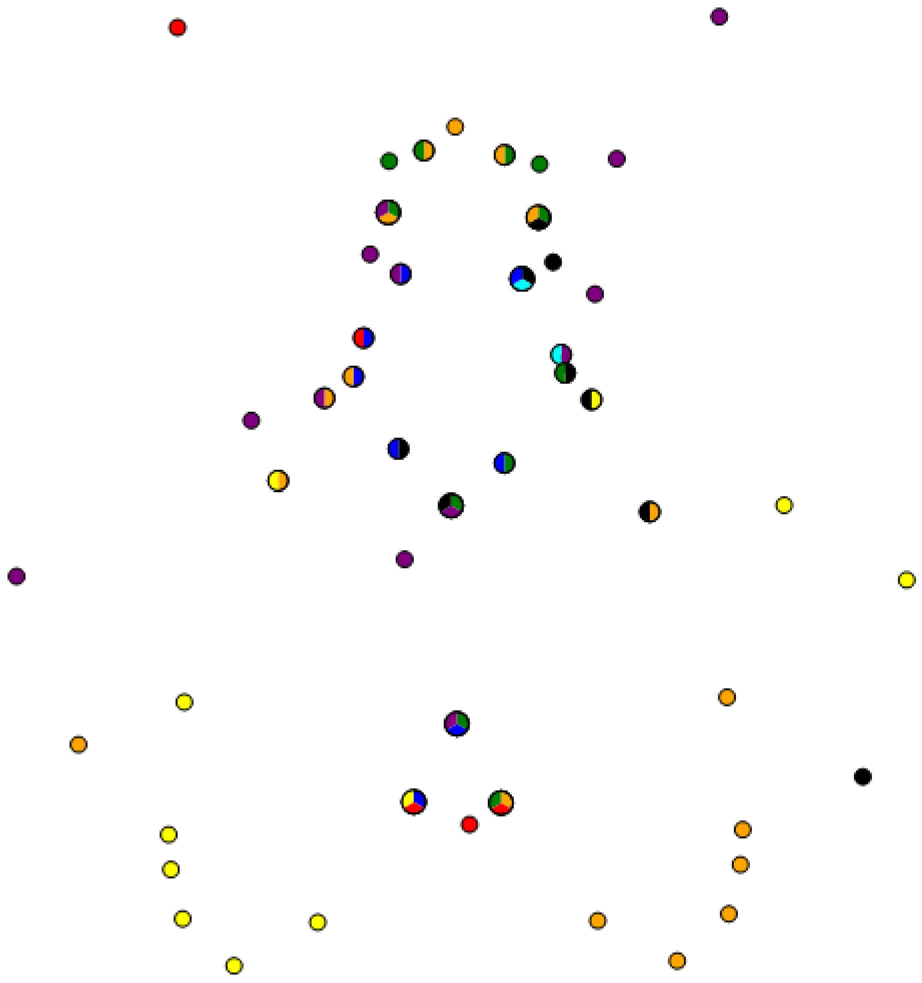


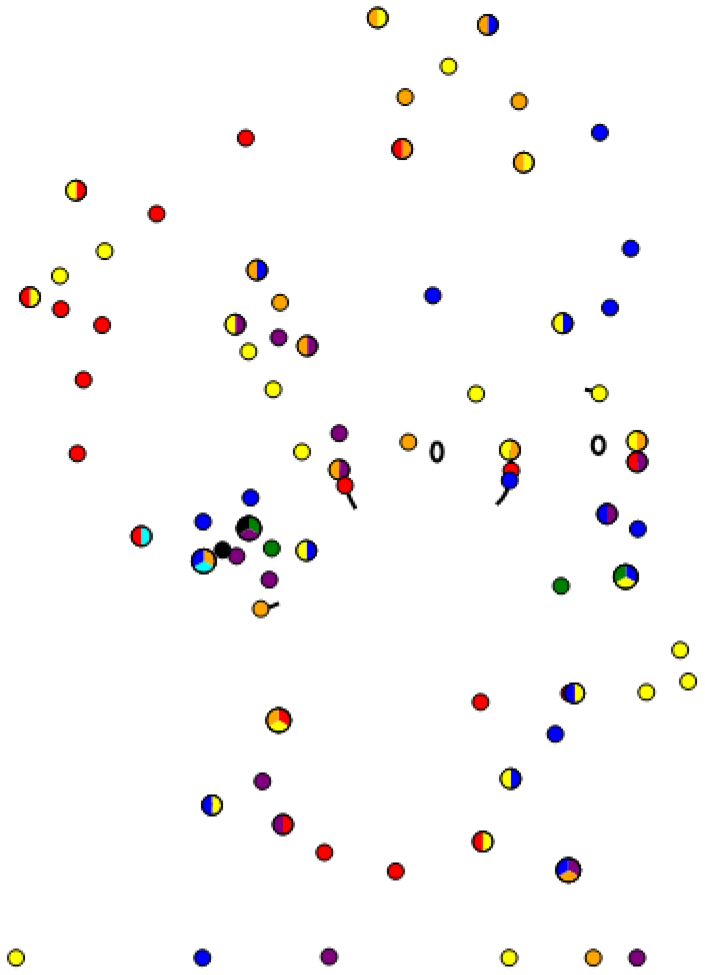


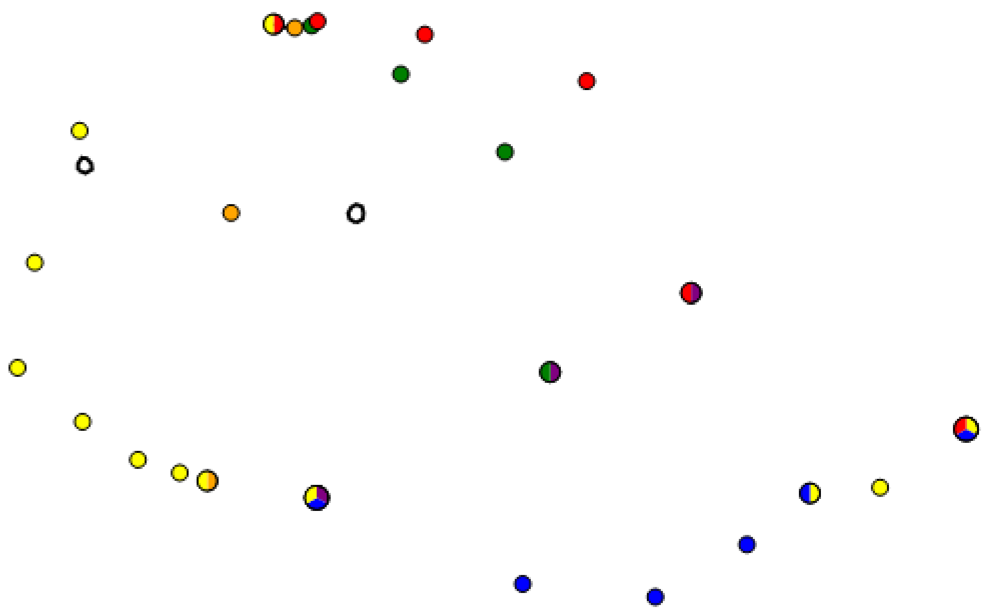


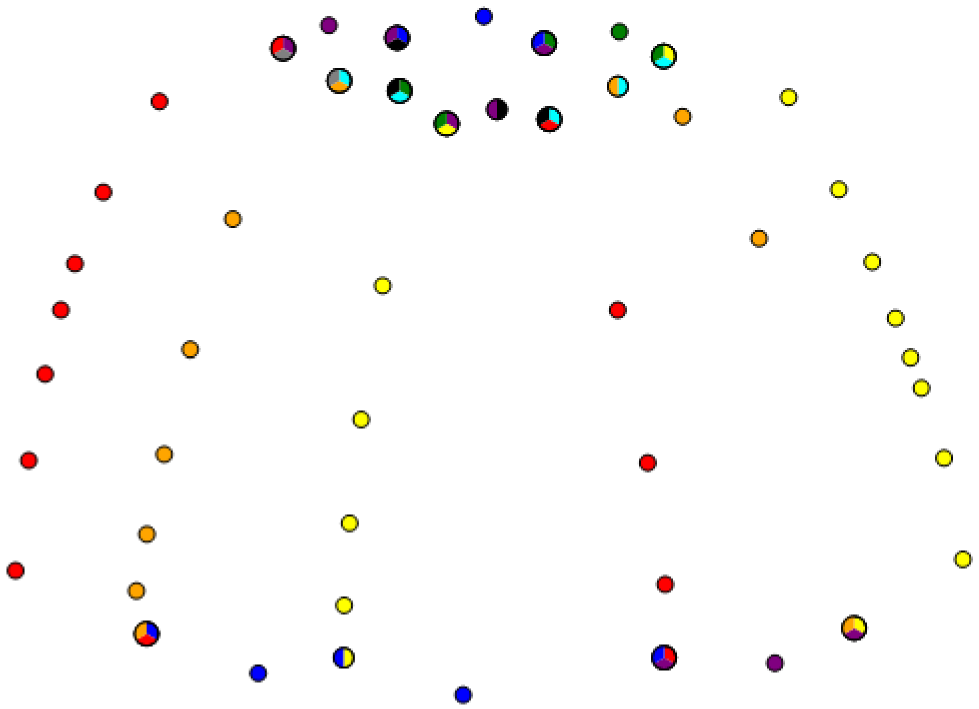


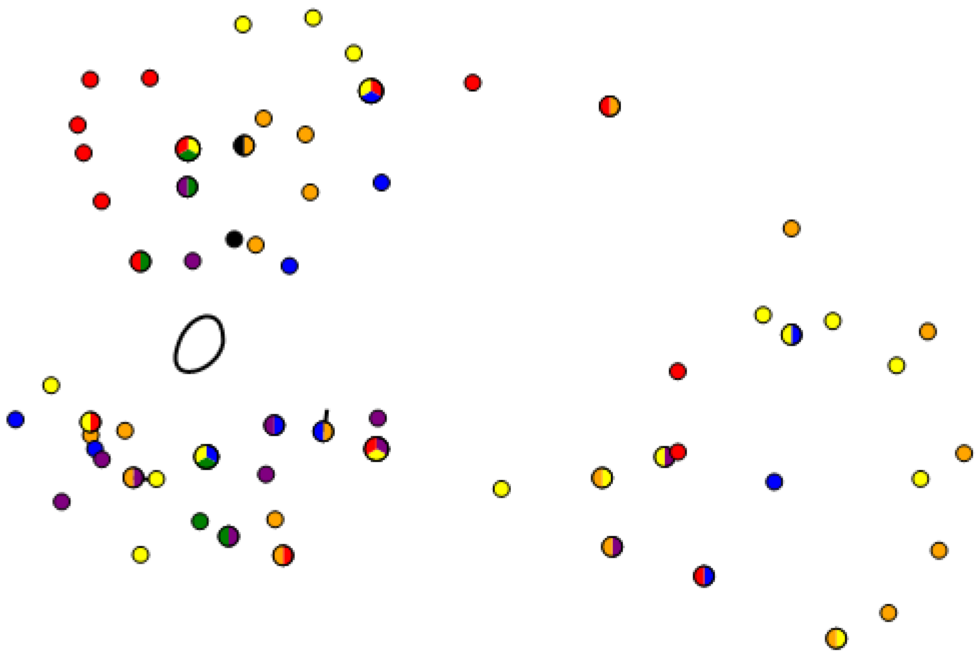


