Approximating Indicative Routes in Weighted Regions

Experimentation Project

Mark Tibboel (0428701)
November 7, 2012

---

1 Introduction

In this paper we will look at the problem of finding a shortest route for a point between a start and goal position while taking the different weighted terrain types into account. This problem has been studied for over 20 years and is formally known as the Weighted Region Problem (WRP) as defined by Mitchell and Papadimitriou [11]. Mitchell and Papadimitriou state the WRP as: Given a straight-line planar (polygonal) subdivision on which a weighted Euclidean metric is defined and a start and goal position, find a minimal length (in the weighted sense) path from the start to the goal. The length in the weighted sense is mostly referred to as the costs of the path. The costs of a path are defined by the sum of all subpaths, where each subpath lies in a different weighted region. The costs of a subpath is defined by the length of the path times the weight of the region. In this paper we search for the best practical method to solve the WRP. The method must give an short path and the method should be fast. The resulting path can be used as input for the Indicative Route Method [7], that will handle collision avoidance with other characters and create smooth paths. First an overview is given of approaches to the WRP in literature. Then an overview is given of the approaches that will be implemented and compared. The comparison experiments are described and the results are given. From the result we will conclude that the Dual Graph method and Steiner approach could be suitable. However, there is still a lot of room for improvement. In the end some extensions from point-based to disk-based path planning are described.

2 Research Question

Which approximation algorithm is best suited for finding ϵ-optimal Indicative Routes in Weighted Regions? A path is an ϵ-approximate path if the path costs are at most $1 + \epsilon$ times the minimal path costs. The different methods will be compared based on the running times and the costs of the resulting paths.

3 Approaches in literature

The weighted region problem was first studied by Mitchell and Papadimitriou [11] in 1991. Mitchell and Papadimitriou gave the definition of the WRP as mentioned above and created an approximation algorithm to solve the WRP. The algorithm makes use of a property that is known from the field of optics. Mitchell and Papadimitriou prove that the shortest path in a weighted region will behave according to Fermat's Principle, which states that a ray of light will always take the path that can be traversed in the least time. Mitchell and Papadimitriou derive a useful lemma from this principle.

Lemma 3.1 *Geodesic paths are piecewise linear, except within regions of weight zero (where subpaths may be arbitrary). The intersection of a geodesic path within the interior of a face with $\alpha_j > 0$ is a (possibly empty) set of line segments.*

Here α is the weight of a face and a geodesic path is a path that is locally optimal. Snell's law of refraction can be derived from Fermat's principle and this law contains a formula that is used to calculate the angles of refraction and incidence when light (or any other wave) goes through a border between two isotropic regions. An isotropic region is a region which has the same properties everywhere. The WRP also contains isotropic regions, because the terrain consists of regions with a uniform weight. A second lemma is derived from Snell's law.

Lemma 3.2 *Let α_i and α_j be the weights of two faces incident to an edge e and assume that $\alpha_c = min\{\alpha_i, \alpha_j\}$ and that $\alpha_i, \alpha_j < +\infty$. If p is a geodesic path that passes through the interior of edge e, then p obeys Snell's Law at edge e.*

These two lemma's are always valid for the WRP and are the basis for the approximation algorithms that solve the problem. The algorithm from Mitchell and Papadimitriou solves the WRP for a finite triangulation of the plane, where all edges and faces have an assigned weight, within a error tolerance $\epsilon > 0$ with respect to the shortest path. The algorithm uses continuous Dijkstra [9, 10] to extend a wavefront where every point on the wavefront has the same distance to the source. Events are triggered at collisions between the wavefront and vertices or edges from the triangulation. The total running time of the algorithm is $O(n^8 log(\frac{nNW}{w\epsilon}))$, where n is the number of vertices, N is the maximum integer coordinate of any vertex of the triangulation, and w and W are the minimum and maximum weights. Mata and Mitchell [8] devised Pathnet 6 years later. This algorithm makes use of cones around all the vertices, which limit the paths that extend from a vertex. From these cones they create critical links to critical vertices. The Pathnet can be constructed in $O(kn^3)$, where k is the number of cones. Once the Pathnet has been constructed it can be queried with Dijkstra's algorithm in order to find the approximate shortest path in $(O(nlogn))$ time. In 1998, Aleksandrov, Lanthier and Maheshwari [1] came up with a new discretization scheme through the use of Steiner points. These points where added on the edges of the triangulation with a logarithmic distribution. The approximate shortest path was found by applying an improved version of Dijkstra's algorithm to the graph of Steiner points and vertices. Dijkstra's algorithm was improved through the use of Fibonacci heaps. The total running time of this algorithm is $O(mnlog(mn) + nm^2)$, where $m = O(log_\delta(L/r))$. Reif and Sun [12] improved this result in 2000. They experimented with uniform and logarithmic distributions of Steiner points and concluded a running time of $O(n^3 logn)$ for uniform discretization and a running time of $O(nmlogm)$, where $m = O(log_\delta(L/r))$ for logarithmic discretization, L is the length of the longest edge, r is ϵ times the minimum distance from any vertex to the boundary of the union of its incident faces and $\delta \geq 1 + \epsilon sin\theta$, where θ is the minimum angle between any two adjacent edges of the surface. In 2001 [14] and 2003 [15] Sun and Reif came with two techniques that can improve several existing algorithms. BUSHWHACK is an improved graph-search algorithm that is faster than Dijkstra's algorithm, because it uses the geometric properties of the graph as explained in section 4.3.3. The second technique is called the adaptive discretization method and it improves the performance of approximation algorithms by placing discretization points densely only in areas that may contain optimal paths. Aleksandrov, Maheshwari and Sack [2] improved the Steiner points approach in 2005 by adding Steiner points to the bisectors of the triangles. This improvement lead to a running time of $O(C(P)\frac{n}{\sqrt{\epsilon}} log(\frac{n}{\epsilon}) log(\frac{1}{\epsilon}))$, where $C(P)$ captures geometric parameters and the weights of the faces of the triangulation P. Several small improvements have been made by Cheng et al. [3] and Sun and Reif [16]. So far several methods have been created that take different approaches to find an optimal ϵ-approximate path as fast as possible. In this paper we will compare different approaches to discover their strengths and weaknesses.

4 Algorithms

This section will describe the algorithms that are implemented in the framework. All algorithms consist of two phases. In the first phase a graph is created from the input data. Three different graphs are used by the different algorithms: Grid, Steiner points and Dual graph. The Steiner points and Dual graph require an extra step in which the initial planar subdivision is triangulated. In the second phase a graph-search algorithm is used to find the shortest path. Three different graph-search methods are used: Dijkstra, A* and Alternative BUSHWHACK.

4.1 Preliminaries

We start with a planar subdivision P in the 2-dimensional Euclidean space. All polygons $t_1, ..., t_n$ have associated weights $w_1, ..., w_n$. Each weight represents the costs per unit distance traveled. The minimum weight must be larger than 0 in order for all paths to be piecewise linear, as stated in Lemma 3.1. The cost of traveling along on edge is equal to the length times the minimum weight of the two incident polygons. We will assume that the polygons are convex and that they don't overlap.

4.2 Graph Construction methods

To effectively execute a search for an optimal path in the planar subdivision P, we first need to discretize P. The discretization can be done by converting P to a grid or by selecting specific points as graph nodes. The discretized data will contain less information than the original planar subdivision, but it will always contain enough information to approximate the shortest path. The discretized data can easily be saved in a graph structure which allows us to search efficiently.

4.2.1 Grid

A straightforward method to discretize a search space is by laying a grid over the planar subdivision and sampling each cell of the grid. The size of the grid can be adjusted to alter the precision of the resulting paths. A surface can be supersampled by using a larger grid (i.e. by using smaller cells). This will result in a bigger graph and slower searches, but the resulting paths will be a better approximation of the shortest path. In contrast, we can undersample the surface by taking a smaller grid. This will speed up the search, but the resulting path will be coarser.

4.2.2 Triangulation

The next two methods require a triangulation of the scene. The Steiner Points require triangles, because the additional points need to be added to the bisectors of the triangles. The Dual graph doesn't necessarily need a triangulation, but a triangulated scene yields more nodes in the Dual graph. This will improve the optimality of the final path. A Delaunay Triangulation is used to prevent the creation of very long and small triangles. These triangles will decrease the optimality of the resulting path. A Delaunay Triangulation always maximizes the minimum angle over all triangulations of P [4]. However, we do want to keep the original edges of the polygons in the resulting graph. A Constrained Delaunay Triangulation keeps the original edges intact and to maximizes the minimum angle over all triangulations. This is a good choice for this project.

