Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Marina Bachran

Online Kernsuche in dreidimensionalen Terrains

4. September 2006

– Diplomarbeit –

Betreuer : Prof. Dr. rer. nat. Rolf Klein

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, den 4. September 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Ein großes Themengebiet in der algorithmischen Geometrie und der Robotik ist die Erkundung unbekannter Umgebungen durch einen Roboter. Ein Aspekt ist dabei den Bereich in einer Umgebung zu finden, von dem aus die komplette Umgebung einsehbar ist, sofern ein solcher Bereich überhaupt existiert. Dieser Bereich wird *Kern* genannt. In Strategien, die einen Weg in diesen Kern suchen, erkundet ein Roboter nach und nach die Umgebung und erweitert somit sukzessiv seinen Sichtbarkeitsbereich, bis er die gesamte Umgebung einsehen kann und er eine Position im Kern erreicht hat. Typisch für die angewendeten Strategien ist, dass aufgrund der wechselnden Sichtbarkeitsinformation ständig neue Bewegungsentscheidungen getroffen werden. Wegen dieser Vorgehensweise werden solche Strategien als Online-Strategien bezeichnet.

Eine häufig untersuchte Umgebung ist ein einfaches Polygon. Das Ziel der vorliegenden Arbeit ist es, basierend auf den Beobachtungen für ein einfaches Polygon, Strategien zu entwickeln und zu bewerten, mit deren Hilfe ein Roboter einen Weg in den Kern eines dreidimensionalen Terrains findet. Der Kern eines solchen Terrains ist ein nach oben offenes Polyeder. Die aus einer solchen Strategie resultierende Weglänge wird mit der optimalen Länge des kürzesten Weges in den Kern verglichen, um die Güte der Strategie abzuschätzen. Als Vorlage für die Entwicklung von Online-Strategien für das dreidimensionale Terrain wurde die *CAB* Strategie gewählt. Diese bestimmt in einem einfachen Polygon den Weg von einem Startpunkt in den Kern.

Das dreidimensionale Terrain beschreibt eine Berglandschaft, die durch Polygone angenähert wird. Um zu garantieren, dass immer ein Kern eines solchen Terrains existiert, werden Höhlen und senkrechte Berghänge ausgeschlossen. Der Roboter kann auf einem beliebigen Punkt auf der Terrainoberfläche starten. Er wird als punktförmig modelliert und kann fliegen. Weiterhin ist er mit einer unbeschränkten Rundumsicht ausgestattet. Seine Aufgabe ist es, anhand einer Online Strategie einen möglichst kurzen Weg in den Kern des Terrains zu finden. Für die Beurteilung der Güte der Online Strategie wird lediglich die Länge des zurückgelegten Weges einbezogen. Weitere Kosten, wie sie z. B. für die Planung des Weges anfallen, werden vernachlässigt. Die Beurteilung findet durch einen Vergleich mit der optimalen Lösung einer Offline Strategie statt. Die Lösung der Offline Strategie besteht in dem hier beschriebenen Szenarium aus einer geradlinigen Bewegung vom Startpunkt des Roboters in den Kern. Die Geradlinigkeit der Bewegung folgt aus der Definition des Kerns, da von jedem Punkt des Kerns das gesamte Terrain sichtbar ist, also insbesondere der Startpunkt des Roboters. Damit ist die Länge des kürzesten Weges in den Kern der minimale euklidische Abstand des Startpunktes zu dem Kern. In

einer Online Strategie ist dem Roboter das Terrain unbekannt. Aus diesem Grund wird er in den meisten Fällen einen längeren Weg zurücklegen, als den der Lösung der Offline Strategie. Fehlentscheidungen, die er auf seinem Weg trifft, verursachen zusätzliche Kosten. In dieser Arbeit wird daher eine untere Schranke für die Gesamtkosten einer Online Strategie im Vergleich zur optimalen Lösung bewiesen werden.

Im Folgenden wird die Gliederung dieser Arbeit ausgeführt. In Kapitel 2 werden zunächst die Grundlagen erläutert, die für das Verständnis der nachfolgenden Kapitel notwendig sind. In diesem Kapitel wird zudem die oben erwähnte *CAB* Strategie beschrieben. Kapitel 3 stellt eine Offline Strategie für die Kernsuche im dreidimensionalen Terrain vor. Diese Offline Strategie bildet die Grundlage für die Beurteilung von Online Strategien. Anschließend werden in Kapitel 4 mehrere Online Strategien vorgestellt. Das Kapitel beinhaltet zunächst eine genauere Untersuchung von Kriterien bezüglich der Sichtbarkeit im dreidimensionalen Terrains. In diesem Zusammenhang wird unter anderem der Begriff des Horizonts eingeführt, der eine wichtige Grundlage für die vorgestellten Online Strategien darstellt. In Bezug auf die Online Strategien werden zunächst einige einfache Strategien vorgestellt und deren Schwachstellen aufgezeigt. Basierend auf den Erkenntnissen daraus werden Vorschläge gemacht, die in komplexe Online Strategien integriert werden können. Eine solche komplexe Strategie wird vorgestellt und bewertet. Daraufhin folgt eine Diskussion über die Übertragung von Ideen der *CAB* Strategie auf das dreidimensionale Terrain. Abgeschlossen wird das Kapitel mit einer allgemeinen Diskussion der Weglänge von Online Strategien für die Kernsuche. Kapitel 5 beschäftigt sich mit der Implementierung der in den Kapiteln 3 und 4 vorgestellten Strategien. Es werden einige wichtige Elemente und Probleme der Implementierung vorgestellt. Das zugehörige Applet wird zukünftig unter *www.geometrylab.de/Kernel3D/* im Internet zu finden sein. Das Benutzerhandbuch zu dem Applet ist im Anhang. Die Diplomarbeit wird mit der Zusammenfassung der wesentlichen Ergebnisse in Kapitel 6 abgeschlossen.

Kapitel 2

Grundlagen

In diesem Kapitel werden Grundlagen für die darauf folgenden Kapitel erläutert. In Abschnitt 2.1 wird zunächst der Sichtbarkeitsbegriff in einfachen Polygonen eingeführt. Abschnitt 2.2 beschreibt die Anwendung dieses Begriffs auf ein dreidimensionales Terrain. Darauf folgt in Abschnitt 2.3 die Definition einer vollständigen Sicht innerhalb des Polygons oder Terrains. In Abschnitt 2.4 werden die Begriffe Offline und Online Algorithmus eingeführt, in Abschnitt 2.5 wird schließlich ein Online Algorithmus zur Suche des Kerns innerhalb eines einfachen Polygons vorgestellt.

2.1 Die Sichtbarkeit in einem einfachen Polygon

Der Sichtbarkeitsbereich beschreibt den Teil eines Raums bzw. Objektes der von einem darin liegenden Punkt p direkt einsehbar ist. Dies bedeutet, dass das Liniensegment zwischen dem Punkt p und einem beliebigen Punkt des einsehbaren Teilraums vollständig innerhalb des Raums bzw. Objektes liegt.

In Abbildung 2.1 ist der Punkt a von a' aus sichtbar. Der Punkt b sieht den Punkt b', da das Liniensegment (b, b') vollständig in O liegt und den Rand von O berührt, aber nicht echt schneidet. Zwischen den Punkten c und c' besteht jedoch keine Sichtbarkeit, da das Liniensegment (c, c') den Rand des Objektes O echt schneidet.

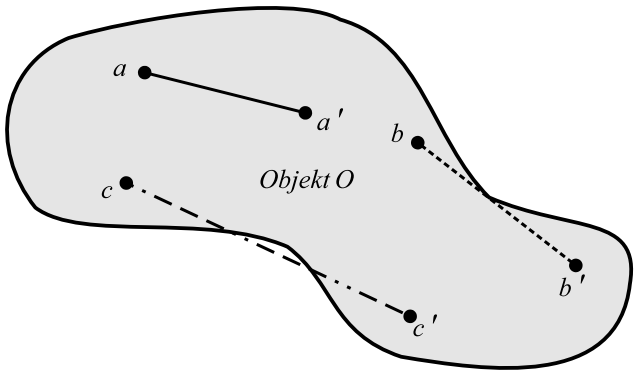

Abbildung 2.1: *Sichtbarkeit.*

Im Folgenden wird diese Beschreibung der Sichtbarkeit auf ein einfaches Polygon angewendet, wie es in [2] beschrieben ist.

> **Definition 2.1:** Ein einfaches Polygon ist eine abgeschlossene Region, die von einer einzelnen polygonalen Kette, welche keine Selbstschnitte aufweist, umschlossen wird. Insbesondere enthält diese Region keine Löcher.

Für die Sichtbarkeit in einem einfachen Polygon nach [1] gilt:

> **Definition 2.2:** Sei P ein einfaches Polygon. Ein Punkt $q \in P$ ist von p aus *sichtbar*, wenn das Liniensegment pq ganz in P enthalten ist.

Und somit:

> **Definition 2.3:** Die Menge $vis(p)$ aller von p aus sichtbaren Punkte heißt das *Sichtbarkeitspolygon* von p in P.

Abbildung 2.2 zeigt ein Beispiel für ein solches Sichtbarkeitspolygon $vis(q)$ eines Punktes q des Polygons P. Dieses ist hellgrau eingefärbt. Die Kanten des Sichtbarkeitspolygons werden aus Kanten bzw. Kantenteilen des Polygons P und aus künstlichen Kanten, die sich auf spitze Ecken des Polygons stützen, gebildet. Eine solche spitze Ecke hat einen Innenwinkel größer als π und schränkt somit den Sichtbereich von q im Polygon ein.

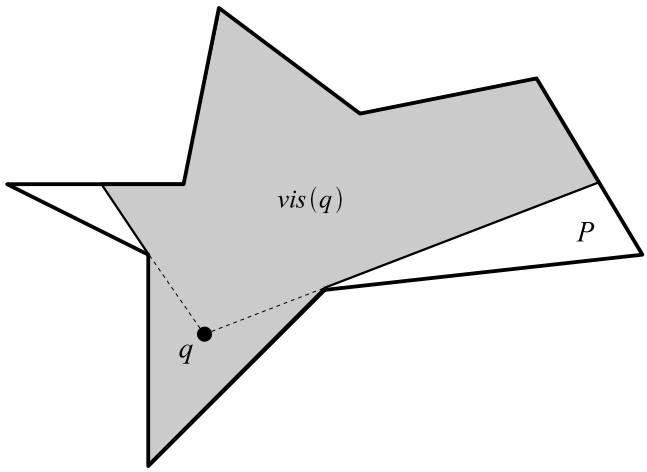

Abbildung 2.2: *Das Sichtbarkeitspolygon $vis(q)$ in P.*

2.2 Die Sichtbarkeit in einem dreidimensionalen Terrain

Bevor die Sichtbarkeit in einem dreidimensionalen Terrain definiert wird, werden zunächst die Eigenschaften dieses Terrains erläutert. Anschaulich stellt es eine Gebirgslandschaft

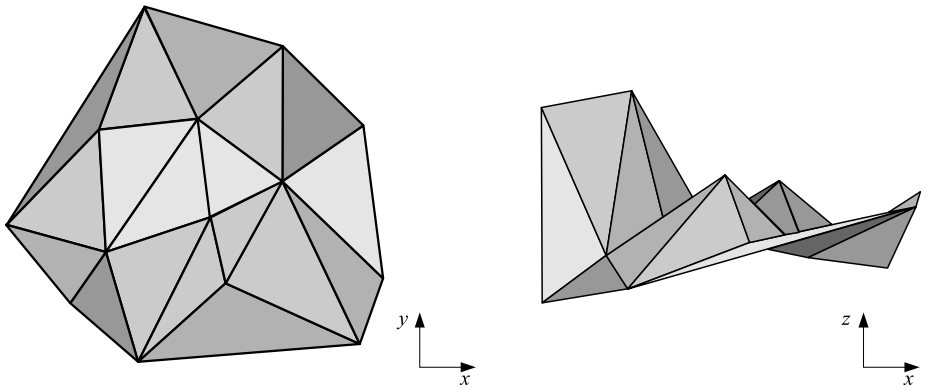

Abbildung 2.3: *Ein Terrain wird durch Polygone angenähert.*

mit Bergen und Tälern dar, wobei keine Höhlen erlaubt sind. Weiterhin werden keine senkrechten Berghänge zugelassen. Ein solches Terrain wird in Abbildung 2.3 dargestellt.

Mit T wird im Folgenden immer das Terrain bezeichnet, während $O(T)$ die Oberfläche des Terrains T ist. Der Bereich oberhalb der Oberfläche wird mit $L(T)$ benannt, die Erdmasse unterhalb der Oberfläche wird durch $E(T)$ angegeben. Zusätzlich ist der Abschluss $\overline{L(T)} = O(T) \cup L(T)$ von Bedeutung. Das Terrain T besteht somit aus den drei Teilmengen $L(T)$, $O(T)$ und $E(T)$. Die hier eingeführten Bezeichnungen sind in Abbildung 2.4 dargestellt. Künftig wird davon ausgegangen, dass $O(T)$ durch einfache Polygone gegeben ist, deren Eckpunkte in einer Ebene liegen. Diese Polygone werden von nun an auch als Facetten bezeichnet. Durch Triangulierung ist $O(T)$ damit immer durch Dreiecke darstellbar. Im Folgenden wird implizit angenommen, dass $O(T)$ trianguliert ist. Dabei sind die Z-Koordinaten der Normalenvektoren der Oberflächenfacetten positiv. Die Normalenvektoren zeigen also in $L(T)$ hinein.

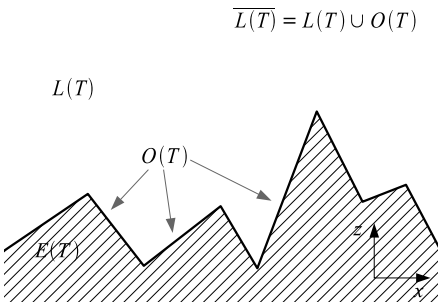

Abbildung 2.4: *Bezeichnungen im Terrain.*

Sichtbarkeit zwischen zwei Punkten p und q in einem dreidimensionalen Terrain bedeutet, dass sowohl p als auch q in derselben Menge und zwar in $\overline{L(T)}$ enthalten sein müssen und dass das Liniensegment (p, q) die Oberfläche $O(T)$ berühren darf, aber nicht

echt schneidet.

Definition 2.4: Sei T ein dreidimensionales Terrain. Ein Punkt $q \in \overline{L(T)}$ ist von einem Punkt $p \in \overline{L(T)}$ aus *sichtbar*, wenn das Liniensegment pq ganz in $\overline{L(T)}$ enthalten ist.

Der Begriff der Sichtbarkeit wird daher für eine Facette der Oberfläche wie folgt definiert:

Definition 2.5: Sei T ein dreidimensionales Terrain, F eine Facette von $O(T)$ und p ein Punkt aus $\overline{L(T)}$. Ferner sei E die zu F zugehörige Ebene. F ist sichtbar, wenn $d(E,p) \geq 0$ gilt, und nicht sichtbar, wenn $d(E,p) < 0$ gilt. Dabei ist $d(E,p)$ gleich null, falls $p \in E$ gilt, $d(E,p)$ ist größer als null, wenn p in dem Halbraum von E über E liegt, in den der Normalenvektor von E zeigt. Analog dazu ist $d(E,p)$ kleiner als null, wenn p in dem Halbraum von E liegt, in den der Normalenvektor von E nicht zeigt.

Analog zu dem Sichtbarkeitspolygon in Abschnitt 2.1 wird ein *Sichtbarkeitspolyeder* definiert:

Definition 2.6: Die Menge aller Punkte von $\overline{L(T)}$, die von einem Punkt $p \in \overline{L(T)}$ aus sichtbar sind, werden als Sichtbarkeitspolyeder $visP(p)$ des Punktes p bezeichnet.

Beobachtung 2.1: Das Sichtbarkeitspolyeder $visP(p)$ ist nach oben offen.

Abbildung 2.5 zeigt ein Beispiel für das Sichtbarkeitspolyeder $visP(p)$ eines Punktes p. Zur Vereinfachung der Darstellung wird dabei nur ein Querschnitt des Terrains bzw. des Sichtbarkeitspolyeders gezeigt. Es wird deutlich, dass das Sichtbarkeitspolyeder $visP(p)$ eines Punktes p nicht unbedingt konvex ist. Die Facetten des Polyeders $visP(p)$ entstehen entweder durch Ebenen, die sich auf Kanten des Terrains stützen oder durch Facetten bzw. Facettenteile des Terrains.

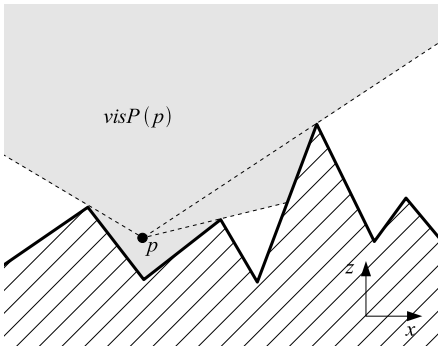

Abbildung 2.5: *Das Sichtbarkeitspolyeder $visP(p)$ eines Punktes im Querschnitt.*

Für Algorithmen zur Bestimmung des Sichtbarkeitspolyeders ist der Einfluss einzelner Facetten der Terrainoberfläche auf die Struktur des Sichtbarkeitspolyeders von Bedeutung.

In diesem Zusammenhang ist ein Begriff hilfreich, welcher die Entfernung von Facetten zu einem Punkt p beschreibt. Die folgende Definition wird in späteren Kapiteln verwendet:

> **Definition 2.7:** Sei T ein dreidimensionales Terrain, e eine Kante aus $O(T)$, $O(T)$ trianguliert und p ein Punkt aus $\overline{L(T)}$. Ferner seien F_1 und F_2 die an e angrenzenden Facetten. Sei nun p_1 der Schwerpunkt der Projektion von F_1 in die X/Y-Ebene, die durch das Weglassen der Z-Koordinate gegeben ist. Entsprechend sei der Schwerpunkt p_2 für F_2 definiert. F_1 heißt näher liegend, wenn $d(p, p_1) \leq d(p, p_2)$ und weiter entfernt, wenn $d(p, p_1) > d(p, p_2)$. Analog heißt F_2 näher liegend, wenn $d(p, p_2) \leq d(p, p_1)$ und weiter entfernt, wenn $d(p, p_2) > d(p, p_1)$. Dabei ist $d(p, p_i)$ der euklidische Abstand der Punkte p und p_i.

2.3 Der Kern

In den Abschnitten 2.1 und 2.2 wurden die Begriffe Sichtbarkeitspolygon und Sichtbarkeitspolyeder eingeführt. Es stellt sich nun die Frage, ob es auch Punkte p gibt, von denen aus das ganze Polygon, bzw. das ganze Terrain einsehbar ist. Für ein einfaches Polygon gilt, dass es für eine bestimmte Klasse von einfachen Polygonen solche Punkte p gibt.

> **Definition 2.8:** Ein einfaches Polygon P heißt *sternförmig*, wenn es einen Punkt p mit $vis(p) = P$ gibt.

Die Gesamtheit aller dieser Punkte wird der *Kern* von P genannt:

> **Definition 2.9:** Der Kern ist die Gesamtheit aller Punkte p mit $vis(p) = P$, also
>
> $$ker(P) = \{p \in P; vis(p) = P\}$$

Die beiden zuvor aufgeführten Definitionen sind [1] entnommen. Abbildung 2.6 zeigt ein Beispiel für ein Polygon P mit seinem Kern. Nicht jede Kante des Polygons trägt zum Kern bei. Durch die Verlängerung der Polygonkanten zu Geraden werden jeweils zwei Halbebenen definiert. Jeweils eine der beiden Halbebenen liegt auf der Innenseite der zugehörigen Polygonkante. Der Schnitt dieser Halbebenen ergibt den Kern des Polygons.

In [1] werden einige Algorithmen zur Bestimmung des Kerns eines einfachen Polygons vorgestellt. Dies gilt auch für den in Abschnitt 2.5 beschriebenen Online-Algorithmus.

Zuvor wurde der Kern eines einfachen Polygons betrachtet. Im Folgenden soll dazu eine Analogie im dreidimensionalen Terrain gefunden werden. Charakteristisch für einen Punkt p im Kern eines Polygons P ist, dass $vis(p) = P$ gilt. Daher gilt für den Kern im dreidimensionalen Terrain:

> **Definition 2.10:** Der Kern eines dreidimensionalen Terrains T ist die Gesamtheit aller Punkte p mit $visP(p) = \overline{L(T)}$, also:
>
> $$ker(T) = \{p \in \overline{L(T)}; visP(p) = \overline{L(T)}\}$$

Aufgrund der in Abschnitt 2.2 erläuterten Einschränkungen, dass das Terrain keine Höhlen besitzt und keine senkrechten Berghänge zugelassen werden, existiert immer ein Kern.

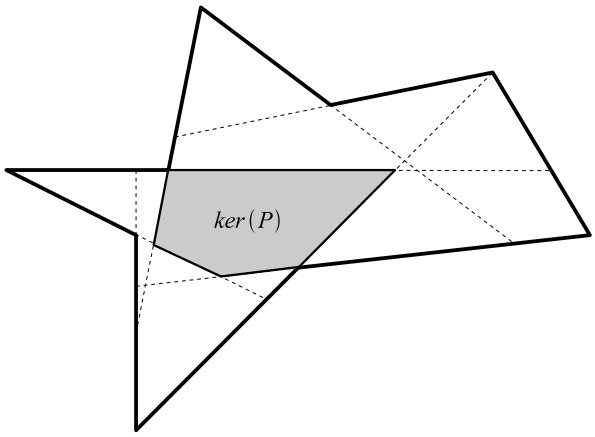

Abbildung 2.6: *Das Polygon P mit dem Kern $ker(P)$.*

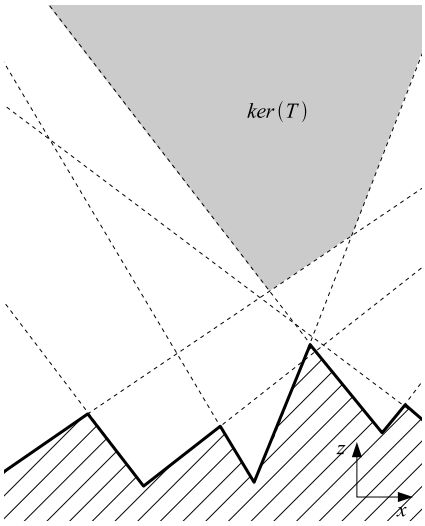

Abbildung 2.7: *Das Terrain T mit dem zugehörigen Kern $ker(T)$.*

Abbildung 2.7 zeigt den Querschnitt eines Terrains T und den zugehörigen Kern $ker(T)$. Der Kern wird begrenzt durch Ebenen, die durch die Facetten von $O(T)$ gegeben sind. Nicht alle Ebenen tragen zum Kern bei. In Analogie zum Halbebenenschnitt beim Polygon kann der Kern des Terrains auch als oberer Halbraumschnitt der zu den Facetten des Terrains zugehörigen Ebenen aufgefasst werden. Wie aus der Abbildung zu entnehmen ist, hat der Kern die folgenden zwei Eigenschaften:

Beobachtung 2.2: Der Kern $ker(T)$ ist konvex und nach oben offen.

Begründung: Jedes Polygon der Terrainoberfläche definiert eine Ebene. Die Facetten des Kerns $ker(T)$ werden durch einige oder alle dieser Ebenen bestimmt. Der Kern ist konvex, wie im folgenden bewiesen wird. Wäre der Kern nicht konvex, so gäbe es zwei Punkte p und q, die innerhalb des Kerns liegen und deren Verbindungslinie jedoch nicht komplett im Kern liegt, also eine Facette des Kerns echt schneidet. Da jede Facette des

Kerns durch eine Ebene bestimmt wird, liegen p und q auf verschiedenen Seiten dieser Ebene. Somit kann einer der beiden Punkte p und q nicht die Oberfläche des zugehörigen Polygons der Terrainoberfläche sehen. Dies steht im Widerspruch zur Definition des Sichtbarkeitspolyeders.

Darüber hinaus ist der Kern nach oben offen, da das Terrain keine Höhlen enthält und damit auch kein Oberflächenpolygon, das das Sichtbarkeitspolyeder nach oben hin begrenzen könnte.

Eine weitere Eigenschaft des Kerns im dreidimensionalen Terrain ist, dass der kürzeste Weg von einem beliebigen Punkt p auf der Oberfläche $O(T)$ des Terrains in den Kern $ker(T)$ entweder in einem Eckpunkt des Kerns endet, oder ein Lot auf eine Fläche oder eine Kante des Kerns ist.

2.4 Offline und Online Algorithmen

In diesem Abschnitt werden allgemeine Unterschiede zwischen Online und Offline Algorithmen beschrieben. Einem Offline-Algorithmus sind alle Angaben zum Problem vor dem Start des Algorithmus bekannt. Somit kann, sofern überhaupt möglich, eine optimale Lösung berechnet werden. Ein Online-Algorithmus hingegen erhält diese Daten jedoch erst nach und nach zur Laufzeit des Algorithmus. Es kann also oft nur versucht werden, eine möglichst gute Lösung für das Problem zu finden. Bei einer Entscheidung zur Laufzeit des Algorithmus kommt es also darauf an, die bisher erhaltenen Daten möglichst gut zu bewerten und aufgrund dieser Bewertung den nächsten Schritt zu planen. Diese Entscheidung muss dann gegebenenfalls im nächsten Schritt wieder korrigiert werden. Daher ist es ein Ziel beim Entwurf eines Online-Algorithmus, den Aufwand solcher Korrekturen gering zu halten.

Eine Bewertung der Güte des Online-Algorithmus erfolgt über den Vergleich mit der optimalen Lösung des Problems. Zu diesem Zweck wird der Begriff der Kompetitivität verwendet. Die Lösung des Online-Algorithmus darf maximal nur C mal so groß sein, wie die optimale Lösung, ansonsten ist der Online-Algorithmus nicht kompetitiv. Die formale Definition von *C-kompetitiv* gemäß [3] lautet:

> **Definition 2.11:** Sei π ein Problem und $P \in \pi$ eine Instanz von π. Ferner sei $OPT(P)$ die optimale Lösung von P. Ein Online Algorithmus ALG zur Lösung von π ist *C-kompetitiv*: $\Leftrightarrow \exists A \; \forall P \in \pi : ALG(P) \leq C * OPT(P) + A$.

2.5 Die Kernsuche im zweidimensionalen Raum

In diesem Abschnitt wird die Strategie *CAB* aus [4] und [5] beschrieben. Ein Roboter mit einem Sichtsystem startet in einer ihm unbekannten Umgebung. Diese Umgebung wird durch ein einfaches Polygon P beschrieben. Der Roboter soll nun einen möglichst kurzen Weg in den Kern des Polygons finden. Eine Lösung existiert nur dann, wenn der Kern nicht leer ist. Das Polygon muss also, wie in Definition 2.8 beschrieben, sternförmig sein. Im Folgenden werden nur sternförmige Polygone betrachtet.

2.5.1 Eine untere Schranke

Die Ausgangssituation ist wie folgt: Ein Roboter startet in dem Punkt s in einem unbekannten, einfachen Polygon P. Sein Ziel ist es, den nächstgelegenen Punkt k von $ker(P)$ zu erreichen. Der vom Roboter zurückgelegte Weg wird anschließend mit der optimalen Lösung verglichen. Die optimale Lösung besteht aus der Weglänge $|sk|$ von der Startposition s des Roboters und dem nächstgelegenen Punkt k des Kerns. Dies lässt sich darüber begründen, dass von k aus das gesamte Polygon einsehbar ist. Somit ist s sichtbar, weshalb auch das Liniensegment (s, k) ganz in P liegen muss.

Der Roboter sieht von seinem Startpunkt aus einen Teil von P ein. In diesem sichtbaren Teil muss der Kern $ker(P)$ enthalten sein. Der Roboter hat jedoch keine Möglichkeit, die Punkte des Kerns von den übrigen Punkten von P zu unterscheiden

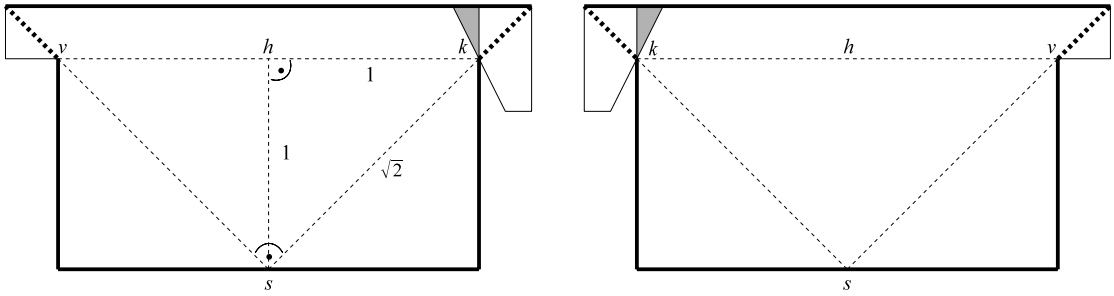

Abbildung 2.8: *Identische Sichtbarkeitspolygone von s und unterschiedliche Kerne [1].*

In Abbildung 2.8 ist ein Beispiel dargestellt, in dem die Sichtbarkeitspolygone gleich sind, die Kerne jedoch auf unterschiedlichen Seiten liegen. Das rechte Polygon ist hierbei durch Spiegelung des linken an der Senkrechten durch s entstanden. Aus diesem Beispiel kann eine *untere Schranke* für den kompetitiven Faktor hergeleitet werden. Der Beweis für diese Behauptung befindet sich in [1]. Die Grundidee des Beweises basiert darauf, dass die beste Online Strategie zuerst zum Segment h läuft, um dann nach rechts oder links in den Kern zu gelangen. Diese Strategie führt zu einer Weglänge von 2, während die optimale Weglänge $\sqrt{2}$ beträgt. Daraus resultiert die untere Schranke von $\sqrt{2}$.

> **Theorem 2.1:** Gibt es überhaupt eine kompetitive Strategie, mit der ein Roboter den nächstgelegenen Punkt des Kerns erreichen kann, so beträgt ihr kompetitiver Faktor mindestens $\sqrt{2}$.

Diese untere Schranke für eine kompetitive Strategie, die einen Weg in den Kern eines Polygons findet, wurde von A. López-Ortiz in [6] auf ≈ 1.486429521 verbessert.

2.5.2 Die Strategie *CAB*

An seiner Ausgangsposition befindet sich der Roboter entweder bereits im Kern oder er sieht nur einen Teil des Polygons P. Es ist ihm jedoch bekannt, dass der Kern, sofern einer existiert, wovon im Folgenden ausgegangen wird, von seiner aktuellen Position aus sichtbar ist. Dies folgt daraus, dass seine Position von jedem Kernpunkt aus sichtbar ist. Weiterhin ist bekannt, dass das Sichtbarkeitspolygon $vis(p)$ bei einer geradlinigen

Bewegung auf den Kern $ker(P)$ zu monoton wächst, bis schließlich ganz P einsehbar ist. Die entscheidende Idee der *CAB*-Strategie ist daher auch, dass der Roboter seinen Weg so wählt, dass $vis(p)$ beständig wächst. Der direkte, gerade Weg in den Kern ist dem Roboter leider nicht bekannt. Die genaue Vorgehensweise wird im Folgenden anhand von Abbildung 2.9 genauer beschrieben.

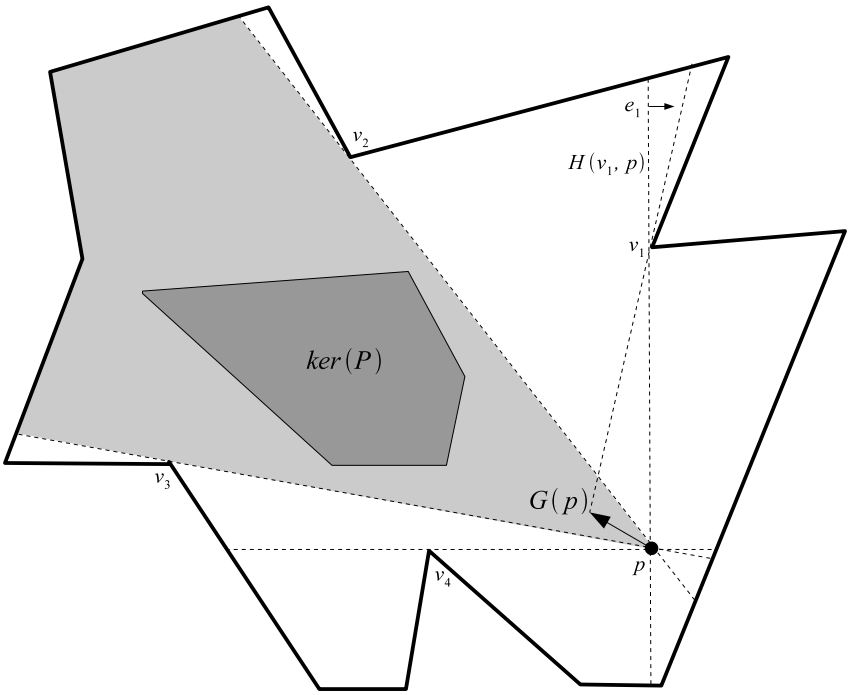

Abbildung 2.9: *Die Strategie CAB führt den Roboter entlang der Winkelhalbierenden in den hellgrau eingezeichneten Bereich $G(p)$ [1].*

Für die Entscheidung, wohin der Roboter sich bewegen soll, werden von p ausgehend künstliche Kanten eingefügt, die sich auf die sichtbaren spitzen Ecken stützen. Spitze Ecken sind Ecken, deren Winkel im Innern des Polygons größer als 180° ist. In dem Beispiel werden dadurch vier künstliche Kanten zu $vis(p)$ hinzugefügt. Jede künstliche Kante e_i zu einer spitzen Ecke v_i definiert zwei Halbebenen. Eine dieser Halbebenen, genannt $H(v_i, p)$, enthält den Kern und liegt auf der von p aus sichtbaren Seite. Wenn nun $vis(p)$ vergrößert werden soll, muss sich p innerhalb von $H(v_i, p)$ bewegen. Dabei rotiert die Kante e_i um v_i. In Abbildung 2.9 wird diese Bewegung für die Ecke v_1 gezeigt.

Das Prinzip der Halbebenen kann auf alle künstlichen Kanten angewendet werden. Für jede spitze Ecke wird die Halbebene $H(v_i, p)$ definiert, die den Kern enthält. Wird der Schnitt aller dieser Halbebenen gebildet, so ist der Kern komplett in diesem Schnitt enthalten. Eine Bewegung des Roboters in diesen Bereich wird im Allgemeinen das Sichtbarkeitspolygon vergrößern. Der Schnitt wird daher

$$\text{Gewinnbereich } G(p) = \bigcap_i H(v_i, p)$$

genannt. Da die künstlichen Kanten, durch die die Halbebenen definiert sind, alle von p ausgehen, ist durch $G(p)$ ein Winkelbereich um p gegeben, der von nicht mehr als zwei künstlichen Kanten begrenzt wird. Die an diesen zwei künstlichen Kanten liegenden spitzen Ecken werden die *wesentlichen Ecken* von $vis(p)$ genannt. Der Roboter versucht der Winkelhalbierenden des Winkelbereichs von $G(p)$ zu folgen. Die wesentlichen Ecken und damit auch die Winkelhalbierende können sich dabei ändern. Aus diesem Grund wurde die Strategie *CAB* (= continuous angular bisector) genannt.

Die Bewegung des Roboters ist im Allgemeinen keine geradlinige Bewegung. Durch die ständige Aktualisierung der Winkelhalbierenden bewegt sich der Roboter im allgemeinen Fall auf einem Hyperbelbogen. Es kann jedoch auch zu einer Bewegung auf einem Kreis- oder Ellipsenbogen kommen. In speziellen Fällen ist auch eine geradlinige Bewegung möglich. Anhand des Beispiels in Abbildung 2.10 sollen die einzelnen Fälle näher erläutert werden. Mit den v_i werden wieder die spitzen Ecken bezeichnet, während die p_i die jeweiligen aktuellen Position des Roboters angeben. Die beiden g_i kennzeichnen einen Sonderfall, der später genauer erklärt wird.