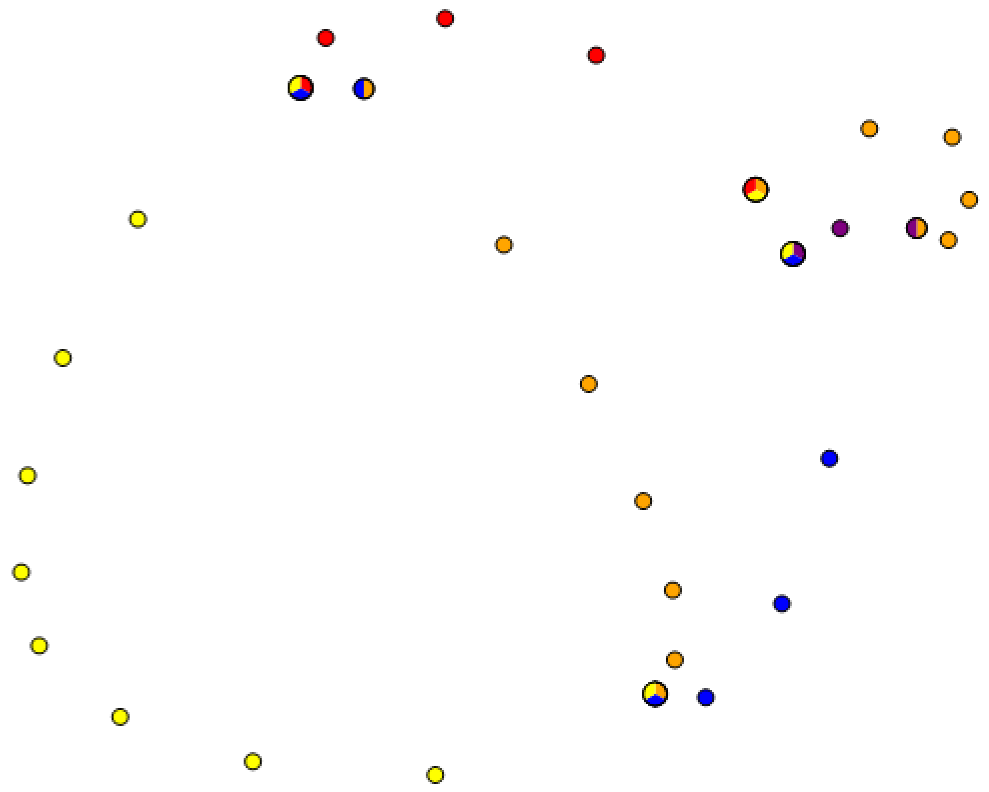


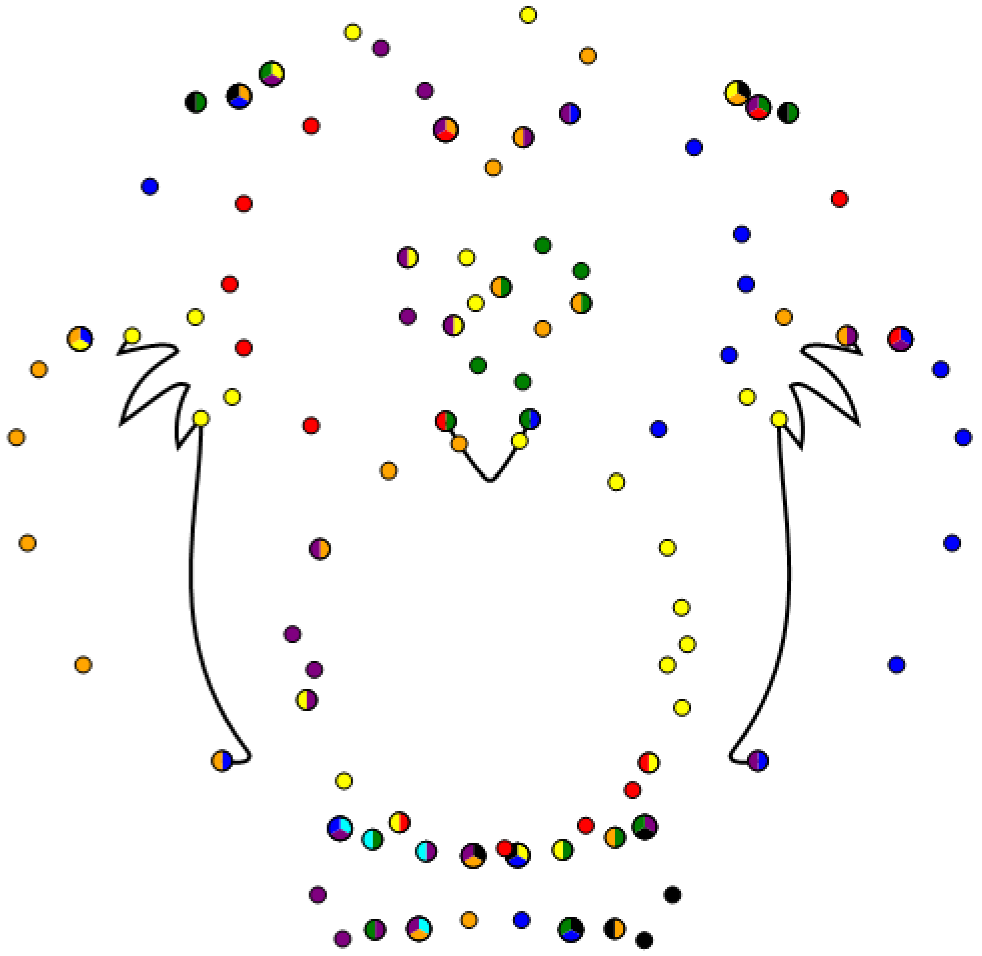


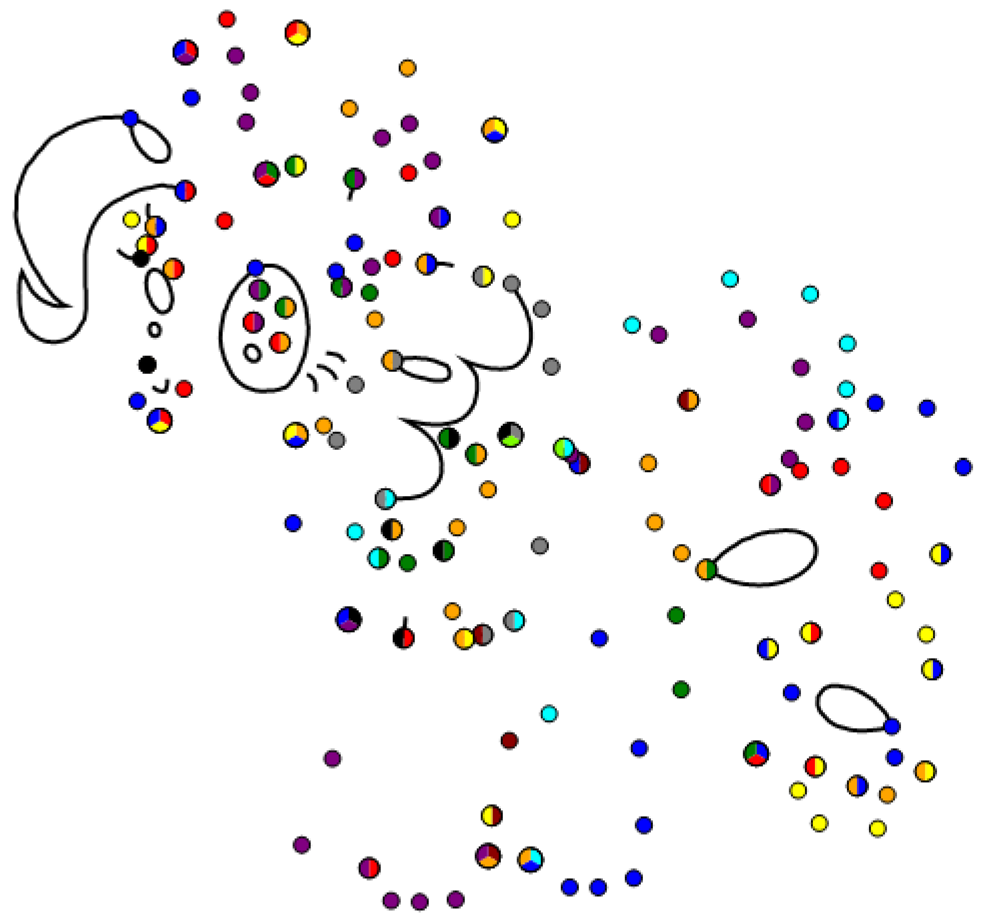


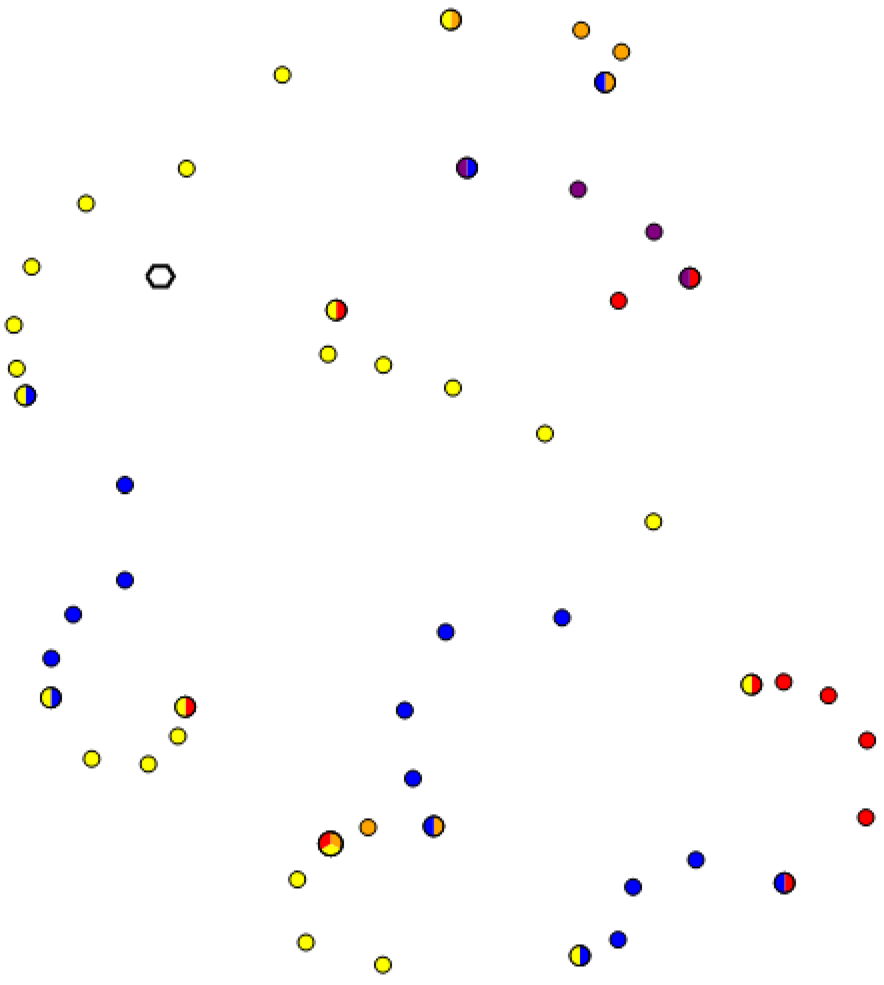


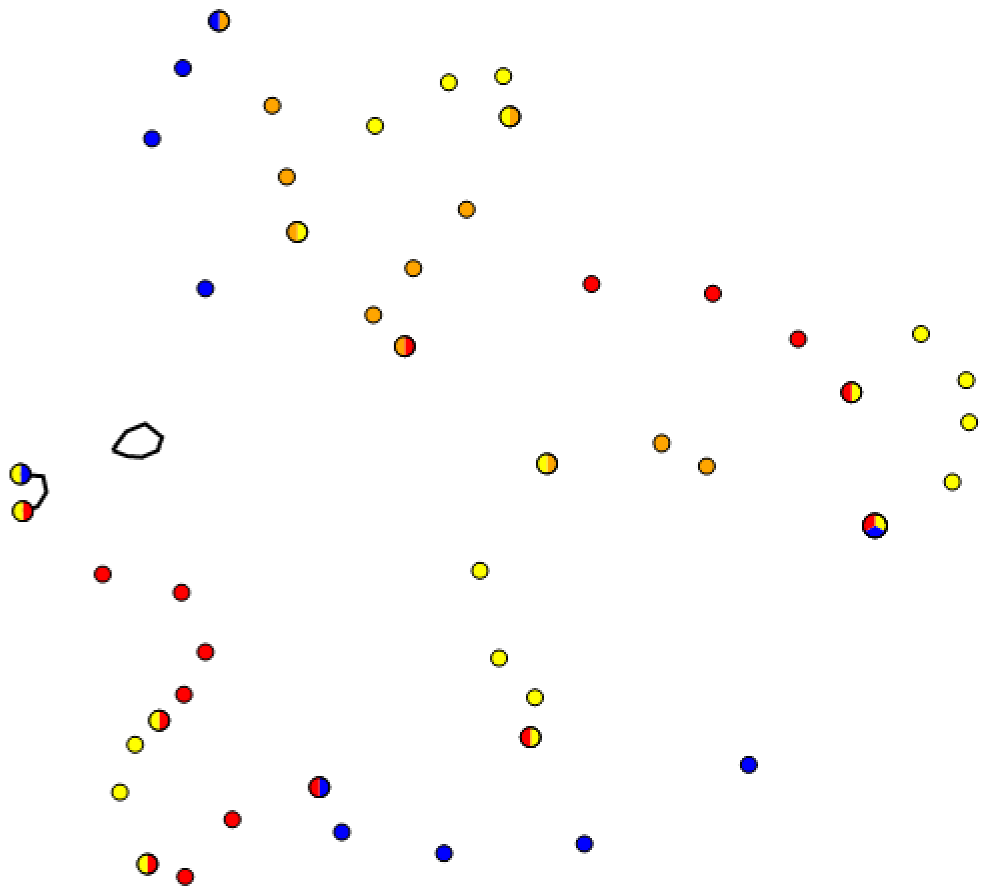