4.2.3 Steiner Points

The Steiner points are used to discretize the search space in order to create a search graph out of the triangulation. This discretization step first defines a *vertex vicinity* that is void of any Steiner

points. Then the Steiner points are placed according to Steiner intervals. Finally, the Steiner points are connected with each other to form a graph.

Vertex vicinity The vertex vicinity is a small region around each vertex that doesn't contain any Steiner points. We refer the reader to Figure 1 for an illustration of the vertex vicinity. The radius of the vertex vicinity of vertex v is defined by:

$$r(v) = \frac{w_{min}(v)}{7w_{max}} d(v).$$

In the formula w is the weight of the faces incident to vertex v. $d(v)$ is the minimum distance from vertex v to all the edges in the set of edges $E(v)$, where $E(v)$ is the set of all edges from triangles incident to v without the edges incident to v. From the radius of the vertex vicinity we can create an isosceles triangle in each triangle incident to v. These isosceles triangles have 2 sides of length $\epsilon r(v)$ that lie on the original triangle and are connected by v. The vertex vicinity $S(v)$ consists of the union of all the isosceles triangles of v.

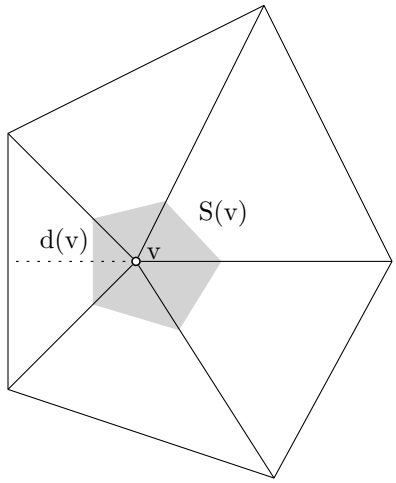

Figure 1: The vertex vicinity $S(v)$ of vertex v. The dashed line represents $d(v)$.

Steiner intervals The Steiner points will be added on the bisectors of all three angles of every triangle in the scene. The Steiner points on one bisector are shown in Figure 2. The position of a Steiner point on a bisector is always defined by the distance from the previous Steiner point to the next point. The first Steiner point p_0 uses the vertex vicinity radius as distance. After the first Steiner point has been placed on the edge of the vertex vicinity, the rest of the Steiner points $p_1 ... p_k$ can be found with the formula:

$$|p_{i-1}p_i| = \sin(\alpha/2)\sqrt{(\epsilon/2)}|vp_{i-1}| \text{ for } i = 1, ..., k,$$

where $|p_{i-1}p_i|$ is the Euclidean distance between the two points p_{i-1} and p_i, α is the angle that is split in two by the bisector, v is the vertex at the origin of the bisector and k is the maximum integer such that the Euclidean distance $|vp_k|$ between points v and p_k is smaller than the length of the bisector.

Connecting Steiner points With all the Steiner points in place we have all the nodes in the graph, but they are still unconnected. Before we start connecting the points we will look at the notion of bisector neighbours.

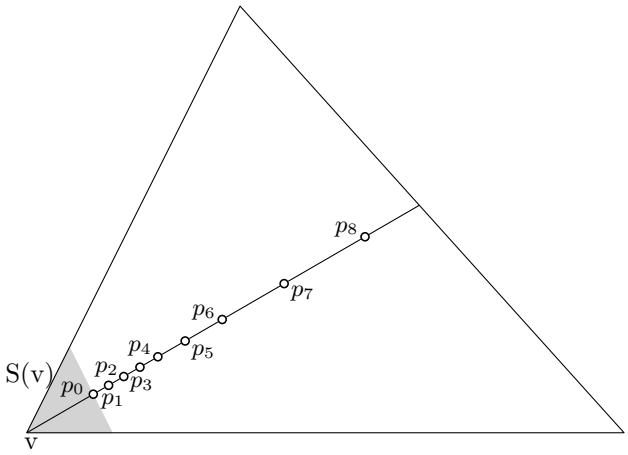

Figure 2: The Steiner points on bisector l.

Bisectors neighbours A bisector is always a neighbour of itself, because points that lie on the same bisector should be connected with each other. A bisector is always a neighbour of all other bisectors that lie in the same triangle. Finally, two bisectors are neighbours if they share an edge e that is incident to the angle that is split by the bisectors (this is also the case in the first two rules). For each pair of bisectors l_1 and l_2 we will connect all Steiner points p on l_1 with all Steiner points q on l_2. All possible neighbours are shown in Figure 3.

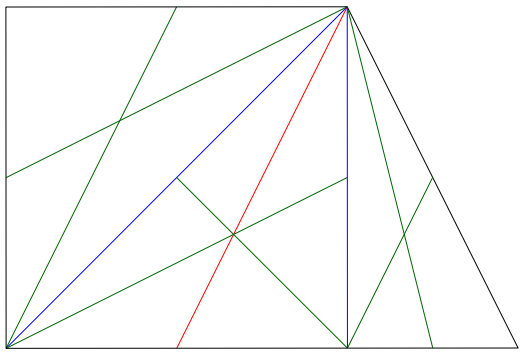

Figure 3: All different neighbours for a bisector (red line). The shared edges are blue lines, the bisector neighbours are green lines and the triangle edges are black lines.

Neighbours within the same triangle The shortest path between p and q depends on the weights of the triangles that are incident to the edge e. If two neighbour bisectors lie in the same triangle, then we define weight w_1 to be the weight of that triangle. Weight w_2 is defined as the weight on the other side of the edge e. If $w_2 \geq w_1$ then an optimal path will always consist of a straight line between p and q. The costs of travel along line segment pq is equal to $|pq| * w_1$. If $w_2 < w_1$ then it is possible for the optimal path to partly lie on the edge e. We can determine whether this is the case by looking at Snell's law. According to this law, an edge-using path can only exists if the angle between the incoming path and the edge is critical. The critical edge is the smallest angle of incidence of a ray of light, where total internal reflection occurs. The same goes for the outgoing path, so the

angles between the incoming path and the edge ϕ, and the outgoing path and the edge ψ should be equal. We can calculate the critical angle by Snell's law: $\sin\phi = \sin\psi = w_2/w_1$. Once we know the critical angle, we can look for the points x and y that lie on e, such that px connects p to edge e and xy lies on e and yq connects q to e. The path pq will now consist of the line segments px, xy and yq as shown in Figure 4. The costs of the path pq is equal to $|px| * w_1 + |xy| * w_2 + |yq| * w_1$. There are some cases where it is not possible to find the points x or y, because the line px or yq under the critical angle doesn't intersect with the edge e or when px and yq intersect as shown in Figure 8. In these degenerate cases we will just take the straight line between p and q.

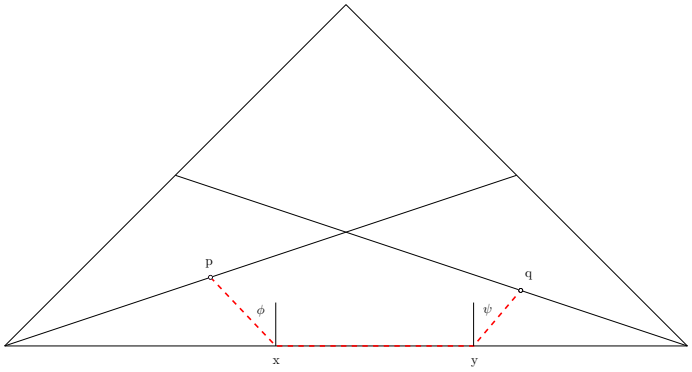

Figure 4: The points p and q are connected with an edge-using path via x and y.

Neighbours in adjacent triangles If the neighbouring bisectors lie in adjacent triangles, then the weights w_1 and w_2 are defined as the weight of the triangle of bisectors l_1 and l_2 respectively. If the weights are equal then the optimal path between p and q will consist of a straight line segment. The costs of the straight line path pq is defined by $|pq| * w_1$. If the weights are not equal then we have to find a point x on edge e, such that the incoming ϕ and outgoing ψ angles satisfy Snell's law $w_2 * sin\phi = w_1 * sin\psi$ as shown in Figure 5. Given x, we can calculate the costs of the path pq as $w_2 * |px| + w_1 * |xq|$.