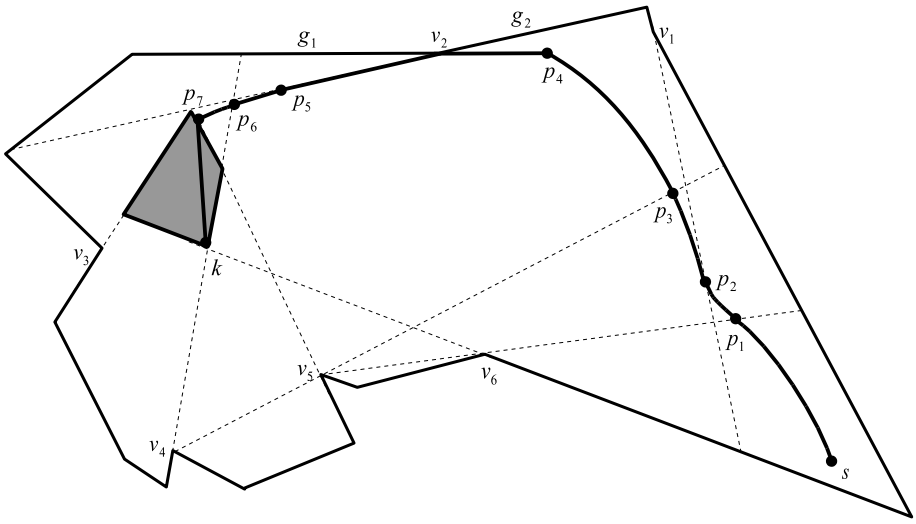

Abbildung 2.10: *Der von der CAB Strategie geplante Weg in den Kern [1].*

Vom Startpunkt s sind die zwei spitzen Ecken v_1 und v_6 nicht voll einsehbar. Es wird die Winkelhalbierende zu dem aktuellen Gewinnbereich gebildet. Folgt der Roboter der jeweils aktuellen Winkelhalbierenden, dann verändert sich die Differenz der Abstände $|pv_1| - |pv_6|$ nicht. Aufgrund dieser Bedingung an die Bewegung auf der Winkelhalbierenden wird eine Hyperbelbahn definiert, die die Brennpunkte v_1 und v_6 hat und sich zum näheren der beiden Punkte krümmt. Das Beispiel aus Abbildung 2.9 stellte hierbei einen Sonderfall dar. Ist der Abstand zu den beiden spitzen Ecken gleich, so degeneriert die Hyperbel zu einer Geraden.

Erreicht der Roboter in dem Beispiel aus Abbildung 2.10 den Punkt p_1, so wird hinter v_6 die Ecke v_5 sichtbar. Damit ändern sich die wesentlichen Ecken, v_5 nimmt nun die Stelle von v_6 ein, da die Halbebene $H(v_5, p)$ ab jetzt ein Stück von $H(v_6, p) \cap H(v_1, p)$ abschneidet. Ab p_1 folgt der Roboter also wieder einem Hyperbelbogen, die Brennpunkte sind diesmal v_1 und v_5 und die Hyperbel krümmt sich in die entgegengesetzte Richtung zu v_1.

In p_2 tritt wieder eine Änderung auf, da der Roboter die spitze Ecke v_1 nun einsehen kann. Damit fällt diese Ecke als wesentliche Ecke weg. Auch v_6 ist schon voll einsehbar. Damit bleibt als wesentliche Ecke nur v_5 übrig. Der Gewinnbereich kann in diesem Fall kein von zwei Geraden begrenzter Winkelbereich sein und besteht aus der Halbebene $H(v_5, p)$. Die Winkelhalbierende steht in diesem Fall senkrecht auf der Geraden durch v_5 und p. Hieraus ergibt sich für die Bewegung des Roboters somit ein Kreisbogen um v_5.

In p_3 tritt das nächste Ereignis auf, da nun die spitze Ecke v_4 sichtbar wird. Nun gibt es wieder zwei wesentliche Ecken, die aber in diesem Fall auf der gleichen Seite des Gewinnbereichs liegen. Von den beiden Punkten liegt nur v_4 auf dem Rand des Gewinnbereichs. Um dem abzuhelfen, wird v_5 nun an der Gerade durch v_5 und p gespiegelt. Damit erhält man den Punkt w_5, wie auch Abbildung 2.11 zeigt. Der Roboter folgt nun der Winkelhalbierenden von v_4 und w_5. Dabei bewegt sich w_5 ständig weiter. Man kann sich überlegen, dass die Summe der Abstände $|pv_4| + |pv_5|$ zu den wesentlichen Ecken gleich bleibt. Dadurch wird eine Ellipse mit den Brennpunkten v_4 und v_5 definiert. Damit ist die zuvor beschriebene Kreisbewegung ein Spezialfall einer Ellipse, bei der die beiden Brennpunkte zusammenfallen.

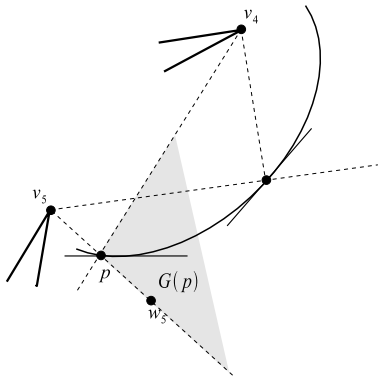

Abbildung 2.11: *v_4 und v_5 sind die beiden Brennpunkte der Ellipsenbahn des Roboters.*

In dem Punkt p_4 tritt ein weiterer Spezialfall auf. Würde der Roboter dem Ellipsenbogen über p_4 hinaus folgen, so würde sich sein Sichtbarkeitspolygon wieder verkleinern. Er würde die Sicht auf die Kante g_1 des Polygons verlieren. Um dies zu verhindern, muss der Roboter an der Verlängerung von g_1 entlang gleiten und ab v_2 an der Verlängerung von g_2. Währenddessen bleiben v_4 und v_5 die wesentlichen Ecken. Der Roboter darf also den Durchschnitt der inneren Halbebenen der Kanten g_1 und g_2 nicht verlassen. Dieser Durchschnitt ist jedoch nicht auf diese zwei Kanten beschränkt. Allgemein wird der Durchschnitt als Haltebereich $E(p)$ bezeichnet und ist der innere Durchschnitt aller inneren Halbgeraden aller Kanten e, die von p aus sichtbar sind und außerdem p in ihrem Inneren oder in ihrer Verlängerung enthalten.

Sobald die Winkelhalbierende nicht mehr in den durch den Haltebereich verbotenen Bereich hinter der Verlängerung von g_2 zeigt, muss der Roboter der Winkelhalbierenden wieder folgen. Dies geschieht in p_5. In p_6 kann der Roboter die spitze Ecke v_4 einsehen und folgt nun wieder einem Kreisbogen um v_5. In p_7 erreicht der Roboter den Kern. Er kann nun das gesamte Polygon einsehen und kann damit auch den kompletten Kern berechnen. Seine Aufgabe ist aber noch nicht beendet, da er den kürzesten Weg in den Kern suchen

sollte. Er besitzt jetzt alle notwendigen Informationen, um sich direkt zu dem Punkt k zu begeben, der der zu s nächste Punkt im Kern ist.

Im nächsten Abschnitt soll diese Strategie analysiert werden. Es soll der kompetitive Faktor, d. h. das Verhältnis des zurückgelegten Weges zum direkten Weg in den Kern, der Strecke $|sk|$, ermittelt werden.

2.5.3 Die Weglänge der Strategie *CAB*

In diesem Abschnitt wird ein kompetitiver Faktor für die *CAB*-Strategie bestimmt. In Bezug auf die Anzahl der Wegstücke gilt laut [1]:

> **Lemma 2.1:** Der von der Strategie *CAB* in einem Polygon mit n Ecken erzeugte Weg besteht aus $O(n)$ vielen Stücken.

Bei der Bestimmung der Weglänge der Stücke insgesamt tritt das Problem auf, dass der Weg unterschiedliche Stücke enthält. Bei den Ellipsenstücken gibt es z.B. die Schwierigkeit, dass das Integral für die Weglänge keine geschlossene Darstellung besitzt. Diese Problematik wird über die Eigenschaft der Wege umgangen, dass sie *selbstnähernd* sind. Das Prinzip der *selbstnähernden* Wege wird in Abbildung 2.12 dargestellt.

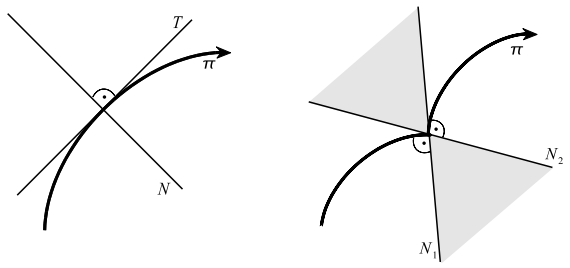

Abbildung 2.12: *Selbstnähernde Wege.*

Ein Punkt p im Inneren eines aus endlichen vielen glatten Stücken bestehenden Weges π besitzt, wie im linken Teil der Abbildung gezeigt, eine eindeutig bestimmte Tangente T und eine auf T senkrecht stehende Normale N. Falls p ein Punkt von π ist, in dem zwei Stücke des Weges aufeinander treffen, so werden durch die beiden Normalen der Stücke ein Normalenbündel definiert. Dies ist im rechten Teil der Abbildung dargestellt. Jede dieser Normalen wird innerhalb von π als Normale durch den Punkt p aufgefasst.

> **Definition 2.12:** Ein Weg π heißt *selbstnähernd*, wenn er die folgenden Eigenschaften erfüllt:
>
> - π ist stückweise glatt
> - π ist von s nach t orientiert
> - für jeden Punkt p und jede Normale N in p gilt: Das restliche Wegstück von p bis t liegt ganz in der abgeschlossenen Halbebene vor der Normalen N

Anschaulich bedeutet dies, dass bei drei auf π aufeinander folgenden Punkten a, b und c gegeben ist, dass b näher an c liegt als a. Ein selbstnähernder Weg π kommt also in jedem

Punkt p seinen 'zukünftigen' Punkten näher. Mehr Informationen über selbstnähernde Kurven finden sich in [7]. Für die von CAB erzeugten Wege gilt:

Theorem 2.2: Jeder von der Strategie CAB erzeugte Weg ist selbstnähernd.

Der vollständige Beweis findet sich in [1] und in [4]. Dieser stützt sich im Wesentlichen auf die drei folgenden Aussagen. Die erste Aussage beschreibt eine besondere Eigenschaft eines jeden Punktes p auf dem Weg π' in den Kern. Der Punkt p ist ein Eckpunkt des Kerns seines Sichtbarkeitspolygons $vis(p)$.

Lemma 2.2: In einer hinreichend kleinen Umgebung eines Punktes $p \in P$ stimmt $ker(vis(p))$ mit dem Durchschnitt der Winkelbereiche $G(p) \cap E(p)$ überein.

Durch das Verfolgen der CAB Strategie bewegt sich der Roboter so, dass sein Sichtbarkeitspolygon wächst, bis der Roboter schließlich das ganze Polygon P einsieht. Dabei schrumpft der Kern des Sichtbarkeitspolygons auf den Kern des gesamten Polygons.

Lemma 2.3: Seien p, q Punkte im Polygon P mit $vis(p) \subseteq vis(q)$. Dann gilt:

$$ker(P) \subseteq ker(vis(q)) \subseteq ker(vis(p)).$$

Durch das Schrumpfen des Kerns des Sichtbarkeitspolyeders ergibt sich folgendes Korollar:

Korollar 2.1: Für jeden Punkt p auf dem Weg des Roboters ist der Rest des Weges ab p ganz in $ker(vis(p))$ enthalten.

Aus den obigen Aussagen ergibt sich somit, dass der zurückgelegte Weg der CAB Strategie selbstnähernd ist. Laut [7] gilt für solche Wege auch:

Theorem 2.3: Ein selbstnähernder Weg kann höchstens $5,3331...$ mal so lang sein wie der Abstand seiner Endpunkte. Diese Schranke ist scharf.

Es folgt damit, dass CAB kompetitiv mit dem Faktor $5,34$ ist. Lee und Chwa beweisen in [8] einen besseren kompetitiven Faktor für die CAB Strategie. Dieser beträgt $\pi + 1$. In [9] wird ein weiterer Online Algorithmus für die Kernsuche in einem einfachen Polygon vorgestellt. Für diesen wurde ein kompetitiven Faktor von $1 + 2\sqrt{2}$ gezeigt.

Kapitel 3

Offline Kernsuche in einem dreidimensionalen Terrain

In diesem Abschnitt wird der Offline-Algorithmus zur Bestimmung des kürzesten Weges in den Kern vorgestellt. Dazu wird zunächst ein Algorithmus zur Bestimmung des Kerns erläutert. Wie in Abschnitt 2.3 bereits beschrieben wurde, tragen nicht immer alle Facetten des Terrains zum Kern bei. Ein Algorithmus lässt sich damit in zwei Aufgabenteile unterteilen. Die erste Aufgabe ist die Bestimmung der Facetten, die für den Kern von Bedeutung sind, während die zweite Aufgabe genaue Bestimmung der Form des Kerns aus diesen Facetten ist. Für beide dieser Aufgaben lässt sich das Prinzip der Dualität nutzen, wie es in Abschnitt 3.1 eingeführt wird. Die Grundidee dabei ist, dass die Bestimmung des Halbraumschnitts, welcher dem Kern entspricht, alternativ auch im dualen Raum durch die Berechnung einer konvexen Hülle durchgeführt werden kann. In Abschnitt 3.2 wird ein Algorithmus zur Bestimmung der konvexen Hülle in dem dualen Raum vorgestellt. Nach der Berechnung der konvexen Hülle sind die Ebenen bekannt, die zum Kern beitragen. Die Bestimmung der Form des Kern lässt sich dann auf unterschiedliche Art und Weise durchführen. Eine Möglichkeit ist die direkte Ermittlung aus der Struktur der konvexen Hülle. In Abschnitt 3.3 wird ein alternativer iterativer Algorithmus vorgestellt, der nicht auf zusätzlichen strukturellen Informationen der konvexen Hülle angewiesen ist. Er kann somit, sofern die Facetten, die zum Kern beitragen, bekannt sind, auch unabhängig von der Dualität genutzt werden. Abschließend wird in Abschnitt 3.4 die Bestimmung der kürzesten Weges in den Kern erläutert.

3.1 Die Dualität von Halbraumschnitt und konvexer Hülle

Für den dreidimensionalen Raum lässt sich eine Dualität zwischen Ebenen und Punkten in einem dualen Raum definieren. Dabei werden der Normalenvektor und der Abstand der Ebene zum Ursprung auf einen dualen Punkt abgebildet. Das hier vorgestellte Prinzip der Dualität wurde [10] entnommen. Definition 3.1 führt die duale Abbildung ein, wie sie

im Folgenden verwendet wird:

Definition 3.1: Sei E eine Ebene, gegeben in der Hesseschen Normalform $ax + by + cz + 1 = 0$. Die dualen Abbildungen sind gegeben durch:

$$\begin{aligned} E = \{(x, y, z) | ax + by + cz + 1 = 0\} &\rightarrow p^* = (a, b, c) \\ E^* = \{(x, y, z) | ax + by + cz + 1 = 0\} &\leftarrow p = (a, b, c) \end{aligned}$$

Somit gilt $E^{**} = E$.

Aus dieser Definition ergeben sich mehrere Zusammenhänge, die in den folgenden Abschnitten von Bedeutung und in Abbildung 3.1 dargestellt sind:

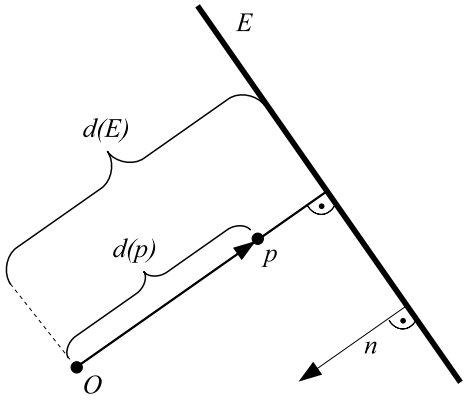

Abbildung 3.1: *Eine Ebene mit ihrem dualen Punkt.*

- Die Richtung des Normalenvektors n der Ebene E ist invers zu der Richtung des Ortsvektors des zu E dualen Punktes p.
- Eine Ebene wird häufig in der Form $ax + by + cz = d_E$ angegeben. Um eine solche Ebene in die Hessesche Normalform überzuführen, muss durch $-d_E$ dividiert werden. Daraus resultiert die folgende Gleichung $\frac{a}{-d_E}x + \frac{b}{-d_E}y + \frac{c}{-d_E}z + 1 = 0$. Dabei entspricht d_E dem Abstand der Ebene zum Ursprung in Einheiten eines Normalenvektors $n = (a, b, c)$. Es ergibt sich ein Zusammenhang zwischen dem Abstand der Ebene vom Ursprung und dem Abstand des Punktes vom Ursprung: $|d(E)| = \frac{1}{d_p}$. Hierdurch ergibt sich als Grenzfall eine Dualität zwischen Ebenen mit einem unendlichen Abstand zum Ursprung und dem Ursprung des dualen Raums.

Eine Anwendung der hier vorgestellten Dualität ist die Berechnung eines Halbraumschnittes mit Hilfe der konvexen Hülle. Für eine erfolgreiche Durchführung der Berechnung muss zusätzlich ein Punkt im Inneren des Halbraumschnittes bekannt sein, was im Folgenden näher erläutert wird. Durch die Transformation in den dualen Raum wird eine Ebene auf einen Punkt abgebildet. Allerdings wird ein Punkt nur über die drei Parameter der x, y und z Koordinaten definiert, während eine Ebene über vier Parameter definiert wird. Diese Parameter sind die x, y und z Koordinaten des Normalenvektors und der Abstand d zum Ursprung. Um diese vier Parameter auf die drei Parameter eines Punktes

abzubilden, wird durch d dividiert. Dadurch geht jedoch die Information der Richtung des Normalenvektors der Ebene verloren. Durch den Schnitt von Halbräumen ist eine Partitionierung gegeben, die den dreidimensionalen Raum in disjunkte Partitionen unterteilt. Die gesuchte Partition ist die, in die alle Normalenvektoren hineinzeigen. Die Information der Richtung des Normalenvektors ist jedoch durch die Transformation in den dualen Raum verloren gegangen. Um die richtige Partition zu berechnen, muss ein Punkt im Voraus in ihrem Inneren bekannt sein. In der vorliegenden Arbeit wurde der Ursprung gewählt. Durch eine Verschiebung des Terrain wird in der vorliegenden Arbeit sichergestellt, dass der Ursprung in dem Halbraumschnitt liegt und als innerer Punkt diesen markiert. Durch die in Abschnitt 2.2 beschriebenen Beschränkungen des Terrains ist gewährleistet, dass der Kern immer erreicht wird, wenn ein Strahl von einem beliebigen Startpunkt senkrecht nach oben geschossen wird. Um die minimale notwendige Verschiebung zu bestimmen, wird der Strahl so gewählt, dass er im Ursprung beginnt und entlang der z-Achse verläuft. Es wird nun jede Ebene mit diesem Strahl geschnitten und die maximale z-Koordinate der Schnittpunkte bestimmt. Es ist anschaulich klar, dass die Szene um zumindest diesen Wert nach unten verschoben werden muss. Dadurch wird der Abstand einer jeden Ebene zum Ursprung positiv. Im Anschluss an die Verschiebung werden nun die dualen Punkte der Ebenen bestimmt. Alle dualen Punkte haben eine negative Z-Koordinate, weil die Z-Koordinate der Normalenvektoren, welche immer positiv ist, durch den negativen Wert des Abstands zum Ursprung dividiert wird.

Wie in Beobachtung 2.2 beschrieben wurde, ist der Halbraumschnitt eines dreidimensionalen Terrains immer offen und eine Begrenzung gegenüber dem Unendlichen ist notwendig. Zu diesem Zweck wird ausgenutzt, dass der Ursprung des dualen Raums dual zu allen unendlich entfernt liegenden Ebenen ist. Aus diesem Grund wird der Ursprung in die Punktemenge zur Berechnung der konvexen Hülle hinzugefügt. Dieser liegt in jedem Fall auf der konvexen Hülle, weil alle dualen Punkte eine negative z-Koordinate haben. Wenn also der Halbraumschnitt von n Ebenen berechnet werden soll, so müssen die n dualen Punkte dieser Ebenen bestimmt werden. Zusammen mit dem Ursprung ergibt sich eine Punktemenge der Mächtigkeit $n+1$, deren konvexe Hülle ermittelt wird. Die Ebenen, deren duale Punkte auf der konvexen Hülle liegen, tragen zu dem Halbraumschnitt bei. Alle übrigen Ebenen begrenzen den Halbraumschnitt nicht.

Aus der Dualität ergeben sich weitere interessante Beziehungen zwischen dem Graphen des Halbraumschnitt und dem dazu dualen Graphen der konvexen Hülle. Aus der Definition der Dualität folgt, dass jeder Knoten auf der konvexen Hülle einer Ebene entspricht, die den Halbraumschnitt begrenzt. Aus der Dualität der beiden Graphen ergibt sich, dass ein Dreieck D im Graphen der konvexen Hülle dual zu einem Knoten p im Graphen des Halbraumschnitts ist. Dieser Knoten p ist der Schnittpunkt der drei Ebenen, die dual zu den Eckpunkten des Dreiecks D sind. Weiterhin ist eine Kante s im Graphen der konvexen Hülle dual zu einer Kante s' im Graphen des Halbraumschnitts. Diese Kante s' ist eine Teilmenge der Schnittgeraden der beiden zu den Endpunkten von s dualen Ebenen. Diese Beziehungen werden in Tabelle 3.1 zusammengefasst.

Abbildung 3.2 illustriert die Möglichkeit, die Struktur des Halbraumschnitts über die Dualität aus der Struktur der konvexen Hülle abzuleiten. Dies wird am Beispiel der Facette $F(p)$ und des Punktes $q(D)$ gezeigt. Der Punkt $q(D)$ auf dem Halbraumschnitt ist der Schnittpunkt der zu den Punkten p, p_1 und p_2 dualen Ebenen. Analog zur Berechnung von $q(D)$ können auch die anderen Eckpunkte berechnet werden, die die Facette $F(p)$ des Halbraumschnitts definieren. Wie in Tabelle 3.1 aufgeführt, sind die Dreiecke der kon-

Konvexe Hülle	Halbraumschnitt
Knoten $p = (a, b, c)$	Ebene $ax + by + cz + 1 = 0$
Kante	Schnittgerade zweier benachbarter Ebenen
Dreieck D	Schnittpunkt $q(D)$ von drei oder mehr Ebenen
Dreiecke mit Eckpunkt $(0, 0, 0)$	virtueller Schnittpunkt im Unendlichen

Tabelle 3.1: *Folgerungen aus der Transformation in den dualen Raum*

vexen Hülle mit dem Eckpunkt $(0, 0, 0)$ dual zu den offenen Enden der Halbgeraden der offenen Facetten des Halbraumschnitts. Der Punkt p in der Abbildung ist ein Endpunkt von zwei solchen Dreiecken. Die zu (p, p_3) benachbarten Segmente (p, p_2) und (p, p_4) der konvexen Hülle sind somit dual zu den beiden Halbgeraden von $F(p)$, da diese Segmente zu Dreiecken gehören, die den Nullpunkt p_3 als Eckpunkt haben, selbst jedoch nicht im Nullpunkt enden. Die Halbgerade $g(p, p_2)$ lässt sich als Schnittgerade der beiden durch die Facetten $F(p)$ und $F(p_2)$ gegebenen Ebenen berechnen. In der Darstellung der konvexen Hülle wurden die Segmente, die im Nullpunkt enden, gestrichelt eingezeichnet. Diese sind zwar Teil der konvexen Hülle, haben jedoch kein Gegenstück im Halbraumschnitt. Liegen zwei benachbarte Dreiecke der konvexen Hülle in derselben Ebene, so ist deren gemeinsames Segment nicht notwendig. Die Dreiecke entsprechen somit demselben Punkt des Halbraumschnittes. In diesem Punkt schneiden sich dann mehr als drei Ebenen.

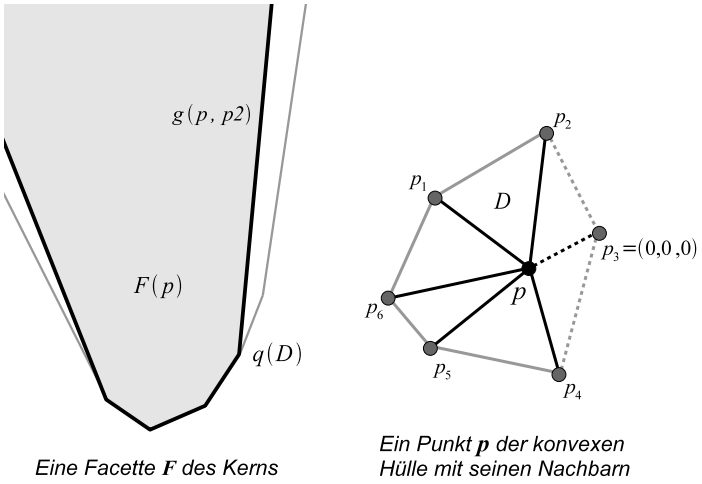

Abbildung 3.2: *Eine Facette des Kerns ist dual zu einem Punkt der konvexen Hülle.*

Mit den hier vorgestellten Eigenschaften der Dualität lässt sich aus der Struktur der konvexen Hülle die Struktur des Halbraumschnittes und damit des Kerns berechnen. Dies kann für einen Algorithmus zur Bestimmung des Kerns eines Terrains ausgenutzt werden.

3.2 Die konvexe Hülle einer dreidimensionalen Punktemenge

An dieser Stelle wird ein Algorithmus zur Bestimmung der konvexen Hülle einer dreidimensionalen Punktemenge beschrieben, wie er in [11] und in [2] vorgestellt wurde. Die konvexe Hülle ist wie folgt definiert:

> **Definition 3.2:** Die konvexe Hülle $CH(M)$ einer Punktemenge M im dreidimensionalen Raum ist die kleinste konvexe Menge, die alle Punkte aus M enthält.

Dies bedeutet anschaulich, dass genau die Punkte aus M auf dem Rand von $CH(M)$ liegen, welche das Papier berühren würden, wenn man die Punktemenge M "in Papier einwickeln" würde.

Im Folgenden werden die einzelnen Schritte des Algorithmus erläutert. Diese lassen sich in eine Initialisierung, bestehend aus mehreren Teilschritten, und darauf folgende Iterationsschritte unterteilen, in denen die einzelnen Punkte nacheinander in die konvexe Hülle des vorherigen Iterationsschrittes hinzugefügt werden.

Initialisierung der konvexen Hülle

Die Initialisierung beginnt mit der Konstruktion eines unregelmäßigen Tetraeders CH_{Tet}, bestehend aus vier nicht koplanaren Punkten. Bei der Konstruktion wird Wert darauf gelegt, dass die vier Punkte des Tetraeders Teil der resultierenden konvexen Hülle $CH(M)$ sind. Hierzu werden zuerst die zwei Punkte x_{min} und x_{max} aus M gewählt, die die kleinste bzw. größte X-Koordinate besitzen. Diese Punkte liegen auf jeden Fall auf der konvexen Hülle. Die Auswahl dieser Punkte erfolgt anhand von lexikographischen Vergleichen. Diese Art des Vergleichs wird im Folgenden immer angewendet, wenn Punkte miteinander verglichen werden. Durch die Punkte x_{min} und x_{max} wird die Gerade g gelegt. Mit Hilfe von g wird der dritte Punkt g_{max} des Tetraeders bestimmt. Dieser hat den maximalen Abstand aller Punkte aus M zu g. Durch die drei bisher bestimmten Punkte wird die Ebene E definiert, die der Bestimmung des vierten Punktes e_{max} dient. Dieser ist der Punkt aus M der den maximalen Abstand zu E hat. Somit wurden alle vier Punkte des Tetraeders bestimmt.

Sonderfälle, wie z. B. dass alle Punkte aus M in einer Ebene, auf einer Geraden oder in einem einzigen Punkt liegen, müssen in dieser Diplomarbeit nicht gesondert betrachtet werden, da sie in diesem Zusammenhang nicht auftreten können.

Im darauf folgenden Schritt werden zunächst die Eckpunkte von CH_{Tet} aus M entfernt. CH_{Tet} ist die initiale konvexe Hülle, der nach und nach die übrigen Punkte hinzugefügt werden. Die konvexe Hülle wird mit Hilfe einer $DCEL$ Datenstruktur verwaltet, wie sie in [1] und in [12] beschrieben wird. Die Facetten dieser $DCEL$ sind dabei immer Dreiecke. Die Kanten der Dreiecke werden durch zwei Halbkanten repräsentiert. Die Information einer Kante wird auf diese beiden Halbkanten so aufgeteilt, dass die rechte Halbkante eine Referenz auf die auf der rechten Seite anliegende Facette speichert sowie eine Referenz auf die rechte Vorgängerkante und die rechte Nachfolgerkante. Analog speichert die linke Halbkante Referenzen auf die linke Facette, die linke Vorgängerkante und die rechte Nachfolgerkante. Die linke und die rechte Halbkante besitzen die gleichen Endpunkte, zeigen

jedoch in entgegengesetzte Richtungen. Dadurch besteht jedes Dreieck aus eigenen Halbkanten. Diese Halbkanten sind bei jedem Dreieck entgegen dem Uhrzeigersinn orientiert. Die Halbkanten verweisen dabei jeweils auf ihre Zwillingskante.

Für ein späteres effizientes Einfügen der Punkte wird ein so genannter Konfliktgraph aufgebaut. Bei diesem wird jedem Punkt aus M ein Dreieck von CH_{Tet} zugeordnet, das so genannte Konfliktdreieck. Dabei erhalten die Punkte auch Referenzen auf ihr Konfliktdreieck. Die so vorgenommene Zuordnung ermöglicht eine effiziente Identifizierung der Dreiecke mit denen ein Punkt in Konflikt steht. Dies wird bei der Beschreibung der Iterationsschritte näher erläutert. Die Zuordnung erfolgt anhand eines Punktes c im Innern von CH_{Tet}. Die hier benutzte Methode zur Bestimmung von c ist die Berechnung des arithmetischen Mittels der vier Eckpunkte von CH_{Tet}. Wie in Abbildung 3.3 dargestellt, wird nun ein Strahl von c aus durch jeden Punkt aus M geschossen. Das Dreieck, dass von diesem Strahl geschnitten wird, ist das dem Punkt zugehörige Konfliktdreieck. In der Abbildung werden die Punkte p_1 und p_2 auf diese Weise dem Dreieck T_1 zugeordnet. Punkt p_3 liegt im Inneren von CH_{Tet}. Der Strahl von c durch p_3 trifft zuerst p_3 und dann das Dreieck T_1. Der Punkt p_3 ist also schon in der konvexen Hülle enthalten, kann daher aus M gelöscht werden und braucht in den folgenden Iterationsschritten nicht mehr berücksichtigt zu werden. Für jeden Punkt werden die Dreiecke auf diese Weise nacheinander überprüft. Sobald das Dreieck gefunden wurde, dass für den Punkt zuständig ist oder wäre, kann die Suche für diesen Punkt abgebrochen werden.

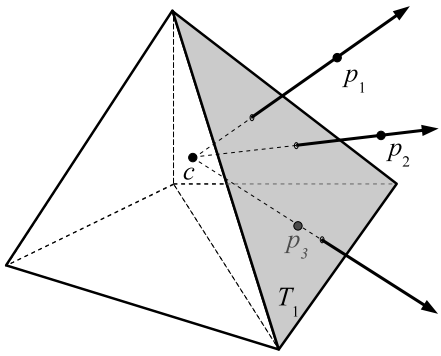

Abbildung 3.3: *Initialisierung des Tetraeders.*

Als letzter Initialisierungsschritt wird für die Punkte aus M eine Sortierung eingeführt, wodurch die Einfügereihenfolge der Iterationsschritte gegeben wird. Diese Einfügereihenfolge soll dafür sorgen, dass beim Einfügen der Punkte ein möglichst geringer Aufwand beim Aktualisieren der bisherigen konvexen Hülle anfällt. Zu diesem Zweck werden die Punkte anhand ihres Abstandes zu ihrem Konfliktdreieck sortiert. Bei den Iterationsschritten wird der Punkt mit dem maximalen Abstand zu seinem Konfliktdreieck jeweils als nächstes eingefügt. Da dieser Punkt zur konvexen Hülle gehört, beschränkt sich das Verfahren beim Einfügen auf die Punkte, die zur konvexen Hülle gehören. Alle anderen Punkte, die im Inneren der konvexen Hülle liegen, können vorher aus M entfernt werden. Ein Nachteil bei diesem Verfahren ist die Aktualisierung für die Einfügereihenfolge aufgrund der Änderungen im Konfliktgraphen. Dennoch wird durch die geschickte Auswahl der Einfügereihenfolge eine gute Laufzeit erwartet. Für eine auch in der Theorie günstige Laufzeit kann die Reihenfolge der Punkte aus der Menge M auch randomisiert werden.

Dies verhindert einen zusätzlichen Aufwand für die Aktualisierung der Einfügereihenfolge. Ein Nachteil dabei ist jedoch, dass auch Punkte, die nicht zum Endergebnis der konvexen Hülle gehören, zu einem Zwischenergebnis gehören, somit später wieder entfernt werden müssen und so unnötigen Aufwand verursachen. Mit dem randomisiertem Verfahren kann, wie in [2] gezeigt wurde, eine zu erwartende Laufzeit von $O(n \log n)$ erreicht werden.

Iterative Erweiterung der konvexen Hülle

In jedem Iterationsschritt wird die konvexe Hülle durch jeweils einen Punkt aus M erweitert. Dieser Punkt p_r wird anhand der Einfügereihenfolge aus der Initialisierung ausgewählt.

Zuerst werden die Dreiecke der konvexen Hülle bestimmt, die von p_r aus sichtbar ist, also die Bedingung aus Beobachtung 3.1 erfüllen.

> **Beobachtung 3.1:** Ein Dreieck ist von einem Punkt aus sichtbar, wenn der Punkt über der Ebene liegt, in der das Dreieck eingebettet ist.

Zur Bestimmung der sichtbaren Dreiecke wird mit dem Konfliktdreieck von p_r begonnen. Ausgehend von diesem Konfliktdreieck werden alle von p_r aus sichtbaren Dreiecke bestimmt.

Im Folgenden wird der Sichtbarkeitstest zur Bestimmung der sichtbaren Dreiecke ausführlicher beschrieben. Ziel der Überprüfung ist dabei, möglichst nur die Dreiecke zu überprüfen, die Teil des sichtbaren Bereiches sind und unnötige Tests mit nicht sichtbaren Dreiecken zu vermeiden. Weiterhin sollen Dreiecke in einem Iterationsschritt nur einmal überprüft werden. Zu diesem Zweck werden die einzelnen Iterationsschritte durchnummeriert. Bei den Dreiecken wird die Nummer des jeweiligen Iterationsschrittes gespeichert, in dem das Dreieck überprüft wurde. Anhand dieses Wertes kann somit bestimmt werden, ob das Dreieck in dem aktuellen Iterationsschritt bereits überprüft wurde. Der Sichtbarkeitstest wird, wie zuvor erwähnt, mit dem Konfliktdreieck T von p_r begonnen. Die Zwillinge aller Kanten von T werden auf einem Stapel abgelegt. Dieser Stapel enthält alle Kanten, an denen ein Dreieck angrenzt, welches noch zu überprüfen ist. Solange der Stapel nicht leer ist, wird das oberste Element s entnommen und das anliegende Dreieck D auf Sichtbarkeit von p_r aus überprüft. Wurde D in dem aktuellen Iterationsschritt bereits überprüft, so wird mit dem nächsten Element des Stapels fortgefahren. Für den Fall, dass D von p_r aus sichtbar ist, so werden die Zwillinge der Kanten von D mit Ausnahme des Zwillings von s auf den Stapel gelegt. Alle Dreiecke, die in dem aktuellen Iterationsschritt als sichtbar identifiziert werden, werden zu der Menge $D_{Sichtbar}$ hinzugefügt. Diese enthält auch das Konfliktdreieck T. Dieser Sichtbarkeitstest gewährleistet, dass alle sichtbaren Dreiecke gefunden werden. Abbildung 3.4 zeigt das Beispiel einer konvexen Hülle zu der ein Punkt hinzugefügt wird. Die sichtbaren Dreiecke sind weiß gekennzeichnet.

Abbildung 3.4 zeigt weiterhin die Projektion des Horizontes H von p_r auf einen Schirm. Der Horizont beschreibt den Übergang von dem sichtbaren zu dem nicht sichtbaren Bereich der konvexen Hülle. Der Horizont besteht aus genau den Kanten, hinter denen beim Sichtbarkeitstest ein nicht sichtbares Dreieck festgestellt wurde. Die Projektion des Horizontes ist ein geschlossenes konvexes Polygon.