B Results

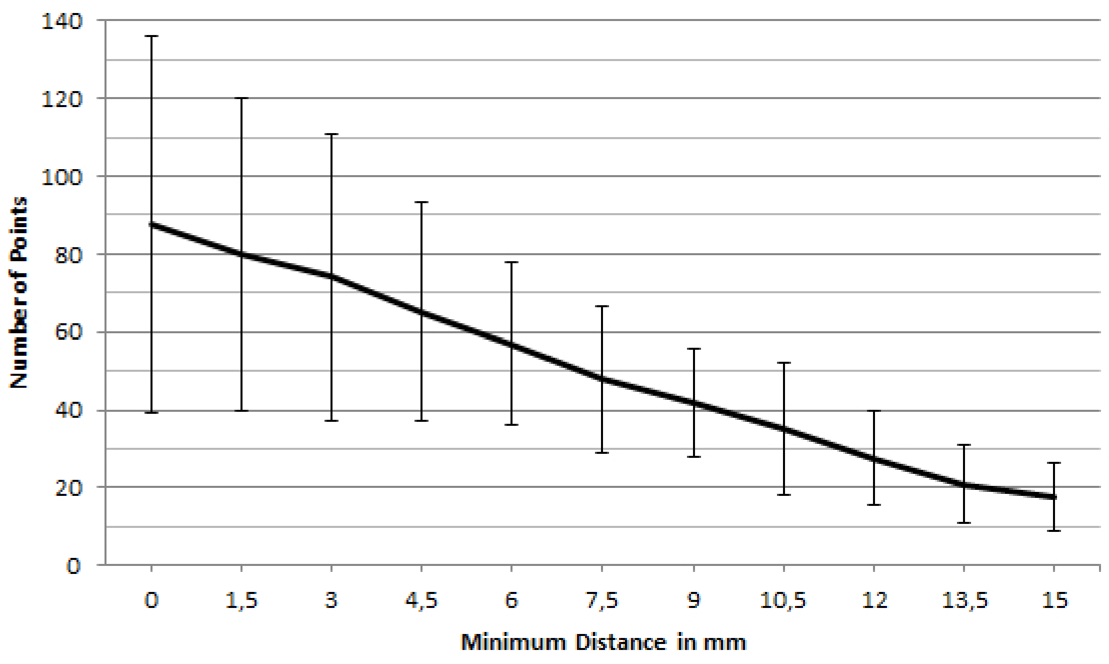


Figure 19: The effect of changing the d_{min} parameter on the total number of points.

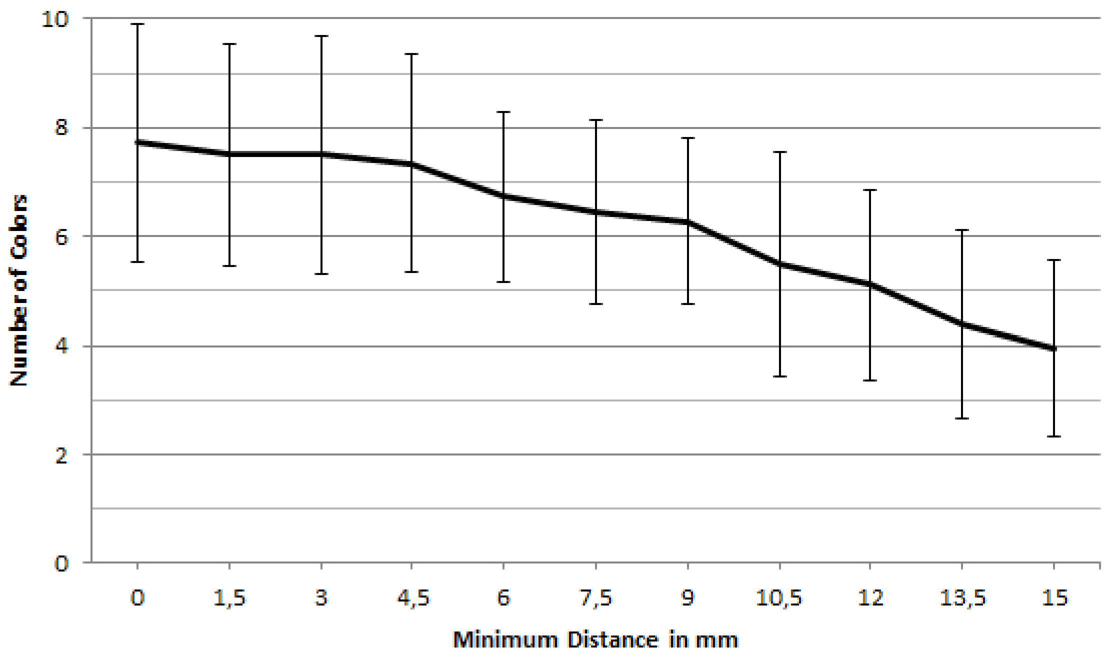


Figure 20: The effect of changing the