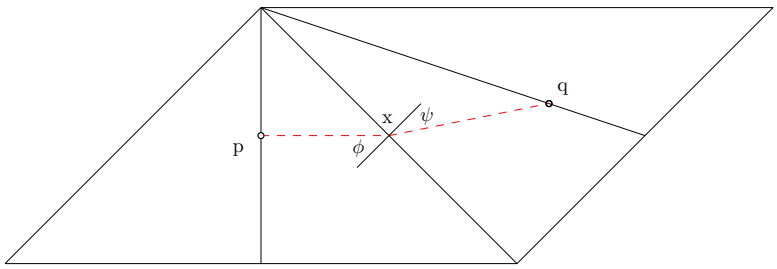

Figure 5: The points p and q are connected via x.

4.2.4 Dual graph

A Dual graph is a graph that can be derived from any other graph. Every triangle in the normal graph becomes a node in the dual graph. Every two triangles that share an edge will be connected in the Dual graph [4]. Dual graphs are very sparse, because there will only be one node per triangle.

4.2.5 Dual Graph Extra

To increase the number of nodes in the graph, we also created a Dual graph Extra that creates additional nodes on the edges of the original triangles. Each edge will become a node in the Dual graph Extra, that will be located halfway on that edge. All nodes that belong to the same triangle will be connected with each other. A node that lies on an edge will belong to 2 triangles, if there are triangles on both sides of the edge.

4.3 Graph-search Algorithms

Once the graph has been constructed, the shortest path in the graph can be found with several graph search algorithms. In this section we will describe Dijkstra's algorithm, A* and Alternative BUSHWHACK. Before we can start the search we will have to connect the start and goal to the graph. The grid uses the cells that contains the start and goal position as start and goal cells. The Dual graph connects the start and goal positions to the two nodes that are closest, unless that closest node is the Dual node of an obstacle triangle. The Steiner approach adds the start and goal positions to the DCEL after the triangulation. The triangle that contains the start or goal position will be split into 3 new triangles with the start or goal position as a vertex.

4.3.1 Dijkstra

Dijkstra's algorithm was founded by the Dutch computer scientist Edsger Dijkstra in 1959 [5]. Dijkstra's algorithm will always find the shortest path between two nodes in a graph if it exists. Dijkstra's algorithm starts with setting a distance value for all nodes to infinity, except for the start node. The start node will have a distance value of zero and will be set as 'current node'. Throughout the algorithm a set of nodes will be maintained. This set is called the unvisited set and it starts out with all nodes except the start node. In each iteration the distance value for all nodes connected to the current node are calculated by taking the edge costs plus the distance value of the current node. If a node already has a distance value, then it will only be overwritten if the new value is lower. The algorithm chooses the node with the lowest distance value as the new current node and removes this node from the unvisited set. The algorithm stops when the selected node is the goal node or when the lowest distance value is equal to infinity. If the lowest distance value is equal to infinity, then there exists no path from the start node to the goal node in the graph. The generalized version of Dijkstra is called A*.

4.3.2 A*

The A* search algorithm [6] is a fast and flexible search algorithm. The algorithm maintains two lists of nodes: the open and the closed set. Nodes in the open set are available for exploration and nodes and the closed set have already been explored. During the execution of the algorithm each node in the open set will have four attributes. The first attribute g represents the cost to travel to this node from the start node. The second attribute h represents the cost to travel from this node to the goal node. The third attribute f is the sum of g and h. The final attribute, $parent$, points to the previous node from which this node was discovered. The algorithm starts at the start position, so g can always be calculated by looking at the g-value of the parent and adding the travel cost between the node and his parent. We can't calculate h, because we don't know the shortest route from a node to the goal node. Instead, the algorithm uses a heuristic to approximate the distance from the node to the goal. A* starts from the start position and looks at all surrounding nodes. All surrounding nodes are added to the open list and their attributes are set. The node with the lowest f value is chosen as next node and moved from the open to the closed list. From this node the surrounding nodes are added to the open list. These two steps are repeated until the chosen node is the goal node. The path is

then reconstructed by following the parent pointers from the goal node to the start node. This path is reversed such that the start position is at the beginning.

Heuristic The A* algorithm has no predefined heuristic and that's what gives the algorithm its flexibility. An important property of a heuristic is whether it is admissible or not. An admissible heuristic never overestimates the costs to the goal. Non-admissible heuristics can overlook the optimal solution due to the overestimation of the costs to the goal. If the A* algorithm is used with a non-admissible heuristic, then it will behave like a Best-First-Search algorithm. In an unweighted environment the Euclidean distance is an admissible heuristic and gives good results. The Euclidean distance not ideal for a weighted region. If the weights are from (0-1), then the Euclidean distance will give an overestimation and thus be a non-admissible heuristic. If the weights are from (1-∞) then the Euclidean distance will remain admissible, but the estimations could be very low, compared to the actual costs. When a heuristic gives a low estimate, then the A* algorithm will favor nodes with the lowest g-value, making it behave more like Dijkstra's algorithm. A heuristic that always returns 0 will make A* behave exactly the same as Dijkstra's algorithm. Because the goal of these experiments is to find an optimal path, we will have to choose an admissible heuristic. The weights distribution of the surface is unknown, but we do know the minimum weight w_1. This results in a heuristic that takes the Euclidean distance times the minimum weight:

$$h(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2} * w_1.$$

In this formula $h(n)$ is the heuristic for a node n, where g is the goal node.

4.3.3 Alternative BUSHWHACK

BUSHWHACK is a fast search algorithm that uses geometrical properties of the graph to speed up the search. BUSHWHACK uses an interval datastructure to prune the search. A Steiner point on a triangle edge can only be connected to a subset of the points on the other edges. As shown in Figure 6, this will create a cone for each point. Unfortunately, these cones are only valid for homogeneous regions. If we would create these cones for Steiner points on the bisectors, then they could start in one triangle and end in another triangle. If the weights in these triangles are different, then we don't have a homogeneous region and the cones would be invalid. Aleksandrov et al. proposes an alternative version of the BUSHWHACK method that could be used when Steiner points are added to the bisectors of triangles. Before we give the algorithm, we will first define some characteristics of the algorithm.
Given a directed graph $G(V, E)$, a start position s and a goal position g, we define two subsets of V. Subset $S \subseteq V$ consists of nodes to which the shortest path has already been found. Set S^a consists of all nodes that are adjacent to S, but not in S. The set $E(S)$ consist of all edges between S and S^a. For every node u in S we define a *representative* $\rho(u)$ to be a node in S^a such that for all nodes u' in S: $\delta(u') + c(u', \rho(u)) < \delta(u) + c(u, \rho(u))$, where $\delta(u)$ is the minimum cost from the start to node u and $c(u, v)$ is defined as the costs of the edge from u to v. Node u is called the *predecessor* of its representative. Finally, we define a group (l_1, l_2, e) to be the set of all edges that start on bisector l_1 and end at bisector l_2, where l_1 and l_2 are neighbour bisectors sharing edge e.
The algorithm as proposed by Aleksandrov et al. keeps a representative for each group (l_1, l_2, e) for all points u in S, where u lies on bisector l_1. The representatives from all nodes in S are combined in one priority queue R. The algorithm starts with the connection of the start position and goal position with the closest Steiner points t_s and t_g, respectively. Because the start and goal positions aren't on a bisector, we then conduct the search from t_s to t_g. Point t_s is added to S and the representatives $\rho(t_s)$ are added to R. Then the main loop starts and runs until t_g is found. In every iteration we take the representative $\rho(u)$ with the lowest cost from R and add it to S. The predecessor of this node is set as the parent of $\rho(u)$. New representatives for $\rho(u)$ are discovered and all predecessors of $\rho(u)$ are given

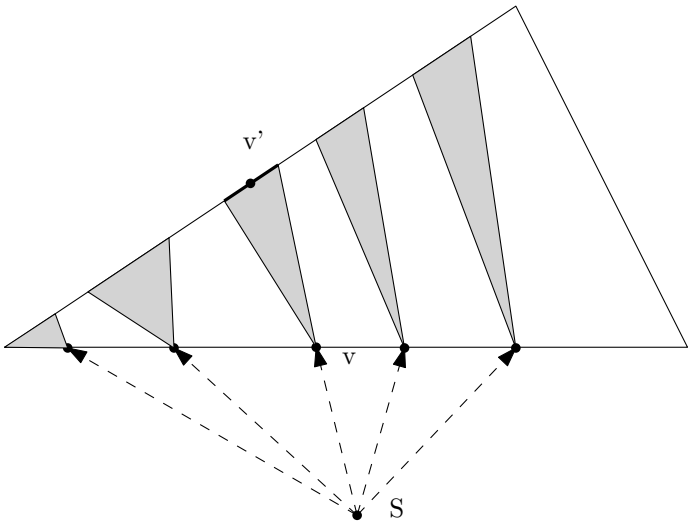

Figure 6: The BUSHWHACK algorithm uses intervals to prune the search

a new representative. These new representatives are added to R. The loop ends when the t_g is found. The final path can be reconstructed by following the parent pointers. The outline of the algorithm is given in Algorithm 1.