Nach Abschluss des Sichtbarkeitstests wird die Aktualisierung der konvexen Hülle durchgeführt. Zu diesem Zweck werden alle sichtbaren Dreiecke $D_{Sichtbar}$ aus der bisherigen

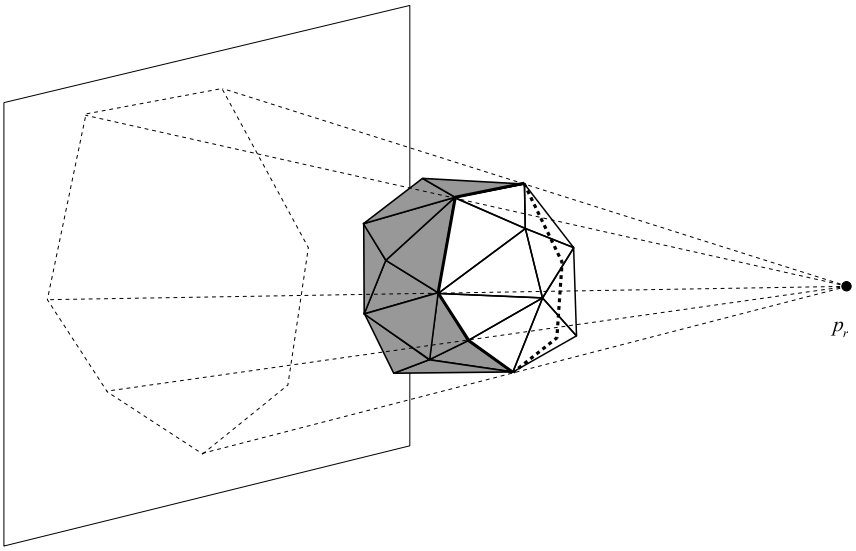

Abbildung 3.4: *Der Horizont auf einer konvexen Hülle zu Punkt p.*

konvexen Hülle entfernt. Die Punkte aus M, die den Dreiecken aus $D_{Sichtbar}$ zugeordnet waren, werden für eine Neuzuordnung zwischengespeichert.

Wie in Abbildung 3.5 dargestellt, werden die neuen Dreiecke der konvexen Hülle aus den einzelnen Horizontkanten erstellt. Dabei bildet jeweils eine Horizontkante mit dem Punkt p_r ein neues Dreieck.

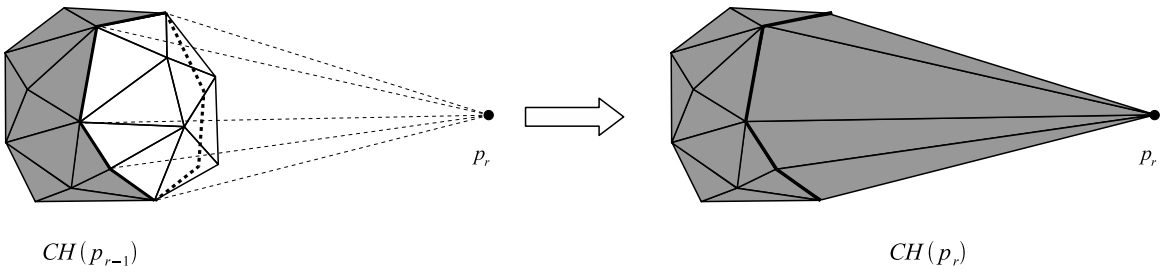

Abbildung 3.5: *Hinzufügen eines Punktes zur konvexen Hülle.*

Nachdem die neuen Dreiecke der konvexen Hülle hinzugefügt wurden, findet die Neuzuordnung der zuvor zwischengespeicherten Punkte aus M statt. Die Zuordnung erfolgt dabei analog zu dem entsprechenden Initialisierungsschritt, wobei nur die neuen Dreiecke berücksichtigt werden müssen. Punkte, die von p_r aus hinter einem neuen Dreieck liegen, werden nicht neu zugeordnet, da sie im Inneren der konvexen Hülle liegen. Sie werden deshalb aus M entfernt.

Mit der Neuzuordnung der Punkte ist ein Iterationsschritt abgeschlossen und es wird, sofern vorhanden, der nächste Punkt p_{r+1} aus M eingefügt.

3.3 Die Bestimmung des Kerns

Durch die in den vorherigen Abschnitten beschriebenen Verfahren können die Ebenen des Terrains bestimmt werden, die zum Kern beitragen. Die Struktur des Kerns kann anschließend aus diesen Ebenen berechnet werden. Dies kann einmal wie Abschnitt 3.1 beschrieben wurde, über die dort definierte Dualität anhand der Struktur der konvexen Hülle erfolgen. Ein selbstentwickelter, alternativer Algorithmus zur Bestimmung des Kerns soll in diesem Abschnitt vorgestellt werden. Dieser Algorithmus kann angewendet werden, wenn die Ebenen bekannt sind, die zum Halbraumschnitt beitragen und der Halbraumschnitt nach oben offen ist. Dabei werden Eigenschaften des Kerns ausgenutzt, die im Folgenden erläutert werden. Wie in Abschnitt 2.3 festgestellt wurde, ist der Kern ein konvexes Polyeder. Das bedeutet, dass die Ebenen vom tiefsten Punkt des Kerns ausgehend in positive Z-Richtung immer steiler werden. Zukünftig wird davon ausgegangen, dass die Normalenvektoren der Facetten des Kerns in das Kerninnere zeigen. Der Winkel dieser Normalenvektoren zur Z-Achse nimmt also vom tiefsten Punkt des Kerns in positive Z-Richtung zu. Dies wird in Abbildung 3.6 dargestellt. Aufgrund dieser Beobachtung wird der Kern von unten nach oben schrittweise aufgebaut. Die Ebenen werden, sortiert nach den Winkeln ihrer Normalenvektoren zur Z-Achse, nach und nach eingefügt. Es wird mit der Ebene begonnen, deren Normalenvektor den kleinsten Winkel zur Z-Achse hat.

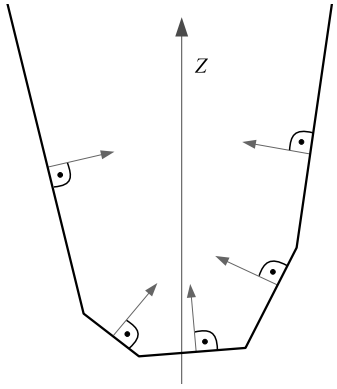

Abbildung 3.6: *Die Normalen auf den Facetten des Kerns.*

Die Initialisierung

In einem Initialisierungsschritt werden die Ebenen nach dem zuvor genannten Kriterium sortiert. Die ersten drei Ebenen werden in der Initialisierung der Kernstruktur verwendet. Die restlichen Ebenen werden in den später folgenden Iterationsschritten der Reihe nach hinzugenommen.

Der initiale Kern besteht somit, wie in Abbildung 3.7 dargestellt, aus den drei Facetten F_0, F_1 und F_2. Die Kernstruktur wird in einer $DCEL$ Graphenstruktur gespeichert, wie sie in [1] und in [12] beschrieben wird. Wie schon bei der konvexen Hülle wird auch hier jede Kante durch zwei Halbkanten repräsentiert. Im Folgenden werden als Halbgeraden nur die ausgehenden Halbkanten bezeichnet, das heißt die Halbkanten, die nur einen Anfangspunkt besitzen, jedoch keinen Endpunkt. In der Abbildung sind somit e_0, e_2 und

e_4 die Halbgeraden des initialen Kerns. Die Halbgeraden werden für die Iterationsschritte zusätzlich in einer speziellen Datenstruktur gespeichert. Diese Datenstruktur sortiert die Halbgeraden anhand des Winkels der Projektion ihres Richtungsvektors in der X/Y-Ebene. Dies ermöglicht eine effiziente Suche nach der Position im Graphen an der eine Ebene in den Iterationsschritten eingefügt werden muss.

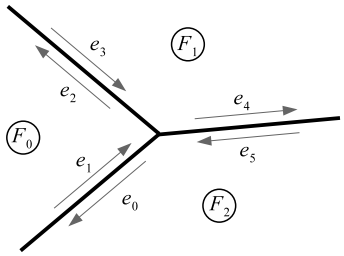

Abbildung 3.7: *Initialisierung des Kerns.*

Die Iterationsschritte

In jedem Iterationsschritt wird die Ebene E der verbleibenden Ebenen dem bisherigen Kern hinzugefügt, deren Normalenvektor den kleinsten Winkel zur Z-Achse hat. Zur Bestimmung der Position in der $DCEL$ Graphenstruktur an der E eingefügt werden soll, wird zunächst der Normalenvektor der Ebene in die X/Y-Ebene projiziert und invertiert. Das Resultat wird im Folgenden mit n_E bezeichnet. Im Anschluss wird die Position in der Halbgeradenmenge bestimmt, an der der Normalenvektor eingefügt werden würde, wenn er eine Halbgerade wäre. Die Vorgängerhalbgerade v und die Nachfolgerhalbgerade w von dieser Position werden für die spätere Überprüfung vorgemerkt. In der Abbildung 3.8 sind dies die Halbgeraden h und j.

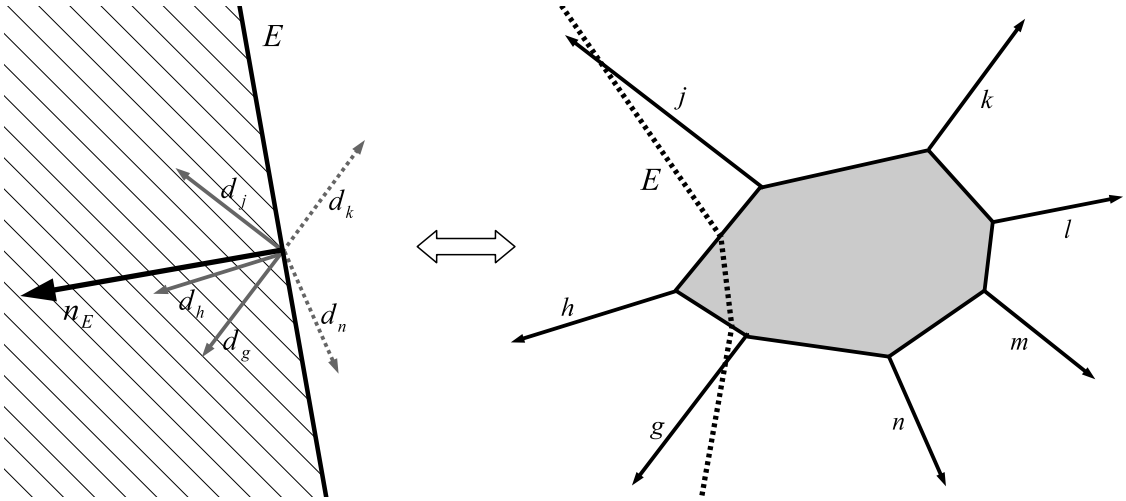

Abbildung 3.8: *Konflikttest.*

Nachdem die Position bestimmt wurde, an der E eingefügt werden muss, ist es notwendig, die Halbgeraden zu bestimmen, die mit E in Konflikt stehen, d. h. die von E geschnitten werden. Zu diesem Zweck wird mit v und allen Vorgängern von v ein Konflikttest durchgeführt, bis eine Halbgerade gefunden wurde, mit der kein Konflikt besteht. Dasselbe wird mit w und seinen Nachfolgern durchgeführt. Der Konflikttest wird in Abbildung 3.8 dargestellt. Die Richtungsvektoren d_g, d_h und d_j zeigen in den schraffierten Bereich von E, d. h. die zugehörigen Halbgeraden g, h und j werden E schneiden. Hierbei ist zu berücksichtigen, dass E in der Darstellung in den Ursprung verschoben wurde, jedoch seine Neigung zur Z-Achse beibehalten hat. Die Richtungsvektoren sind in diesem Zusammenhang auch als nicht eingebettet in die X/Y-Ebene anzusehen. Für den Konflikttest werden die Richtungsvektoren als Ortsvektoren interpretiert. Liegt der Endpunkt eines Ortsvektors über oder in E, so besteht ein Konflikt. Die Richtungsvektoren die einen Konflikt mit E haben, sind in der Abbildung durchgängig gezeichnet. Beim Test von d_k und d_n wird der Konflikttest jeweils beendet, weshalb diese Richtungsvektoren gestrichelt gezeichnet wurden. Die Richtungsvektoren zu den Halbgeraden l und m wurden nicht eingezeichnet, da diese nicht auf einen Konflikt getestet werden.

Die Halbgeraden mit einem Konflikt mit E werden nacheinander abgearbeitet. Bei dieser Abarbeitung kann für jede Halbgerade einer der drei Fälle auftreten:

- **Eckschnitt**: E schneidet durch den Anfangspunkt der Halbgeraden
- **Einfacher Schnitt**: E schneidet durch die Halbgerade, jedoch nicht im Anfangspunkt
- **Tiefer Schnitt**: E schneidet die Verlängerung der Halbgeraden, hinter ihrem Anfangspunkt

In Abbildung 3.8 wurde ein Konflikt mit den Halbgeraden g, h und j festgestellt. Es werden nicht zuerst alle Halbgeraden gesucht, die einen Konflikt verursachen. Sobald eine Halbgerade gefunden wurde, die einen Konflikt mit E hat, wird dieser behandelt. Bei der Behandlung von g entstehen zwei neue Halbgeraden, die linke davon wird bei der Behandlung von h mit der rechten der beiden neuen Halbgeraden von h zu einer abgeschlossenen Kante verbunden. Beim Einfügen einer Ebene können maximal zwei neue Halbgeraden entstehen, wobei mindestens eine Halbgerade entfernt wird. Daraus folgt, dass die Komplexität der Menge der Halbgeraden in $O(n)$ ist. Wie die drei Fälle, die bei dem Konflikttest einer jeden Halbgeraden auftreten können, im Einzelnen behandelt werden, wird im Folgenden näher erläutert.

Fall 1: Eckschnitt

In diesem Fall wird der Startpunkt einer Halbgeraden durch die Ebene E geschnitten. Dies wird in Abbildung 3.9 dargestellt, welche die Projektion des Kerns in die X/Y-Ebene zeigt und zwar vor und nach dem Einfügen der Ebene E. Der Kern besteht in der Ausgangssituation aus einer geschlossenen Facette F_3 und fünf offenen Facetten, wozu F_1 und F_2 gehören. Die Ebene E schneidet die Facetten F_1 und F_2, trennt dabei die Halbgerade g ab und schneidet den Punkt C. Die Halbgerade g wird daher sowohl aus der Menge der Halbgeraden, als auch aus der $DCEL$ Datenstruktur entfernt. Es entstehen die Halbgeraden r und s in dem Punkt C, die zu der Menge der Halbgeraden und zu

der $DCEL$ Datenstruktur hinzugefügt werden. Weiterhin wird die Facette F_E der $DCEL$ hinzugefügt. In diesem Fall schneiden sich in C vier, d. h. mehr als drei Ebenen. Der Punkt C erhält dadurch mehr als drei ausgehende Kanten.

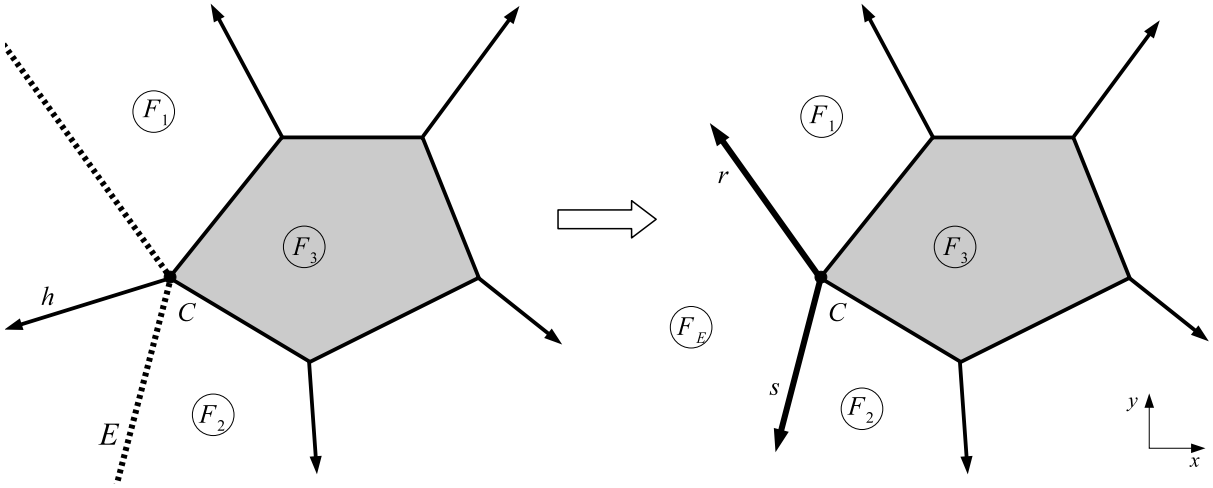

Abbildung 3.9: *Fall 1: Die neue Ebene schneidet durch einen Eckpunkt.*

Fall 2: Einfacher Schnitt

Ein einfacher Schnitt ist vergleichbar mit einem Eckschnitt. Auch hier wird eine Halbgerade h des bisherigen Kerns geschnitten. Nur wird in diesem Fall die Halbgerade nicht abgeschnitten, sondern zu einer Kante h' verkürzt. Dieser Fall ist in Abbildung 3.10 veranschaulicht. Die Halbgerade h wird hier aus der Menge der Halbgeraden entfernt, ihr wird der Punkt C als Endpunkt hinzugefügt und sie bleibt in der $DCEL$ Datenstruktur enthalten. Weiterhin entstehen die Halbgeraden r und s, die beide vom Startpunkt C ausgehen. Die Halbgeraden r und s werden in die Menge der Halbgeraden und zusammen mit der neuen Facette F_E und dem neuen Schnittpunkt C in die $DCEL$ Datenstruktur eingefügt.

Fall 3: Tiefer Schnitt

Der Fall des tiefen Schnittes ist komplexer als die ersten beiden Fälle. Wie in Abbildung 3.11 gezeigt wird, schneidet die Ebene E die Verlängerung der Halbgeraden h hinter ihrem Startpunkt. Hierdurch verschwindet diese Halbgerade komplett aus dem bisherigen Kern und die Kanten i und j werden verkürzt. Weiterhin entstehen die Schnittpunkte C_1 und C_2, die Kante t, sowie die Halbgeraden r und s.

Eine nahe liegende Möglichkeit wäre, diesen Fall direkt abzuarbeiten und die Menge der Halbgeraden bzw. die $DCEL$ Datenstruktur entsprechend anzupassen. Abbildung 3.12 zeigt jedoch, dass in einem solchen Fall der Schnitt noch tiefer in den bisherigen Kern hinein erfolgen kann. Für die Art des Schnittes besteht die Einschränkung, dass keine Facette des bisherigen Kerns herausgeschnitten werden darf. Dies würde der Voraussetzung für den Algorithmus widersprechen, dass jede Ebene zu dem Kern beiträgt.

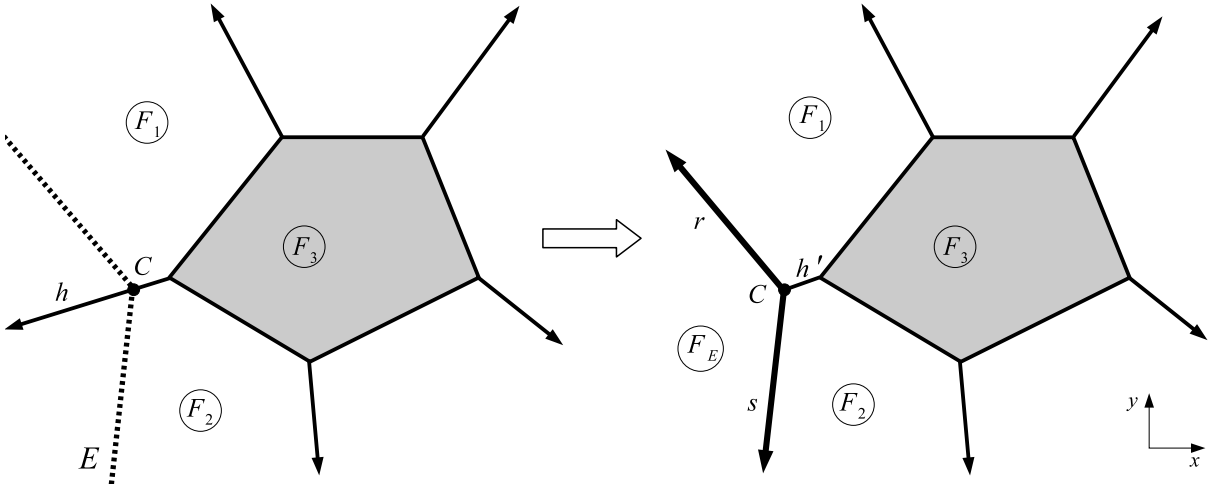

Abbildung 3.10: *Fall 2: Eine Halbgerade wird abgeschnitten.*

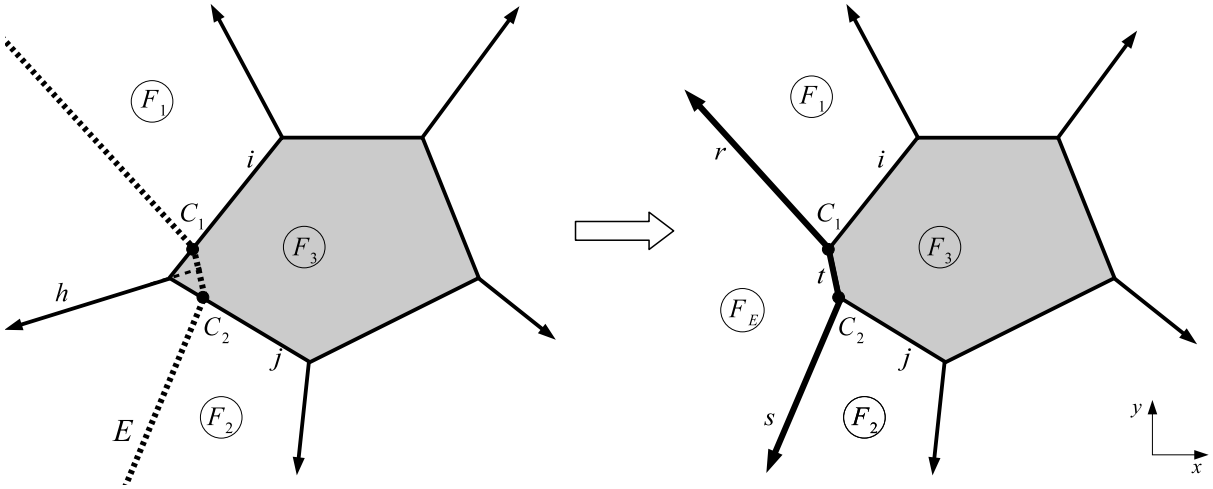

Abbildung 3.11: *Fall 3: Eine Ecke wird abgeschnitten.*

Aufgrund solcher tiefen Schnitte wurde eine rekursive Lösung implementiert. Bei dieser Lösung wird die aktuelle Halbgerade h aus der Menge der Halbgeraden und der $DCEL$ Datenstruktur zusammen mit ihrem Startpunkt gelöscht. Auf diese Art und Weise entstehen zwischenzeitlich zwei Halbgeraden i und j, bei denen auch ein Konflikttest durchgeführt wird. Im Fall eines tiefen Schnittes wie in Abbildung 3.12 kann dies dazu führen, dass auch eine weitere Kante – hier i – herausgeschnitten wird. Da i zwischenzeitlich als eine Halbgerade angesehen wurde und der Schnittpunkt hinter der Verlängerung der Halbgeraden lag, wird auch auf sie wiederum der Fall des tiefen Schnittes angewandt. Die Rekursion endet jeweils bei den Kanten g, j und k, da bei ihnen der Fall des einfachen Schnittes angewendet wird. Zu Berücksichtigen bei dieser rekursiven Lösung ist jedoch, dass bei den jeweiligen Rekursionsschritten nicht immer zwei Halbgeraden entstehen, sondern die Halbgerade des jeweiligen benachbarten Konfliktpunktes C_i abgeschlossen wird und damit auch nicht zu der Menge der Halbgeraden hinzugefügt wird. Nach Abschluss der

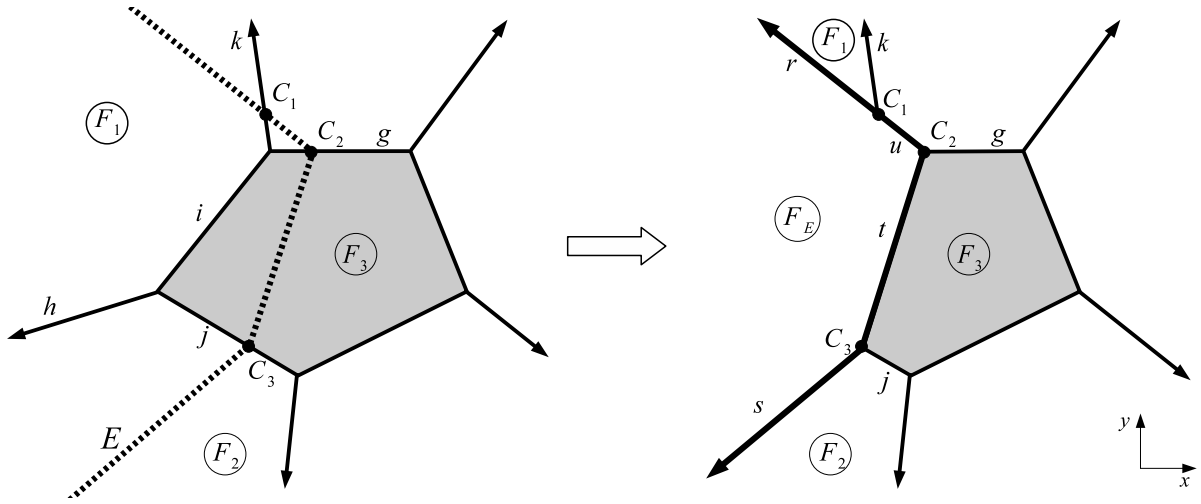

Abbildung 3.12: *Die Ebene E schneidet tief in den Kern.*

Rekursion wurde somit die Halbgerade h aus der Menge der Halbgeraden entfernt und die neuen Halbgeraden r und s wurden hinzugefügt. Die $DCEL$ Datenstruktur wurde durch die einzelnen Rekursionsschritte entsprechend aktualisiert.

Laufzeitanalyse

Die Abschätzung der Laufzeit kann in zwei Schritten durchgeführt werden. Zunächst wird die Laufzeit der Initialisierung bestimmt und dann die der Iterationsschritte. Daraus ergibt sich die Gesamtlaufzeit des Algorithmus. Bei der Initialisierung besteht der wesentliche Aufwand in der Sortierung der Ebenen anhand der Winkel ihrer Normalenvektoren. Somit ist die Laufzeit der Initialisierung $O(n \log n)$, wobei n sich auf die Anzahl der Ebenen bezieht, von denen der Kern bestimmt werden soll. Für die Laufzeit der Iteration wird zunächst der Aufwand für jeden Iterationsschritt abgeschätzt. Dieser wird dann mit $(n-3)$ multipliziert, da noch $(n-3)$ Ebenen nach der Initialisierung eingefügt werden müssen. Bei jedem Iterationsschritt werden zunächst die Konflikthalbgeraden bestimmt. Die Anzahl der Halbgeraden liegt in $O(n)$, wie zuvor gezeigt wurde. Daher ist die Anzahl der Konflikttests auch in $O(n)$. Aus der Eulerschen Formel lässt sich folgern, dass die Anzahl der Kanten in dem Graphen in $O(n)$ liegt. Für eine Ebene können somit maximal $O(n)$ Konflikttests mit Konfliktbehandlung durchgeführt werden. Somit ist der Aufwand für diesen Anteil der Iterationsschritte auch in $O(n)$. Damit ergibt sich eine Gesamtlaufzeit von $O(n^2)$. Aufgrund der Folgerung aus der Eulerschen Formel ([1]), dass eine Facette in einem solchen Graphen im Mittel nur 6 Kanten hat, ist es zu erwarten, dass die tatsächliche Laufzeit des Algorithmus besser ist, als es die zuvor angegebene Komplexität impliziert.

In [13] wird ein weiterer randomisierter, iterativer Algorithmus vorgestellt, der den Aufbau eines Halbraumschnittes in einer zu erwartenden Laufzeit von $O(n \log n)$ erlaubt. Für diesen Algorithmus ist die Verwendung der Dualität nicht notwendig.

3.4 Der Weg in den Kern

Die Bestimmung des kürzesten Weges in den Kern kann anhand der Kernfacetten durchgeführt werden. Der kürzeste Weg in den Kern von einer Roboterposition R aus ist entweder ein Lot auf eine Facette des Kerns, ein Lot auf eine Kante des Kerns oder der Weg zu einem Eckpunkt des Kerns.

Eine einfache Möglichkeit die Wegstrecke zu bestimmen, ist, den Punkt des Kerns zu bestimmen, der R am nächsten ist. Hierzu kann zunächst überprüft werden, ob sich R im Inneren oder auf dem Rand des Kerns befindet. Ist dies der Fall, so ist der Roboter bereits im Kern und es muss keine Wegstrecke berechnet werden. Trifft dies nicht zu, so wird zu jeder Facette F_i der Punkt q_i bestimmt, der R am nächsten ist. Sollte die Strecke von R zu q_i ein Lot auf die Facette F_i sein, so ist der kürzeste Weg gefunden. Wenn zu jeder Facette F_i der Punkt q_i bestimmt wurde, der R am nächsten liegt und dabei kein Lot auf eine Facette gefunden wurde, so bildet der kleinste Abstand $d(q_i, R)$ den kürzesten Weg in den Kern. Der kürzeste Weg mit dem kleinsten Abstand $d(q_i, R)$ ist in diesem Fall ein Lot auf eine Kante einer Facette des Kerns oder der Weg zu einem Eckpunkt einer Facette des Kerns. Diese Berechnung kann in $O(n)$ durchgeführt werden, wobei n die Anzahl der Facetten des Kerns ist.

Kapitel 4

Online Strategien für die Kernsuche

Online Strategien für die Kernsuche in einem dreidimensionalen Terrain müssen mit unvollständigen Informationen über das Terrain arbeiten. Daher ist eine Bestimmung des optimalen Weges nicht immer möglich. Im Vergleich zu Offline Algorithmen, die den optimalen Weg liefern, ist eine längere Wegstrecke zum Kern zu erwarten. Lässt sich ein linearer Zusammenhang zu der optimalen Weglänge erreichen, so wird, wie in Abschnitt 2.4 definiert, von einer konstant-kompetitiven Online Strategie gesprochen.

Die Entscheidung, welcher Weg in den Kern gewählt wird, kann aufgrund des aktuellen Sichtbereiches des Roboters getroffen werden. Dabei ist zu berücksichtigen, dass sich dieser Sichtbereich mit der Bewegung des Roboters ändert. Für die Strategie ist somit von Bedeutung, dass der Roboter aufgrund der veränderten Sichtbarkeit eine zuvor getroffene Entscheidung für die Bewegung revidiert und auf die neu verfügbaren Informationen anpasst. Hierbei ist auch von Bedeutung, inwiefern der Roboter in der Lage ist, die veränderte Sichtbarkeit zum richtigen Zeitpunkt zu erkennen, was z. B. in einer realen Umgebung aufgrund der Einschränkung der Sensorik gegeben ist. Auch für die Verwaltung der Datenstrukturen ist die veränderte Sichtbarkeit nicht trivial.

In Abschnitt 4.1 wird zunächst eine untere Schranke für die zu erwartende Weglänge einer Online Strategie bestimmt. In Abschnitt 4.2 wird dann der Sichtbarkeitsbereich eines Roboters beschrieben und der Begriff des Horizontes eingeführt. Basierend auf diesen Begriffen werden in dem darauf folgenden Abschnitt 4.3 verschiedene einfache Online Strategien vorgestellt, deren Korrektheit gezeigt und auf ihre Kompetitivität eingegangen. In Abschnitt 4.4 werden weitere Ideen vorgestellt, die bei dem Entwurf komplexerer Online Strategien hilfreich sein können. Darüber hinaus wird eine komplexe Strategie vorgestellt, die einige dieser Ideen berücksichtigt. Anschließend wird die Bewegung des Roboters auf Kreisbögen als Alternative zu geradlinigen Bewegungen diskutiert. Das Kapitel wird abgeschlossen mit einer Übertragung der *CAB* Strategie aus Abschnitt 2.5 auf das dreidimensionale Terrain und einer allgemeinen Diskussion über die Kompetitivität von Online Strategien zur Kernsuche im dreidimensionalen Terrain.

4.1 Eine untere Schranke

In diesem Abschnitt wird eine untere Schranke für den kompetitiven Faktor von Online Strategien zur Kernsuche in einem dreidimensionalen Terrain bestimmt. Der Beweis dieser Schranke erfolgt analog zu dem Beweis für die Kernsuche in einem einfachen Polygon, welcher in Abschnitt 2.5.1 referenziert wurde.

Abbildung 4.1 zeigt ein Terrain mit einem Roboter an der Position R. Der Roboter kann von seiner Startposition R aus den Kern $ker(T)$ des Terrains T sehen. Er weiß jedoch nicht, wo sich dieser befindet. Durch eine Spiegelung des Terrains an der Z-Achse entsteht ein anderes Terrain, in welchem der Roboter von R aus ein identisches Sichtbarkeitspolyeder hat, der Kern sich jedoch an einer anderen Stelle befindet. Die Grundidee des Beweises basiert darauf, dass daher kein Wert für die X- bzw. Y-Komponente der Bewegungsrichtung bevorzugt werden kann und somit immer ein Umweg notwendig ist.

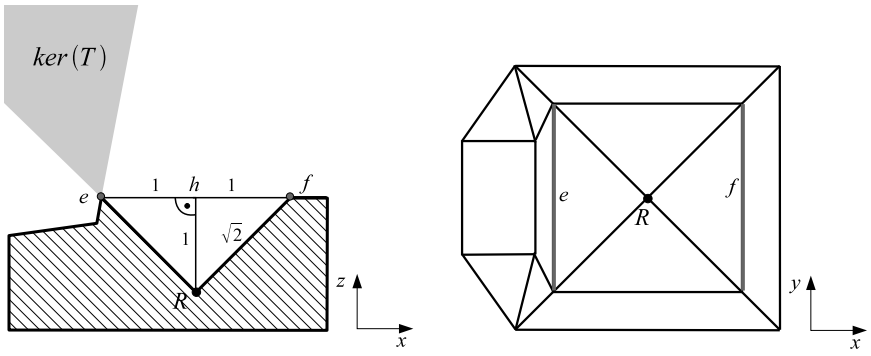

Abbildung 4.1: *Terrain zur Bestimmung einer unteren Schranke.*

Theorem 4.1: Sofern überhaupt eine kompetitive Online Strategie existiert, mit der ein Weg in den Kern gefunden werden kann, so beträgt ihr kompetitiver Faktor mindestens $\sqrt{2}$.

Beweis: Sei S eine kompetitive Strategie, die den Roboter zu dem Punkt des Kerns führt, der seiner Startposition am nächsten ist. Diese Strategie wird auf das Beispiel in Abbildung 4.1 angewendet. Der Roboter kann von seinem Startpunkt aus nicht das ganze Terrain einsehen und weiß nicht, wo sich der Kern befindet. Das Terrain ist so aufgebaut, dass das Sichtbarkeitspolyeder des Roboters bis zur Höhe h ein Tetraeder ist, dessen Grundfläche nach oben gerichtet ist. Die Kantenlänge der Grundfläche beträgt genau zwei, während die Kanten von der Grundfläche zur Spitze die Länge $\sqrt{2}$ haben. Von seiner Startposition aus sieht der Roboter nach oben vier Kanten, die in einem Quadrat der Seitenlänge zwei angeordnet sind. Hinter jeder dieser Kanten befindet sich mindestens eine Ebene, die noch nicht sichtbar ist. Der Roboter muss dabei, unabhängig von seiner Bewegung relativ zur X/Y-Ebene, zunächst bis auf eine Höhe h ansteigen, um drei der vier unbekannten Ebenen zu sehen. Danach muss er noch zu der Kante e fliegen, hinter welche er noch nicht sehen kann. Der in [3] beschriebene Gegenspieler kann, solange der Roboter die Höhe h noch nicht erreicht hat, sich immer noch dafür entscheiden, dass Terrain an der Z-Achse zu spiegeln. Er wählt das Terrain so, dass e auf der linken Seite des Terrains liegt, wenn der Roboter auf die rechte Kante f, die e gegenüberliegt, zusteuert. Wenn der Roboter jedoch auf die linke Seite zusteuert, wird das Terrain so gespiegelt, dass e auf der rechten Seite liegt.

Aufgrund der Tatsache, dass der Gegenspieler das Terrain zu jedem Zeitpunkt spiegeln kann, kann erst festgestellt werden, auf welcher Seite sich der Kern befindet, sobald der Roboter die Höhe h erreicht hat. Hat er die Höhe h erreicht, so wird er feststellen, dass er mindestens noch eine Wegstrecke von eins zurücklegen muss, um den Kern zu erreichen.