d_{min} parameter on the total number of colors.

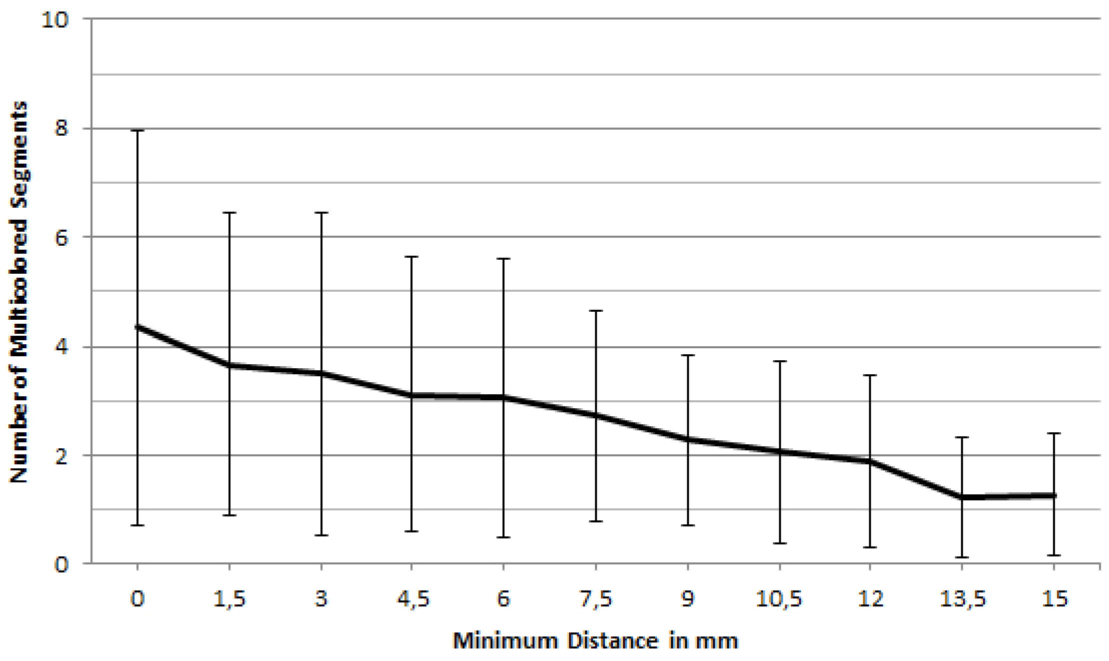


Figure 21: The effect of changing the d_{min} parameter on the number of segments that could not be solved using a single color.