Algorithm 1 AlternativeBUSHWHACK(G, start, goal)

1: Connect *start* with nearest node in G: t_s
2: Connect *goal* with nearest node in G: t_g
3: Add representatives for t_s to R
4: Current node *Cnode* is set to the lowest value in R
5: **while** *Cnode* $\neq$ *goal* **do**
6: Find new representatives for *Cnode*
7: Find new representatives for all predecessors of *Cnode*
8: Add all newly discovered representatives to R
9: *Cnode* is set to the lowest value in R
10: **end while**
11: Reconstruct the path from the parent pointers
12: **Return** Path

5 Implementation

The methods described in the previous section are all implemented in a C++ program. The program takes prix files as input. Prix files use XML to describe (possibly multilayered) planar subdivisions. A custom weight attribute is added to assign a weight to all regions in the planar subdivision. The grid conversion is done by reading from the openGL buffer. The constrained Delaunay triangulation is provided by a little program called Triangle [13]. Triangulated scenes are stored as Doubly Connected Edge Lists [4] for easy processing to Dual Graph or addition of Steiner Points. Dual graph nodes are found by taking the average coordinate of the three vertices of a triangle. The Dual graph nodes could also be taken from the intersection of the bisectors of the angles, but there are no specific properties

of the bisector intersection that we could use to our advantage. We chose the average coordinates, because this is a faster method. The dual graph nodes are connected with a straight line segment, unless this line segment intersects with a triangle that doesn't contain one of the two dual graph nodes. In that case the connecting edge will bend at one of the two shared vertices of the triangles that contain the dual graph nodes, such that the resulting path has the lowest cost. Both paths are visualized in Figure 7. The connections between Steiner points are undefined in some cases as explained in Figure 8. In these cases the Steiner points will be connected with a straight line segment. The search algorithms aren't fully optimized and lower running times could be achieved by further optimizing the data structures used by the search algorithms.

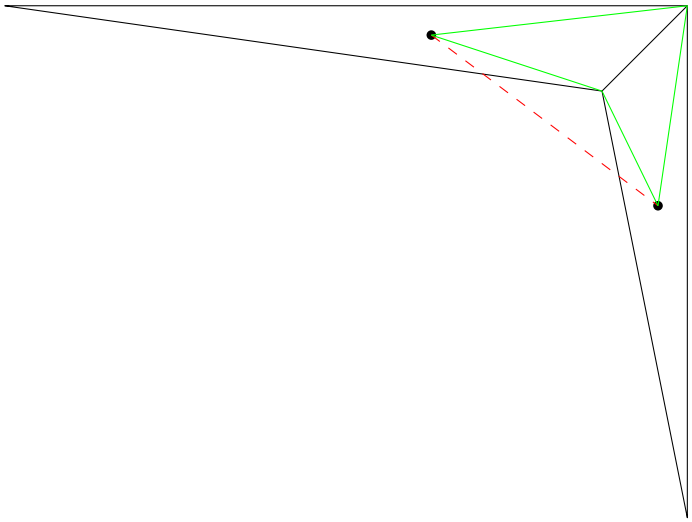

Figure 7: Degenerate Dual Graph case. The (red) straight line connection intersects with another triangle. The cheapest of the two green lines will be chosen as connection.

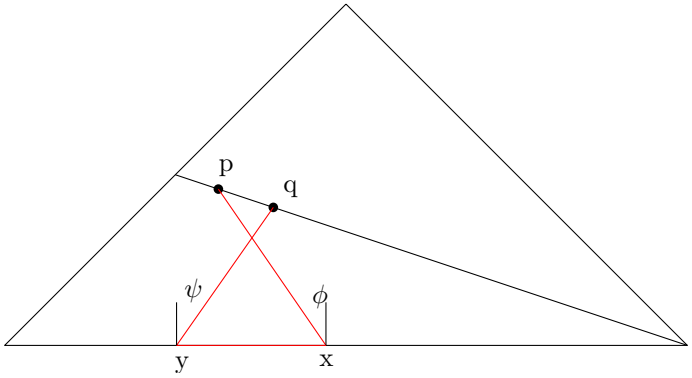

Figure 8: Degenerate Steiner connection case. The (red) straight line connection intersects with itself, due to the mandatory angles ϕ and ψ.

6 Extension to using disks

All of the methods described in this paper take the assumption that the object that moves from the start to the goal position is a point, i.e. it has no actual dimensions. However, if we want to use these methods to find the optimal path for characters in a crowd simulation or unmanned ground vehicles in the army, then we have to take the dimensions of the object into account. In this section we will give several adjustments to the algorithms and data structures such that the methods support path planning in weighted regions for objects with a given radius r. For the extension to disks we will assume that the costs of travel through regions with different weights is always equal to the highest costs of all regions that intersect with the disk.

6.1 Grid

The grid can be extended with additional information in order to support multiple character radii. By only adjusting the grid, we can leave the search algorithms almost unchanged. The additional information will be added to each cell and will consist of different weights for different radii. The weight values for bigger radii are comparable with a subsampled grid. For instance, if a character has a radius of one, then the character will fit inside a 2x2 pixel. The extra weight values can easily be extracted from the OpenGL framebuffer and the extra construction costs will be linear in the number of additional radii. The search algorithms only need a small adjustment. They now need to know the radius of the current character in order to retrieve the right weight value at every cell.

6.2 Dual Graph

The Dual graph can also be extended with additional information. For each radius we need to recompute the costs to travel between 2 nodes. This could be done by sliding a disk from node A to node B along the original edge, as shown in Figure 9. The costs can be found by integrating the weight over the entire edge and adding the two half circles at the beginning and the end. The additional construction costs will depend on the number of edges, times the number of additional radii: $O(E * R)$. The search algorithms only need a small adjustment. They now need to know the radius of the current character in order to retrieve the right weight value at every cell.

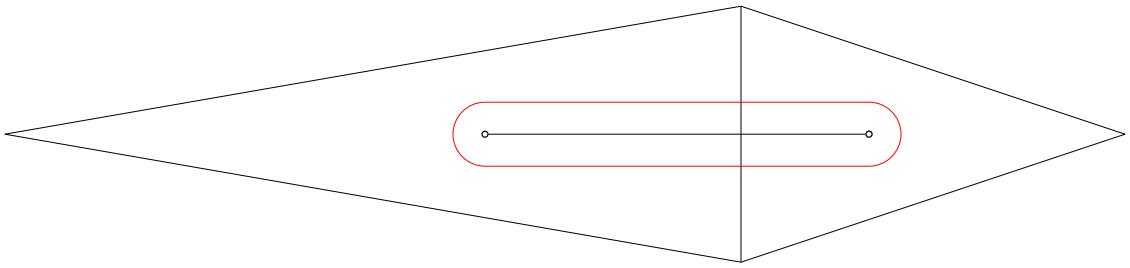

Figure 9: Sliding disk method.

6.3 Steiner points

The Steiner point approach could use the same sliding disk principle as the Dual graph adjustment. However, due to the very high number of edges in the Steiner graphs, this method is not well suited for Steiner graphs. Another problem arises from the assumptions that are made before the Steiner placement as explained in the next Section.