Der kürzeste Weg den eine Online Strategie somit wählen kann, ist $h+1$. Damit kann die tatsächliche Weglänge des Roboters $w(R, e)$ wie folgt mit der optimalen Weglänge $d(R, e)$ verglichen werden:

$$\frac{w(R,e)}{d(R,e)} \geq \frac{h+1}{\sqrt{2}} = \sqrt{2}$$

□

In den folgenden Kapiteln wird zunächst der Sichtbarkeitsbereich des Roboter analysiert, bevor einige einfache Strategien zur Kernsuche vorgestellt werden.

4.2 Die Sichtbarkeit des Roboters im Terrain

In diesem Abschnitt werden Begriffe bezüglich des Sichtbarkeitsbereichs eines Roboters R im dreidimensionalen Terrain erklärt. Ein Terrain, in dem der Roboter nur eine eingeschränkte Sicht hat, lässt sich in zwei Bereiche unterteilen: einen sichtbaren und einen nicht sichtbaren Bereich. Der sichtbare Bereich hat, wie bereits in Abschnitt 2.6 beschrieben, die Form eines Polyeders und wird Sichtbarkeitspolyeder genannt. Von besonderem Interesse sind auch die Teile des Terrains, die für den Übergang zwischen sichtbarem und nicht sichtbarem Bereich verantwortlich sind. Dieser Übergang wird als Horizont bezeichnet. In Abschnitt 4.2.1 wird zunächst beschrieben und definiert, was ein Horizont ist. Abschnitt 4.2.2 geht dann auf Veränderungen des Horizontes bei der Bewegung des Roboters ein und diskutiert Probleme, die daraus entstehen. Anschließend wird in Abschnitt 4.2.3 genauer auf den Sichtbarkeitspolyeder eingegangen und es wird analog zu dem Gewinnbereich der *CAB* Strategie für Polygone ein Sichtbarkeitskegel definiert. Danach wird in Abschnitt 4.2.4 ein Algorithmus zur Bestimmung des Horizonts vorgestellt.

4.2.1 Der Horizont

Ein Horizont definiert, ausgehend von einer Roboterposition R, den Übergang von einem sichtbaren zu einem nicht sichtbaren Bereich des Terrains. Der Horizont ist ein charakteristischer Bestandteil des Sichtbarkeitspolyeders und kann daher als Grundlage für die Konstruktion eines solchen verwendet werden. Aus diesem Grund ist der Horizont auch für die Bewegungsentscheidungen in den Online Strategien von Interesse, wie in späteren Abschnitten genauer erläutert wird. In diesem Abschnitt wird der Begriff des Horizonts eingeführt. Der Horizont eines dreidimensionalen Terrains besteht aus Kanten bzw. Teilkanten der Oberfläche des Terrains. Als notwendiges Kriterium ergibt sich für die angrenzenden Facetten einer Horizontkante, dass der Roboter oberhalb der näher liegenden und unterhalb der weiter entfernten Facette liegt. Eine solche Kante e, wie in Abbildung 4.2 dargestellt, wird von nun an als *blockierende Kante* bezeichnet. Der Roboter R liegt oberhalb der näher liegenden Facette F_1 und unterhalb der weiter entfernten Facette F_2.

Basierend auf den Definitionen 2.5 und 2.7 lässt sich der Begriff der blockierenden

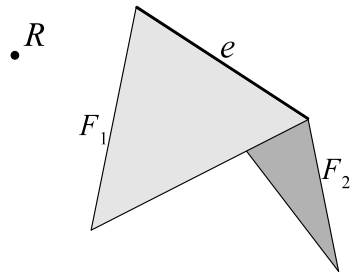

Abbildung 4.2: *Kante e ist eine blockierende Kante.*

Kante wie folgt definieren:

> **Definition 4.1:** Sei T ein dreidimensionales Terrain, R die Position eines Roboters in $\overline{L(T)}$ und e eine Kante aus $O(T)$. Ferner seien F_1 und F_2 die an e angrenzende Facetten des Terrains, wobei F_1 die näher liegende Facette ist. Die Kante e heißt blockierend, wenn F_1 sichtbar ist und F_2 nicht sichtbar ist.

Für eine vereinfachende Darstellung des Horizonts wird, wie in Abbildung 4.3 dargestellt, ein zweidimensionaler Schnitt $M_{R,\rho}$ durch das dreidimensionale Terrain verwendet. Die dabei eingeführten Begriffe sind auch für das Terrain allgemein gültig. Daher wird im Folgenden der Horizont zunächst für einen solchen Schnitt definiert und anschließend wird diese Definition auf das dreidimensionale Terrain erweitert. Ein Terrainschnitt $M_{R,\rho}$ ist durch die Blickrichtung ρ des Roboters an einer Position R innerhalb des Terrains gegeben, ρ ist dabei der Winkel der Blickrichtung in der X/Y-Ebene.

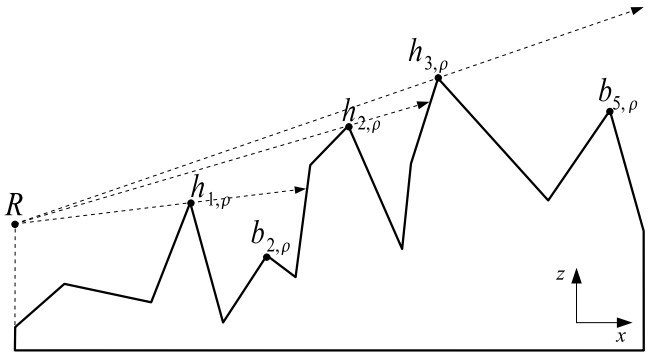

Abbildung 4.3: *Horizonte von R in Richtung ρ*

In einem solchen Schnitt ist eine blockierende Kante als ein Punkt vorhanden. Dieser wird im Folgenden als blockierender Punkt $b_{i,\rho}$ bezeichnet, wobei i den Index in der Abfolge der blockierenden Punkte von R aus angibt und ρ die Blickrichtung des Roboters ist. In dem Fall, dass ein blockierender Punkt $b_{i,\rho}$ zum Horizont gehört, wird er mit $h_{i,\rho}$ bezeichnet. Er gehört zum Horizont, wenn er die folgenden zwei Bedingungen erfüllt:

- $h_{i,\rho}$ ist von R aus sichtbar.

- an $h_{i,\rho}$ findet ein Wechsel bezüglich des sichtbaren Bereiches statt.

Die Horizontpunkte wurden in der Abbildung mit $h_{1,\rho}$, $h_{2,\rho}$ und $h_{3,\rho}$ bezeichnet. Die blockierenden Punkte $b_{2,\rho}$ und $b_{5,\rho}$ gehören hier nicht zum Horizont. Die Horizontpunkte $h_{j,\rho}$ haben die Eigenschaft, dass von R ausgehende Strahlen zuerst den Horizontpunkt als blockierenden Punkt treffen, bevor irgendein anderer Punkt des Terrains geschnitten wird, sofern überhaupt noch ein weiterer Punkt des Terrains geschnitten wird. Der Index j der Horizontpunkte gibt auch hier die Abfolge der Punkte auf der Terrainoberfläche vom Roboter ausgehend in Blickrichtung ρ an. Der erste Horizontpunkt $h_{1,\rho}$ wird dabei als *unterer Horizont* bezeichnet, während der Horizont mit dem größten Index j als *oberer Horizont* bezeichnet wird. In der Abbildung ist der obere Horizont durch $h_{3,\rho}$ gegeben. Es ist anzumerken, dass sowohl die blockierenden Punkte als auch die Horizonte von der Roboterposition R abhängig sind. In Falle des Schnittes wird dies durch die Blickrichtung ρ besonders deutlich. Im Folgenden werden nun die anschaulich eingeführten Begriffe formalisiert. Der Terrainschnitt $M_{R,\rho}$ ist wie folgt definiert:

Definition 4.2: Sei T ein dreidimensionales Terrain, R die Position eines Roboters in $\overline{L(T)}$ und $\rho \in [0, 2\pi[$ eine Blickrichtung des Roboters. Die Schritthalbebene $\Gamma_{R,\rho}$ ist gegeben durch die Richtungsvektoren $\vec{d_1} = (0, 0, 1)$ und $\vec{d_2} = (\cos\rho, \sin\rho, 0)$. Sie ist eingeschränkt auf positive $\vec{d_2}$ Richtung. Der Terrainschnitt $M_{R,\rho}$ ist gegeben durch $\Gamma_{R,\rho} \cap O(T)$.

Die Horizontpunkte auf einem solchen Terrainschnitt sind wie folgt definiert:

Definition 4.3: Sei $M_{R,\rho}$ ein Terrainschnitt. Die Menge $H_{R,\rho}$ aller Horizontpunkte ist die maximale Teilmenge von $M_{R,\rho}$, die die folgenden Eigenschaften besitzt:

- $H_{R,\rho}$ ist endlich
- $H_{R,\rho}$ ist anhand der Abfolge der Punkte auf $M_{R,\rho}$ von R ausgehend sortiert.
- Die Elemente werden mit $h_{j,\rho}$ bezeichnet, wobei j die Position in $H_{R,\rho}$ angibt. Dabei wird $h_{j,\rho}$ als Punkt des j-ten Horizonts bezeichnet.
- Sei g ein Strahl von R durch $h_{j,\rho}$. Es gilt:

  $$\exists \epsilon > 0 : \forall p \in g, d(p, h_{j,\rho}) < \epsilon : p \in \overline{L(T)}$$

  Der Strahl berührt $M_{R,\rho}$ also in einem lokalen Maximum $h_{j,\rho}$ und verläuft in dessen näherer Umgebung oberhalb der Terrainoberfläche.
- Weiterhin gilt:

  $$\forall q \in g, d(h_{j,\rho}, R) > d(q, R) : q \notin (M_{R,\rho} \setminus H_\rho)$$

  Der Strahl g hat also zuvor keinen Punkt von $M_{R,\rho}$ getroffen, der nicht Teil des Horizonts ist.
- Existiert zu einem $h_{j,\rho}$ ein $h_{j-1,\rho}$, so liegt $h_{j-1,\rho}$ unterhalb oder auf dem Strahl g.

Im Folgenden wird die Definition des Horizonts von einem Terrainschnitt auf das gesamte Terrain erweitert. Dabei ergibt sich, wie auch schon beim Terrainschnitt beobachtet, dass nicht alle blockierenden Kanten auch zum Horizont beitragen. Weiterhin kann

es vorkommen, dass nur Teile von blockierenden Kanten Horizontkanten sind. Dies wird in Abbildung 4.4 veranschaulicht. Auf der linken Seite wird die Terrainoberfläche $O(T)$ dargestellt. Die Knoten des Terrains und der Roboter an der Position R wurden zusätzlich mit ihrer Z-Koordinate beschriftet. Die blockierenden Kanten wurden fett eingezeichnet. Die rechte Seite der Abbildung zeigt das gleiche Terrain. Dünn gestrichelt eingezeichnet ist der Rand des Terrains. Der Horizont ist mit durchgezogenen Linien gekennzeichnet, wobei der obere Horizont dicker eingezeichnet wurde. Es ist ersichtlich, dass die Kanten des Terrainrandes nicht zum Horizont gehören. Die von R ausgehenden Strahlen markieren die *radialen Einflussbereiche* der einzelnen Horizontkanten. Die Darstellung im rechten Bereich der Abbildung wird auch als Horizontkarte bezeichnet.

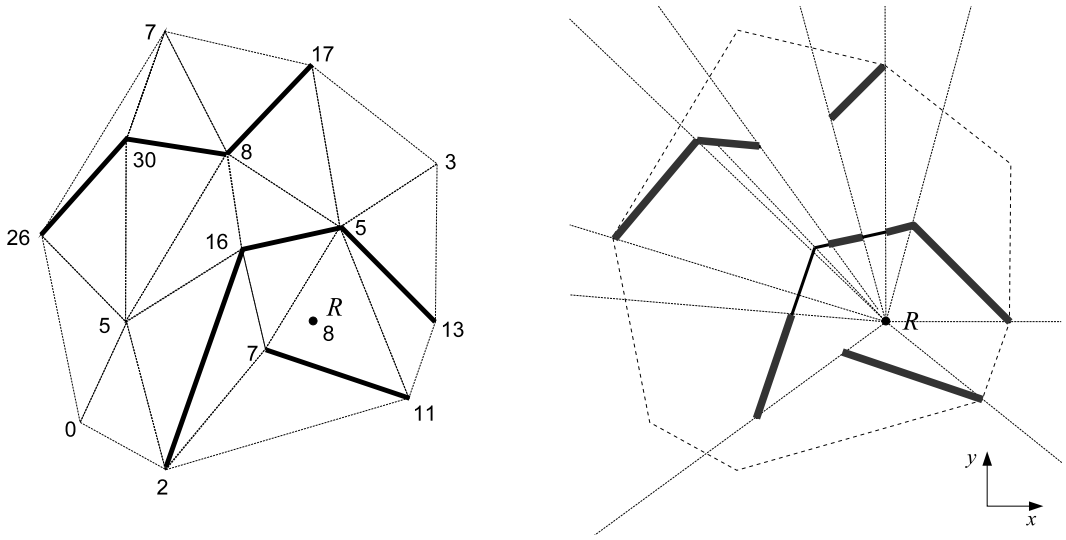

Abbildung 4.4: *Terrain mit blockierenden Kanten und Horizontkarte.*

In dem Beispiel der Abbildung ist nicht für jede Blickrichtung des Roboters im Intervall $[0, 2\pi[$ ein unterer Horizont definiert. Der untere Horizont wird daher als *radial nicht geschlossen* bezeichnet. Für die beiden Fälle, dass der Roboter am Rand des Terrains startet oder er in einer Blickrichtung schon das gesamte Terrain überblicken kann, existiert in einem Blickwinkelbereich kein Horizont und damit auch kein unterer bzw. oberer Horizont. In dem Fall, dass für jeden Blickwinkel im Intervall $[0, 2\pi[$ ein Horizont definiert ist, wird der Horizont als *radial geschlossen* bezeichnet. Existiert für eine Blickrichtung ein Horizont, so gilt aufgrund der Definition des Terrainschnitts automatisch, dass ein unterer und ein oberer Horizont definiert sind. Daraus folgt, dass der untere und der obere Horizont in demselben Blickwinkelbereich um den Roboter herum definiert sind.

Die folgenden Definitionen fassen die vorherigen Erläuterungen zum Horizont zusammen. Der Horizont wird dabei wie folgt definiert:

Definition 4.4: Sei T ein dreidimensionales Terrain und R eine Roboterposition in $\overline{L(T)}$. Der Horizont H ist gegeben durch

$$H = \bigcup_{\rho \in [0,2\pi[} H_\rho$$

Konkret besteht H, wie aus Abbildung 4.4 ersichtlich wird, aus einer Menge von Kanten bzw. Teilkanten des Terrains. In Anlehnung an die Sortierung der Horizontpunkte

im Terrainschnitt wird für die Kanten des Horizonts eine Folge von Teilhorizonten H_i ausgehend von der Roboterposition R definiert:

> **Definition 4.5:** Sei T ein dreidimensionales Terrain und R eine Roboterposition in $\overline{L(T)}$. Der j-te Teilhorizont H_j ist gegeben durch
>
> $$H_j \;=\; \{h_{j,\rho} : \rho \in [0, 2\pi[, h_{j,\rho} \in H_\rho\}$$

Für die einzelnen Teilhorizonte ergibt sich somit:

> **Beobachtung 4.1:** Eine Eigenschaft der hier beschriebenen Folge ist, dass für jeden Teilhorizont H_{j+1} in dem zugehörigen Winkelbereich des Roboters auch ein Teilhorizont H_j definiert ist, wobei H_j auch in zusätzlichen Winkelbereichen definiert sein kann.

Der Begriff des oberen Horizonts H_{max} ist für die in dieser Arbeit vorgestellten Online Strategien von besonderer Bedeutung. Der obere Horizont kann dabei aus unterschiedlichen Teilhorizonten zusammengesetzt sein, da nicht für jede Blickrichtung ρ die gleiche Anzahl von Horizontpunkten vorhanden sein muss. Die Horizontkanten des oberen Horizonts haben in dem zugehörigen Blickwinkelbereich des Roboters den größten X/Y-Abstand, d. h. den größten Abstand innerhalb der Projektion des Terrains und des Roboters in die X/Y-Ebene durch Ignorieren der Z-Koordinate. Formal bedeutet dies:

> **Definition 4.6:** Sei T ein dreidimensionales Terrain und R eine Roboterposition in $\overline{L(T)}$. Der obere Horizont H_{max} ist gegeben durch:
>
> $$H_{max} \;=\; \left\{h : \exists H_\rho, \rho \in [0, 2\pi[, |H_\rho| > 0 : h = h_{|H_\rho|,\rho}\right\}$$

4.2.2 Veränderung des Horizontes

Wie zuvor erwähnt, ist die Horizontstruktur abhängig von der Position des Roboters. Abbildung 4.5 verdeutlicht dies anhand eines Terrainschnittes. Ein Roboter sieht von seiner Position R aus zwei Horizonte, den unteren Horizont $h_{1,\rho}$ und den oberen Horizont $h_{2,\rho}$. Steigt der Roboter, so verändert sich beim Erreichen der Position R' der Horizont. Der Roboter kann nun hinter $h_{2,\rho}$ sehen, weshalb dieser Punkt nicht mehr Teil des Horizonts ist. Der untere Horizont $h_{1,\rho}$ wird somit zusätzlich auch oberer Horizont.

Ein solcher Terrainschnitt veranschaulicht die Bewegung eines Roboters nur im Bezug auf einen seiner Freiheitsgrade entlang der Z-Achse. Zusätzlich ist seine Blickrichtung ρ von Bedeutung. Da ein punktförmiger Roboter jedoch drei Freiheitsgrade hat, kann die Veränderung des Horizontes eines Roboters im dreidimensionalen Terrain deutlich komplexer sein.

Abbildung 4.6 verdeutlicht dies anhand zweier Beispiele. Das Beispiel auf der linken Seite zeigt einen Roboter an einer Startposition R. An dieser Position sind h_1 und h_3 Horizontkanten, während h_2 nicht sichtbar ist. Eine Datenstruktur für den Horizont könnte nun davon ausgehen, dass blockierende Kanten hinter anderen Horizontkanten sichtbar werden können. In diesem Fall würde, wenn der Roboter lediglich ansteigt, h_2 hinter h_3 sichtbar werden. Diese Datenstruktur könnte nicht sichtbare blockierende Kanten z. B. einem radialen Einflussbereich einer Horizontkante für den Roboter zuordnen. Aufgrund

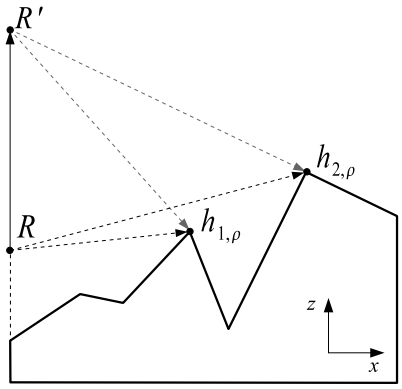

Abbildung 4.5: *Der untere Horizont kann durch die Bewegung des Roboters zum oberen Horizont werden.*

von Bewegungskomponenten in der X/Y-Ebene sind solche Zuordnungen jedoch nicht unbedingt sinnvoll, da sich der radiale Einflussbereich einer Horizontkante mit der Veränderung der Roboterposition verschiebt, wodurch sich die Zuordnung ändern kann oder auch blockierende Kanten neben einer Horizontkante sichtbar werden können. Dieses Prinzip wird in Abbildung 4.6 durch die Roboterposition R' dargestellt. Die Kante h_2 wurde kurzzeitig sichtbar, als der Roboter zwischen h_1 und h_3 einen Ausschnitt von h_2 sehen konnte. Von besonderem Interesse ist dabei, dass nicht unbedingt einer der Eckpunkte von h_2 als erstes sichtbar werden muss, sondern auch, wie hier gezeigt, ein innerer Teil der Kante zuerst sichtbar werden kann. Weiterhin zeigt das Beispiel einer weiteren Roboterposition R'', dass eine Kante wieder aus dem Horizont verschwinden kann, ohne dass ein Einblick hinter diese Kante erfolgt sein muss.

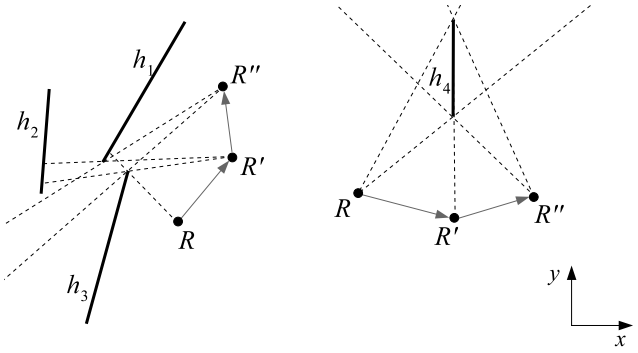

Abbildung 4.6: *Veränderungen am Horizont bei der Bewegung des Roboters.*

Die rechte Seite von Abbildung 4.6 zeigt ein weiteres Beispiel, in welchem eine Horizontkante h_4 aufgrund der Veränderung ihres radialen Einflussbereichs an der Position R' kurzzeitig aus dem Horizont verschwindet. Zuvor, als sich der Roboter an der Position R befand, gehörte die Kante zum Horizont. Bewegt sich der Roboter weiter zur Position R'', kann die Kante erneut zum Horizont gehören. In dieser Position ist die Rolle der angrenzenden Facetten der Kante jedoch vertauscht. Für den Fall, dass eine Online Strategie die Sichtbarkeit immer erweitert, ist dieser Fall für eine Horizontkante ausgeschlossen. Für

blockierende Kanten, die nicht Teil des Horizontes sind und damit nicht sichtbar sind, kann dieser Fall jedoch auch dann auftreten.

Die Problematik, die durch die beiden Beispiele gezeigt wurde und im Wesentlichen auf der Veränderung des radialen Einflussbereichs einer Horizontkante beruht, wird im Folgenden als das Problem der *radialen Verschiebung* bezeichnet. Der Entwurf einer effizienten Datenstruktur zur Speicherung des Horizontes und einer schnellen Aktualisierungsmöglichkeit erscheint aufgrund von radialen Verschiebungen als schwierig. Der Entwurf einer solchen Datenstruktur ist nicht Bestandteil dieser Arbeit.

4.2.3 Sichtbarkeitspolyeder und -kegel

Die wesentlichen Informationen zur Sichtbarkeit im dreidimensionalen Terrain können aus der Horizontkarte gewonnen werden. Aus dieser kann auch das in Definition 2.6 definierte Sichtbarkeitspolyeder abgeleitet werden. Hierzu ist die Definition der *Sichtbarkeitsebene* hilfreich:

> **Definition 4.7:** Sei T ein dreidimensionales Terrain, R eine Roboterposition in $\overline{L(T)}$ und e eine Horizontkante des Terrains. Sei $S_{R,e}$ die Ebene, die durch R und e aufgespannt wird. Der Normalenvektor von $S_{R,e}$ sei so gerichtet, dass seine Z-Komponente positiv ist. $S_{R,e}$ heißt Sichtbarkeitsebene von e bezüglich der Roboterposition R. $S_{R,e}$ teilt das Terrain vom Roboter aus betrachtet hinter der Kante e in einen sichtbaren (oberen) und einen nicht sichtbaren (unteren) Bereich auf. Die Menge aller Sichtbarkeitsebenen bezüglich R wird mit S_R bezeichnet.

Für die Bestimmung des Sichtbarkeitspolyeders ist die Definition eines Sichtbarkeitsdreiecks interessant:

> **Definition 4.8:** Sei e eine Horizontkante im Terrain T und R eine Roboterposition in $\overline{L(T)}$. Ferner seien g und h zwei von R ausgehende Strahlen durch die Endpunkte von e. Dann ist das Sichtbarkeitsdreieck $D_{R,e}$ gegeben als die Teilmenge der Sichtbarkeitsebene $S_{e,R}$, die zwischen den Strahlen g und h liegt, wobei g, h und e in der Teilmenge enthalten sind.

Mit Hilfe des Horizonts kann das Sichtbarkeitspolyeder $visP(R)$ aufgebaut werden. Dazu speichert jede Horizontkante e einen Verweis auf die Horizontkanten und die Kanten des Terrains, die im radialen Einflussbereich von e hinter e liegen. Diese Verweise können während des Algorithmus zur Bestimmung des Horizonts gesetzt werden, was kaum Mehraufwand erfordert und das Laufzeitverhalten des Algorithmus nicht beeinträchtigt. Abbildung 4.7 zeigt ein Beispielterrain, in dem der radiale Einflussbereich der Kante e hellgrau eingefärbt wurde.

Die Facetten des Sichtbarkeitspolyeders werden nacheinander berechnet. Dabei wird mit den Facetten innerhalb des unteren Horizonts begonnen. In Abbildung 4.7 ist dieser Bereich dunkelgrau eingefärbt. Aus der Grafik ist ersichtlich, dass die Facetten des Terrains nicht unbedingt vollständig zum Rand des Sichtbarkeitspolyeders gehören. Die Teile der Facetten, die abgeschnitten werden müssen, können anhand des radialen Einflussbereiches der Horizontkanten des unteren Horizonts bestimmt werden. Zu diesem Zweck kann

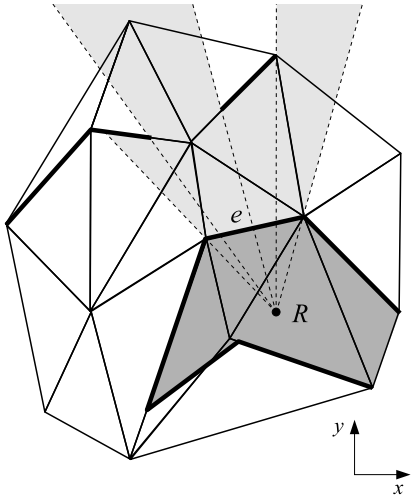

Abbildung 4.7: *Der radiale Einflussbereich von e.*

der dem Roboter näher gelegene Teil des jeweiligen Sichtbarkeitsdreiecks $D_{R,e}$ verwendet werden.

Das Prinzip des weiteren Vorgehens bei der Bestimmung des Sichtbarkeitspolyederrandes soll zunächst anhand des Terrainschnitts aus Abbildung 4.8 gezeigt werden. Die Kanten vor $h_{1,\rho}$ wurden im ersten Schritt des Algorithmus bereits zu dem Rand des Sichtbarkeitspolyeders hinzugefügt. In einem Terrainschnitt ist das Sichtbarkeitspolyeder $visP(R)$ ein Sichtbarkeitspolygon $vis(R)$. Der Bereich, der dadurch schon als Inneres des Sichtbarkeitspolygons bekannt ist, wurde dunkelgrau eingefärbt.

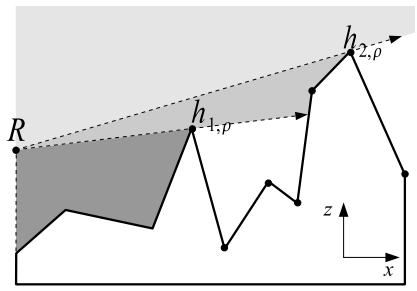

Abbildung 4.8: *Bestimmung des Sichtbarkeitspolygons eines Terrainschnitts.*

Zur Bestimmung des nächsten Randpunktes von $vis(R)$ hinter $h_{1,\rho}$, werden die drei hinter $h_{1,\rho}$ liegenden Eckpunkte auf Sichtbarkeit von R aus überprüft. Die Eckpunkte entsprechen dabei den Kanten des Terrains, die dem Einflussbereich der vorherigen Horizontkante zugeordnet wurden. Da die Eckpunkte unter dem von R ausgehenden Strahl S durch $h_{1,\rho}$ liegen, sind sie nicht sichtbar. Erst der vierte Punkt ist wieder von R aus sichtbar. Somit ist klar, dass S die Terrainoberfläche zwischen dem dritten und dem vierten Punkt geschnitten hat. Der Bereich des Strahls S von $h_{1,\rho}$ aus bis zu diesem Schnittpunkt bildet somit den Rand des Sichtbarkeitspolyeders. Weiterhin gehört der Teil der Terrainoberfläche zwischen dem Schnittpunkt und dem nächsten Horizont $h_{2,\rho}$ zu diesem Rand. Somit ist auch ein weiterer Teil des Inneren von $vis(R)$ bekannt. Dieser wurde in der

Abbildung mittelgrau unterlegt. Dieser Schritt wird solange wiederholt, bis der oberen Horizont erreicht wird. In der Abbildung ist dies bereits bei $h_{2,\rho}$ der Fall. Da der Strahl hinter dem oberen Horizont die Terrainoberfläche nicht wieder schneidet, wird dem Rand des Sichtbarkeitspolygons $vis(R)$ hier eine Halbgerade von $h_{2,\rho}$ in das Unendliche hinzugefügt. Der gesamte Bereich oberhalb des zugehörigen Strahles, in der Abbildung hellgrau eingezeichnet, gehört zum Inneren von $vis(R)$ und ist nach oben hin unbegrenzt.

Bezogen auf das dreidimensionale Terrain entsprechen die Strahlen den zuvor vorgestellten Sichtbarkeitsdreiecken, die das Innere des Sichtbarkeitspolyeders hinter einer Horizontkante nach unten hin beschränken. Der Schnitt eines Sichtbarkeitsdreiecks mit einem Terrain kann allerdings eine komplexe Struktur haben. Dieser entspricht einer Kontur aus Schnitten mit einer oder mehreren Facetten des Terrains. Die Facetten bzw. Teilfacetten, die vom Roboter aus hinter dieser Kontur und vor dem nächsten Horizont liegen, gehören auch zum Rand des Polyeders. Ein zwischenzeitlicher Wechsel in einen nicht sichtbaren Bereich kann nicht erfolgen, da ansonsten ein weiterer Horizont vorhanden wäre.

Mit Hilfe der Zuordnung der Terrainkanten zu den Einflussbereichen der Horizontkanten, wie sie in Abbildung 4.7 eingeführt wurde, lässt sich die Kontur hinter einer Horizontkante e schnell ermitteln. Analog zu dem Beispiel aus dem Terrainschnitt können die e zugeordneten Kanten überprüft werden, ob sie hinter e sichtbar sind. Sobald bei dieser Sichtbarkeitsprüfung ein Wechsel zwischen sichtbarem und nicht sichtbarem Bereich festgestellt wird, kann von der zugehörigen Facette ausgehend die komplette Kontur als Schnitt zwischen Terrain und Sichtbarkeitsdreieck gebildet werden. Auch die Bestimmung der restliche Facetten bzw. Teilfacetten zwischen der Kontur und den nachfolgenden Horizontkanten lässt sich mit Hilfe des Sichtbarkeitsdreiecks einfach bestimmen.

In dem Terrainschnittbeispiel aus Abbildung 4.8 wurde ein von R ausgehender Strahl durch den Punkt des oberen Horizonts nicht abgeschnitten. Ebenso wird ein Sichtbarkeitsdreieck des oberen Horizonts im dreidimensionalen Terrain nicht begrenzt. Beispiele hierfür sind die Winkelbereiche aus Abbildung 4.7, in denen der Einflussbereich der Kante e unbegrenzt ist. Die folgende Beobachtung fasst die Struktur des Sichtbarkeitspolyeders zusammen:

> **Beobachtung 4.2:** Das Sichtbarkeitspolyeder $visP(R)$ zu einer Roboterposition R in einem Terrain T umschließt den ganzen sichtbaren Bereich des Roboters. Es ist nach oben hin offen. Der Rand des Polyeders besteht aus Facetten und Teilfacetten des Terrains, sowie aus Teilen von Sichtbarkeitsdreiecken der Horizontkanten.

Es wird nun eine Teilmenge des Sichtbarkeitspolyeders betrachtet. Diese Teilmenge ist als Analogie zu dem Gewinnbereich $G(p)$ der CAB Strategie aus Abschnitt 2.5 definiert und wird im Folgenden als *Sichtbarkeitskegel* bezeichnet:

> **Definition 4.9:** Sei T ein Terrain und R eine Roboterposition in $\overline{L(T)}$. Dann ist der Rand des Sichtbarkeitskegels B_R gegeben durch die Vereinigung aller Sichtbarkeitsdreiecke des oberen Horizonts. Das Innere des Kegels liegt dabei in Richtung der Normalenvektoren der Sichtbarkeitsdreiecke. Ist der obere Horizont nicht radial geschlossen, so ist der Sichtbarkeitskegel in dem entsprechenden Winkelbereichen nicht definiert.

Der Sichtbarkeitskegel B_R ist eine Teilmenge des Sichtbarkeitspolyeders. In Abbildung 4.9 wird der Sichtbarkeitskegel unter anderem durch Sichtbarkeitsdreiecke auf die

Horizontkanten f und g aufgespannt. Weitere Begrenzungen sind aus dem Terrainschnitt nicht ersichtlich. Eine solche Abbildung kann zu der Vermutung führen, dass der Sichtbarkeitskegel tatsächlich kegelförmig ist. Dies ist jedoch nicht zutreffend. Wie anhand der Horizontkarte aus Abbildung 4.7 zu erkennen ist, ist der obere Horizont im Allgemeinen nicht konvex, wodurch auch der Sichtbarkeitskegel nicht konvex sein muss. Im Folgenden wird ein Beispiel dafür beschrieben.

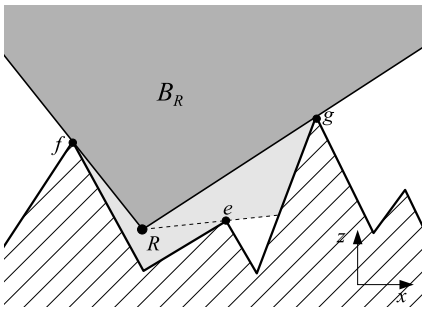

Abbildung 4.9: *Ein einfacher Sichtbarkeitskegel.*

Abbildung 4.10 zeigt ein Terrain, welches aus zwei Bergen besteht. Der Horizont besteht aus den Kanten h_0 bis h_4. Der sichtbare Teil des Terrains wurde hellgrau unterlegt. In der Abbildung wurden die Teile der Sichtbarkeitsdreiecke bis zu den Horizontkanten grau eingezeichnet. Im linken Teil der Abbildung wurde eine Startposition R des Roboters auf der Terrainoberfläche gewählt. Der Sichtbarkeitskegel B_R ist in diesem Fall nicht konvex. Weiterhin ist B_R nicht für alle Blickwinkelbereiche des Roboters definiert. Im rechten Teil des Terrains existiert kein Horizont und somit ist auch der Sichtbarkeitskegel für diesen Bereich nicht definiert. Ein solcher Sichtbarkeitskegel wird als *radial nicht geschlossen* bezeichnet. Im rechten Teil der Abbildung ist der Roboter zu der Position R' aufgestiegen. Der Sichtbarkeitskegel ist nach außen gewendet. Er ist also nicht nur nach oben offen, sondern auch zu den Seiten hin. Die Form eines Sichtbarkeitskegels erinnert daher nur entfernt an einen Kegel.

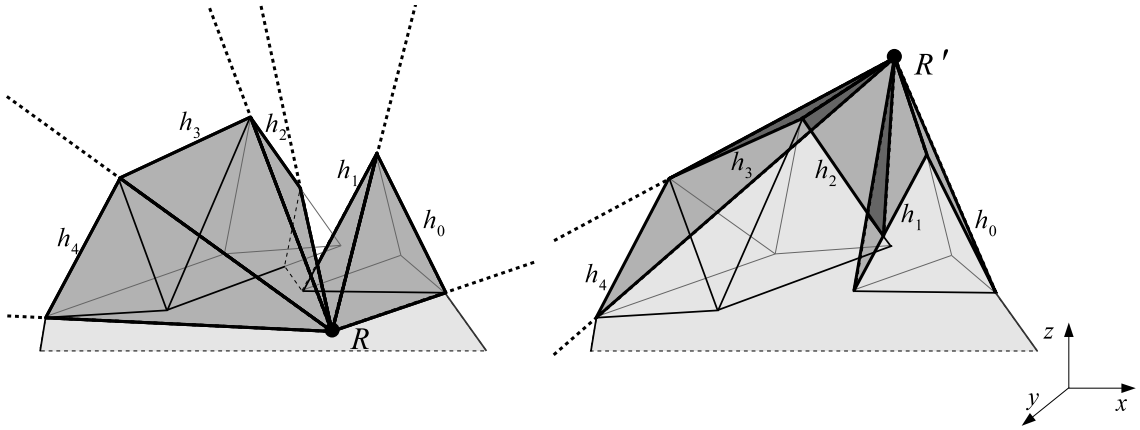

Abbildung 4.10: *Der Sichtbarkeitskegel ist nicht konvex.*

Wie bereits erwähnt, ist der Sichtbarkeitskegel eine Analogie zum Gewinnbereich $G(p)$ der *CAB* Strategie. Die Online Strategien der Abschnitte 4.3.3 und 4.4.6 berücksichtigen den Aufbau des Sichtbarkeitskegels und lehnen sich dabei an die Idee des Bisektors der *CAB* Strategie an.