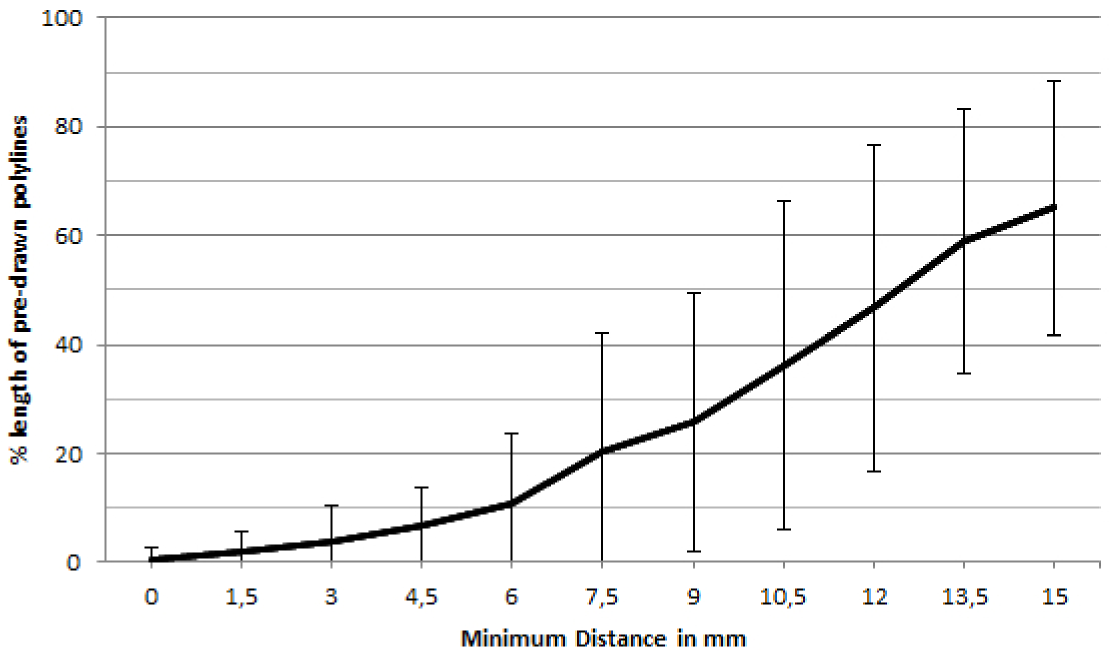


Figure 22: The effect of changing the d_{min} parameter on the percentage distance of the input that will be pre-rendered.

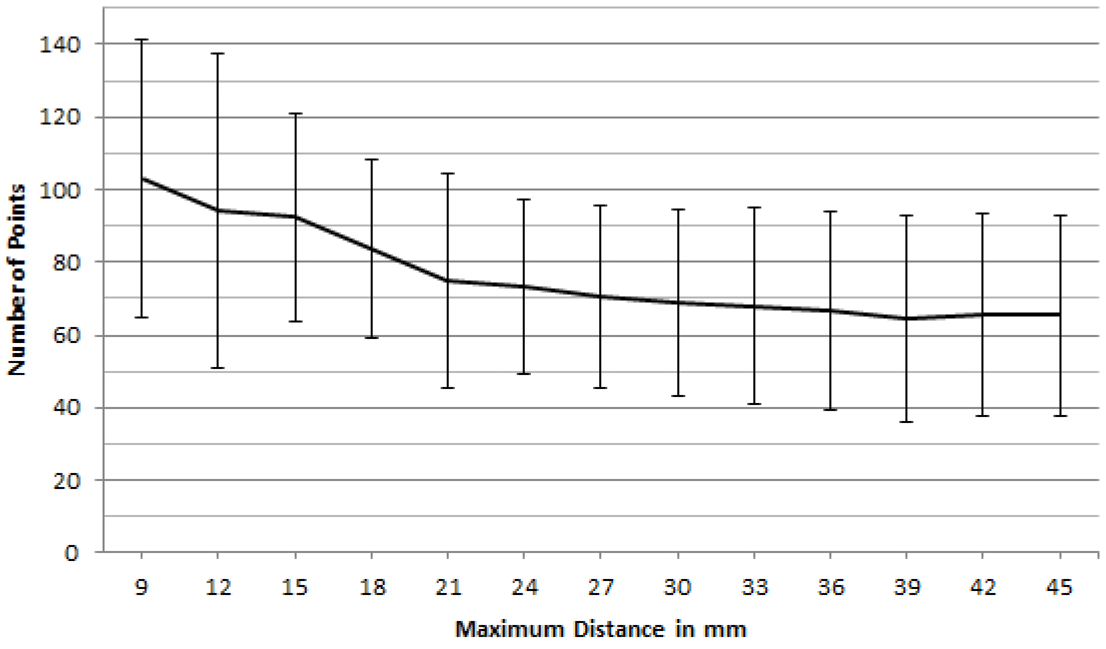


Figure 23: The effect of changing the d_{max} parameter on the total number of points.

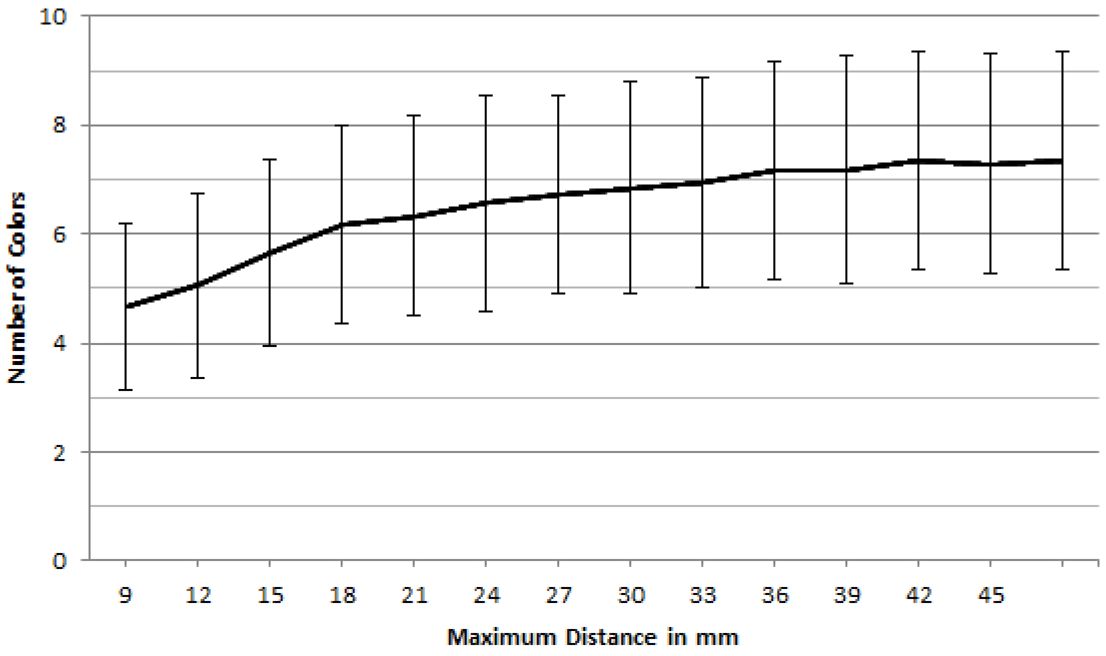


Figure 24: The effect of changing the d_{max} parameter on the total number of colors.