6.4 Connections within the same triangle

The Steiner method states that the cost of travel along an edge is equal to the lowest weight of the two faces incident to the edge times the length of the path. When a disk with a certain radius is used to travel along an edge it will always intersect with both incident faces and the cost of travel will be equal to the highest weight of both faces times the length of the path. If we move the path in such a way that the triangle lies completely on the cheapest side, then the basis of this rule could still be applied. After we translate the points x and y into the cheapest triangle by a length equal to the radius, then we have to check whether the disk position at x or y intersects with any other triangles of a higher weight. If this is not the case, then all position in between x and y will also be valid. The cost of the path is equal to the original formula for two Steiner points p and q that lie on bisectors of the same triangle:

6.5 Connections between two triangles

The connections between neighbouring bisectors in different triangles could be used to create corridors for disks with a certain radius as shown in Figure 10. These corridors define a set of pairs of nodes for which the costs could be calculated without the sliding disk principle. If we have two bisectors l_a and l_b that lie in two different triangles T_a and T_b and a radius r, then we first find four boundary nodes. The first boundary node a is the node on l_a that is closest to vertex v incident to l_a and the distance between a and v is greater than r. The second boundary node b is the node on l_a that is closest to the end of bisector l_a, while the distance from b to the end of l_a is greater than r. Boundary nodes c and d are found in the same way, but then on bisector l_b. The corridor can then be found between the four nodes. For each Steiner point p on l_a that lies in between a and b (including a and b) and each Steiner point q on l_b that lies in between c and d, the costs to travel between these two points can be calculated as follows. First find all triangles H that are currently intersected by the disk. Then find the translation of the disk from p to q until the disk no longer intersects with the triangle that has the highest weight value in H. Calculate the costs so far by take the translation distance times the highest weight value in H. Repeat this step until the highest weight in H is equal to the weight of the triangle that contains l_a. At this point the disk can be translated until it touches the edge of T_b that is not the shared edge of l_a and l_b. The cost of this part is equal to the cost of the same path without a defined radius. From this point onward the disk moves up until q and whenever the disk intersects with a new triangle, this triangle will be added to H and the travel costs will be calculated with the highest weight value in H. With this method we can calculate the costs to travel from p to q in linear time for each radius.

6.6 Degenerate cases

If the change of the path between two Steiner points on the same triangle will lead to an intersection with another triangle that has a higher value or if one of the two Steiner points in two different triangles lies outside the corridor, then the path costs will have to be calculated with the sliding disk method.

6.7 Construction time analysis

These adjustments don't give a guaranteed improvement over the sliding disk method, because a high enough radius will lead to degenerate cases for all paths. However, this method has the possibility to reduce the calculations needed for each edge, by applying constant time operations for all edges that lie within a corridor.

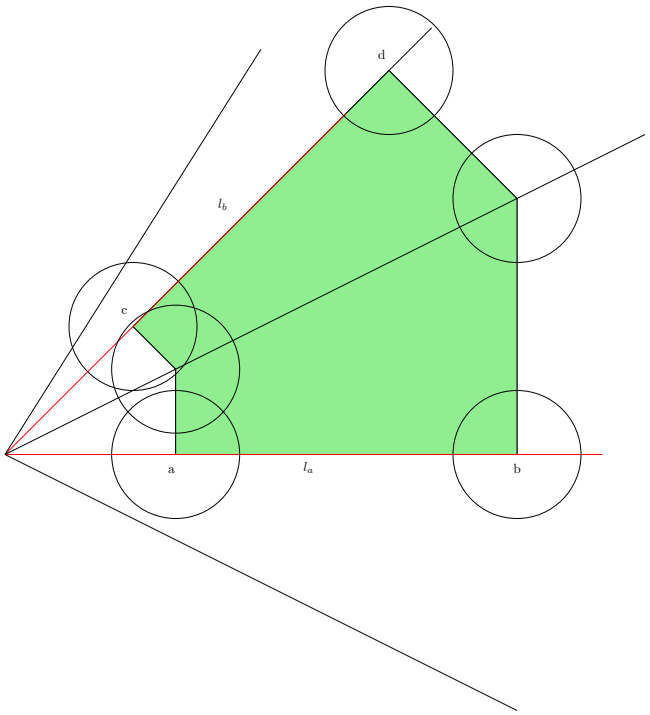

Figure 10: The corridor between bisectors l_a and l_b with boundary points a, b, c and d.

7 Experiments

Several experiments have been conducted to compare the different methods based on path costs and running times. All our experiments are run on an AMD Phenom II x4 3.4Ghz processor with 4 Gb RAM. All running times are averaged over 50 runs in order to decrease the error in the C++ high precision timer to 0.12 ms. All pictures of scenes use a range of colors to show the weight of different regions. The colors range from white for a weight of 1 to blue for the maximum weight.

7.1 Scene1

The first set of experiments were conducted on Scene1. The goal of these experiments was to find good parameter values for later experiments and to check the correctness of the implementation of the various algorithms. Scene1 is a small scene that consists of 4 regions with weights between one and eight. The dimensions of the scene are 100x50.

Grid The resulting paths of the grid methods are shown in Figure 11 for start position (20,40) and goal position (80,35) and Figure 12 for start position (5,5) and goal position (95,45). The running times and costs are shown in Table 1 and Table 2. The construction times of the grid with different pixel values are shown in Table 3. The start and goal positions aren't exactly the same in the figure, because larger pixels lead to larger start and goal areas. Dijkstra and A* compute exactly the same path. The subsampling clearly leads to a big improvement for the running times. The path costs are only slightly larger when larger pixels are used.

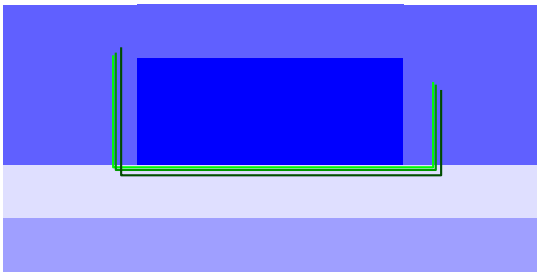

Figure 11: The resulting paths of the different pixel dimensions on Scene1 with start position (20,40) and goal position (80,35). The light green line used 1x1 pixels, the seagreen line used 2x2 pixels and the darkgreen line used 4x4 pixels.

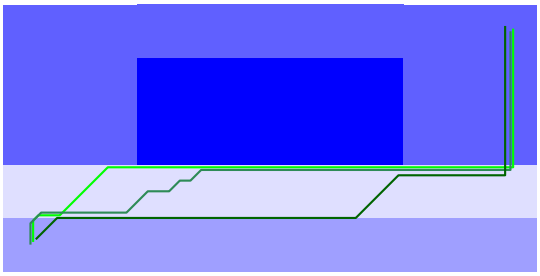

Figure 12: The resulting paths of the different pixel dimensions on Scene1 with start position (5,5) and goal position (95,45). The light green line used 1x1 pixels, the seagreen line used 2x2 pixels and the darkgreen line used 4x4 pixels.

Pixelsize	1x1		2x2		4x4	
Method	A*	Dijkstra	A*	Dijkstra	A*	Dijkstra
Costs	241	241	242	242	244	244
Time(ms)	140	144	8.1	8.2	1.4	1.3

Table 1: Costs and runtimes of the grid methods on Scene1 with start position (20,40) and goal position (80,35).

Pixelsize	1x1		2x2		4x4	
Method	A*	Dijkstra	A*	Dijkstra	A*	Dijkstra
Costs	236	236	222	222	230	230
Time(ms)	135	167	7.8	9.0	0.63	0.67

Table 2: Costs and runtimes of the grid methods on Scene1 with start position (5,5) and goal position (95,45).

Pixelsize	1x1	2x2	4x4
Time(ms)	0.90	0.86	0.68

Table 3: Construction times of the grid with different resolutions.

Dual Graph The resulting paths of the Dual graph methods are shown in Figure 13 for start position (20,40) and goal position (80,35) and Figure 14 for start position (5,5) and goal position (95,45). The

running times and costs are shown in Table 4 and Table 5. The Dual graph of this scene contains 16 nodes and 78 edges. The Dual graph is constructed in 19.3 ms. The Dual graph Extra contains 64 nodes and 632 edges and is constructed in 18.5 ms. The methods are fast, but the resulting path costs are high for the Dual graph methods. This is caused by the sparse graph that is created by the Dual graph. The Dual graph of Scene1 is shown in Figure 15.