In der *CAB* Strategie ist es möglich, dass der Roboter die Sicht auf Polygonteile wieder verliert, wenn er ohne weitere Einschränkung in den Gewinnbereich $G(p)$ hineinläuft. Dieses Problem wird durch den Haltebereich $E(p)$ gelöst, welcher bei einer Bewegung nicht verlassen werden darf. Analog hierzu kann die Sicht eines Roboters, der in den Sichtbarkeitskegel hinein fliegt, eingeschränkt werden. In Abbildung 4.9 verliert der Roboter z. B. die Sicht auf die Kante e, sobald er sich rechts von der Kante g innerhalb des Sichtbarkeitskegels befindet. Um dies zu vermeiden, wird, analog zu dem Haltebereich $E(p)$ aus der *CAB* Strategie, ein weiteres Kriterium für die Bewegung des Roboters angegeben. Der Roboter bestimmt zu seiner aktuellen Position R von allen bereits gesehenen und den aktuell sichtbaren Facetten einen vorläufigen Kern ker_{tmp}. Dieser vorläufige Kern darf nicht verlassen werden, um möglichst die Sicht auf bereits einsehbare Terrainteile zu erhalten. Eine Online Strategie, die diese Regel befolgt und zusätzlich die Sicht des Roboters auf das Terrain vergrößert, indem Blick auf neue Facetten ermöglicht, arbeitet somit korrekt.

Es ist jedoch anzumerken, dass auch die Verwendung von vorläufigen Kernen nicht garantieren kann, dass die Sicht auf bereits gesehene Facetten und Kanten des Terrains erhalten wird. Dieses Problem besteht aufgrund der *radialen Verschiebung*, welche in Abschnitt 4.2.2 beschrieben wurde, und begründet sich darauf, dass die Berechnung eines vorläufigen Kerns auf den zu Facetten zugehörigen Ebenen beruht und nicht auf den Facetten selbst. Im Beispielterrain aus Abbildung 4.11 geht die Sicht auf eine Facette verloren während der Roboter sich von der Position R zur Position R' bewegt und dabei den aktuellen vorläufigen Kern nicht verlässt. An Position R konnte der Roboter durch das vordere Hindernis hindurch die drei Kanten e, f, g und die beiden Facetten F_1 und F_2 sehen. Nach seiner Bewegung zu der Position R' hin, hat er die Sicht auf die Kanten f und g und die Facette F_2 verloren. Die Bewegung zu R' hin kann jedoch nicht durch eine Einschränkung der Bewegung auf den vorläufigen Kern verhindert werden. Die Öffnung in dem Hindernis, durch welche der hintere Teil des Terrain sichtbar ist, erinnert an ein Fenster. Daher wird im Folgenden die hier beschriebene Problematik als *Fenstereffekt* bezeichnet.

In dem Fall, dass die Facette F_2 zu dem vorläufigen Kern des Roboters an der Position R beigetragen hat und nur sichtbare Facetten bei dessen Berechnung verwendet würden, so würde F_2 bei der Berechnung des vorläufigen Kerns des Roboters an der Position R' nicht mehr berücksichtigt. Somit würde eine wichtige Information über die Sichtbarkeit in dem Terrain verloren gehen und eine Korrektheit dürfte nur schwer zu erreichen sein. Daher ist es wichtig, dass alle jemals sichtbar gewesenen Facetten in diese Berechnung einfließen. Der vorläufige Kern bekommt dadurch die zusätzliche Funktion als Gedächtnis des Roboters.

Für die Korrektheit eines Algorithmus sind weder die Sichtbarkeit auf eine Kante noch die auf eine Facette von entscheidender Bedeutung. Wichtig ist lediglich, dass eine Sicht auf die Ebenen erhalten bleibt, die durch bereits gesehene Facetten des Terrains gegeben sind. Er muss also weiterhin oberhalb dieser Ebenen bleiben. Dies kann dadurch gewährleistet werden, dass sich der Roboter im vorläufigen Kern aufhält und diesen nicht verlässt. Bewegt sich der Roboter nun im vorläufigen Kern so weiter, dass er hinter eine weitere Horizontkante e sehen kann, so befindet er sich oberhalb einer weiteren Ebene,

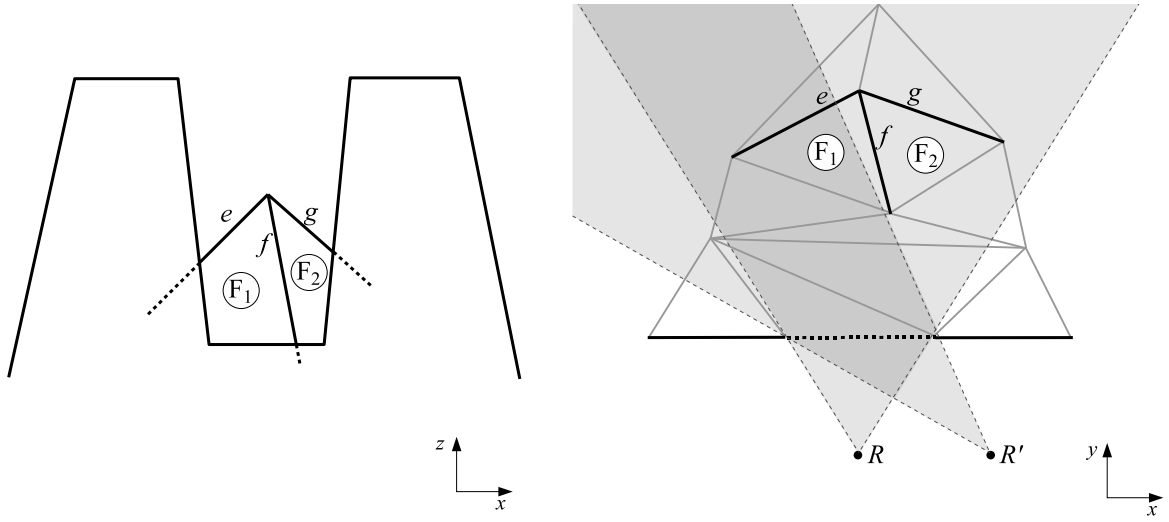

Abbildung 4.11: *Der Fenstereffekt.*

nämlich der Ebene, die durch die weiter entfernt liegenden Facette von e gegeben ist. Dies garantiert, dass der vorläufige Kern weiter eingeschränkt wird und die Folge der vorläufigen Kerne gegen den Kern des Terrains konvergiert.

Das Sichtbarkeitspolyeder und der Sichtbarkeitskegel sind für die Entscheidungen, die von den in dieser Arbeit vorgestellten Online Strategien getroffen werden, nur indirekt von Bedeutung. Für diese Entscheidungen ist der Begriff des Horizonts ausreichend. Sowohl Sichtbarkeitspolyeder als auch Sichtbarkeitskegel sind jedoch für das Verständnis und die Visualisierung der Strategien wichtig. Das Sichtbarkeitspolyeder wird zudem für die Bestimmung der Facetten, deren zugehörige Ebenen zum vorläufigen Kern beitragen, verwendet.

4.2.4 Algorithmus zur Bestimmung des Horizonts

Im Folgenden wird ein Verfahren beschrieben, welches den Horizont bestimmt und eine Horizontkarte erstellt. In [14] wird die Komplexität des Horizonts in Bezug auf die Horizontkanten mit $O(n^3)$ abgeschätzt, wobei n die Anzahl der Kanten im Terrain ist. Floriani und Magillo beschreiben dort einen Algorithmus, der den Horizont durch einen radialen Sweep aufbaut. Die Anzahl der Ereignisse ist in $O(n+k)$, wobei k die Anzahl der Konflikte der blockierenden Kanten ist. Das führt zu einem Algorithmus dessen Laufzeit in $O((n+k)\log n)$ ist, im schlimmsten Fall ist dabei $k = O(n^2)$. In vielen in der Praxis vorkommenden Fällen soll die Laufzeit jedoch besser sein. An dieser Stelle wird jedoch ein eigener alternativer Algorithmus vorgestellt, der den Horizont von der Roboterposition ausgehend nach außen hin aufgebaut.

Initialisierung

In der Initialisierung werden die Kanten des Terrains identifiziert, die blockierende Kanten sind und somit potenziell zum Horizont gehören. Gleichzeitig werden die blockierenden Kanten sortiert, um dann während der Iterationsschritte in den Horizont eingefügt zu

werden.

Zunächst werden einige Vorüberlegungen und Definitionen für den Algorithmus vorgestellt. Abbildung 4.12 zeigt zwei intuitive Sortierkriterien für die blockierenden Kanten, die jedoch nicht alle möglichen Fälle berücksichtigen und damit nicht verwendet werden können. Sie sind dennoch als Vorüberlegungen für ein korrektes Sortierkriterium interessant. In der Abbildung werden zwei Szenarien dargestellt, in denen jeweils eines der beiden Kriterien fehlschlägt. Das Sortierkriterium $\tilde{d}_i$ sortiert die Kanten nach ihrem euklidischen Abstand zur Roboterposition R innerhalb der Projektion auf die X/Y-Ebene. Existiert kein Lot von R auf die Kante, so wird der nähere Endpunkt der Kante als Abstand genommen. Aus diesem Sortierkriterium folgt für das linke Szenario eine Kantenreihenfolge von b_1, b_3, b_2. Dies führt dazu, dass in späteren Schritten des Algorithmus' die Sichtbarkeit von b_2 *hinter* b_3 getestet wird, was zu einem falschen Ergebnis führt. In dem rechten Szenarium führt dieses Sortierkriterium hingegen zu einer geeigneten Reihenfolge.

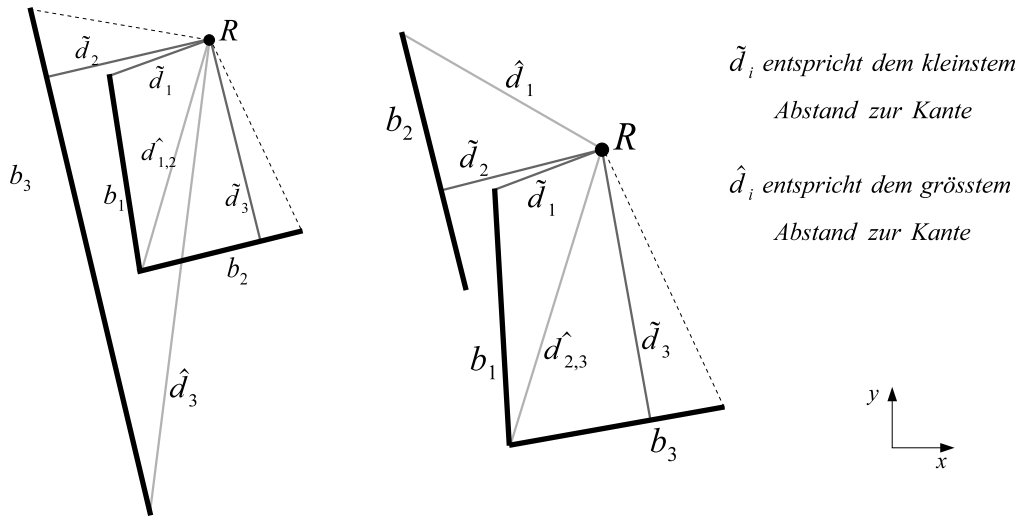

Abbildung 4.12: *schlechte Sortierungen für die blockierenden Kanten.*

Das zweite Sortierkriterium sortiert die Kanten nach dem größeren Abstand zur Roboterposition, d.h. anhand des Endpunktes der Kante, der den größeren Abstand zu R bezüglich der X/Y-Projektion hat. In dem linken Szenario liefert dieses Kriterium zwar ein geeignetes Ergebnis, in dem rechten Szenarium ist die Reihenfolge von b_2, b_1, b_3 jedoch nicht korrekt. Dies führt in späteren Schritten des Algorithmus dazu, dass überprüft wird, ob b_1 *hinter* b_2 sichtbar ist, was wiederum zu einem falschen Ergebnis führt.

Für das Sortierverfahren, welches im Folgenden vorgestellt wird, und für die weiteren Schritte des Algorithmus ist der Begriff des *Winkelbereichs* einer Kante von Bedeutung. Dieser wird unter anderem auch als Sortierkriterium innerhalb der Horizontdatenstrukturen des Algorithmus verwendet. Abbildung 4.13 zeigt den Winkelbereich einer Kante e bezüglich einer Roboterposition R.

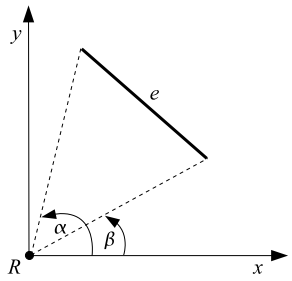

Abbildung 4.13: *Der Winkelbereich der Kante e umfasst die Winkel von β bis α.*

Der Winkelbereich ist dabei wie folgt definiert:

> **Definition 4.10:** Sei T ein dreidimensionales Terrain, R eine Roboterposition in $\overline{L(T)}$ und e eine blockierende Kante. Der Winkelbereich ψ bezüglich der Projektion des Terrains in die X/Y Ebene ist gegeben durch das Intervall$[\beta, \alpha]$ mit α und β als Winkel zwischen der X-Achse und den Strahlen von R durch die beiden Endpunkte der Kante. Ferner sei die Zuordnung der Winkel zu den Endpunkten so gewählt, dass die Kante in dem Bereich liegt, der durch ψ gegeben ist.

Zur Bestimmung der blockierenden Kanten und ihrer Abarbeitungsreihenfolge A für die Iterationsschritte des Algorithmus' werden alle Kanten des Terrains ausgehend von der Roboterposition R durchgegangen. Es wird bei dem hier vorgestellten Verfahren angenommen, dass die Oberfläche des Terrains trianguliert ist. Der Initialisierungsschritt wird anhand der X/Y-Projektion des Terrains durchgeführt. Das Terrainbeispiel aus Abbildung 4.14 wird zur Erklärung verwendet.

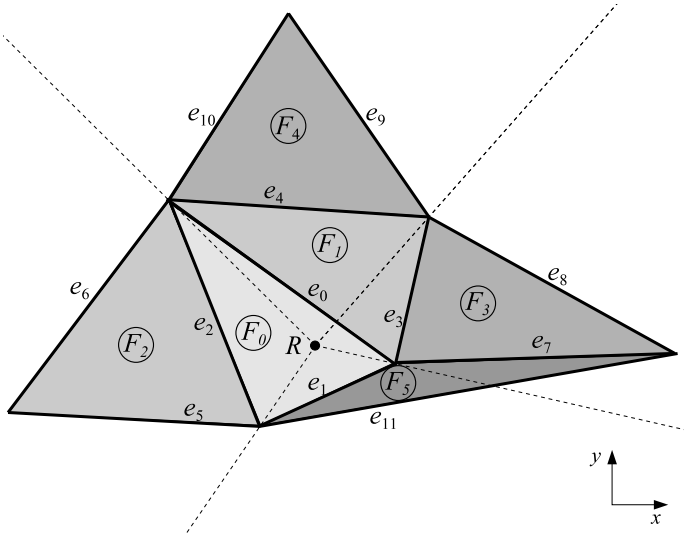

Abbildung 4.14: *Abarbeitungsreihenfolge der Kanten e_j*

Während des Initialisierungsschrittes werden vier Datenstrukturen verwendet:

- U ist die Schlange der abzuarbeitenden Facetten.
- V ist die Menge der bereits erreichten aber noch nicht abzuarbeitenden Facetten.
- W ist die Schlange der abzuarbeitenden Kanten.
- C speichert die noch nicht erreichten Kanten des Terrainrandes.

Die Datenstrukturen V und W werden leer initialisiert, während U zu Beginn die Facette F_0 enthält. Dies ist die Facette, in der sich die Roboterposition R befindet. In C sind zunächst alle Kanten des Terrainrandes enthalten. Für das Durchwandern des Terrains werden nun abwechselnd die Schlangen U und W abgearbeitet, solange bis beide Schlangen leer sind. Die Reihenfolge in dem Beispiel, in welcher die Kanten und Facetten in die jeweiligen Schlangen eingefügt werden, ist durch die zugehörigen Indizes gegeben.

Bei der Abarbeitung der Schlange U wird diese nach und nach geleert. Für jede Facette F_k wird geprüft, ob die an die Facette angrenzenden Kanten bereits abgearbeitet wurden. Dies ist genau dann der Fall, wenn die von R aus weiter entfernt liegende Facette einer dieser Kanten gerade F_k ist. In diesem Fall wird eine solche Kante ignoriert. Ist dies nicht der Fall, so wurde die Kante noch nicht abgearbeitet. Sie wird daher in die Schlange W hinzugefügt.

Nach der Abarbeitung der Schlange U erfolgt die Abarbeitung der Schlange W. Diese wird ebenfalls nach und nach geleert und füllt gegebenenfalls wieder U. Für jede Kante e_j aus W wird zunächst geprüft, ob sie eine blockierende Kante ist. Ist dies der Fall, so wird sie an das Ende der Abarbeitungsreihenfolge der blockierenden Kanten A eingefügt. Anschließend wird überprüft, ob die weiter entfernt liegende Facette F_l zu U oder zu V hinzugefügt werden muss. Es werden dabei die folgenden vier Fälle unterschieden:

- Die Kante e_j hat keine weiter entfernt liegende Facette und gehört zum Rand des Terrains. In diesem Fall wird die Kante weder zu U noch zu V hinzugefügt. Die Kante wird jedoch aus C entfernt, da dieser Abschnitt des Terrainrandes somit erreicht wurde.
- Die Facette F_l ist nicht in V enthalten. Weiterhin wird die Kante e_j durch das Liniensegment zwischen R und dem Knoten von F_l, der nicht zu e_j gehört, geschnitten. In diesem Fall sind die beiden anderen Kanten der Facette über e_j sichtbar. Die Facette kann daher sofort abgearbeitet werden und wird zu U hinzugefügt. Dieser Fall tritt in dem Beispiel bei den Kanten e_0, e_2, e_3 und e_4 auf.
- Die Facette F_l ist nicht in V enthalten, die Kante e_j wird jedoch nicht von dem zuvor beschriebenen Liniensegment geschnitten. Daraus folgt, dass noch eine zweite Kante erreicht werden muss, bis vom Roboter aus alle Punkte der Facette erreicht werden können. Die Facette F_l wurde also bereits erreicht, kann aber noch nicht abgearbeitet werden. Daher wird F_l zu V hinzugefügt. Dieser Fall tritt in dem Beispiel bei der Kante e_1 auf.
- Die Facette F_l wurde bereits von einer anderen Seite erreicht und ist daher in V enthalten. Da die Facette somit bereits von zwei Seiten erreicht wurde, muss auch die dritte Kante der Facette komplett von R aus erreichbar sein. Sie wird daher aus V entfernt und zu U hinzugefügt. Dieser Fall tritt in dem Beispiel bei der Kante e_7 auf.

Sind die beiden Schlangen nach diesen Abarbeitungsschritten leer, so ist mit Hilfe von C zu überprüfen, ob der Terrainrand bereits komplett erreicht wurde. Ist dies der Fall, so wurden alle blockierenden Kanten gefunden und die Abarbeitungsreihenfolge wurde festgelegt. Existiert jedoch noch eine Kante in C, so ist ein Spezialfall eingetreten und die Kante mit dem kleinsten Abstand zu R wird aus C entfernt und zu W hinzugefügt. Die beiden Abarbeitungsschritte für die Schlangen werden dann erneut fortgesetzt, bis diese Schlangen beide wiederum leer sind. Dies wird so lange fortgesetzt, bis auch C leer ist.

Iterationsschritte

In den Iterationsschritten werden die entsprechenden Teile der blockierenden Kanten anhand ihrer Abarbeitungsreihenfolge aus der Initialisierung nach und nach in den Horizont eingefügt. Zur Verwaltung des Horizonts werden drei Datenstrukturen verwendet, in denen die Horizontkanten enthalten sind. Dies sind eine Datenstruktur für den unteren Horizont, eine die den kompletten Horizont enthält und eine weitere für den oberen Horizont H_{max}. In einem Iterationsschritt werden für die aktuelle blockierende Kante e zunächst alle Winkelbereiche der Kante zum Horizont hinzugefügt, in denen noch kein Horizont definiert wurde. Diese müssen auch zum oberen und unteren Horizont hinzugefügt werden.

Ist für Winkelbereiche der Kante e bereits ein Horizont definiert, so muss die Kante gegebenenfalls in mehrere Teile unterteilt werden und die Teilstücke müssen getrennt zum Horizont hinzugefügt werden. Abbildung 4.15 zeigt ein Beispiel, in dem die Kante e die Kante f aus H_{max} überlappt und in zwei Stücke e_1 und e_2 unterteilt wird. In diesem Fall hat e_1 einen Konflikt mit f und es muss überprüft werden, ob e_1 hinter f ganz oder auch nur teilweise sichtbar ist. Im Winkelbereich des überhängenden Stückes e_2 muss ferner überprüft werden, ob weitere Kanten aus H_{max} existieren, mit denen e_2 in Konflikt steht. In der Abbildung wird durch gestrichelte Linien angedeutet, welche Kantenteile zum resultierenden oberen Horizont gehören.

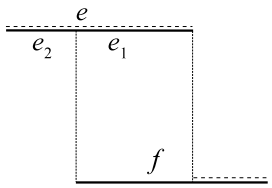

Abbildung 4.15: *Die Kante e überlappt f und wird, ebenso wie f, in zwei Stücke geteilt.*

Wie an dem obigen Beispiel gezeigt wurde, kann die Kante e in mehrere Teilstücke unterteilt werden. Jedes dieser Teilstücke e_i kann entweder, wie zuvor geschildert, direkt in alle drei Horizontdatenstrukturen eingefügt werden oder es besteht ein Konflikt mit einer bestehenden Horizontkante f_j aus H_{max}. Jedes Teilstück e_i, welches hinter einer Horizontkante f_j liegt, muss auf Sichtbarkeit hinter f_j überprüft werden. Dabei kann e_i in einen sichtbaren und einen nicht sichtbaren Teil unterteilt werden. Existiert ein sichtbares Teilstück von e_i, so wird für den zugehörigen Winkelbereich ein Teil von f_j aus dem oberen Horizont entfernt. Wie in Abbildung 4.16 gezeigt wird, existieren drei Fälle für die Aktualisierung der Horizontkante f.

Das sichtbare Teilstück von e_i wird sowohl in die Datenstruktur des Gesamthorizonts, als auch in die Datenstruktur von H_{max} eingefügt. Abbildung 4.17 zeigt, dass e mit meh-

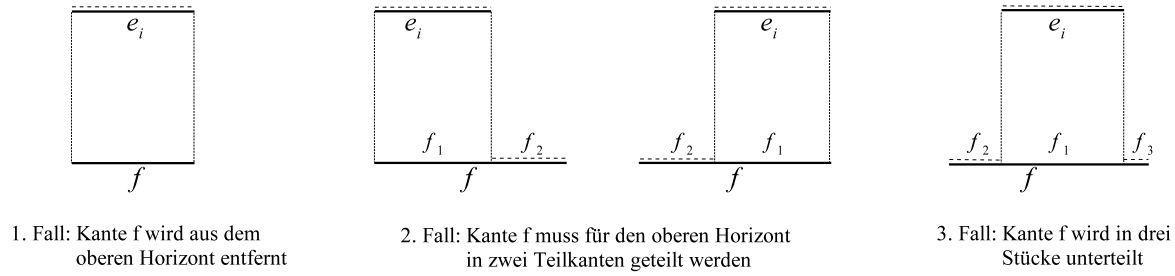

Abbildung 4.16: *Die Kante e hat einen Konflikt mit der Kante f des oberen Horizonts.*

reren Kanten des oberen Horizonts in Konflikt stehen kann und somit für die Kanten mehrere Konflikttests durchgeführt werden müssen.

Abbildung 4.17: *Die Kante e hat einen Konflikt mit mehreren Kanten des oberen Horizonts.*

Besteht ein Konflikt und muss ein Sichtbarkeitstest durchgeführt werden, so ist eine Überprüfung anhand des bisherigen oberen Horizonts ausreichend, da dieser wie in Abschnitt 4.2.1 den gleichen Winkelbereich wie der untere Horizont abdeckt und somit auch den maximalen Winkelbereich des oberen Horizonts abdeckt. Weiterhin wurde die Abarbeitungsreihenfolge der blockierenden Kanten so gewählt, dass die blockierende Kante immer hinter dem oberen Horizont liegt. Dies bedeutet, dass in einem Terrainschnitt der blockierende Punkt oberhalb eines vom Roboter ausgehenden Strahls durch den bisherigen oberen Horizont liegen muss. Liegt der blockierende Punkt unterhalb dieses Strahls, so ist er nicht sichtbar, da er vom bisherigen oberen Horizont verdeckt wird. Nachdem alle blockierenden Kanten in den Horizont eingefügt wurden, ist der Horizont erstellt.

Der in diesem Abschnitt beschriebene Algorithmus wurde lediglich für eine übersichtliche Erklärung in eine Initialisierung und in eine Iteration unterteilt. Da im Initialisierungsschritt die blockierenden Kanten zur Abarbeitungsreihenfolge in der gleichen Reihenfolge hinzugefügt werden, wie sie in den Iterationsschritten abgearbeitet werden, können die Iterationsschritte bereits während der Initialisierung ausgeführt werden. Ein Algorithmus zur Bestimmung des Sichtbarkeitspolyeders kann ebenfalls den Initialisierungsschritt ausnutzen und die Reihenfolge der Facetten in der Schlange U verwenden. Dabei kann der jeweils parallel erzeugte obere Horizont genutzt werden.

4.3 Einfache Strategien

4.3.1 Die Strategie *Straight Up*

Die Grundidee der Online Strategie *Straight Up* basiert auf der Terraineigenschaft, dass keine senkrechten Facetten zugelassen sind und auf Beobachtung 2.2, die besagt, dass der Kern $ker(T)$ konvex und nach oben offen ist. Eine direkte Folgerung aus diesen Eigenschaften ist, dass es ausreichend ist, senkrecht nach oben zu fliegen, um in den Kern zu gelangen. *Straight Up* verfolgt diese einfache Strategie. Die Korrektheit der Strategie ist somit offensichtlich. Wie im Folgenden bewiesen wird, ist die Strategie jedoch nicht kompetitiv. Abbildung 4.18 zeigt dabei die Grundidee des Beweises.

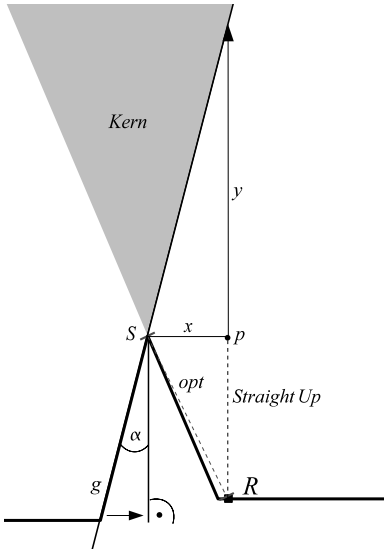

Abbildung 4.18: *Die Online Strategie "Straight Up" ist nicht kompetitiv.*

In der Abbildung wird die rechte Seite des Kerns durch die Gerade g begrenzt. Dies ist die Seite, die der Roboterposition R zugewandt ist. Eine Verkleinerung des Winkels α führt dazu, dass diese noch steiler wird und der Weg, den der Roboter zurücklegen muss, sich somit verlängert. Die Steigung $m(g) = \frac{y}{x}$ der Geraden g wird genutzt, um diesen Sachverhalt zu beschreiben:

$$\lim_{\alpha \to 0} m(g) = \lim_{\alpha \to 0} \left(\frac{y}{x}\right) = \lim_{y \to \infty} \left(\frac{y}{x}\right) = \frac{\lim_{y \to \infty} y}{x}$$

Um kompetitiv zu sein, muss die Online Strategie *Straight Up* bezüglich seiner Weglänge die Formel aus Definition 2.11 erfüllen. Es ergibt sich für die größte mögliche Steigung von g:

$$\nexists C : \lim_{y \to \infty} y + dist(R,p) = C \cdot opt = \sqrt{x^2 + dist(R,p)^2}$$

Dabei wird mit $dist(R,p)$ der euklidische Abstand zwischen R und p bezeichnet. Es wurde $x = dist(p,S)$ gewählt, wobei S der Scheitelpunkt des Berges ist und p der Punkt des Roboterweges auf Höhe von S ist. Da der Grenzwert $\lim_{y \to \infty} y$ gegen unendlich strebt, und damit die Strecke, die der Roboter zurücklegen muss, um in den Kern zu gelangen,

auch gegen unendlich strebt, lässt sich kein linearer Zusammenhang mit der optimalen Lösung des Problems finden. Somit ist bewiesen, dass die Online Strategie *Straight Up* nicht kompetitiv ist.

□

Durch die Bewegung entlang der Z-Achse werden nicht alle drei Freiheitsgrade des Roboters ausgenutzt. Es wird hingegen nur ein Einziger verwendet. Schwierigkeiten bezüglich der Aktualisierung einer Horizontdatenstruktur entlang der Wegstrecke, wie sie in Abschnitt 4.2.2 erläutert wurden, bestehen bei dieser Strategie nicht. Der Entwurf einer geeigneten und effizient zu aktualisierenden Horizontdatenstruktur sollte daher für diese Strategie nicht schwierig sein.

4.3.2 Die Strategie *Go To Next*

Im vorherigen Kapitel wurde bewiesen, dass die Strategie *Straight Up* zwar korrekt arbeitet, jedoch nicht kompetitiv ist. Diese Strategie lässt sich verfeinern, indem der Roboter nicht einfach senkrecht nach oben fliegt, sondern den für ihn sichtbaren Bereich berücksichtigt. In der Strategie *Go To Next* wird der Horizont berechnet und anschließend nacheinander jede Facette hinter einer Kante des oberen Horizonts eingesehen, bis das ganze Terrain sichtbar ist. Im Folgenden wird diese Strategie detailliert beschrieben und seine Korrektheit gezeigt.

Vor jeder Bewegungsentscheidung der Strategie wird der Horizont berechnet. Dann wird zu jeder Kante e des oberen Horizonts die Distanz zu dem Punkt auf e berechnet, der dem Roboter räumlich am nächsten ist. Anschließend steuert der Roboter den so bestimmten Punkt an. Sobald er hinter die Horizontkante e sehen kann, auf der dieser Punkt lag, bestimmt er den jetzt aktuellen Horizont und berechnet wiederum den ihm am nächsten liegenden Punkt auf einer Kante des neuen oberen Horizonts. Dieser Punkt ist das nächste Ziel, welches er ansteuert. Befindet sich der Roboter im Verlauf der Strategie oberhalb des nächsten Zielpunktes, so ist die Z-Komponente des Bewegungsvektor negativ. In diesem Fall wird die Z-Komponente auf null gesetzt, da eine größere Z-Koordinate bei gleichen X- und Y-Koordinaten eine bessere Sicht im Terrain bedeutet und die Wegstrecke zum Zielpunkt im Kern außerdem kürzer ist als bei einem Bewegungsvektor mit negativer Z-Komponente. Die Bewegungskomponenten in X/Y-Richtung bleiben dabei erhalten.

Befolgt der Roboter ausschließlich diese Regeln, so kann es vorkommen, dass er die Sicht auf Teile des Terrain wieder verliert. Aus diesem Grund berechnet der Roboter für alle Teile des Terrains, die zuvor schon einmal sichtbar gewesen sind oder aktuell sichtbar sind, einen vorläufigen Kern ker_{tmp}. Diese Idee wurde bereits in Abschnitt 4.2.3 vorgestellt. Ähnlich zur *CAB* Strategie in Lemma 2.2 gilt auch hier, dass der Roboter sich immer auf dem Rand von ker_{tmp} befindet. Der vorläufige Kern ker_{tmp} darf nicht verlassen werden, um zu garantieren, dass der Roboter oberhalb einer Ebene bleibt, die einer bereits gesehenen Facette zugehörig ist. Dadurch kann die bereits bestehende Sicht nicht wieder zu stark eingeschränkt werden. Führt der Bewegungsvektor zum nächsten Zielpunkt aus ker_{tmp} heraus, so wird die Bewegungsrichtung in X- und Y-Richtung beibehalten, die Z-Komponente wird dabei jedoch so angepasst, dass der Roboter an dem Rand von ker_{tmp} nach oben gleitet, bis er hinter die entsprechende Horizontkante e sehen kann. Hierbei kann es auch vorkommen, dass die Z-Komponente mehrfach angepasst werden muss und der Roboter entlang unterschiedlicher Facetten des vorläufigen Kerns entlang gleitet. Die Sequenz bestehend aus der Berechnung des Horizont und des vorläufigen Kerns und der

Bewegung auf den Zielpunkt der Horizontkante zu, wird solange wiederholt, bis der Roboter den Zielpunkt erreicht hat.

Die Online Strategie *Go To Next* arbeitet korrekt, da an den Roboter die Bedingung gestellt wird, ker_{tmp} nicht zu verlassen und der Roboter seine Sichtbarkeit durch das Ansteuern der Kanten des oberen Horizonts beständig erweitert. Dadurch wird der Kern auf jeden Fall erreicht. Die Erweiterung der Sicht erfolgt zumindest immer dann, wenn der Roboter hinter eine weitere Kante des oberen Horizonts schauen kann. Der vorläufige Kern ker_{tmp} wird dann durch mindestens eine weitere Ebene eingeschränkt. Sobald der Roboter hinter alle Horizontkanten sehen kann und auch keine weiteren Horizontkanten durch die Bewegung sichtbar wurden, sieht er das komplette Terrain.

Abbildung 4.19 veranschaulicht in einem zweidimensionalen Terrain die Arbeitsweise von *Go To Next*. Der Roboter findet nach vier Schritten in den Kern. In der Grafik sind die vorläufigen Kerne $ker^i_{tmp}(T)$ in verschiedenen Grautönen eingezeichnet, die er nach dem $i-$ten Schritt berechnet. Es gilt:

$$ker^0_{tmp}(T) \supseteq ker^1_{tmp}(T) \supseteq ker^2_{tmp}(T) \supseteq ker^3_{tmp}(T) \supseteq ker(T)$$

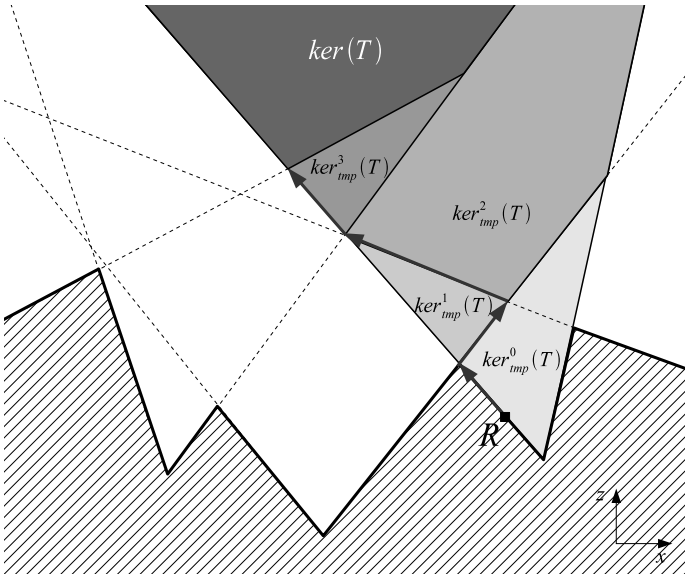

Abbildung 4.19: *Die Online Strategie "Go To Next".*

Nach jedem Schritt sieht der Roboter hinter eine weitere Horizontkante und erweitert damit die Sicht auf mindestens eine weitere Ebene, die den vorläufigen Kern weiter begrenzt. Der vorläufige Kern schrumpft in Analogie zu Lemma 2.3 immer weiter auf den endgültigen Kern $ker(T)$ zusammen.

Die Strategie *Go To Next* ist nicht kompetitiv, ebenso wenig wie die Verfeinerung dieser Strategie, die im nächsten Abschnitt erläutert wird. Der Beweis dazu wird deshalb am Ende des nächsten Abschnittes für beide Strategien zusammen beschrieben.