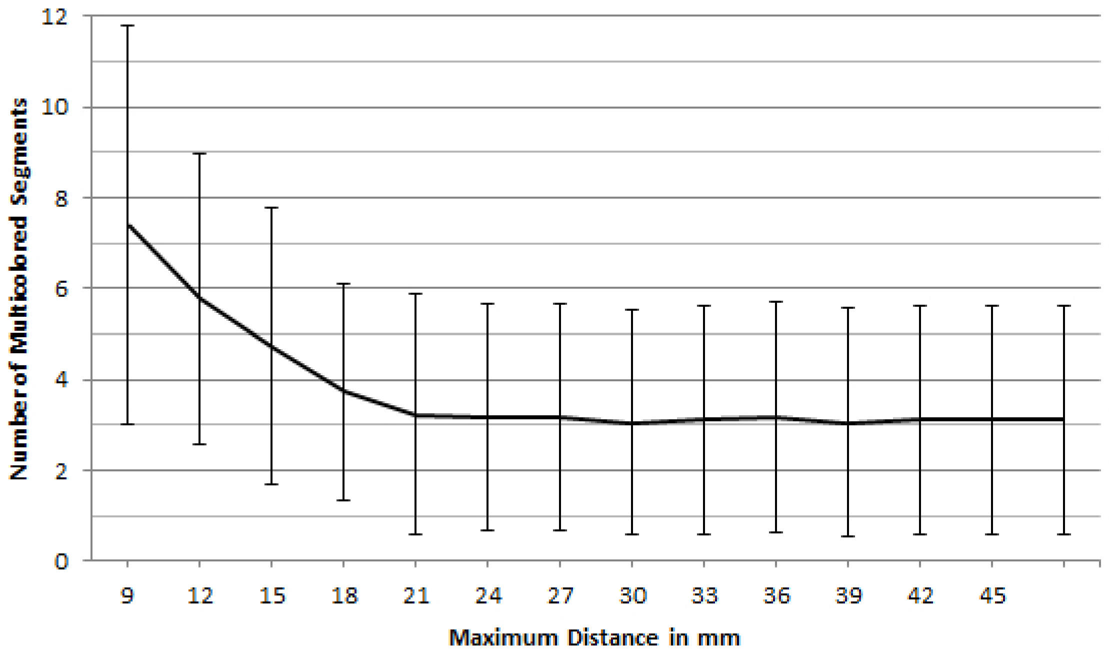


Figure 25: The effect of changing the d_{max} parameter on the number of segments that could not be solved using a single color.

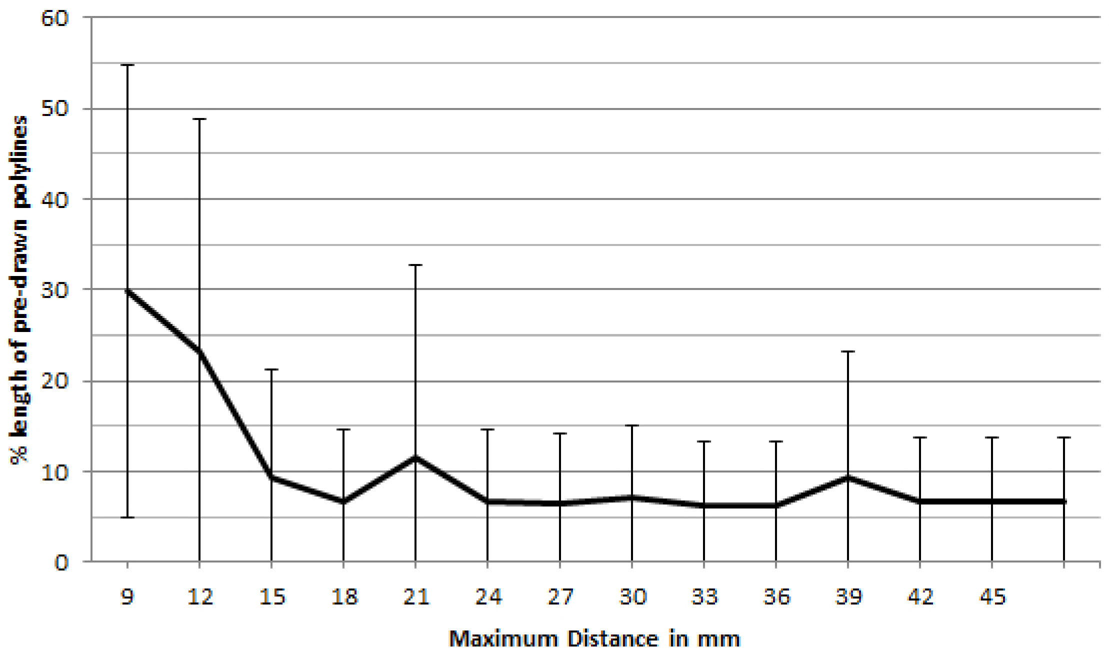


Figure 26: The effect of changing the d_{max} parameter on the percentage distance of the input that will be pre-rendered.

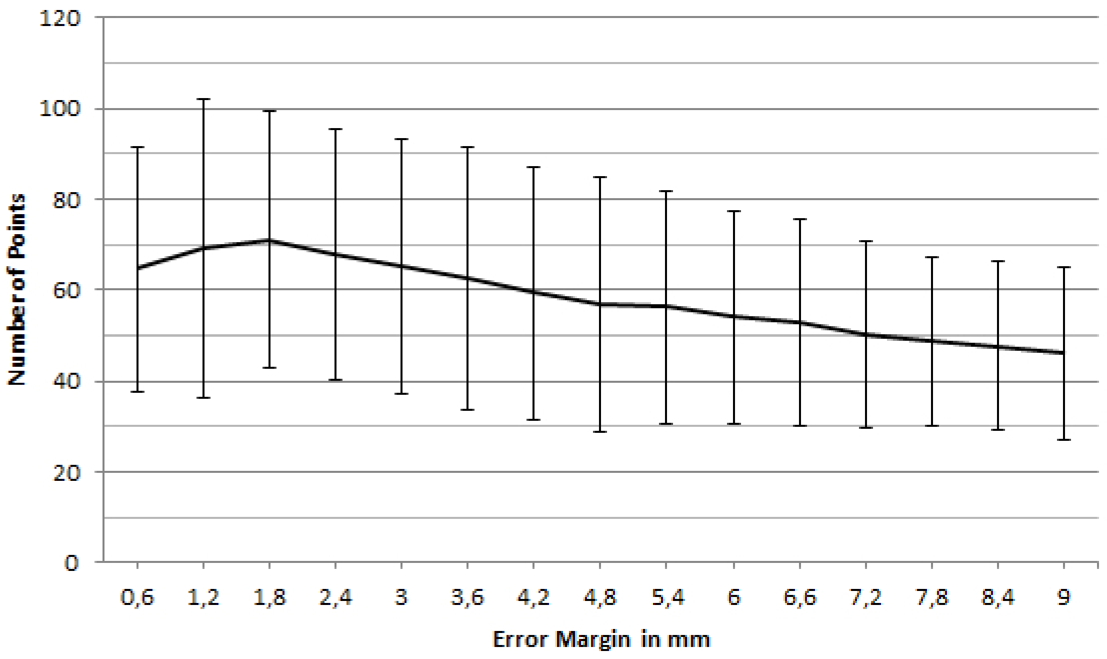


Figure 27: The effect of changing the ϵ parameter on the total number of points.