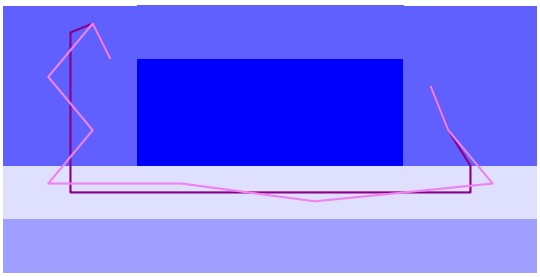

Figure 13: The resulting paths of the dual graph methods with start position (20,40) and goal position (80,35). The pink line used the Dual graph and the purple line used the Dual graph Extra. Dijkstra and A* return exactly the same path.

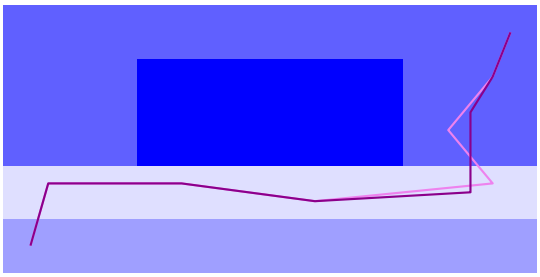

Figure 14: The resulting paths of the dual graph with start position (5,5) and goal position (95,45). The pink line used the Dual graph and the purple line used the Dual graph Extra. Dijkstra and A* return exactly the same path.

Graph	Dual Graph		Dual Graph Extra	
Method	A*	Dijkstra	A*	Dijkstra
Costs	392	392	356	356
Time(ms)	0.92	0.96	0.13	0.22

Table 4: Costs and runtimes of the Dual graph methods on Scene1 with start position (20,40) and goal position (80,35).

Steiner points The resulting paths of the Steiner point methods are shown in Figure 16 for start position (20,40) and goal position (80,35) and Figure 17 for start position (5,5) and goal position (95,45). The time and costs are shown in Table 6 and Table 7. Dijkstra, A* and Alternative BUSHWHACK all return exactly the same path for each ϵ value. The graph with added Steiner points contains 160 nodes and 2812 edges for $\epsilon = 16384$ ($=2^{14}$) up until 2175 nodes and 758794 edges for $\epsilon = 1$. The construction times of the graph with different ϵ values are shown in Table 8. If we compare the results

Graph	Dual Graph		Dual Graph Extra	
Method	A*	Dijkstra	A*	Dijkstra
Costs	266	266	243	243
Time(ms)	0.07	0.12	0.12	0.22

Table 5: Costs and runtimes of the Dual graph methods on Scene1 with start position (5,5) and goal position (95,45).

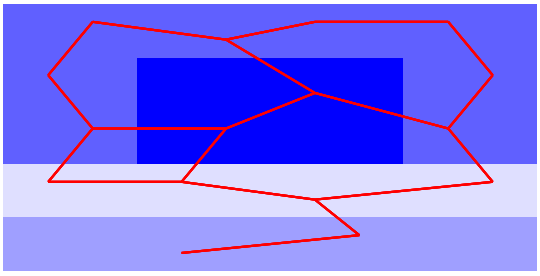

Figure 15: The Dual graph for Scene1 is a very sparse graph.

of the different ϵ values with the Dual graph and Grid methods, then we can conclude that ϵ values lower than 16 can not compete with the other methods based on running times. The higher ϵ values speed up the search a lot, while the increase in pathcosts is very low. Sometimes the pathcosts are even lower with a higher ϵ value, because the Steiner points are positioned in favor of a certain path. The high construction and running times are caused by the very dense graph that is created from all the Steiner points. The Steiner graph for Scene1 with $\epsilon = 32$ is shown in Figure 18.

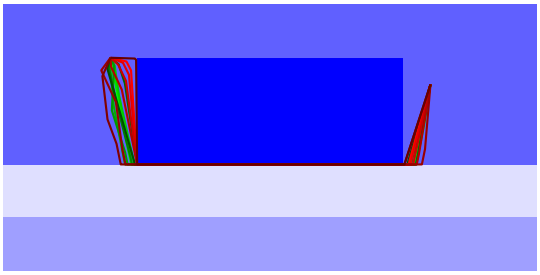

Figure 16: The resulting paths of the different epsilon values on Scene1 with start position (20,40) and goal position (80,35). ϵ values from 1 to 256 range from light green to dark green, ϵ values from 512 to 16834 range from light red to dark red.

7.2 Forest

For the next experiments we will only use the fastest search methods, because the paths are the same for the grid, Dual graph or Steiner graph. From the experiments on Scene1 we can conclude that A* is the fastest method for the grid and Dual graph. Dijkstra is the fastest method for the Steiner graph. The Forest scene consists of a forest like environment with bushes, several puddles, fallen trees and a panoramic view. There are two versions of the Forest scene. The planar subdivision of these two versions are the same, but the weights are adjusted to personal preferences of the character. The dimensions of the scene are 290x410. The Dual graph of this scene contains 208 nodes and 1202 edges.

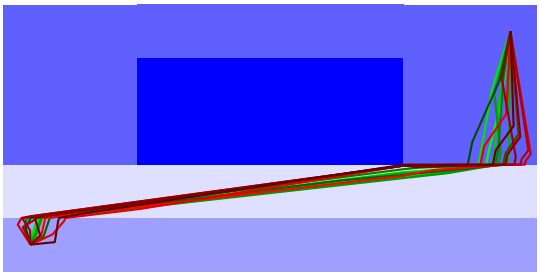

Figure 17: The resulting paths of the different epsilon values on Scene1 with start position (5,5) and goal position (95,45). ϵ values from 1 to 256 range from light green to dark green, ϵ values from 512 to 16834 range from light red to dark red.

ϵ	1			4			16			64		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	232			232			233			234		
Time(ms)	1318	1274	4817	201	197	813	37	36	174	8.0	8.7	52
ϵ	128			256			512			1024		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	233			242			238			234		
Time(ms)	4.9	4.7	27	2.2	2.0	17.3	1.4	1.3	11.4	0.94	0.88	7.8
ϵ	2048			4096			8192			16384		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	232			237			236			252		
Time(ms)	0.70	0.67	6.2	0.54	0.51	4.9	0.40	0.38	3.8	0.33	0.33	4.2

Table 6: Costs and runtimes of the Steiner point methods on Scene1 with start position (20,40) and goal position (80,35). Method A is A*, D is Dijkstra and B is Alternative BUSHWHACK.

ϵ	1			4			16			64		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	227			228			230			238		
Time(ms)	1159	1167	5514	171	174	815	28	28	150	6.5	6.4	42
ϵ	128			256			512			1024		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	231			234			235			241		
Time(ms)	3.23	3.21	26	1.86	1.77	14.5	1.21	1.21	11.2	0.86	0.83	7.7
ϵ	2048			4096			8192			16384		
Method	A	D	B	A	D	B	A	D	B	A	D	B
Costs	240			236			236			241		
Time(ms)	0.61	0.58	5.4	0.51	0.48	4.7	0.38	0.35	3.7	0.31	0.30	3.2

Table 7: Costs and runtimes of the Steiner point methods on Scene1 with start position (5,5) and goal position (95,45). Method A is A*, D is Dijkstra and B is Alternative BUSHWHACK.

The Dual graph Extra contains 832 nodes and 9652 edges. The number of Steiner points depends on the weights and is different for the two versions.

epsilon	1	4	16	64	128	256	512	1024	2048	4096	8192	16384
Time(ms)	5962	1755	608	261	175	135	107	88	77	66	57	53

Table 8: Construction times of the Steiner graph with different ϵ values.