4.3.3 Die Strategie *Weighted Angles*

In diesem Abschnitt wird die Strategie *Weighted Angles* vorgestellt. Sie basiert auf derselben Idee, wie die Strategie *Go To Next* und versucht ebenfalls hinter die Kanten des oberen

Horizonts zu schauen. Während *Go To Next* immer die nächste Horizontkante nimmt, versucht *Weighted Angles* alle Kanten des oberen Horizonts in die Bewegungsentscheidung zu integrieren. Dieses Vorgehen soll etwaige Fehlentscheidungen durch die Auswahl einer einzigen Kante vermeiden. *Weighted Angles* gewichtet die Kanten des oberen Horizonts anhand ihres Abstandes zum Roboter und ihrem Winkelbereich, wie er in Definition 4.10 definiert wurde. Der Roboter fliegt dadurch in den aktuellen Sichtbarkeitskegel hinein. Die Idee, die Kanten auf diese Art und Weise zu gewichten, ist in Anlehnung an das Konzept des Bisektors in der *CAB* Strategie entstanden. Ein Ziel dieser Gewichtung ist, dass die resultierende Wegstrecke der Strategie geglättet wird, indem starke Richtungswechsel, wie sie bei *Go To Next* auftreten können, vermieden werden. In den folgenden Abschnitten wird die Strategie beschrieben, ihre Korrektheit gezeigt und nachgewiesen, dass sie nicht kompetitiv ist.

Wie auch bei *Go To Next* wird zunächst der Horizont zur aktuellen Roboterposition im Terrain berechnet. Daraufhin wird auf jeder Kante h_i des Horizonts der Punkt bestimmt, der den kleinsten euklidischen Abstand zur Roboterposition R hat. Die Länge des der Horizontkante h_i zugehörigen Richtungsvektors v_i entspricht dabei dem zuvor bestimmten Abstand zwischen Roboter und Horizontkante. Jeder Vektor v_i gibt also die Richtung an, in der die zugehörige Kante h_i liegt und die Länge des Weges dahin, wie in Abbildung 4.20 dargestellt. Die Richtungsvektoren v_i werden mit dem Winkelbereich α_i der Horizontkante h_i gewichtet:

$$\widehat{v}_i = \frac{\alpha_i}{2\pi} \cdot v_i$$

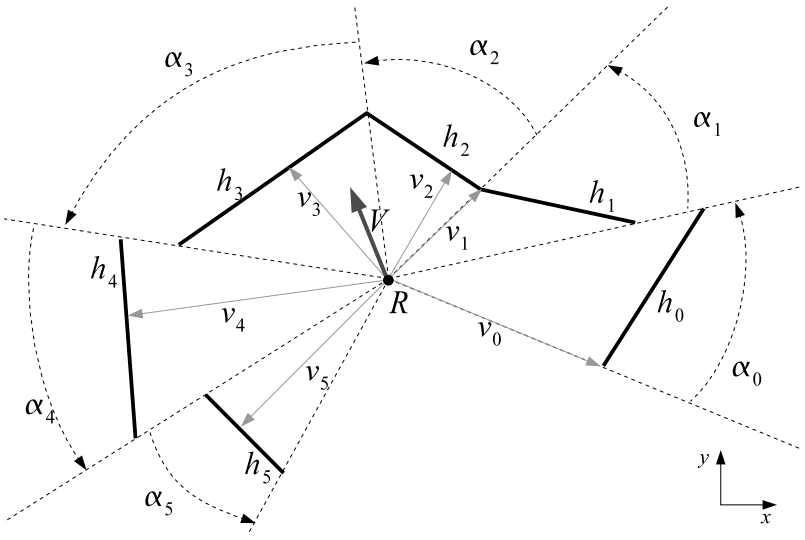

Abbildung 4.20: *Die Online Strategie "Weighted Angles".*

Zeigt die Z-Komponente eines $\widehat{v}_i$ in negative Z-Richtung, so wird diese Komponente mit der gleichen Begründung wie bei *Go To Next* auf null gesetzt. Der endgültige Bewegungsvektor V wird anschließend als Summe der $\widehat{v}_i$ berechnet.

$$V = \sum \widehat{v}_i = \sum \frac{\alpha_i}{2\pi} \cdot v_i$$

Der Bewegungsvektor V wird daraufhin solange zur Bewegung genutzt, bis der Roboter hinter eine der oberen Horizontkanten h_i sehen kann. Dabei wird der Spezialfall, dass V der Nullvektor ist, so behandelt, dass V auf $(0, 0, 1)$ gesetzt wird und der Roboter somit senkrecht nach oben fliegt. Außerdem wird die Einschränkung der Bewegung auf einen vorläufigen Kern ker_{tmp} genutzt, wie sie bei der vorher beschriebenen Strategie *Go To Next* eingesetzt wurde. Der Bewegungsvektor V wird also gegebenenfalls so korrigiert, dass der Roboter am Rand des Kerns nach oben gleitet. Sobald der Roboter hinter eine der oberen Horizontkanten sehen kann, wird der Horizont neu berechnet und ein neuer Bewegungsvektor V wird bestimmt. Dies wird solange wiederholt, bis der Roboter hinter alle Kanten des oberen Horizonts sehen kann und damit das ganze Terrain einsieht. Ist dies erfüllt, so wurde der Kern erreicht.

Die Korrektheit von *Weighted Angles* ist aufgrund der Verwendung der Einschränkung der Bewegung auf die vorläufigen Kerne und der Erweiterung der Sichtbarkeit gegeben. In Bezug auf die Kompetitivität wird im Folgenden gezeigt, dass weder die Strategie *Go To Next* noch die Strategie *Weighted Angles* kompetitiv ist. Abbildung 4.21 zeigt ein zweidimensionales Terrain, in dem sich der Roboter auf gleicher Höhe mit der letzten Horizontkante e befindet, hinter die er noch nicht gesehen hat. Der Roboter steht am Rand des Terrains, der nach Definition nicht zum Horizont gehört. Bei beiden Strategien wird eine Bewegung in Richtung der Kante e erfolgen. In dem Beispiel der Abbildung kann der Roboter den Bereich hinter der Kante e erst dann einsehen, wenn er diese erreicht hat. Der kürzeste Weg in den Kern ist jedoch ein Lot auf den Kern, in der Abbildung mit d_{opt} bezeichnet. Wird das Szenarium so angepasst, dass die nicht sichtbare Ebene hinter e flacher wird, so wird der optimale Weg kürzer. Eine solche Anpassung des Szenariums entspricht einer Verkleinerung des Winkels α. Die Wegstrecke d_w von der Roboterposition R zur Kante e verändert sich bei der Anpassung von α jedoch nicht. Für eine Grenzwertbetrachtung folgt dadurch:

$$\lim_{\alpha \to 0}(d_{opt}) = 0 \Longrightarrow \lim_{\alpha \to 0}\left(\frac{d_w}{d_{opt}}\right) \to \infty$$

□

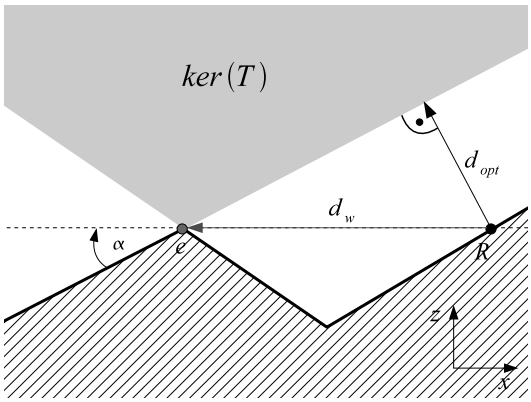

Abbildung 4.21: *Weighted Angles und Go To Next sind nicht kompetitiv.*

Es kann somit für beide Strategien kein kompetitiver Faktor angegeben werden. In dieser Strategie werden Vektoren verwendet, die auf den dem Roboter am nächsten Punkt

einer Kante zeigen. Es können aber auch alternativ zu jeder Kante die Vektoren verwendet werden, die auf die beiden Endpunkte der Kante zeigen. Aber auch bei dieser Wahl der Vektoren tritt das zuvor beschriebene Problem auf.

4.4 Komplexe Strategien

In diesem Abschnitt werden zunächst mehrere Ideen vorgestellt, die in komplexere Online Strategien integriert werden können, deren Korrektheit garantieren oder hilfreich für den Entwurf einer kompetitiven Strategie sein könnten, sofern eine solche existiert. Dabei werden auch Erkenntnisse aus den zuvor eingeführten einfachen Strategien diskutiert. Anschließend wird in Abschnitt 4.4.6 eine komplexe Strategie vorgestellt, die viele der beschriebenen Ideen beinhaltet. Abschnitt 4.4.7 diskutiert die Bewegung des Roboters auf Kreisbögen anstatt auf Geraden. Eine Übertragung der *CAB* Strategie auf das dreidimensionale Terrain wird in Abschnitt 4.4.8 behandelt. In Abschnitt 4.4.9 dann werden Ideen zur Kompetitivität einer Online Strategie zur Kernsuche entwickelt und es wird ein möglicher Ansatz für einen Beweis skizziert, dass es eine kompetitive Strategie gibt.

4.4.1 Gewichtung von Bewegungsrichtungen anhand von Lot-Vektoren

In Abschnitt 4.3.3 wurde eine Strategie vorgeschlagen, die für jede Kante des oberen Horizonts einen Bewegungsvektor bestimmt und diese dann durch eine Gewichtung zu einem resultierenden Bewegungsvektor zusammenfasst. Dieses Prinzip wurde in Abbildung 4.22 veranschaulicht. In diesem Beispiel zeigt der Vektor v_0 auf einem Endpunkt der Kante h_0. Anhand dieses Beispiels soll eine Alternative für die Bestimmung der Vektoren v_i vorgestellt werden.

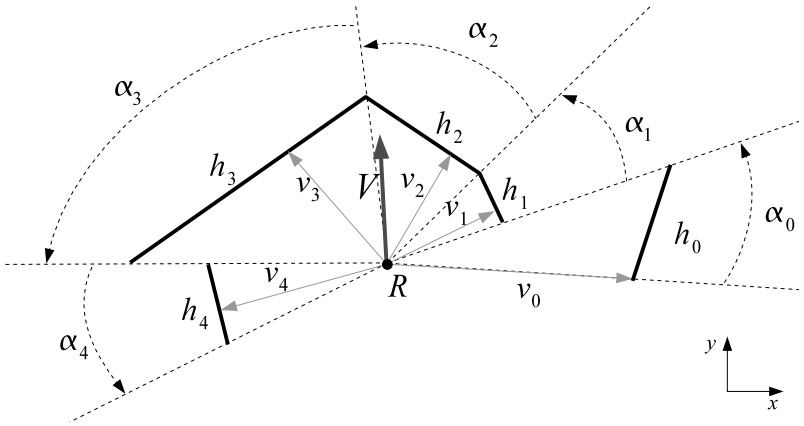

Abbildung 4.22: *Die Strategie Weighted Angles.*

Eine Bewegung in Richtung von v_0 scheint nicht optimal zu sein, da eine Bewegung in Richtung des Lots auf die Gerade, die die Kante h_0 verlängert, eine kürzere Wegstrecke und somit einen schnelleren Einblick in den Bereich hinter h_0 erlaubt. Daher wird in dieser alternativen Strategie der zugehörige Lotvektor als Bewegungsvektor v_0 gewählt. Eine Gewichtung anhand des Winkelbereichs, den h_0 abdeckt, kann auch bei dieser alternativen

Bestimmung von v_0 verwendet werden, da auch hier ein Gewinn für den Sichtbarkeitsbereich bei Einblick hinter die Horizontkante existiert. Es bleibt jedoch zu überprüfen, ob eine solche Gewichtung eine günstigere Wegstrecke zur Folge hat.

Eine weitere wichtige Eigenschaft des Lotvektors ist, dass dieser zusammen mit der zugehörigen Horizontkante eine Sichtbarkeitsebene definiert. Bei der Bewegung des Roboters gibt der Lotvektor die Drehrichtung der Sichtbarkeitsebene an. Falls die Horizontkante waagerecht verläuft, wird diese Drehrichtung durch den Winkel des Lotvektors zur Z-Achse beschrieben. Für schräg verlaufende Horizontkanten wird die Drehrichtung analog definiert.

4.4.2 Korrektheit durch vorläufige Kerne

Ein zentraler Aspekt, der auch bei komplexeren Strategien gewährleistet sein muss, ist die Korrektheit der Strategie. Um diese Korrektheit zu gewährleisten, ist das bereits in Abschnitt 4.2.3 vorgestellte Verfahren geeignet, in dem die Bewegung des Roboters auf den vorläufigen Kern ker_{tmp} beschränkt wird. Dies garantiert, dass der Sichtbarkeitsbereich des Roboters nicht wieder zu stark eingeschränkt wird. Ein einmal sichtbarer Bereich bleibt dabei entweder sichtbar oder der Roboter merkt sich, dass eine Facette des Terrains schon einmal gesehen wurde und verwendet diese auch bei der Berechnung von späteren vorläufigen Kernen. Erweitert die Strategie zusätzlich den Sichtbarkeitsbereich durch Einblicke hinter weitere Horizontkanten, so entsteht eine Folge von vorläufigen Kernen, wobei das letzte Folgenglied der Kern des Terrains ist, womit die Korrektheit gegeben ist.

4.4.3 Berücksichtigung des gesamten Horizonts

Eine weitere mögliche Ergänzung zu einer Strategie ist, nicht nur den oberen Horizont zu berücksichtigen, sondern den gesamten Horizont. In Abbildung 4.23 ist ein Beispiel für einen Horizont zu einer Roboterposition R abgebildet. Es sind auch die Lotvektoren v_i auf die Kanten h_i des oberen Horizonts eingezeichnet. Der Lotvektor v_u bildet das Lot zur Kante h_u des unteren Horizonts. Dieser zeigt nicht auf die Kante selbst, sondern auf deren Verlängerung. Der Weg direkt zur Kante wäre länger, als der Weg zur Verlängerung von h_u. Wie im vorigen Punkt bereits erläutert wurde, wäre der direkte Weg zur Kante h_u länger, als das Lot auf die Verlängerung von h_u Um hinter die h_u zu sehen, reicht der Weg entlang v_u somit aus. Würde der untere Horizont, wie in der Abbildung durch die Richtung des Bewegungsvektors V angedeutet, nicht in die Berechnung einfließen, so würde der Roboter diese Richtung nicht berücksichtigen. Das ist insbesondere dann nachteilig, wenn hinter h_u eine steile Ebene befindet, die die Position des Kerns wesentlich beeinflusst.

4.4.4 Minimale Steigung

Eine weitere Möglichkeit für komplexere Strategien könnte die Verwendung einer minimalen Steigung von 45° in Bewegungsrichtung sein. Die Idee hinter diesem Vorschlag besteht darin, dass der Erfolg einer Strategie im Wesentlichen von der richtigen Wahl der Bewegungsrichtung in der X/Y-Ebene abhängt. Eine Steigung, wie sie hier vorgeschlagen wird, führt zu einem zusätzlichen kompetitiven Faktor von maximal $\sqrt{2}$. Diese Idee wird im Folgenden 45° *Kriterium* genannt.

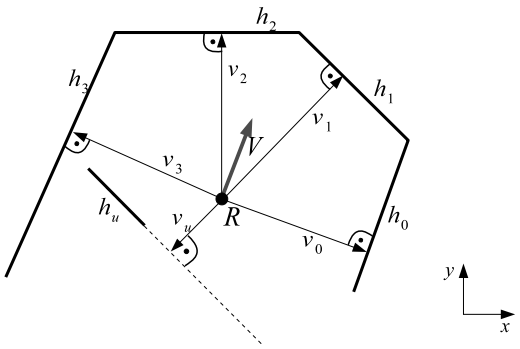

Abbildung 4.23: *Der Roboter bezieht nur den oberen Horizont in die Bewegungsentscheidung mit ein.*

Als Grundlage der *StraightUp* Strategie wurde genutzt, dass sich die Sichtbarkeit bei zunehmender Höhe verbessert. Ein Umkehrschluss daraus ist, dass eine Roboter nicht an Höhe verlieren sollte. Somit sind negative Steigungen für einen Roboter nicht sinnvoll und die Steigung des Roboters in Bewegungsrichtung ist auf das Intervall $[0° : 90°]$ eingeschränkt. Eine Bewegung mit einer Steigung von mindestens 45° ist die Winkelhalbierende dieses Intervalls.

Abbildung 4.24 zeigt, dass damit der Fehler im Vergleich zu einem kürzesten Weg der Steigung 0° maximal $\sqrt{2}$ ist.

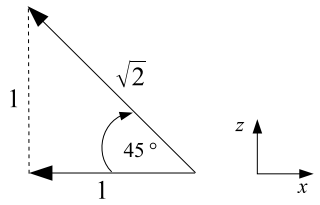

Abbildung 4.24: *Ein Weg mit* 45° *Steigung macht höchstens einen Fehler von* $\sqrt{2}$.

Durch diese Bedingung an den Bewegungsvektor des Roboters wird, wie in Abbildung 4.25 dargestellt, ein Kegel mit einem Innenwinkel von 90° aufgespannt. Dieser Kegel hat immer einen Schnitt mit dem Kern. Dies folgt aus den Beschränkungen an das Terrain, das keine senkrechten Facetten enthalten darf. Die Aufgabe der Kernsuche lässt sich somit auf die Suche nach diesem Schnitt reduzieren. Der Kegel wird stetig an die Position des Roboters angepasst. Bei einer Bewegung ausschließlich in diesem Kegel wird die Bewegung des Roboters zusätzlich von dem aktuellen vorläufigen Kern und vom Terrain beschränkt. In der Abbildung wurde der optimale Weg in den Kern d_{opt} eingezeichnet. Es wird deutlich, dass in diesem Fall der Roboter einen kompetitiven Weg zurücklegt, wenn er sich auf dem linken Rand des Kegels bewegt. Der kompetitive Faktor dieses Weges lässt sich unter Verwendung des Satzes des Pythagoras, wie zuvor schon gezeigt, mit $\sqrt{2}$ abschätzen.

Im Folgenden wird ein weiterer Ansatz beschrieben, der auch Steigungen berücksichtigt, die durch das Terrain bereits vorgegeben sind. Zu jeder Kante h_i des oberen Horizonts kann, wie schon zuvor erläutert, ein Lotvektor v_i bestimmt werden. Dieser Vektor hat die Eigenschaft, dass er in der Sichtbarkeitsebene S_{R,h_i} liegt. Es wird nun der Vektor w_i

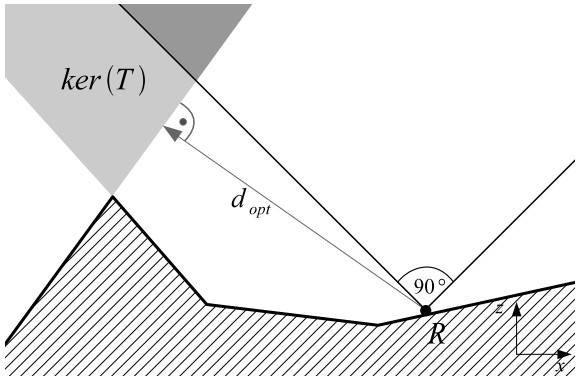

Abbildung 4.25: *Die Bewegung des Roboters verläuft innerhalb des Kegels.*

bestimmt, der die Winkelhalbierende der Vektoren v_i und $e_z = (0, 0, 1)$ darstellt. Dies illustriert Abbildung 4.26. Falls die Z-Koordinate eines v_i kleiner als null ist, wird sie auf null gesetzt. In diesem Fall wird das zugehörige w_i eine Steigung von 45° erhalten, was somit dem 45° Kriterium entspricht. Anschließend wird die Summe über alle w_i berechnet:

$$\vec{W} = \sum \vec{w}_i, \text{ mit den Winkelhalbierenden } \vec{w}_i$$

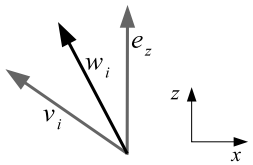

Abbildung 4.26: *Der Vektor w_i ist die Winkelhalbierende der Vektoren e_z und v_i.*

Durch dieses Vorgehen soll erreicht werden, dass der Roboter schneller an Höhe gewinnt und somit auch schneller einen besseren Überblick über das Terrain hat. Dieser Ansatz wird im Folgenden als *Winkelhalbierenden Kriterium* bezeichnet.

4.4.5 Begrenztes *Straight Up*

Wie in Abschnitt 4.3.1 gezeigt wurde, ist eine reine *Straight Up* Strategie nicht kompetitiv. Es wird nun jedoch gezeigt, dass es möglich ist, eine Strategie in mehrere Phasen einzuteilen, wobei in einer oder mehrerer dieser Phasen die Strategie *Straight Up* verfolgt wird, und dennoch ein kompetitive Strategie, sofern überhaupt möglich, entstehen kann. Im Folgenden werden Kriterien angegeben, unter welchen Bedingungen die Strategie *Straight Up* verfolgt werden kann und dass sich dann dadurch der kompetitiver Faktor einer kombinierten Strategie um maximal eins erhöht.

In Abbildung 4.27 sind zwei Horizontkarten abgebildet. Die linke Karte zeigt ein Szenarium, in dem der Horizont und damit auch der Sichtbarkeitskegel nicht radial geschlossen sind. In diesem Fall ist eine Vorzugsrichtung für die Bewegung nach links hin gegeben. Dies lässt sich anhand des Winkels α zwischen zwei benachbarten Lotvektoren v_i erkennen, der größer als 180° ist. Der Rand des Sichtbarkeitspolyeders wird neben dem bereits

sichtbaren Terrain nur durch die Horizontkanten h_0, h_1 und h_2 und deren zugehörigen Sichtbarkeitsdreiecke definiert. Es ist also davon auszugehen, dass eine Bewegung in die Vorzugsrichtung sinnvoll ist, um die Sicht zu vergrößern. In diesem Fall wird davon gesprochen, dass der Horizont *nicht vollständig beschränkend* ist. Der rechte Teil der Abbildung zeigt eine zweite Horizontkarte, in der kein Winkel zwischen jeweils zwei benachbarten v_i größer als 180° ist. In diesem Fall lässt sich keine Vorzugsrichtung finden. Der Horizont wird dann als *vollständig beschränkend* bezeichnet.

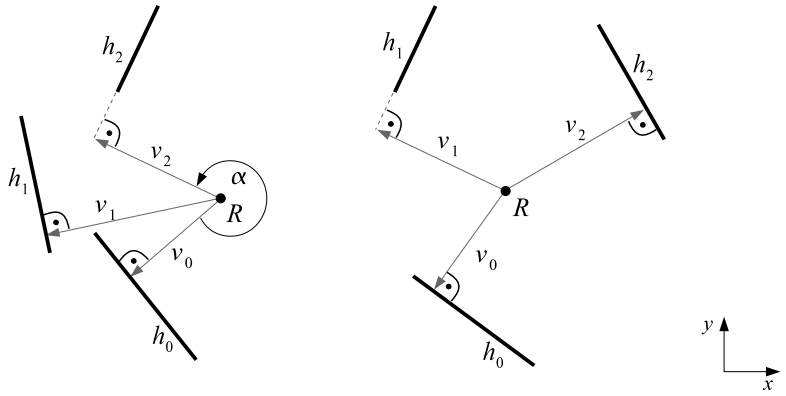

Abbildung 4.27: *Die linke Horizontkarte hat eine Vorzugsrichtung, die rechte nicht.*

Die beiden folgenden Definitionen fassen diese Begriffe genauer. Zunächst wird dabei der Begriff der Sichtbarkeitshalbebene eingeführt:

> **Definition 4.11:** Sei T ein Terrain, R eine Roboterposition und H der aktuelle Horizont. Ferner sei E die Menge der Sichtbarkeitsebenen zu R und den Horizontkanten aus H. Sei $h_i \in H$, $E_i \in E$ die zugehörige Sichtbarkeitsebene und v_i der Lotvektor von R auf h_i. Der Lotvektor v_i ist also auch ein Richtungsvektor von E_i. Die Sichtbarkeitshalbebene L_i sei definiert als Teilmenge von E_i eingeschränkt auf positive v_i Richtung von R aus. Sei L die Menge aller Sichtbarkeitshalbebenen L_i.

Basierend auf der obigen Definition wird nun bestimmt, was ein vollständig beschränkender Horizont ist:

> **Definition 4.12:** Sei T ein Terrain, R eine Roboterposition, H der aktuelle Horizont und L die Menge der Sichtbarkeitshalbebenen. Sei *HS* der Halbraumschnitt, der durch die Sichtbarkeitshalbebenen gegeben ist. Dann ist K die Begrenzung von *HS*, eingeschränkt auf den Bereich, dessen Bestandteile auch Teile von Halbebenen aus L sind. Ist die Begrenzung K für jede Blickrichtung von R aus definiert, so wird der Horizont als *vollständig beschränkend* bezeichnet.

Ist der Horizont vollständig beschränkend, so kann eine Phase erfolgen, in der *Straight Up* angewendet wird, wobei die gesamte Weglänge in den Kern kompetitiv ist, sofern der

Rest der Wegstrecke dies nicht verhindert. Der folgende Beweis stützt sich im Wesentlichen auf die folgende Beobachtung:

> **Beobachtung 4.3:** Sei T ein Terrain, R eine Roboterposition und H der aktuelle Horizont. Solange H vollständig beschränkend ist, so ist die Z-Koordinate von R kleiner oder gleich der Z-Koordinate eines beliebigen Punktes aus dem Kern von T.

Für den Beweis der Beobachtung werden Eigenschaften des Horizonts betrachtet. Der Horizont ist vollständig beschränkend. Also existiert für jede Blickrichtung des Roboters eine Sichtbarkeitshalbebene L_i. Abbildung 4.28 zeigt, wie die L_i beim Aufsteigen des Roboters zuerst sehr steil sind und dann nach und nach auf die hinter den Horizontkanten anliegenden Facetten F_i geklappt werden.

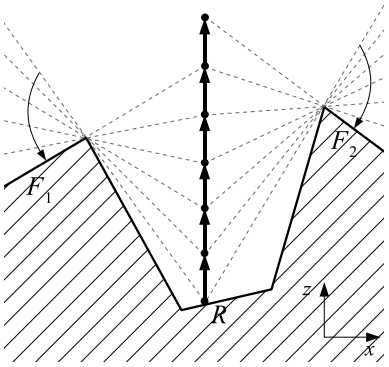

Abbildung 4.28: *Kontinuierliche Anpassung der Sichtbarkeitsebenen.*

Durch die Ebenen, die zu den F_i gehören, wird ein Polyeder definiert. Solange sich der Roboter innerhalb dieses Polyeders befindet, ist sein Horizont vollständig beschränkend. Dieses Polyeder wurde in Abbildung 4.29 hellgrau eingezeichnet. Der Roboter erreicht den Rand des Polyeders, sobald eine seiner Sichtbarkeitshalbebenen L_i auf eine Facette F_i geklappt wurde. Dadurch muss der Horizont neu berechnet werden und es wird überprüft, ob der Horizont noch vollständig beschränkend ist.

Solange der Horizont des Roboters voll beschränkend ist, gilt, dass der Roboter unterhalb der Ebenen der zuvor erwähnten F_i ist. Es wird nun der untere Halbraumschnitt HI_u aller F_i berechnet. Der untere Halbraum einer Ebene ist der Halbraum, in den der Normalenvektor der Ebene nicht zeigt. Analog dazu ist der obere Halbraumschnitt HI_o definiert. In HI_u ist der Roboter enthalten, während HI_o den Kern enthält. Daraus folgt, dass der Roboter unterhalb des Kerns ist, also dass seine Z-Koordinate kleiner oder gleich der Z-Koordinate des untersten Punktes des Kerns ist.

□

Wird die Strategie *Straight Up* ausschließlich in Situationen eingesetzt, in denen die Beobachtung 4.3 erfüllt ist, so liegt die Endposition des Roboters unterhalb des Kerns oder auf der Höhe des untersten Kernpunktes. Die Weglänge einer optimalen Strategie, um in den Kern zu gelangen, muss damit länger sein, als die Teilstrecke, die mit der *Straight Up* Strategie zurückgelegt wurde. Somit wird der kompetitive Faktor um nicht mehr als eins erhöht. Dieses Prinzip wird aus Abbildung 4.30 deutlich.

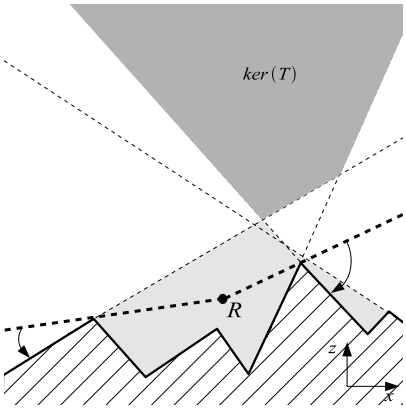

Abbildung 4.29: *P ist ein Punkt des Kerns mit kleinster Z-Koordinate*

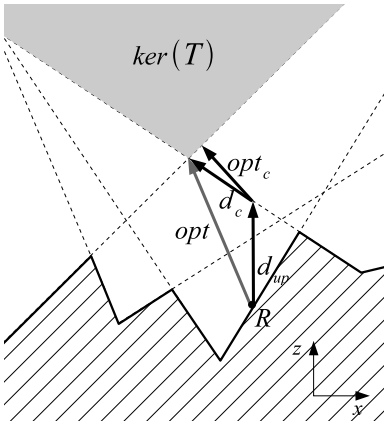

Abbildung 4.30: *Begrenztes Straight Up.*

In dem in der Abbildung gezeigten Beispiel wird die Strategie des *begrenzten Straight Up* eingesetzt. Der Roboter fliegt solange senkrecht nach oben, bis der Horizont nicht mehr vollständig beschränkend ist, da auf der rechten Seite keine Horizontkante mehr existiert. Anschließend kann der Roboter eine Phase beginnen, in der eine andere Strategie eingesetzt wird. Es wird angenommen, dass eine solche Strategie kompetitiv ist und einen kompetitiven Faktor von C besitzt. Würde ausschließlich diese kompetitive Strategie eingesetzt, so wäre die Weglänge durch $C \cdot opt$ nach oben hin begrenzt. Sei d_{up} die Summe der Wegstücke, die mit *begrenztem Straight Up* zurückgelegt wird. So gilt $d_{up} \leq opt$. Ferner sei X die Summe der Wegstücke, die mit der kompetitiven Strategie zurückgelegt wurden. Diese Summe ist durch $C \cdot opt_c$ nach oben hin begrenzt, wobei opt_c der optimale Weg für die restlichen Stücke in den Kern ist, in denen das *begrenzte Straight Up* nicht angewendet wird. Für opt_c gilt $opt_c \leq opt$. Dies lässt sich anhand der Z-Koordinaten der einzelnen Wegstrecken nachvollziehen, da *begrenztes Straight Up* die Z-Koordinate verändert, jedoch immer unterhalb des Kerns bleibt, bzw. höchstens die Höhe des untersten Punktes des Kerns erreicht. Daher gilt:

$$\Delta Z_{opt} = d_{up} + \Delta Z_{d_c} \Rightarrow \Delta Z_{opt} \geq \Delta Z_{d_c} \Rightarrow opt \geq d_c \Rightarrow opt \geq opt_c$$

Für den gesamten Weg π einer kompetitiven Online Strategie, die ein *begrenztes Straight Up* einsetzt, ergibt sich:

$$\pi = d_{up} + X \leq opt + X \leq opt + c \cdot opt_c \leq opt + c \cdot opt = (c+1) \cdot opt$$

Der kompetitive Faktor wird insgesamt also um nicht mehr als eins erhöht.

□

Daraus folgt, dass es Situationen gibt, in denen eine *Straight Up* Phase nicht schlecht ist und nicht verhindert, dass die gesamte Strategie nicht mehr kompetitiv ist. Es bleibt zu überprüfen, ob eine Strategie tatsächlich einen höheren kompetitiven Faktor durch den Einsatz von *begrenztem Straight Up* erhält und wie viel größer dieser gegebenenfalls ist. Die hier bewiesene Erhöhung ist lediglich eine obere Grenze.

4.4.6 Die Strategie *Weighted Perpendiculars*

In diesem Abschnitt wird eine weitere Online Strategie vorgestellt, die sich auf die Beobachtungen stützt, die in den vorherigen Abschnitten erläutert wurden. Die Strategie ist in zwei Phasen aufgeteilt. In der ersten Phase wird die *Straight Up* Strategie verfolgt und es ist keine Vorzugsrichtung gegeben. In der zweiten Phase, in der die Strategie meint, eine gute Vorzugsrichtung erkannt zu haben, wird die Strategie der gewichteten Lotvektoren aller Horizontkanten verwendet. Als Erweiterung wird zusätzlich das 45° *Kriterium* gegebenenfalls in Kombination mit dem Winkelhalbierenden Kriterium gewählt, um das Problem mit der Kompetitivität, welches bei *Weighted Angles* beobachtet wurde, zu vermeiden. Im Folgenden werden die Phasen genauer erläutert.

Die Unterscheidung zwischen den Phasen wird anhand der Existenz einer Vorzugsrichtung getroffen. Falls der Winkel zwischen je zwei benachbarten Lotvektoren kleiner ist als 180°, so existiert keine Vorzugsrichtung und der Roboter R beginnt eine *Straight Up* Phase, die abbricht, sobald eine Vorzugsrichtung vorhanden ist. Dabei werden die Lotvektoren aller Kanten des Horizonts einbezogen und nicht nur die des oberen Horizonts. Durch eine solche Phase wird, wie zuvor beschrieben, der kompetitive Faktor um höchstens eins erhöht.

Der Roboter wechselt in die andere Phase, sobald eine Vorzugsrichtung in der X/Y-Ebene für den Bewegungsvektor u vorhanden ist. Eine Vorzugsrichtung ist nur dann gegeben, wenn ein Winkel α in der X/Y-Ebene zwischen zwei benachbarten Lotvektoren vorhanden ist, der größer als 180° ist. Eine Eigenschaft eines solchen Lotvektors ist, dass er in der Sichtbarkeitsebene der zugehörigen Horizontkante liegt, jedoch nicht in deren Sichtbarkeitsdreieck. Die Strategie bildet die Summe V über die Lotvektoren, gewichtet anhand der Winkelbereiche ihrer zugehörigen Horizontkanten. Dabei wird die Z-Koordinate eines Lotvektors v_i auf null gesetzt, falls diese kleiner als null war. Der Lotvektor wird also zumindest waagerecht verlaufen. Auf den Ergebnisvektor V der Summe der gewichteten Lotvektoren wird entweder das 45° *Kriterium* oder das *Winkelhalbierenden Kriterium* angewendet. Gleichzeitig gilt, wie schon bei anderen in dieser Arbeit beschriebenen Online Strategien, dass der vorläufige Kern ker_{tmp} nicht verlassen werden darf.

Die Strategie *Weighted Perpendiculars* arbeitet korrekt. Das folgt aus den zuvor erläuterten Argumenten. Obwohl diese Strategie in vielen Fällen ein besseres Ergebnis liefern dürfte, als die zuvor beschriebenen Strategien, ist sie nicht kompetitiv. Abbildung 4.31 zeigt ein Terrainbeispiel, in dem der Roboter von seiner Startposition R fast das ganze

Terrain einsieht. Er sieht nur die Facette F_1 hinter der Horizontkante e noch nicht. In einem solchen Fall wird entweder das 45° Kriterium angewendet, oder es wird der Vektor w berechnet, der den Winkel zwischen der Senkrechten in R und dem Lotvektor auf die Kante e halbiert. Der Weg des Roboters verläuft solange entlang dieses Vektors, bis er die Facette F_1 einsieht. Dies ist der Fall, sobald der Roboter die zur Facette F_1 zugehörige Ebene E_{F_1} berührt. Es wird im Folgenden erläutert, wie der Weg des Roboters beliebig verlängert werden kann, während der optimale Weg in den Kern gleich bleibt. Die Kante e wird in der Ebene E_{F_1} nach oben verschoben, wie in der Abbildung durch die hellgrau eingezeichnete Kante e'' veranschaulicht. Dabei ändert sich die Steigung der Facette F_2, während die Steigung der Facette F_1 erhalten bleibt. Um die Kante e noch weiter in der Ebene E_{F_1} nach oben zu verschieben, kann zusätzlich auch noch die Facette F_3 angepasst werden, wie dies durch die Position der Kante e' in der Abbildung angedeutet wird. Durch diese Verschiebung von e wird der Vektor w, der die Bewegungsrichtung des Roboters angibt, immer weniger steil und der Weg des Roboters wird immer länger. Da jedoch E_{F_1} gleich bleibt, verändert sich dabei der optimale Weg in den Kern nicht. Daraus folgt, dass die Online Strategie *Weighted Perpendiculars* nicht kompetitiv ist.