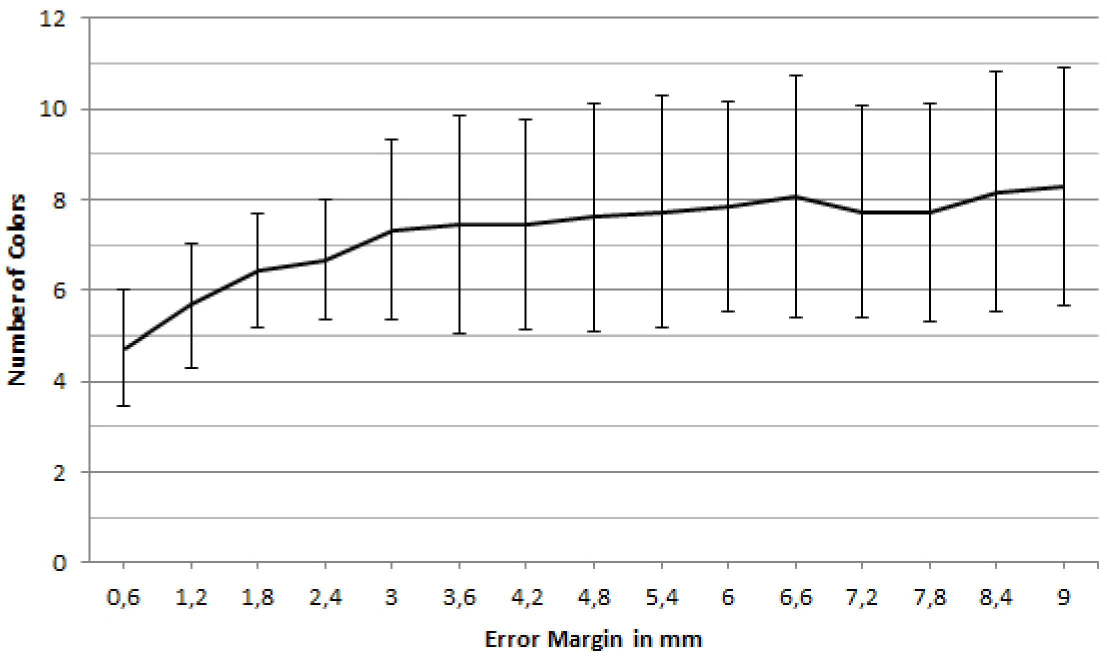


Figure 28: The effect of changing the ϵ parameter on the total number of colors.

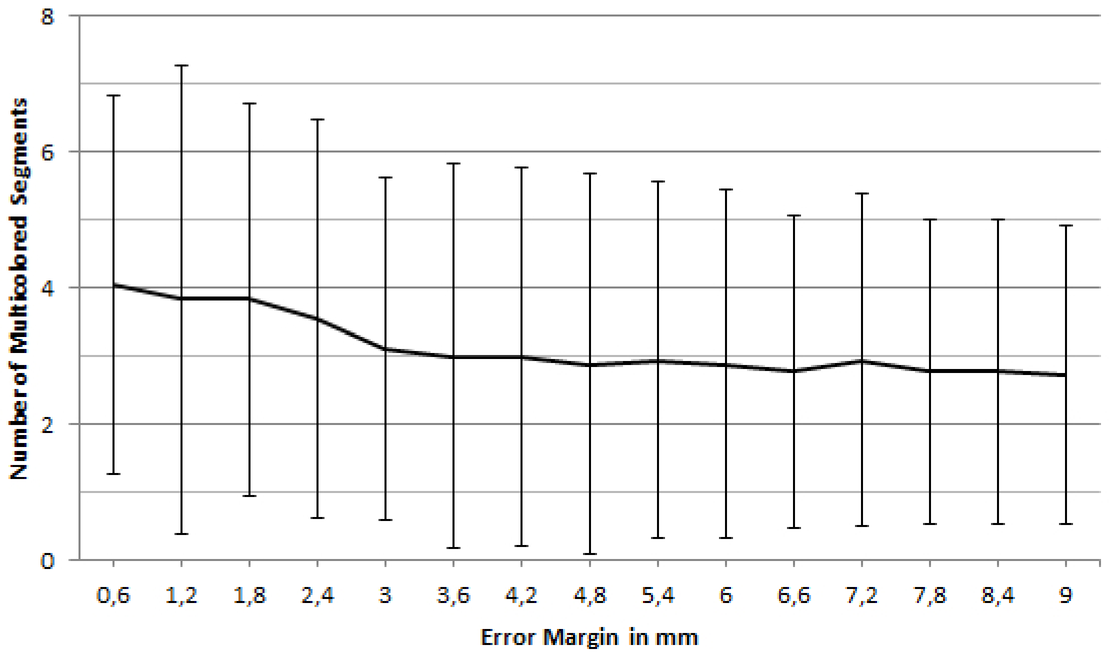


Figure 29: The effect of changing the ϵ parameter on the number of segments that could not be solved using a single color.

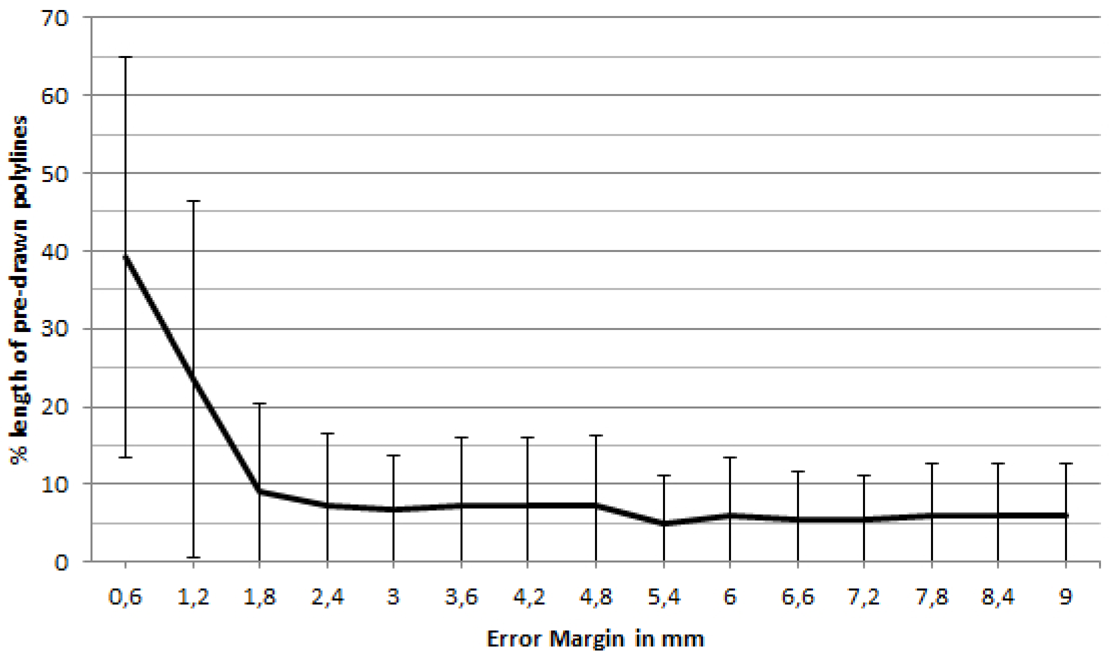


Figure 30: The effect of changing the ϵ parameter on the percentage distance of the input that will be pre-rendered.

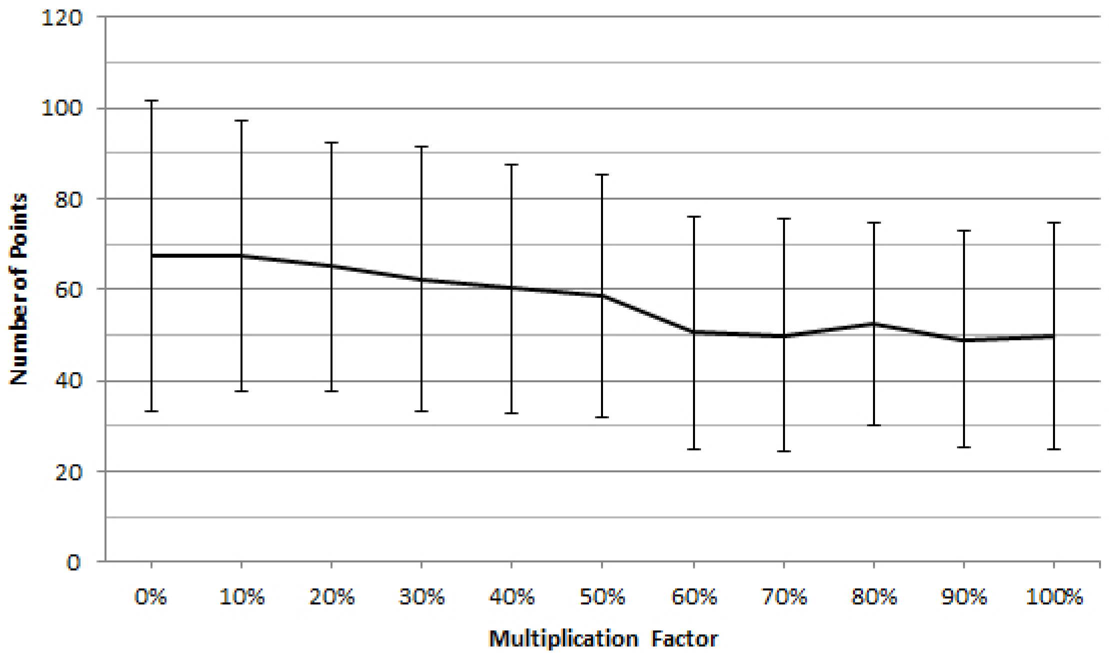


Figure 31: The effect of changing the ρ parameter on the total number of points.