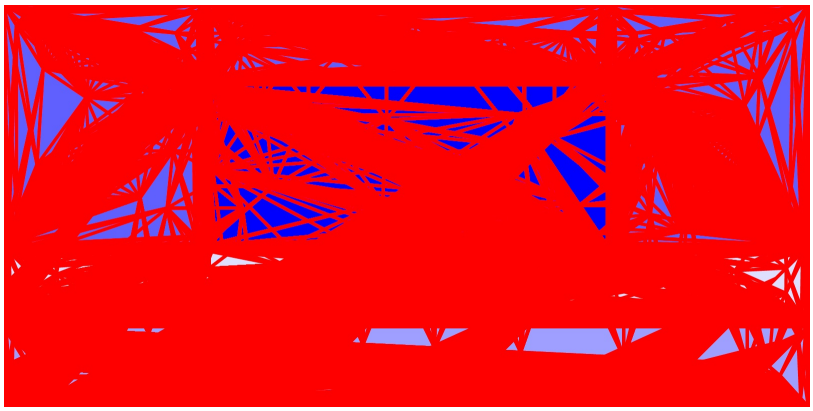

Figure 18: The Steiner graph for Scene1 is a very dense graph. This graph is created with ϵ=32.

7.2.1 Forest child

The first version is based on the preferences of a child. This version has low weights (weight = 1) for puddles and fallen trees, because the child likes to play on them. The panoramic view has a high weight (weight = 10), because the child doesn't care about the panoramic view. The path has a weight of 2 and the bushes have a weight of 30. This version has 1832 nodes and 36896 edges for $\epsilon = 16384$ up until 3773 nodes and 161432 edges for $\epsilon = 64$. The fastest and cheapest paths are shown in Figure 19. All running times and path costs are shown in Table 9. Bar charts of Table 9 are shown in Figure 21 and Figure 22. The construction times are given in Table 9. The Dual graph methods are the fastest methods, but the path costs are a little bit high. The Steiner methods returns the cheapest paths, but the running times are nowhere near the Dual Graph methods.

Graph type	Method	Pixelsize	ϵ	Construction(ms)	Costs	Search(ms)
Grid	A*	2x2		6.2	1155	140
Grid	A*	4x4		6.2	1180	24
Dual	A*			26	1581	0.63
Dual Extra	A*			26	1348	3.85
Steiner	Dijkstra		64	15315	1112	266
Steiner	Dijkstra		128	11153	1118	169
Steiner	Dijkstra		256	8526	1125	114
Steiner	Dijkstra		512	6817	1123	83
Steiner	Dijkstra		1024	5335	1131	60
Steiner	Dijkstra		2048	4439	1132	46
Steiner	Dijkstra		4096	3864	1139	39
Steiner	Dijkstra		8192	3445	1138	34
Steiner	Dijkstra		16384	3185	1141	30

Table 9: Costs and runtimes of the fastest search methods on Forest Child with start position (60,2) and goal position (200,2).

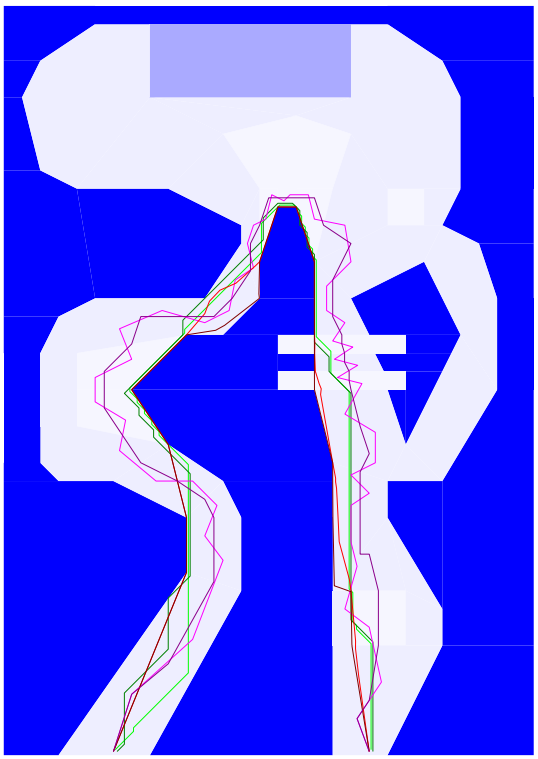

Figure 19: The resulting paths of both grid methods (2x2 is light green and 4x4 is dark green), Dual graph method (pink), Dual graph Extra (purple) and Steiner with $\epsilon = 64$ (light red) and $\epsilon = 16384$ (dark red) on Forest Child with start position (60,2) and goal position (200,2).

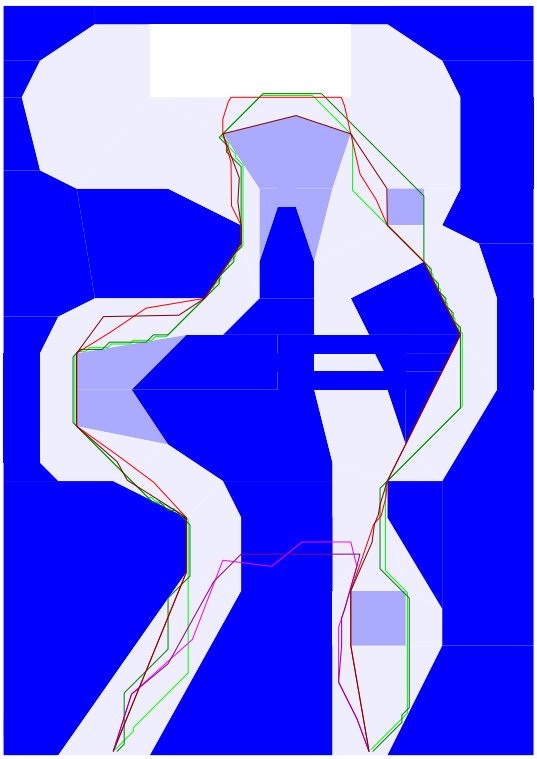

Figure 20: The resulting paths of both grid methods (2x2 is light green and 4x4 is dark green), Dual graph method (pink), Dual graph Extra (purple) and Steiner with $\epsilon = 64$ (light red) and $\epsilon = 16384$ (dark red) on Forest Parent with start position (60,2) and goal position (200,2).

7.2.2 Forest adult

The other version is based on the preferences of an adult. The adult version had high weights (weight = 10) for the puddles and fallen trees, because the adult doesn't want to get wet or play on fallen trees. The panoramic view has a very low weight (weight = 1). The path has a weight of 2 and the bushes have a weight of 30. This version has 1805 nodes and 35752 edges for $\epsilon = 16384$ up until 3636 nodes and 149996 edges for $\epsilon = 64$. The fastest and cheapest paths are shown in Figure 20. All running times and path costs are shown in Table 10. Bar charts of Table 10 are shown in Figure 23 and Figure 24. The Dual graph methods are still the fastest methods, but they now take a huge shortcut. This was an unexpected result, that could be fixed by increasing the weight of the bushes. There are no real obstacles, but just regions with very high weight, so this result is still a valid result. The Steiner methods perform just as well is in the Forest child scene.

7.2.3 Zigzag

The last experiment is the Zigzag. This scene appears in many papers and clearly shows that the shortest route isn't the optimal route in a weighted scene. The scene's dimensions are 150x50. The

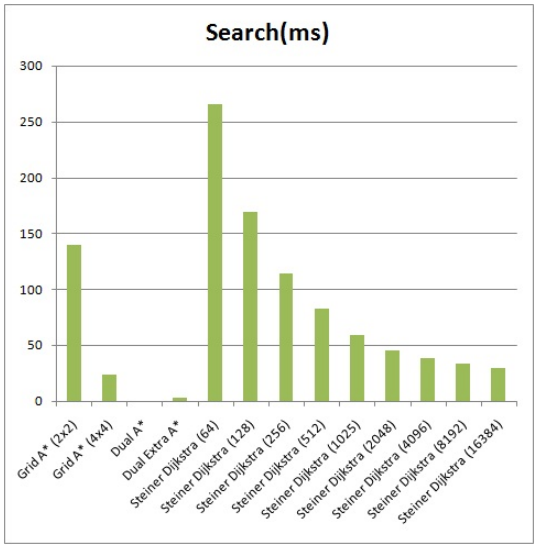

Figure 21: Bar chart of the search times of Table 9.

Figure 22: Bar chart of the costs of Table 9.