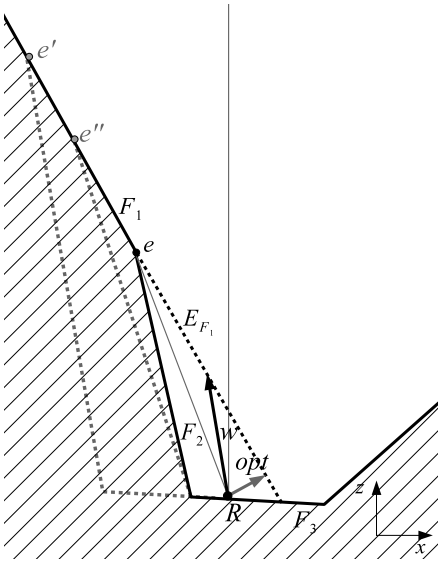

Abbildung 4.31: *Die Strategie Weighted Perpendiculars ist nicht kompetitiv.*

4.4.7 Bewegung auf Kreisbögen

In [15] werden unter anderem einfache Online Strategien im zweidimensionalen Raum vorgestellt, bei der ein Roboter R an einer Wand startet und die Aufgabe hat, hinter eine an dieser Wand liegende Ecke c zu sehen. Der Roboter bewegt sich dazu auf einem Kreisbogen. Eine mögliche Wahl für den Mittelpunkt dieses Kreisbogens ist dabei der Mittelpunkt der Strecke (R, c) vom Roboter zur Ecke. Es wird gezeigt, dass die Online Strategie in diesem Fall kompetitiv mit dem Faktor $\frac{\pi}{2}$ ist. Übertragen auf das dreidimensionale Terrain bewegt sich der Roboter R auf einem Kreisbogen auf eine Horizontkante h zu und nicht wie zuvor auf einer Geraden.

Im Folgenden wird zunächst eine mögliche Wahl für den Mittelpunkt eines solchen Kreisbogens vorgestellt und beschrieben, wie eine einfache Strategie, in der sich der Roboter auf Kreisbögen bewegt, aussehen könnte. Anschließend wird die Wahl der Lage und des Mittelpunkts eines solchen Kreisbogens diskutiert. Abbildung 4.32 zeigt das Terrainbeispiel, mit dem gezeigt wurde, dass *Go To Next* und *WeightedAngles* nicht kompetitiv sind. Es wird deutlich, dass der Fehler der beiden zuvor genannten Strategien mit der Bewegung auf einem Kreisbogen vermieden wird. Dabei wurde der Kreisbogen wie folgt bestimmt. Sei $\vec{v}$ der Vektor, der von R aus auf den R am nächsten liegenden Punkt von h zeigt. Dann ist der Mittelpunkt des Kreisbogens genau der Endpunkt des Vektors $\vec{v}_m = \frac{1}{2}\vec{v}$. Die Verwendung des hier beschriebenen Kreisbogens vermeidet auch den Fehler der *Weighted Perpendiculars* Strategie.

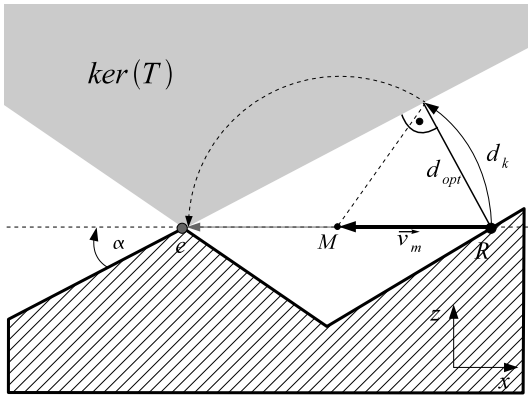

Abbildung 4.32: *Bewegung des Roboters auf Kurven.*

Wenn allerdings der Horizont radial geschlossen ist oder der Horizont aus mehr als einer Kante besteht, existiert für jede Horizontkante ein solcher Kreisbogen. Es stellt sich die Frage, auf welche Art und Weise über diese Kreisbögen gemittelt werden soll, um die Bewegungsentscheidung für den nächsten Schritt zu treffen. Eine Möglichkeit wäre, eine modifizierte *Go To Next* Strategie auszuführen, bei der der Roboter sich nicht auf Geraden zur nächsten Horizontkante bewegt, sondern auf einem Kreisbogen. Da eine solche Strategie auch die Regel enthält, dass der Roboter den vorläufigen Kern ker_{tmp} nicht verlassen darf, arbeitet eine solche modifizierte *Go To Next* Strategie korrekt.

In den einfachen Strategien aus [15] wird deutlich, dass die Wahl des Mittelpunkts des Kreisbogens Einfluss hat auf den kompetitiven Faktor der Strategie. Der Mittelpunkt des Kreisbogens bestimmt auch den Zielpunkt des Kreisbogens, in dem dieser endet. Im Folgenden wird nun eine Möglichkeit diskutiert, mit deren Hilfe ein Kreisbogen konstruiert werden kann, der mehrere Horizontkanten mit einbezieht und nicht, wie oben beschrieben, jede Horizontkante einzeln einsieht. Abbildung 4.33 zeigt ein Terrainbeispiel, in dem der Roboter von seiner Position R aus drei Horizontkanten sieht. Der Winkel α zwischen den benachbarten Lotvektoren auf die Kanten h_1 und h_3 ist größer als 180°. Somit ist, wie in Abschnitt 4.4.5 beschrieben, eine Vorzugsrichtung gegeben durch die Winkelhalbierende g des zu α inversen Winkels. Diese Winkelhalbierende g schneidet die Horizontkanten, bzw. deren Verlängerungen in den drei grau eingezeichneten Schnittpunkten S_1, S_2 und S_3. Dabei ist zu beachten, dass diese Schnittpunkte im Allgemeinen nicht auf einer Geraden liegen. Durch die verschiedenen Schräglagen der Horizontkanten wird es in den meisten

Fällen notwendig sein, einen von R ausgehenden Strahl s mit dem gleichen Winkel in der X/Y-Ebene wie g durch jeden dieser Schnittpunkte zu führen. Es kann nun einer der drei Schnittpunkte als Zielpunkt des resultierenden Kreisbogens gewählt werden. Ob dabei nun die Wahl auf den Schnittpunkt S_i mit der größten Höhe oder auf den Schnittpunkt S_i mit der größten Entfernung von R fällt, kann in zukünftigen Arbeiten untersucht werden.

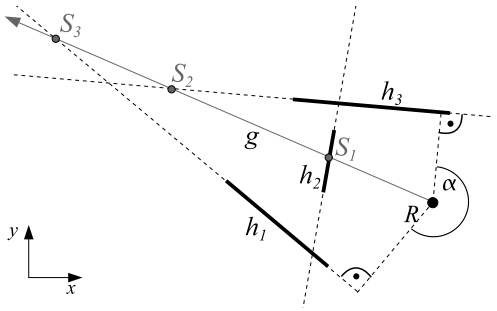

Abbildung 4.33: *Die Winkelhalbierende des zu α inversen Winkels schneidet die Horizontkanten bzw. deren Verlängerungen.*

Zusätzlich zur Wahl des Mittelpunkts ist hier noch die Wahl der Lage des Kreisbogens im dreidimensionalen Raum wichtig. Mit der Lage des Kreisbogens ist der Winkel der Ebene, in der der Kreisbogen liegt, zur Z-Achse gemeint. Besteht der Horizont nur noch aus einer Kante, oder werden die Horizontkanten nacheinander angesteuert, wie in der zuvor vorgestellten modifizierten *GoToNext* Strategie, dann kann der Kreisbogen so gewählt werden, dass er in der Ebene liegt, die senkrecht zur Sichtbarkeitsebene der Horizontkante steht und entlang des Lotvektors dieser Horizontkante verläuft. Weiterführende Arbeiten könnten sich z. B. mit der Wahl einer geeigneten Lage des Kreisbogens beschäftigen. Dabei ist die gleichzeitige Berücksichtigung mehrerer Horizontkanten besonders interessant.

Eine mögliche Verbesserung dieser Strategie kann darin bestehen, für den Einblick hinter eine Horizontkante nicht über die ganze Wegstrecke einen Kreisbogen zu verwenden, sondern zuerst geradlinig auf den Punkt M auf dem Kreisbogen zu zufliegen, der den Kreisbogen in zwei gleich lange Stücke teilt. Dieses Prinzip wird in Abbildung 4.34 verdeutlicht.

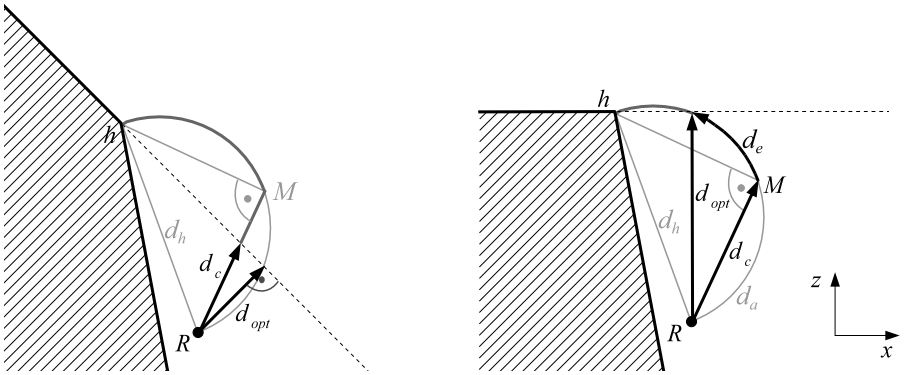

Abbildung 4.34: *Optimierung der Kreisbogenstrategie.*

In der Abbildung werden zwei Fälle dargestellt, die bei dieser Bewegung auftreten

können. Im linken Teil der Abbildung ist der optimale Weg mit der Länge d_{opt} kürzer als der Weg zu dem Punkt M. Hierbei legt der Roboter nur die Strecke d_c zurück und erreicht M nicht, da er schon zuvor Einblick auf die Facette hinter h erhält. Es ergibt sich dabei zwischen d_c und d_{opt} ein Dreieck, bei dem sich an d_{opt} ein rechter Winkel befindet und d_c die Hypotenuse ist. Daher gilt:

$$d_c \leq \sqrt{2} \cdot d_{opt}$$

Ein kompetitiver Faktor C_1 für diesem Fall ist also $\sqrt{2}$.

In dem rechten Teil der Abbildung ist der optimale Weg mit der Länge d_{opt} länger als der Weg zu dem Punkt M. In diesem Fall ist der Einblick hinter den Horizont erst auf dem Abschnitt, auf dem eine Kreisbewegung erfolgt, möglich. Wäre bis zu diesem Zeitpunkt ausschließlich eine Kreisbewegung ausgeführt worden, so würde für die zurückgelegte Wegstrecke gelten:

$$\begin{aligned}
& d_e + d_a \leq \frac{\pi}{2} \cdot d_{opt} \\
\Rightarrow \quad & d_e + \frac{\pi}{4} \cdot d_h \leq \frac{\pi}{2} \cdot d_{opt} \\
\Rightarrow \quad & d_e \leq \frac{\pi}{2} \cdot d_{opt} - \frac{\pi}{4} \cdot d_h \\
\Rightarrow \quad & d_e + d_c \leq \frac{\pi}{2} \cdot d_{opt} - \frac{\pi}{4} \cdot d_h + \frac{1}{\sqrt{2}} d_h \\
\Rightarrow \quad & d_e + d_c \leq \frac{\pi}{2} \cdot d_{opt} - \left(\frac{\pi}{4} - \frac{1}{\sqrt{2}} \right) \cdot d_h
\end{aligned}$$

Da die Wegstrecke des optimalen Weges auf dem Kreis endet, gilt der Satz des Thales und somit auch $d_h > d_{opt}$. Da die geklammerte Konstante größer als null ist, bleibt die Ungleichung beim Ersetzen von d_h durch d_{opt} erhalten. Es gilt somit:

$$\begin{aligned}
& d_e + d_c \leq \frac{\pi}{2} \cdot d_{opt} - \left(\frac{\pi}{4} - \frac{1}{\sqrt{2}} \right) \cdot d_{opt} \\
\Rightarrow \quad & d_e + d_c \leq \left(\frac{\pi}{4} + \frac{1}{\sqrt{2}} \right) \cdot d_{opt}
\end{aligned}$$

Ein kompetitive Faktor C_2 für diesen zweiten Fall ist somit:

$$C_2 = \left(\frac{\pi}{4} + \frac{1}{\sqrt{2}} \right) \approx 1.4925$$

Da C_2 größer ist als C_1 gilt somit für die gesamte Strategie ein kompetitiver Faktor von $C = C_2$. Die hier vorgestellte Strategie kann genauso, wie die einfachere Strategie nur einen Kreisbogen zu verwenden im Zweidimensionalen verwendet werden, um den Bereich hinter einer Ecke zu erkunden. Für diese Problemstellung wurde mit C somit auch eine weitere obere Schranke gefunden, die niedriger ist, als die der reinen Kreisbewegung.

Abschließend wird nun eine weitere Möglichkeit vorgestellt, mit der die in diesem Abschnitt vorgestellte Bewegung eventuell noch weiter optimiert werden kann. In Abschnitt 4.4.4 wurde begründet, warum eine minimale Steigung von 45° sinnvoll sein kann. Bewegt der Roboter sich auf einem Kreisbogen, so ist seine Steigung zuerst über 45°, um dann abzunehmen. Um dies zu verhindern, fliegt der Roboter auf einer Geraden mit einer Steigung von 45° weiter, sobald seine Steigung bei der Bewegung auf dem Kreisbogen unter 45° fallen würde. Das 45° Kriterium kann gegebenenfalls auch am Anfang der Bewegung eingesetzt werden, falls die Steigung der geradlinige Bewegung auf M zu kleiner als 45° sein sollte. In diesem Fall wird die geradlinige Bewegung solange durchgeführt, bis ein anderer Punkt als M auf dem Kreisbogen erreicht wird. In weiterführenden Arbeiten kann untersucht werden, ob eine solche Modifikation der Strategie zu den erwarteten Verbesserungen führt.

4.4.8 Übertragung der *CAB* Strategie

In diesem Kapitel werden zunächst Analogien und Unterschiede zwischen einem einfachen Polygon und einem dreidimensionalen Terrain beschrieben. Danach wird diskutiert, wie die *CAB* Strategie auf die Kernsuche im dreidimensionalen Terrain übertragen werden kann.

Wie schon aus Kapitel 2 ersichtlich, entsprechen die Eckpunkte eines Polygons den Kanten des dreidimensionalen Terrains. Weiterhin entspricht eine Kante des Polygons einer Facette des dreidimensionalen Terrains. In Abbildung 4.35 wird auf der linken Seite ein Roboter an der Position R in einem einfachen Polygon P mit dem Sichtbarkeitspolygon $vis(R)$ dargestellt. Sobald der Roboter eine der gestrichelten Linien überquert, gewinnt er die Sicht auf die zuvor nicht sichtbare Kante hinter einer Ecke von P. In der Abbildung ist dies der Fall bei den Kanten e, f und g. Er gewinnt also einen Einblick in den Bereich hinter einer Ecke. Eine solche Sichtbarkeitsänderung entspricht im dreidimensionalen Terrain dem Durchfliegen einer Ebene E, die zu einer weiter entfernt liegenden Facette einer Horizontkante gehört. Es ist somit ein Einblick in das Gebiet hinter der Horizontkante möglich. In der Abbildung auf der rechten Seite wurden solche Ebenen als gestrichelte Linien dargestellt.

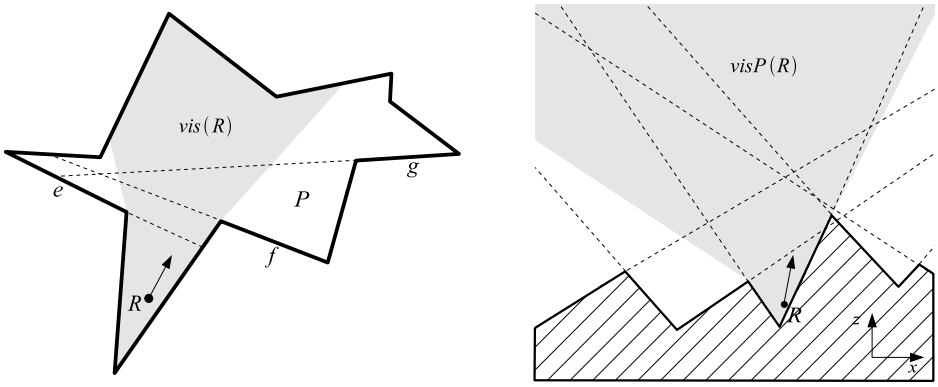

Abbildung 4.35: *Sicht auf weitere Kanten oder Facetten.*

Die Bewegung des Roboters ist in den beiden Szenarien des Polygons und des Terrains davon geprägt, dass immer wieder neue Bewegungsentscheidungen getroffen werden

müssen. Diese findet aufgrund einer veränderten Sichtbarkeitssituation statt. Im einfachen Polygon muss der Gewinnbereich $G(p)$ und damit auch der Bisektor neu bestimmt werden, sobald ein neuer spitzer Eckpunkt sichtbar wird. Im dreidimensionalen Terrain muss der Horizont neu berechnet werden, sobald eine neue blockierende Kante sichtbar wird. Der Horizont ändert sich dabei kontinuierlich, es müssen aber nur wesentliche Änderungen berücksichtigt werden. Es muss nicht berücksichtigt werden, wenn sich der Ausschnitt einer Kante ändert, der eine Horizontkante bildet. Nur falls eine neue Horizontkante hinzukommt oder verschwindet, muss der Horizont neu berechnet werden. Im Vergleich dazu kann eine Ecke des Polygons nur entweder sichtbar oder nicht sichtbar sein. Eine Ecke kann sich während der Bewegung des Roboters jedoch nicht verändern.

Im Folgenden wird beschrieben, wie ermittelt werden kann, wann neue Bewegungsentscheidungen in den beiden Szenarien getroffen werden müssen. Im Polygon orientiert sich der Roboter bei der *CAB* Strategie an den sichtbaren spitzen Eckpunkten, um seine Bewegungsentscheidungen zu treffen. Abbildung 4.36 zeigt, dass der Roboter eine neue Bewegungsentscheidung treffen muss, sobald er eine der gestrichelten Linien überquert. Diese Linien entstehen aus Geraden, die durch eine nicht sichtbare Ecke und eine Polygonecke geht, die auf dem Rand des Sichtbarkeitspolygons liegt. Auf diese Weise kann leicht entschieden werden, wann eine neue Bewegungsentscheidung getroffen werden muss.

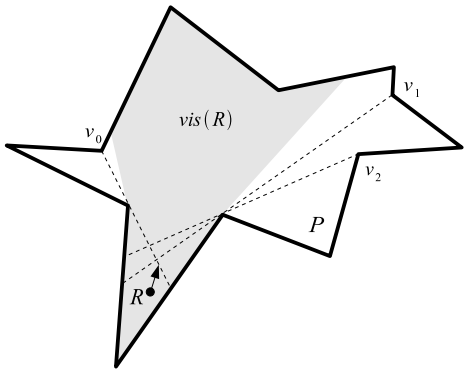

Abbildung 4.36: *Sobald der Roboter R eine der gestrichelten Linien überquert, sieht er eine neue Ecke des Polygons P.*

Im dreidimensionalen Terrain ist es komplexer eine Situation zu identifizieren, in der eine neue Bewegungsentscheidung getroffen werden muss. Während in einem Polygon eine Ecke entweder sichtbar oder nicht sichtbar sein kann, kann im dreidimensionalen Terrain eine Kante e jedoch auch nur teilweise sichtbar sein. Für jede Kante existieren somit drei Ereignisse:

- ein Teil der Kante wird sichtbar.
- die Kante wird vollständig sichtbar.
- Die Sicht auf eine Kante geht verloren.

Abbildung 4.37 zeigt ein Beispiel für eine Roboterbewegung, bei der die Kante e teilweise sichtbar wird. Von der Position R aus sieht der Roboter die Kante e nicht. Sobald

er sich zu R' begibt, sieht er einen Teil von e, der sich nicht mit dem Teil von e überschneidet, der von der Position R'' aus sichtbar ist. Um festzustellen, ab wann der Roboter einen Teil von e sieht, wird ein Polyeder $visP(e)$ definiert, welches eine Teilmenge von $\overline{L(T)}$ ist. Zu $visP(e)$ gehören die Punkte aus $\overline{L(T)}$ von denen aus mindestens ein Punkt der Kante e sichtbar ist. Eine Teilmenge $visP_C(e)$ von $visP(e)$ enthält die Punkte aus $\overline{L(T)}$, von denen aus alle Punkte der Kante e sichtbar sind. Ein Polyeder $visP(e)$ ist zusammenhängend und nach oben offen. Die Berechnung eines solchen $visP(e)$ ist nicht Teil dieser Diplomarbeit. Für den Fall, dass eine Kante e blockierend ist und der Roboter sich in $visP(e)$ hineinbewegt, so verändert sich der Horizont und ein Ausschnitt oder die gesamte Kante e wird Teil des aktuellen Horizonts.

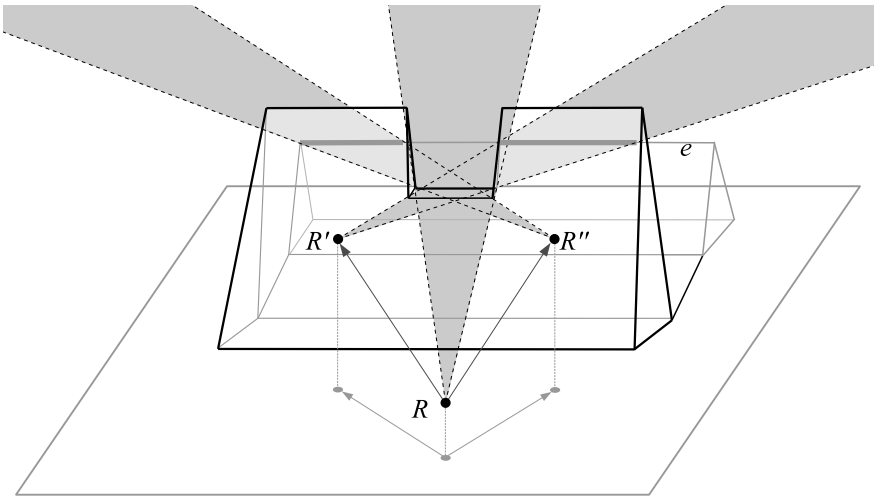

Abbildung 4.37: *Abhängig von der Position erhält der Roboter Sicht auf Teile von e.*

Mit Hilfe von $visP(e)$ lässt sich feststellen, wann eine neue Kante zum Horizont hinzukommt. Es lässt sich damit jedoch nicht bestimmen, welcher Teil der Kante e zum Horizont der aktuellen Roboterposition R beiträgt. Der Teil von e, der von R aus sichtbar ist, verändert sich bei der Bewegung des Roboters kontinuierlich. Es kann sogar vorkommen, dass ein Teil von e, der zuvor sichtbar gewesen ist, wieder hinter einem näher liegenden Horizont verschwindet. In Abbildung 4.37 ist dies der Fall, wenn der Roboter sich von R' zu R'' bewegt. Wie bereits in Abschnitt 4.2.2 erläutert wurde, kann es außerdem auch vorkommen, dass die Sicht auf eine Kante e des Horizonts wieder verloren wird. Dieses dritte Ereignis kann, genauso wie eine neue sichtbare Kante, mit Hilfe von $visP(e)$ erkannt werden, indem das erneute Durchfliegen des Randes dieses Sichtbarkeitspolyeders festgestellt wird.

Bei einer Übertragung der *CAB* Strategie von einem einfachen Polygon auf das dreidimensionale Terrain müssen die hier vorgestellten Analogien und Unterschiede berücksichtigt werden. Eine weitere Schwierigkeit bei der Übertragung der *CAB* Strategie ist die Übertragung der Idee des Bisektors. Im einfachen Polygon bewegt sich der Roboter auf dem Bisektor zweier Geraden, die den Gewinnbereich $G(p)$ bestimmen. Wie schon zuvor erwähnt, entspricht $G(p)$ im dreidimensionalen Terrain dem Sichtbarkeitskegel $B(R)$. In Abschnitt 4.2.3 wurde festgestellt, dass $B(R)$ im Allgemeinen nicht konvex ist. Ein einfacher Versuch der direkten Übertragung der Idee des Bisektors ist, den Schwerpunkt des

Polygons, das durch den oberen Horizont gegeben ist, anzusteuern. Da in einem nicht konvexen Polygon der Schwerpunkt jedoch außerhalb des Polygons liegen kann, kann es vorkommen, dass der Roboter sich in die Terrainoberfläche hineinbewegt. Aufgrund dieses und weiterer Probleme, die bei einer solchen einfachen Strategie entstehen, wurde in Abschnitt 4.3.3 durch eine gewichtete Summe von Vektoren auf die jeweils nächsten Punkte der Kanten des oberen Horizonts die Idee des Bisektors aufgegriffen und in abgewandelter Form realisiert. Auch durch diese Strategie konnte keine Kompetitivität erreicht werden.

4.4.9 Kompetitivität

Laut Aufgabenstellung war der Beweis der Kompetitivität nicht Inhalt dieser Diplomarbeit. Während der Bearbeitung der vorherigen Kapitel wurden jedoch einige Beobachtungen gemacht, die für einen möglichen Beweis von Interesse sind und nun vorgestellt werden. So kann z. B. eine Strategie, bestehend aus einer Kombination von *Weighted Perpendiculars* mit einer Kreisbogenbewegung, durchaus kompetitiv sein. Auf den ersten Blick erscheint hierbei eine getrennte Beobachtung der Bewegungsrichtungen in der X/Y-Ebene und in Z-Richtung sinnvoll. Sollte es möglich sein, bei den Bewegungskomponenten in der X/Y-Ebene eine Kompetitivität festzustellen, so könnte es auch leicht möglich sein, ein geeignetes Kriterium für die Bewegung in Z-Richtung zu entwickeln, so dass die gesamte Bewegung kompetitiv ist. Eine Strategie kann z.B. bei einem nicht *vollständig beschränkendem* Horizont eine Bewegungsrichtung V festlegen, um einen Einblick hinter die noch existierenden Horizontkanten zu erhalten. Dies kann wie bei der Strategie *Weighted Perpendiculars* erfolgen. Dieses Prinzip ist nochmals in Abbildung 4.38 dargestellt. Eine Abwandlung von *Weighted Perpendiculars* könnte auch die Winkelhalbierende des zu α inversen Winkels als Bewegungsrichtung wählen.

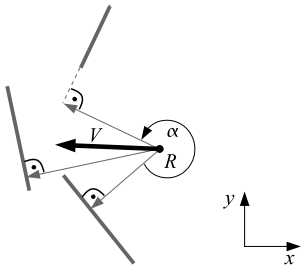

Abbildung 4.38: *Vorzugsrichtung V der Bewegung.*

Wird nun davon ausgegangen, dass sich der Winkelbereich, in dem der Horizont beschränkend ist, bei der Bewegung des Roboters reduziert, so lässt sich mit Hilfe von selbstnähernden Kurven, welche in Abschnitt 2.5.3 beschrieben wurden, eine Kompetitivität feststellen. Dies ist in Abbildung 4.39 dargestellt. Die Abbildung zeigt die Projektion eines vom Roboter zurückgelegten Weges in die X/Y-Ebene. Jeder Pfeil steht dabei für den Einblick hinter eine Horizontkante. Dieser Weg ist selbstnähernd. Dabei sind die Winkelintervalle, die durch die Reduktion des Horizonts wegfallen, grau unterlegt. Es wird deutlich, dass die Vorzugsrichtung nach dem Einblick hinter eine neue Horizontkante eingeschränkt wird.

Das Problem bei solchen Bewegungen ist jedoch, dass durch den Bereich hinter einer aktuellen Horizontkante der Horizont wieder stärker beschränkend werden kann. Dieses

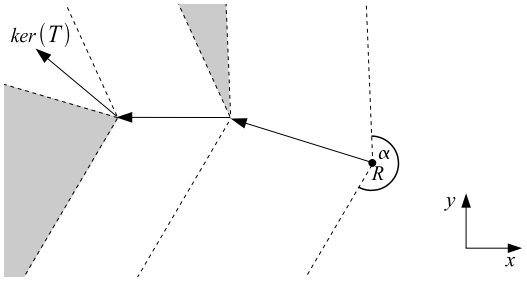

Abbildung 4.39: *Beispiel eines selbstnäherndes Weges.*

Prinzip ist in Abbildung 4.40 dargestellt. Auf der linken Seite ist eine Sicht von oben auf das Beispielterrain abgebildet, während auf der rechten Seite eine Seitenansicht gezeigt wird.

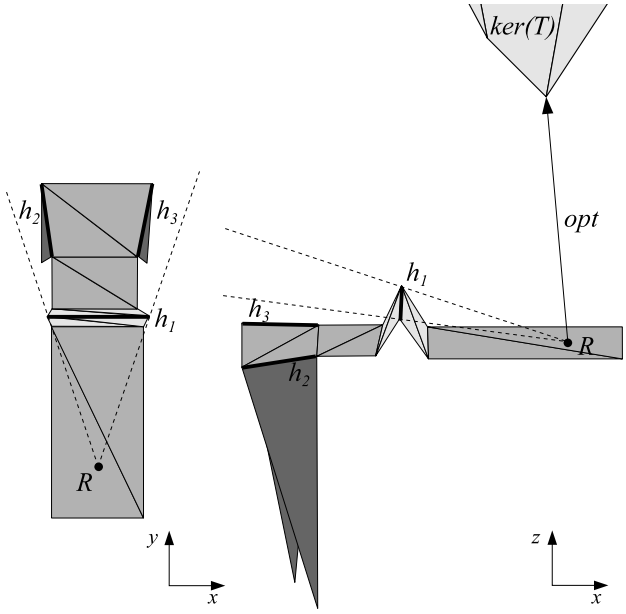

Abbildung 4.40: *Der nächste Punkt des Kerns liegt fast senkrecht über dem Roboter.*

Der Roboter sieht von seiner Position R aus nur die Horizontkante h_1. Er kann also nur diese Kante in seine Bewegungsentscheidung einfließen lassen. Ist die weiter entfernt liegende Facette hinter h_1 sehr steil, so muss der Roboter sehr weit auf h_1 zufliegen, um einen Einblick hinter h_1 zu bekommen. Hinter h_1 werden die Horizontkanten h_2 und h_3 sichtbar. Die weiter entfernt liegenden Facetten von h_2 und h_3 sind in diesem Beispiel noch viel steiler als die von h_1. Der Horizont ist nun wieder hinter der Roboterposition beschränkend. Der Roboter wird daher zunächst auf h_1 zufliegen, um dann später wieder in die entgegengesetzte Richtung auf den Kern zu zufliegen. Eine Projektion des zurückgelegten Weges in die X/Y-Ebene zeigt somit einen Weg, der nicht selbstnähernd ist.

Wie an dem obigen Beispiel deutlich wird, ist eine getrennte Betrachtung der Bewegung in der X/Y-Ebene und in die Z-Richtung problematisch. Daher scheint es notwendig

zu sein, für einen Kompetitivitätsbeweis alle drei Dimensionen gleichzeitig zu berücksichtigen. Ein interessanter Ansatz für einen solchen Beweis könnte eine Übertragung der selbstnähernden Kurven in den dreidimensionalen Raum sein. Im Folgenden wird daher eine Idee skizziert, wie eine solche Übertragung vorgenommen werden könnte. Die Idee der selbstnähernden Wege in der Ebene besteht im Wesentlichen aus der Definition der Normalen in jedem Wegpunkt und der Aussage, dass jeder spätere Wegpunkt auf der Seite der Normalen liegt, in die der Roboter hineinläuft. Übertragen auf den dreidimensionalen Raum entspricht eine solche Normale einer Ebene, deren Normalenvektor dem Bewegungsvektor eines Wegpunktes entspricht. Analog wird zudem gefordert, dass jeder spätere Wegpunkt oberhalb dieser Ebene liegt. Über diese Definition wird erreicht, dass sich der Roboter bei seiner Bewegung immer mehr dem Ziel nähert. Bei einer Literaturrecherche wurden leider keine Aussagen über selbstnähernde Wege und deren Weglänge im dreidimensionalen Raum gefunden. Ein wichtiger Punkt für einen Kompetitivitätsbeweis könnte daher eine Übertragung der Aussagen bezüglich der Weglänge von selbstnähernden Kurven in den dreidimensionalen Raum sein, wie sie in Theorem 2.3 für selbstnähernde Kurven in der Ebene getroffen wurden.

Existiert ein solches Theorem auch für den dreidimensionalen Fall, so ist es lediglich notwendig eine Strategie zu finden, die korrekt arbeitet und einen selbstnähernden Weg beschreibt. Wir nehmen zunächst an, dass eine Strategie S, angelehnt an *Go To Next* aus Abschnitt 4.3.2, einen Einblick hinter die nächstgelegene Horizontkante ermöglicht. Bei der Bewegung berücksichtigt S außerdem das Kriterium bezüglich der vorläufigen Kerne, um, wie in Abschnitt 4.4.2 gezeigt, die Korrektheit zu gewährleisten. Der Zielpunkt jeder Teilbewegung entspricht dabei dem Zielpunkt der optimalen Lösung, um einen Einblick hinter eine Horizontkante zu bekommen. Ferner sei die Lösung für die Teilbewegungen kompetitiv mit dem Faktor C_T zu der jeweils optimalen Lösung für die Teilbewegung. Da dieser Faktor für jedes Teilstück gilt, reicht es für einen Beweis die jeweils optimalen Wegstücke zu betrachten und auf einen daraus gewonnenen kompetitiven Faktor C_{Opt} zu multiplizieren. Es ergibt sich dann $C = C_T \cdot C_{Opt}$. Eine kompetitive Strategie, um einen Einblick hinter eine Horizontkante zu gewinnen, wurde bereits in Abschnitt 4.4.7 vorgestellt. Um zu beweisen, dass eine Folge von optimalen Teilbewegungen eine selbstnähernde Kurve beschreibt, müssen drei Fälle berücksichtigt werden:

1. Der optimale Weg ist durch das Lot auf die Ebene der Facette hinter der Horizontkante gegeben.

2. Das Lot würde das Terrain schneiden. Der optimale Weg ist daher kein Lot auf die Ebene der Facette hinter der Horizontkante.

3. Das Kriterium bezüglich der vorläufigen Kerne hat die Bewegung beeinflusst.

Im ersten Fall ist es offensichtlich, dass die Bedingung für eine selbstnähernde Kurve erfüllt ist. Im zweiten Fall ist zu berücksichtigen, dass das Terrain nicht nur den optimalen Weg beeinflusst hat, sondern auch die Gestalt des Kerns. Es ist leicht nachzuvollziehen, dass auch in diesem Fall die Bedingung erfüllt ist. Auf eine ausführliche Begründung wird an dieser Stelle jedoch verzichtet. Das gleiche gilt für den dritten Fall. In diesem Fall wird der vorläufige Kern durch die Facette hinter dem Horizont weiter eingeschränkt. Der Endpunkt P der Bewegung befindet sich somit am Rand des neuen vorläufigen Kerns, wobei die in P senkrechte Normalenebene den neuen vorläufigen Kern nur berührt.

Die hier vorgestellte Beweisskizze zeigt, dass eine Strategie S kompetitiv ist, sofern es auch einen kompetitiven Faktor für selbstnähernde Wege im dreidimensionalen Raum gibt.

Kapitel 5

Implementierung und Applet

5.1 Wesentliche Elemente der Architektur

In diesem Abschnitt werden wesentliche Elemente des Java Applets erläutert. Da in den vorhergehenden Kapiteln die Hauptalgorithmen schon beschrieben wurden, werden hier nur einige Eckpunkte des Programms vorgestellt. Zuerst wird in Abschnitt 5.1.1 die Struktur einiger Teile des Applets skizziert, die mit Java3D erstellt wurden. Anschließend werden in Abschnitt 5.1.2 die Geometrieobjekte beschrieben, die der Darstellung und der Programmierung der Algorithmen zu Grunde liegen. In Abschnitt 5.1.3 wird beschrieben, wie weitere Online Strategien dem Applet hinzugefügt werden können.

5.1.1 Java3D Szenengraph

Das Applet *Kernel3D* stellt ein Terrain in zwei verschiedenen Ansichten dar. Zum einen ist dies eine Sicht auf das Terrain, in dem die Facetten des Terrains farbig dargestellt werden. In dieser Sicht ist es möglich, den Kern, den Horizont und die Online bzw. Offline Wege in den Kern anzuzeigen. Zum anderen gibt es eine Editiersicht des Terrains. Hier werden die Knoten und Kanten des Terrains dargestellt und können verschoben und hinzugefügt werden. In jeder dieser Ansichten wird das Terrain zweimal in verschiedenen Blickwinkeln dargestellt. Diese beiden Sichten sind durch zwei verschiedene Szenengraphen entstanden. Im Folgenden wird beschrieben, was ein Szenengraph ist und wie er aufgebaut wird.