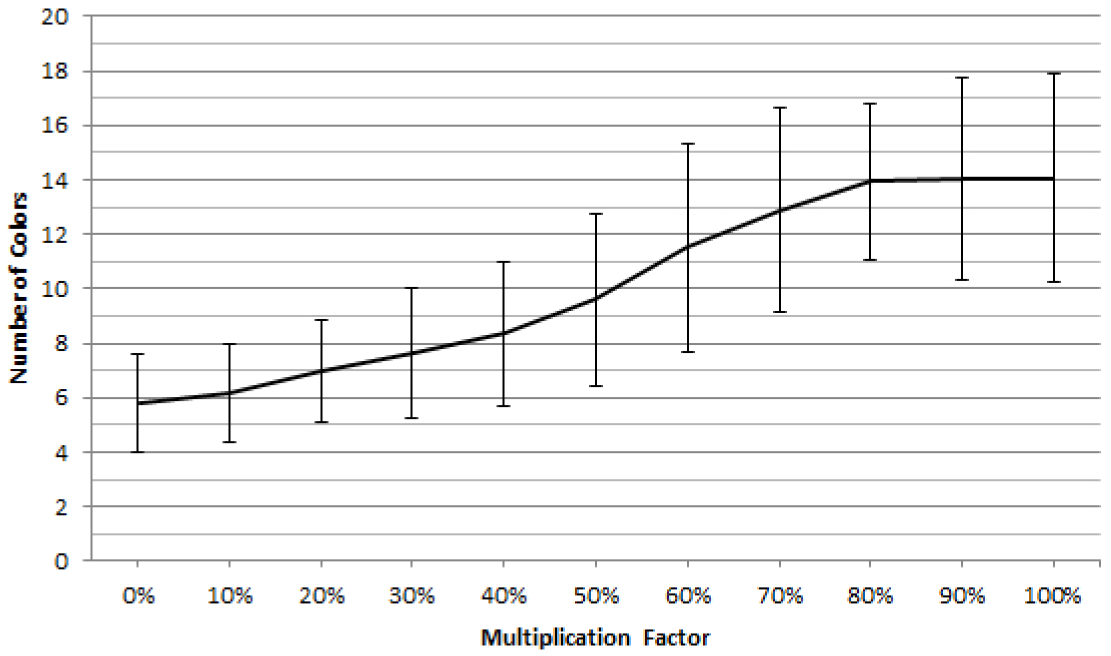


Figure 32: The effect of changing the ρ parameter on the total number of colors.

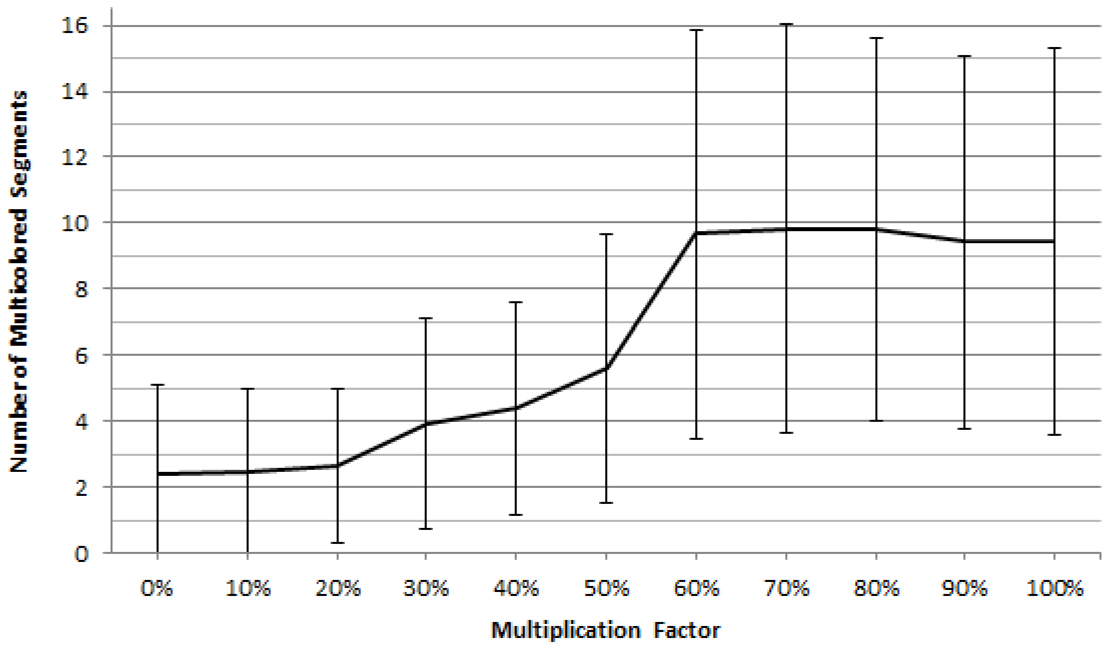


Figure 33: The effect of changing the ρ parameter on the number of segments that could not be solved using a single color.

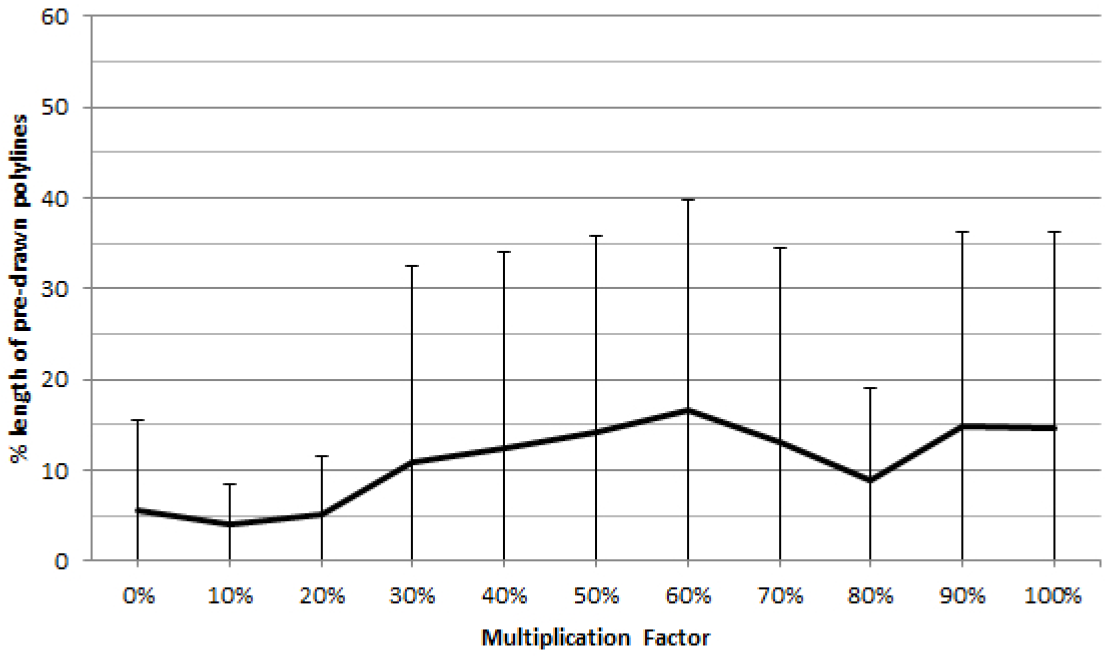


Figure 34: The effect of changing the ρ parameter on the percentage distance of the input that will be pre-rendered.