Graph type	Method	Pixelsize	ϵ	Construction(ms)	Costs	Search(ms)
Grid	A*	2x2		6.2	1834	145
Grid	A*	4x4		6.1	1844	23
Dual	A*			26	2173	0.60
Dual Extra	A*			27	2019	3.85
Steiner	Dijkstra		64	14052	1742	233
Steiner	Dijkstra		128	10149	1751	141
Steiner	Dijkstra		256	7859	1752	100
Steiner	Dijkstra		512	6155	1746	71
Steiner	Dijkstra		1024	4869	1762	52
Steiner	Dijkstra		2048	4091	1760	41
Steiner	Dijkstra		4096	3613	1756	35
Steiner	Dijkstra		8192	3278	1762	31
Steiner	Dijkstra		16384	3037	1759	28

Table 10: Costs and runtimes of the fastest search methods on Forest Child with start position (60,2) and goal position (200,2).

Dual Graph consists of 55 nodes and 302 edges. The Steiner graph has 481 nodes and 8978 edges for $\epsilon = 16384$ up until 957 nodes and 35370 edges for $\epsilon = 64$. The fastest and cheapest paths are shown in Figure 25. All running times and path costs are shown in Table 11. The start position is at (2,25) and the goal is at (148,25). The Dual graph method doesn't follow the expected path and returns the highest path costs. The grid methods outperforms the Steiner method based on path costs. However, when the grid is supersampled to speed up the search, then the difference in path costs becomes very small.

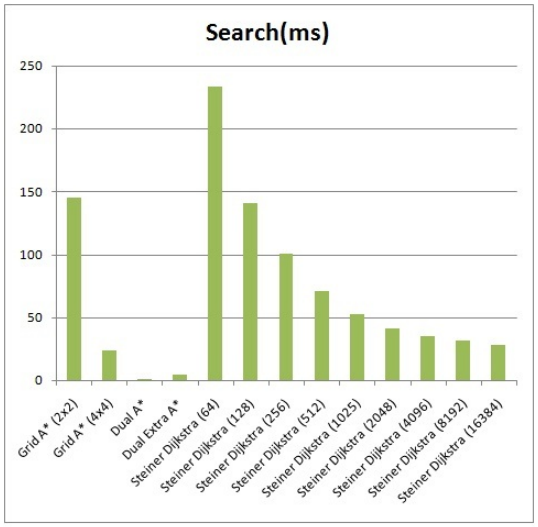

Figure 23: Bar chart of the search times of Table 10.

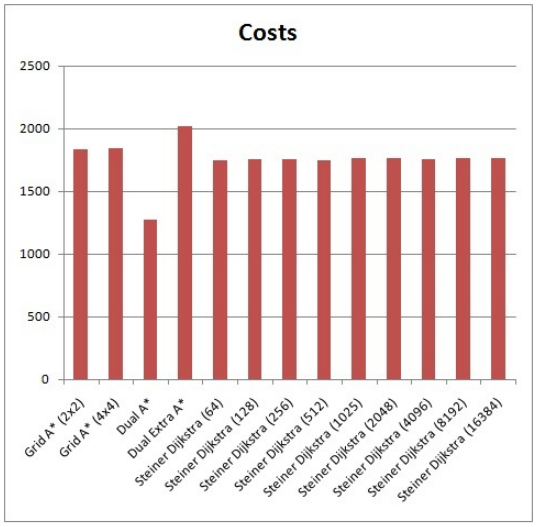

Figure 24: Bar chart of the costs of Table 10.

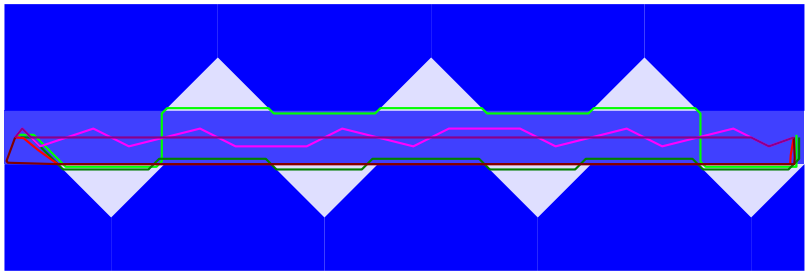

Figure 25: The resulting paths of both grid methods (2x2 is light green and 4x4 is dark green), Dual graph method (pink), Dual graph Extra (purple) and Steiner with $\epsilon = 64$ (light red) and $\epsilon = 16384$ (dark red) on Zigzag.

8 Conclusions

From the experiments we can conclude that the Dual graph method is the fastest method and the Steiner approach returns the cheapest paths. In terms of speed, no method came close to the running times of the Dual graph method. The Dual graph method is so fast because the graph is very sparse. This is also its weakness, because the paths generated are never close to the optimal costs. In the zigzag experiment the Dual graph method still took the shortest route instead of the optimal path. The Steiner approach outperforms the Grid method on path costs and running times if the ϵ is high enough. However, the Grid method has the lowest construction costs and might be an good choice if every search requires a new grid/graph. The Steiner approach with a very high ϵ and the Dual graph method are both well suited for the IRM method. The Dual graph method could be improved by adding more nodes to improve the costs of the resulting paths. The Steiner method might give even better results with even higher ϵ values, but the proof of the ϵ-approximation is only valid for ϵ values between 0 and 1, according to Aleksandrov, Maheshwari and Sack [2].

Graph type	Method	Pixelsize	ϵ	Construction(ms)	Costs	Search(ms)
Grid	A*	1x1		0.91	788	323
Grid	A*	2x2		0.91	1228	18.3
Dual	A*			20	2498	0.19
Dual Extra	A*			20	2370	0.57
Steiner	Dijkstra		64	1058	1272	16.0
Steiner	Dijkstra		128	789	1305	10.3
Steiner	Dijkstra		256	628	1299	7.6
Steiner	Dijkstra		512	491	1274	5.2
Steiner	Dijkstra		1024	414	1292	4.1
Steiner	Dijkstra		2048	364	1291	3.4
Steiner	Dijkstra		4096	303	1353	2.7
Steiner	Dijkstra		8192	285	1351	2.5
Steiner	Dijkstra		16384	259	1350	2.2

Table 11: Costs and runtimes of the fastest search methods on Zigzag.

8.1 Future work

The Dual graph method could be optimized for lower cost paths by adding even more nodes to the graph. These additional nodes could be added next to the nodes that are added halfway by the Dual graph Extra method. The number of additional nodes on each edge could depend on the length of each edge. The Steiner approach could be improved by altering the BUSHWHACK method. If the cones of the BUSHWHACK method could bend along with the angles as defined by Snell's law, then the search on Steiner graphs could be a lot faster.

References

[1] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.R. Sack. In *Algorithm Theory SWAT'98*, volume 1432 of *Lecture Notes in Computer Science*, pages 11–22. 1998.

[2] L. Aleksandrov, A. Maheshwari, and J.R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, 2005.

[3] S. Cheng, H. Na, A. Vigneron, and Y. Wang. Approximate shortest paths in anisotropic regions. *SIAM Journal on Computing*, 38(3):802–824, 2008.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

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

[6] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Ieee Transactions On Systems Science And Cybernetics*, 4(2):100–107, 1968.

[7] I. Karamouzas, R. Geraerts, and M. Overmars. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 113–120, 2009.

[8] C.S. Mata and J.S.B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 264–273, 1997.

[9] J.S.B. Mitchell. *Planning shortest paths*. dissertation, 1986.

[10] J.S.B. Mitchell, D.M. Mount, and C.H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, August 1987.

[11] J.S.B. Mitchell and C.H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[12] J. Reif and Z. Sun. An efficient approximation algorithm for weighted region shortest path problem. In *Proceedings of the 4th Workshop on Algorithmic Foundations of Robotics*, 2000.

[13] J.R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. 1996.

[14] Z. Sun and J. Reif. Bushwhack: An approximation algorithm for minimal paths through pseudo-euclidean spaces. In *Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, pages 160–171. 2001.

[15] Z. Sun and J. Reif. Adaptive and compact discretization for weighted region optimal path finding. In *Fundamentals of Computation Theory*, volume 2751 of *Lecture Notes in Computer Science*, pages 258–270. 2003.

[16] Z. Sun and J.H. Reif. On robotic optimal path planning in polygonal regions with pseudo-euclidean metrics. *IEEE transactions on systems man and cybernetics Part B Cybernetics a publication of the IEEE Systems Man and Cybernetics Society*, 37(4):925–936, 2007.