Um in Java3D Objekte darzustellen, muss ein so genannter Szenengraph aufgebaut werden. Im Folgenden werden die Begriffe aus Java3D jeweils in Klammern hinter der deutschen Umschreibung angegeben. Der Szenengraph (Scene Graph) ist ein Baum und hat ein virtuelles Universum (VirtualUniverse) zur Wurzel, welches als Sohn einen Schauplatz (Locale) hat. Ein Schauplatz hat zwei Söhne. Der linke Sohn bildet die Wurzel zu dem Inhaltszweig (Content Branch Graph). Dieser Teilgraph enthält die Objekte, die dargestellt werden sollen, zusammen mit ihren Translationen und Rotationen. In dem Applet *Kernel3D* sind dies z. B. die Kugeln, die für die Knoten in dem Terrain stehen. Der rechte Sohn des Schauplatzes definiert die Parameter der Sicht auf die Szene, dass ist z. B. die Blickrichtung auf die Szene (view branch graph). Detailliertere Informationen zu dem Szenengraph finden sich in [16] und in [17]. Es wurde die Java3D Version 1.3.2 verwendet. Zum Nachschlagen während der Javaprogrammierung wurde außerdem [18] benutzt.

Zum Aufbau des Szenengraphen wurden die Java3D Klassen zu einfacheren Klassen zusammengefasst. Der dadurch entstandene Szenengraph ist in Abbildung 5.1 dargestellt.

Dabei wurden die selbst erstellten Klassen in hellgrau unterlegten Ellipsen abgebildet, während die Entsprechungen von Java3D in weißen Ellipsen jeweils darunter gezeigt werden. Die Klasse *Location* fasst im Wesentlichen die Java3D Klassen *Universe* und *Locale* zusammen. Dabei werden die Einstellungen bei den beiden Java3D Klassen vorgenommen, die für das Projekt benötigt werden. Der linke Sohn der *Location* ist die *Scene* die eine *BranchGroup* aus Java3D erweitert und alle Objekte (*SceneObject*) logisch zusammenfasst. Sie bietet die Möglichkeit ihre Objekte zu drehen und zu verschieben. Die einzelnen Objekte der Szene enthalten Zustandsvariablen, die angeben, ob das Objekt verschiebbar, drehbar oder überhaupt mit der Maus anklickbar ist. Jedes Szenenobjekt hat wiederum eine Gestalt, die durch ein *ScenePrimitive* gegeben ist. Das *ScenePrimitive* beinhaltet dazu die Java3D Klasse *Shape*. Die Klassen *SceneObject* und *ScenePrimitive* sind abstrakte Klassen. Um die Funktionalität der Klassen nutzen zu können, müssen sie abgeleitet werden. In dem Applet wurden alle geometrischen Objekte so erstellt. Es sind im Wesentlichen:

- die Kanten und die Knoten des Terrains
- das Terrain
- der Kern
- die Achsen
- die Kanten für die Online und Offline Wege

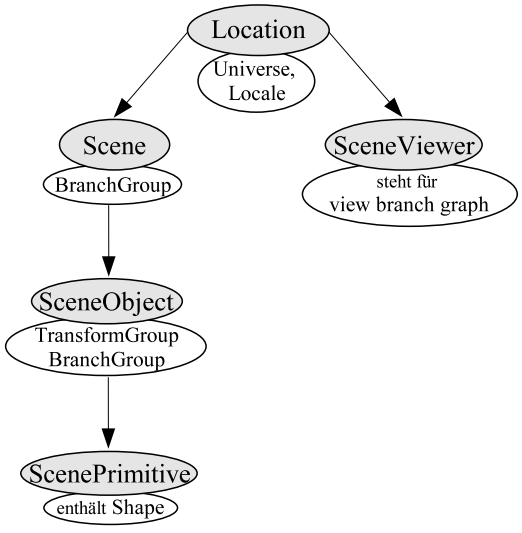

Abbildung 5.1: *Der Scene Graph des Applets*

5.1.2 Geometrie und DCEL

In den Algorithmen wurde, wie schon in der Beschreibung der Algorithmen erwähnt, eine *DCEL* benötigt. Außerdem traten immer wieder Situationen auf, in denen Geraden und

Ebenen miteinander geschnitten wurden und Punkte lokalisiert werden mussten. Dazu wurde das Paket *Geometry* angelegt. Es enthält Basisklassen für eine Ebene (*Plane3D*), eine Gerade (*Line3D*) und einen Vektor (*Vektor3D*). Diese Basisklassen enthalten immer wieder benötigte Methoden zur Interaktion dieser Objekte. Davon abgeleitet wurden *DCELs*, die für die verschiedenen Anwendungen spezialisiert wurden. So brauchte die konvexe Hülle eine DCEL im dreidimensionalen Raum, während die DCEL des Terrains als planarer Graph darstellbar sein muss. Eine dritte DCEL entstand, um den Graphen des Kerns abzubilden. Abbildung 5.2 zeigt diese Hierarchie.

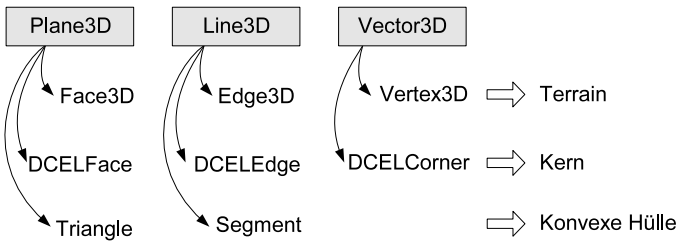

Abbildung 5.2: *Die Basisklassen und die diese erweiternden Klassen.*

Die Formeln zu Schnitten von Ebenen, Geraden stammen aus [19] oder wurden aus Formeln aus dem Buch abgeleitet.

5.1.3 Die Klasse *AbstractOnlineAlgo*

In diesem Abschnitt wird beschrieben, wie mit Hilfe der abstrakten Klasse *AbstractOnlineAlgo* eine Strategie in das Applet integriert werden kann. Zuerst muss die Klasse *Strategies* um einen Algorithmentyp erweitert werden. Danach wird eine neue Klasse erstellt, die die Klasse *AbstractOnlineAlgo* erweitert und zwei Methoden implementiert. Zuerst wird ein Konstruktor implementiert, der als Parameter ein Terrain (*QEDS3D*) und eine Roboterposition (*Vector3D*) hat und den Konstruktor der Basisklasse mit diesen Parametern und dem Algorithmentyp aus der Klasse *Strategies* aufruft. Danach muss die Methode *start()* überschrieben werden, in der die eigentliche Strategie ausgeführt wird. Dabei müssen für jeden Schritt, den die Strategie ausführt, die Vektoren *robotPositions*, *offlineResults*, *onlinePath*, *horizons*, *visFaces* und *tempKernel* gültig gefüllt werden. D. h. für jeden Schritt der Strategie müssen die Zustände des Online und des Offline Weges enthalten sein. Dabei findet sich an der i-ten Stelle jedes Vektors der Zustand des i-ten Schrittes. Um diese Zustände zu setzen, sollten zwei Hilfsmethoden verwendet werden: *addRobotPosition()* und *addHorizon()*. Diese Hilfsmethoden füllen schon einen Teil der zuvor genannten Datenstrukturen. Nähere Informationen sind der Dokumentation des Quellcodes zu entnehmen.

Um die Strategie in die grafische Oberfläche des Applets zu integrieren muss ein neuer Eintrag in die Klasse *MenuBar* hinzugefügt werden. In dieser Klasse muss zudem ein neuer Eintrag in der *actionPerformed()* Methode vorgenommen werden, der bei dem *SceneViewer* die neue Strategie über den Strategietyp aus der Klasse *Strategies* setzt.

5.2 Herausforderungen bei der Implementierung

In diesem Abschnitt werden einige Herausforderungen der Implementierung erläutert. Dazu gehört unter anderem das Behandeln der Ungenauigkeit bei der Verwendung von Fließkommazahlen. Außerdem wurde ein AVL-Baum benötigt, der einige über die Funktionalität eines normalen AVL-Baums hinausgehende Fähigkeiten besitzt. Da das Picking in der verwendeten Java3D Version unzureichend war, musste eine eigene Klasse dafür geschrieben werden.

5.2.1 Problem mit der Ungenauigkeit von Fließkommazahlen

Das Problem der Ungenauigkeit bei Rechnungen mit Fließkommazahlen trat hauptsächlich beim Rechnen mit Ebenen auf. Da das Terrain trianguliert ist, wird jede Facette des Terrains über die Eckpunkte eines Dreiecks bestimmt. Für die Hessesche Normalform der Ebene, die zu einer Facette gehört, musste der Normalenvektor der Ebene bestimmt werden. Dadurch, dass die Reihenfolge die Eckpunkte des Dreiecks variieren kann, gab es voneinander abweichende Ergebnisse für einen Normalenvektor. Der Vergleich von zwei Normalenvektoren einer Ebene ergab dann das falsche Ergebnis. Das Problem der Double Ungenauigkeit trat auch auf bei dem Test, ob ein Punkt in einer Ebene liegt oder ein Punkt auf einer Geraden. Um diese Probleme zu beheben, entstand die Klasse *Numeric.java*. In Listing 5.1 ist ein Auszug aus dieser Klasse angegeben.

```
1  static long maxDist = 32;
2
3  public static boolean isEqual(double d1, double d2) {
4    if (Double.isNaN(d1) || Double.isNaN(d2))
5      return false;
6    if (d1 == Double.POSITIVE_INFINITY)
7      return (d2 == Double.POSITIVE_INFINITY);
8    if (d1 == Double.NEGATIVE_INFINITY)
9      return (d2 == Double.NEGATIVE_INFINITY);
10   long a = Double.doubleToLongBits(d1);
11   long b = Double.doubleToLongBits(d2);
12   if (a == b)
13     return true;
14   if (a < 0)
15     a = Long.MIN_VALUE - a;
16   if (b < 0)
17     b = Long.MIN_VALUE - b;
18   long dist = Math.abs(a - b);
19   if (dist == Long.MIN_VALUE)
20     return false;
21   return (dist <= maxDist);
22 }
```

Listing 5.1: Auszug aus der Klasse Numeric.java

Es gibt verschiedene Möglichkeiten, um das Problem beim Vergleichen von Fließkommazahlen zu lösen. Die Verwendung eines festen ϵ ist die einfachste Lösung. Ist die Differenz zweier Zahlen kleiner als das gewählte ϵ, so werden die Zahlen als gleich betrachtet. In dieser Arbeit wurde jedoch eine Lösung gewählt, die berücksichtigt, dass die Zahlen in der Fließkommazahldarstellung nahe bei der null dichter sind, als in den weit von der null entfernten Bereichen. Es wird eine feste Zahl D gewählt, die angibt, wie viele Zahlen zwischen den beiden Vergleichspartner liegen dürfen. Die Anzahl der Zahlen orientiert sich dabei daran, wie viele Zahlen in der Fließkommadarstellung tatsächlich zwischen den Vergleichspartnern liegen. Die Vorgehensweise wird im Folgenden erläutert.

Laut [20] sind Fließkommazahlen so definiert, dass sie lexikographisch sortiert sind. Diese Ordnung bleibt erhalten, wenn man die Fließkommazahldarstellung als Integerzahldarstellung interpretiert. Diese Integerzahl ist im 'Betrag und Vorzeichen'-Format gespeichert. Die Zahldarstellungen der beiden zu vergleichenden Zahlen werden also als Integerzahlen interpretiert und dann miteinander verglichen. Im Allgemeinen verwenden Prozessoren jedoch das Zweierkomplement und nicht das 'Betrag und Vorzeichen'-Format. Wenn nun die beiden Vergleichspartner negativ sind, ergibt der Vergleich das Gegenteil des erwünschten Ergebnisses, wie aus Abbildung 5.3 ersichtlich. Deshalb müssen die Vergleichspartner in das Zweierkomplement umgerechnet werden, falls sie kleiner als null sein sollten.

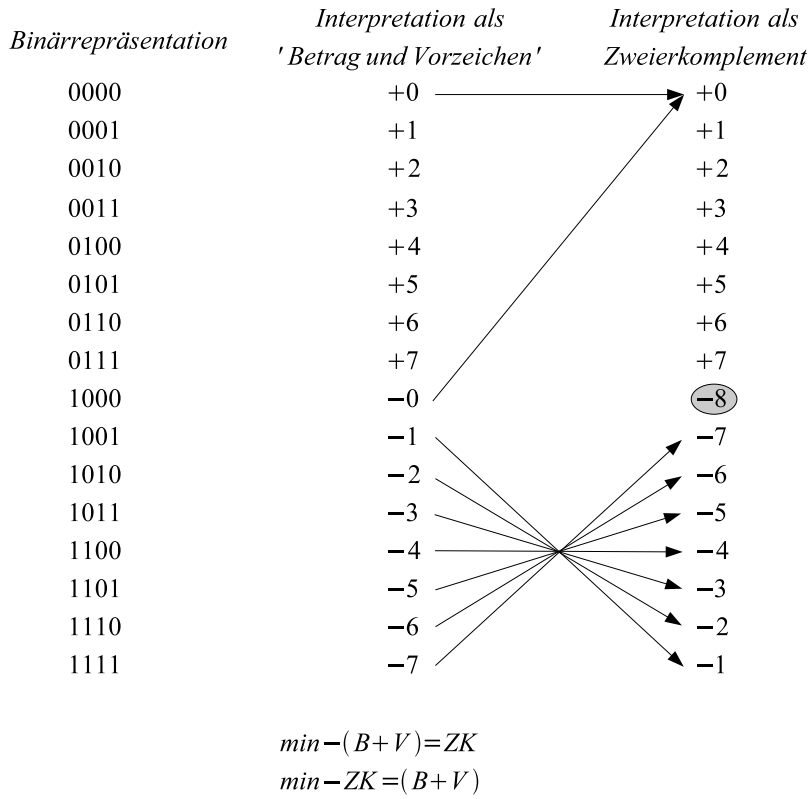

Abbildung 5.3: *Zahldarstellung als Integer und im Zweierkomplement.*

5.2.2 Erweiterter AVL-Baum

Zum Speichern von Objekten während der Algorithmen wurde immer wieder ein AVL-Baum benötigt. Ein einfacher AVL-Baum wurde dazu um zwei Fähigkeiten erweitert. Es wurde ein Iterator programmiert, der es erlaubt, den Baum bequem mit Aufrufen wie *current()* und *next()* zu durchwandern. Zudem ermöglicht dieser Iterator die in Java 5 eingeführte vereinfachte *for* Schleife zu verwenden. Die zweite Erweiterung ermöglicht es, zwei Objekte nach unterschiedlichen Kriterien miteinander zu vergleichen. Damit war es z. B. möglich eine Kante zum einen nach ihrer Steigung mit einer anderen Kante zu vergleichen, oder auch nach lexikographisch nach den Endpunkten.

5.2.3 Picking in Java3D

Das *Picking* und die *Pickbehavior* definiert das Verhalten der Objekte, wenn diese angeklickt werden. Dabei wird unterschieden, welche Maustaste, bzw. Tastaturtaste betätigt wurde und ob das Objekt einfach angeklickt oder gezogen wird. Die Klasse *PickBehavior* des Projektes verarbeitet diese Maus- und Tastaturereignisse und leitet die Ereignisbehandlung ein. In der verwendeten Java3D Version 1.3.2 steht nur ein sehr rudimentäres Picking zur Verfügung. Deshalb musste ein eigenes Picking-Modul geschrieben werden. Die Herausforderung bestand dabei im Umrechnen zwischen den drei Koordinatensystemen des Programms.

Diese Koordinatensysteme werden nun vorgestellt. Abbildung 5.4 zeigt das Koordinatensystem des Bildschirms bzw. der Maus. Dieses Koordinatensystem ist zweidimensional, die X Werte stammen aus dem Intervall $(0, B)$, wobei B die Breite der Leinwand ist und die Y Werte sind aus dem Intervall $(0, H)$, mit der Höhe H der Leinwand. Diese Leinwand ist dabei das Java3D Objekt $Canvas3D$. Der linke obere Punkt ist der Ursprung in diesem Koordinatensystem. In der Abbildung wurde er mit *TopLeft* beschriftet, während der rechte untere Punkt der Leinwand mit *BottomRight* und den entsprechenden Koordinaten beschriftet wurde. Das System hat also zwei Achsen, eine in die positive X-Richtung und eine in die negative Y-Richtung. Die Mausereignisse werden in diesem Koordinatensystem übergeben.

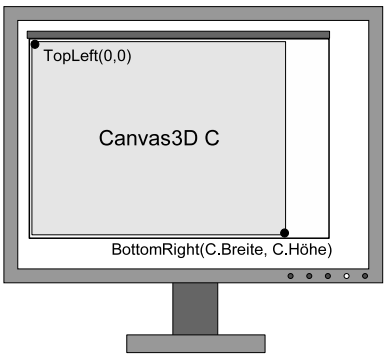

Abbildung 5.4: *Das Maus- und Bildschirmkoordinatensystem*

Abbildung 5.5 zeigt das Koordinatensystem, dass innerhalb von Java3D benutzt wird, um dreidimensionale Objekte auf ein Rechteck (Image Plate) zu projizieren und sie darzustellen. Im Gegensatz zum Mauskoordinatensystem hat dieses System seinen Ursprung

in der unteren linken Ecke, die in der Abbildung mit *BottomLeft* beschriftet wurde. Das System hat zwei Achsen, die in die positive X- bzw. Y-Richtung zeigen.

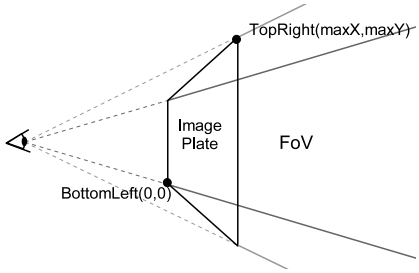

Abbildung 5.5: *Das Koordinatensystem der Image Plate*

Die Koordinaten der Objekte im Applet werden in Weltkoordinaten angegeben. Dieses Koordinatensystem ist in Abbildung 5.6 dargestellt. Das Weltkoordinatensystem hat drei Achsen. Es können Kameras von verschiedenen Positionen auf eine Szene gerichtet werden, um diese aus unterschiedlichen Blickwinkeln zu betrachten. Wie aus der Grafik ersichtlich, ist der Nullpunkt der Schnittpunkt der drei Koordinatenachsen.

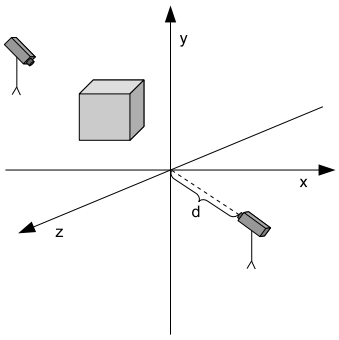

Abbildung 5.6: *Das Weltkoordinatensystem.*

Die Umrechnung zwischen diesen drei Koordinatensystemen ergibt sich aus den unterschiedlichen Positionen der Nullpunkte der Systeme.

Kapitel 6

Zusammenfassung

Die Aufgabenstellung dieser Diplomarbeit bestand im Wesentlichen aus zwei Teilen. Zum einen sollte ein Offline Algorithmus für die Kernsuche in einem dreidimensionalen Terrain implementiert werden. Zum anderen sollten einfache Online Strategien entwickelt werden und deren Korrektheit gezeigt werden. Dabei sollte auch überprüft werden, ob sich die *CAB* Strategie für die Kernsuche in einem Polygon auf die Problemstellung der Kernsuche im dreidimensionalen Terrain übertragen lässt. Im Folgenden wird zusammenfasst, wie diese Aufgaben gelöst wurden.

Nachdem in Kapitel 2 die für das Verständnis wichtigen Grundlagen präsentiert wurden, wurde in Kapitel 3 eine Offline Lösung für die Kernsuche im dreidimensionalen Terrain vorgestellt. Der Kern besteht aus dem Halbraumschnitt der zu den Facetten des Terrains zugehörigen Ebenen. Dieser Halbraumschnitt ist nach oben offen und hat die Form eines Polyeders. Der Kern wurde über die Dualität zwischen den Graphen der konvexen Hülle und dem Graphen des Halbraumschnitts bestimmt. Dabei wurde ein eigener Algorithmus vorgestellt, der den Kern berechnet, wenn bekannt ist, welche Facetten des Terrains zum Kern beitragen. Der Weg in den Kern ist entweder die Wegstrecke zu einem Eckpunkt des Kerns, ein Lot auf eine Kante oder auf eine Facette des Kerns. Dieser Weg kann nach der Berechnung des Kerns mit linearem Zeitaufwand berechnet werden und hat die Länge des minimalen euklidischen Abstandes zwischen dem Startpunkt des Roboters und dem Endpunkt des Weges im Kern.

In Kapitel 4 wurde zuerst der Horizont zu einer Roboterposition definiert. Der Horizont beschreibt dabei den Übergang von einem sichtbaren zu einem nicht sichtbaren Bereich im Terrain. Er besteht aus Kanten und Teilkanten des Terrains. Von besonderem Interesse ist hierbei besonders der obere Horizont, der für die Bewegungsentscheidungen der Online Strategien diente. In diesem Zusammenhang wurde ein Algorithmus zur Bestimmung des Horizonts vorgestellt. Dieser Horizont ändert sich kontinuierlich bei der Bewegung des Roboters. Durch die in Abschnitt 4.2.2 definierte *radiale Verschiebung* ist es schwierig die wesentlichen Ereignisse der Änderung des Horizonts und damit der Sicht bei einer Bewegung des Roboters zu bestimmen. Der vom Roboter aus sichtbare Bereich seiner Umgebung wird Sichtbarkeitspolyeder genannt. Analog zu dem Gewinnbereich $G(p)$ der *CAB* Strategie wurde eine Teilmenge des Sichtbarkeitspolyeders, der Sichtbarkeitskegel, definiert. Dieser Sichtbarkeitskegel wird in einigen Online Strategien für die Bewegungsentscheidung des Roboters verwendet. Einige einfache Strategien für die Kernsuche im dreidimensionalen Terrain wurden in Abschnitt 4.3 vorgestellt. Diese Strategien arbeiten zwar korrekt, sind jedoch nicht kompetitiv. Anschließend wurden in Abschnitt 4.4 Ide-

en vorgestellt, die in komplexen Online Strategien enthalten sein können. In Abschnitt 4.4.6 wurde dann eine solche komplexe Strategie vorgestellt. Diese ist allerdings auch nicht kompetitiv. Die vorgestellten Online Strategien basierten auf geradlinigen Bewegungen. Eine Bewegung auf Kreisbögen scheint jedoch viel versprechender zu sein. Dies wurde in Abschnitt 4.4.7 diskutiert. Die Wahl des Mittelpunktes für solche Kreisbögen und der Ebene, in die ein Kreisbogen eingebettet ist, kann Bestandteil zukünftiger Arbeiten sein, ebenso die Berücksichtigung mehrerer Horizontkanten für die Konstruktion eines Kreisbogens. In Abschnitt 4.4.8 wurde schließlich beschrieben, welche Probleme bei der Übertragung der *CAB* Strategie auf das dreidimensionale Terrain auftreten. Die Entwicklung einer geeigneten Datenstruktur, die alle wesentlichen Änderungen bezüglich der Sichtbarkeit berücksichtigt, ist für eine solche Übertragung wichtig. Durch den so genannten Fenstereffekt aus Abschnitt 4.2.3 erscheint dies jedoch schwierig. Abschließend wurde in Abschnitt 4.4.9 eine Beweisskizze für eine mögliche Online Strategie für die Kernsuche im dreidimensionalen Terrain angegeben. Zukünftige Arbeiten könnten dabei für einen Beweis die selbstnähernden Kurven, die bisher nur für die Ebene definiert wurden, auf den dreidimensionalen Raum übertragen. Damit würde der Beweis dann komplettiert.

Parallel zum theoretischen Teil der Diplomarbeit entstand ein Java-Applet. Einige wesentliche Bestandteile dieses Applets wurden in Abschnitt 5 vorgestellt. Unter anderem wurde dort beschrieben, wie neue Online Strategien in das bestehende Applet integriert werden können. Die Bedienung des Applets wird in Anhang A erklärt.

Anhang A

Benutzerhandbuch

Das Applet unterteilt sich in eine Ansicht, in der das Terrain mit verschiedenen Online Algorithmen exploriert und besichtigt werden kann und in eine Ansicht, in der das Terrain editiert werden kann. Es werden nun zunächst die Maussteuerung und die gemeinsamen Menüpunkte beschrieben. Anschließend werden die Menüs für den Exploriermodus und für den Editiermodus erläutert.

A.1 Die Maussteuerung und allgemeine Menüpunkte

Zunächst wird die Steuerung des Programms mit der Maus beschrieben. Mit der gedrückten linken Maustaste können Objekte bewegt werden. Auch das Terrain kann so bewegt werden. Um näher an das Terrain heranzukommen oder es aus größerer Ferne zu betrachten, wird die Maus mit der gedrückten mittleren Maustaste bewegt. Alternativ kann auch die linke Maustaste und die Taste *Ctrl* gedrückt gehalten werden. Das Terrain kann mit der gedrückten rechten Maustaste oder mit der gedrückten linken Maustaste und der Taste *Alt* gedreht werden.

Das Hauptmenü ***File*** enthält die folgenden Unterpunkte:

- *New:* Erstellt ein neues Terrain, das aus einer einzelnen Pyramide besteht.
- *Load:* Lädt ein Terrain. Die Datei muss die Endung *.q3d* haben und ein spezielles Dateiformat. Die Datei muss mit dem Schlüsselwort *QEDS3D* beginnen. Anschließend werden die Kanten des Terrains angegeben. Jede Kante wird in eine eigene Zeile geschrieben. Die Liste der Kanten wird von geschweiften Klammern umschlossen.
- *Save:* Speichert ein Terrain in eine Datei. Es muss die Endung *.q3d* angegeben werden.
- *Close:* Schließt das Programm.

Im Hauptmenü ***Edit*** bietet Möglichkeiten, das Terrain zu editieren. Das Applet wird in der Explorieransicht gestartet. Der Menüpunkt *Enable Edit Mode* in diesem Menü dient dazu, von dieser Ansicht in die Editieransicht zu wechseln. Analog dazu kann von der Editieransicht in die Explorieransicht gewechselt werden, in dem der Menüpunkt *Disable Edit Mode* angeklickt wird.

Das Hauptmenü ***Edit*** enthält den gemeinsamen Unterpunkt *Random Terrain*. Beim Anklicken dieses Menüpunktes wird ein Dialog geöffnet, in dem die Parameter des Algorithmus zur Generierung eines Zufallsterrains eingestellt werden können. Dies sind die Anzahl der Knoten des Terrains und die minimalen und maximalen Koordinaten der Knoten des neuen Terrains.

Das Hauptmenü ***View*** enthält den gemeinsamen Unterpunkt *Fit To View*, der beim Anklicken bewirkt, dass das Terrain zentriert und vollständig darstellt.

A.2 Die Explorieransicht

In der Explorieransicht ist immer ein Roboter vorhanden. Dieser kann mit der linken Maustaste auf dem Gelände verschoben werden. Der Roboter ist nicht Anklickbar, wenn er sich über dem Terrain befindet, weil gerade eine Online Strategie angewendet wurde. In diesem Fall muss der Roboter zuerst wieder auf das Terrain heruntergeflogen werden, indem der Schieber ganz nach links gezogen wird.

Das Hauptmenü ***Edit*** enthält die folgenden Untermenüs:

- *Load Robot Position:* Lädt eine Roboterposition aus einer Datei mit der Endung *.pos*
- *Save Robot Position:* Speichert eine Roboterposition ein eine Datei, dabei muss die Endung *.pos* angegeben werden.

Das Hauptmenü ***View*** enthält die Unterpunkte:

- *Show Kernel3D:* Der Kern des Terrains wird angezeigt, ebenso der optimale Weg in den Kern.
- *Show Temporary Kernel:* Der vorläufige Kern zur aktuellen Roboterposition wird angezeigt.
- *Show Contributing Faces:* Die Facetten des Terrains, die zum Kern beitragen werden zur Veranschaulichung anders eingefärbt
- *Show Horizon:* Der Horizont zur aktuellen Roboterposition wird angezeigt. Dabei sind die blockierenden Kanten blau, der obere Horizont rot und der übrige Horizont grün dargestellt.
- *Show Visible Faces:* Die von der aktuellen Roboterposition aus sichtbaren Facetten des Terrains werden farblich hervorgehoben.
- *Show Online Path:* Nach dem Auswählen einer Online Strategie kann bestimmt werden, ob der von der Online Strategie zurückgelegte Weg nachgezeichnet werden soll. Als Standard ist diese Option angeschaltet.

Das Hauptmenü ***Strategy*** enthält die folgenden Unterpunkte:

- *Optimal:* Zeigt den optimalen Weg in den Kern an.
- *Straight Up:* Zeigt den Weg dieser Online Strategie an.

- *Go To Next:* Zeigt den Weg dieser Online Strategie an.
- *Weighted Angles:* Zeigt den Weg dieser Online Strategie an.
- *Weighted Perpendiculars:* Zeigt den Weg dieser Online Strategie an.

Abbildung A.1 zeigt ein Bildschirmfoto des Programms. Im linken Teil des Programms wird eine Sicht von oben auf ein Terrain gezeigt, während im rechten Teil des Programms eine Seitenansicht desselben Terrains abgebildet wird. In beiden Ansichten wird zusätzlich der Kern und ein optimaler Weg von der Roboterposition auf dem Terrain in den Kern angezeigt. Über den Schieber unter den beiden Ansichten können sowohl die einzelnen Schritte als auch der gesamte Weg einer Strategie nachvollzogen werden. In der Statusleiste am unteren Programmrand ist auf der linken Seite der Name der aktuellen Strategie eingetragen, während in der Mitte der kompetitive Faktor des aktuellen Weges der ausgewählten Strategie angezeigt wird. Auf der rechten Seite der Statusleiste ist zu entnehmen, wieviele Schritte die ausgewählte Online Strategie in den Kern braucht und welcher der Schritte gerade angezeigt wird.

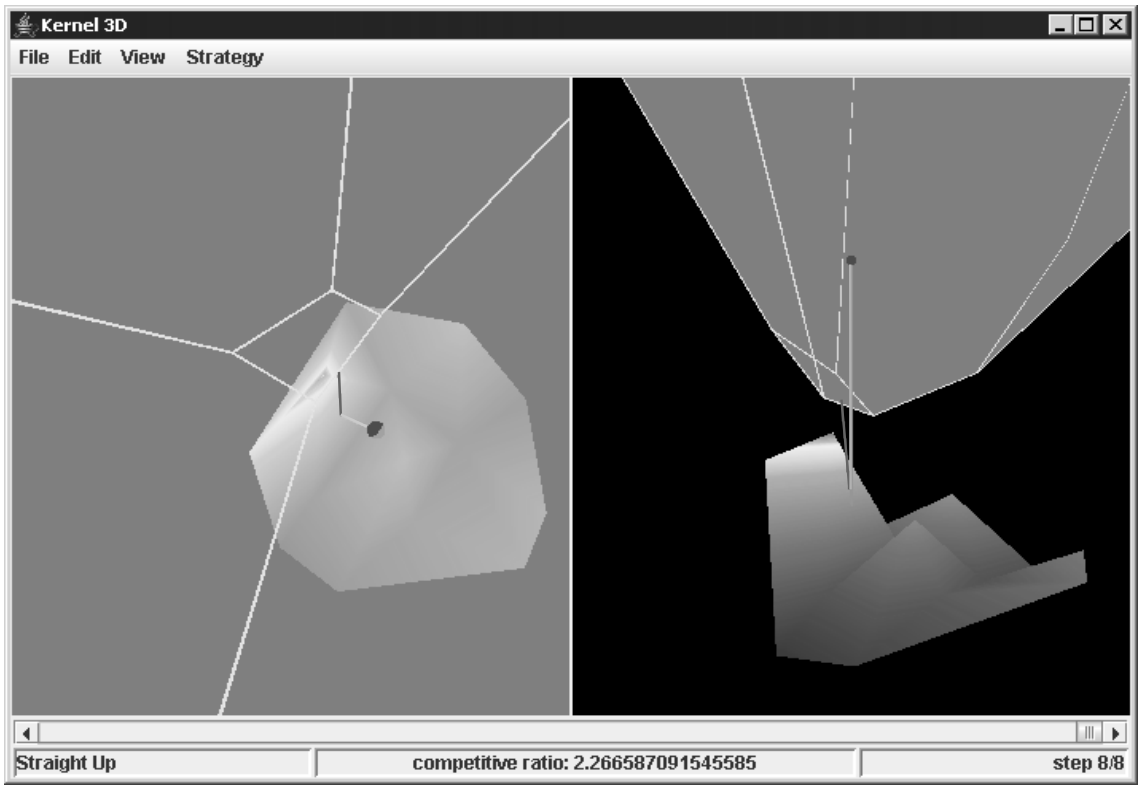

Abbildung A.1: *Das Applet zeigt ein Terrain in zwei verschiedenen Ansichten.*

In der Explorieransicht ist immer ein Roboter vorhanden. Dieser kann mit der linken Maustaste auf dem Terrain verschoben werden. Der Roboter ist nicht anklickbar, wenn er sich über dem Terrain befindet, das kommt dann vor, wenn gerade eine Online Strategie angewendet wird. In diesem Fall muss der Roboter zuerst wieder auf das Terrain heruntergeflogen werden, indem der Schieber ganz nach links gezogen wird.

A.3 Die Editieransicht

In der Editieransicht hat das Hauptmenü ***Edit*** zusätzlich die folgenden Unterpunkte:

- *Add Node:* Nach dem Anklicken dieses Menüeintrags können Knoten zu dem Terrain mit einem Klick der linken Maustaste hinzugefügt werden. Mit einem Klick mit der rechten Maustaste wird dieser Modus verlassen.
- *Delete Node:* Nach dem Anklicken dieses Menüeintrags können Knoten aus dem Terrain mit einem Klick der linken Maustaste gelöscht werden. Mit einem Klick mit der rechten Maustaste wird dieser Modus verlassen.
- *Flip Edge:* Nach dem Anklicken dieses Menüeintrags können Kanten geflippt werden, wenn dadurch keine Höhlen im Terrain entstehen. Das heißt, dass eine Diagonale in einem Viereck nach dem Flippen die beiden anderen gegenüberliegenden Eckpunkte verbindet. Mit einem Klick mit der rechten Maustaste wird dieser Modus verlassen.

Literaturverzeichnis

[1] R. Klein, *Algorithmische Geometrie*. Springer Verlag, 2005.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*. Springer Verlag, 2000.

[3] R. Klein, "Online Algorithmen." Vorlesung an der Universität zu Bonn, 2003.

[4] C. Icking, R. Klein, and E. Langetepe, "Searching for the Kernel of a Polygon: A Competitive Strategy Using Self-Approaching Curves," Technical Report 211, Fernuniversität Hagen, 1997.

[5] E. Langetepe, *Design and Analysis of Strategies for Autonomous Systems in Motion Planning*. PhD thesis, Department of Computer Science, FernUniversität Hagen, 2000.

[6] A. López-Ortiz, *On-line Target Searching in Bounded and Unbounded Domains*. PhD thesis, Univ. Waterloo, Waterloo, Canada, 1996.

[7] C. Icking, R. Klein, and E. Langetepe, "Self-Approaching Curves," tech. rep., Department of Computer Science, 1999.

[8] J.-H. Lee and K.-Y. Chwa, "Tight Analysis of a Self-Approaching Strategy for the Online Kernel-Search Problem," *Inform. Process. Lett.*, vol. 69, pp. 39–45, 1999.

[9] J.-H. Lee, C.-S. Shin, J.-H. Kim, S. Y. Shin, and K.-Y. Chwa, "New Competitive Strategies for Searching in Unknown Star-Shaped Polygons," in *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pp. 427–429, 1997.

[10] V. Bayer, "Survey of Algorithms for the Convex Hull Problem," tech. rep., Department of Computer Science, 1999.

[11] G. Miller, "3D Convex Hull." lecture, 2003.

[12] E. Langetepe and G. Zachmann, *Geometric Data Structures for Computer Graphics*. AK Peters, 2005.

[13] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[14] L. D. Floriani and P. Magillo, "Computing Point Visibility on a Terrain Based on a Nested Horizon Map," in *Proc. ACM Symposium on Applied Computing (SAC), Phoenix Civic Plaza* (E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, eds.), pp. 318–322, 1994.

[15] C. Icking, R. Klein, and L. Ma, "An Optimal Competitive Strategy for Looking Around a Corner."

[16] Sun Microsystems, "Java3d Tutorial." http://java.sun.com/developer/onlineTraining/java3d/.

[17] Sun Microsystems, "Java3d API Documentation." http://download.java.net/media/java3d/javadoc/1.3.2/index.html.

[18] Sun Microsystems, "Java 1.5 API Documentation." http://java.sun.com/j2se/1.5.0/docs/api/.

[19] Bronstein, Semendjajew, Musiol, and Mühlig, *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 1999.

[20] "IEEE standard for Binary Floating-Point Arithmetic," 1985.