Algorithmische Geometrie
Wintersemester 2020/2021
Thomas Bläsius & Marcus Wilhelm

Karlsruher Institut für Technologie

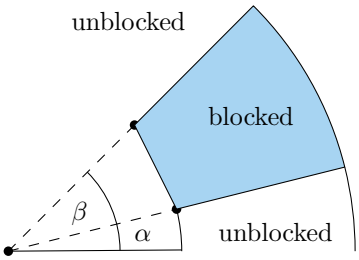
Musterlösung zum Übungsblatt 4

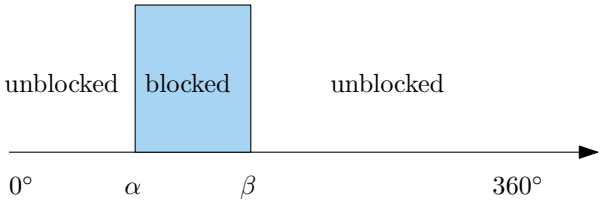
Erstellt von Robin Andre und Marius Hegele

Aufgabe 1: Mission Impossible

6 Punkte

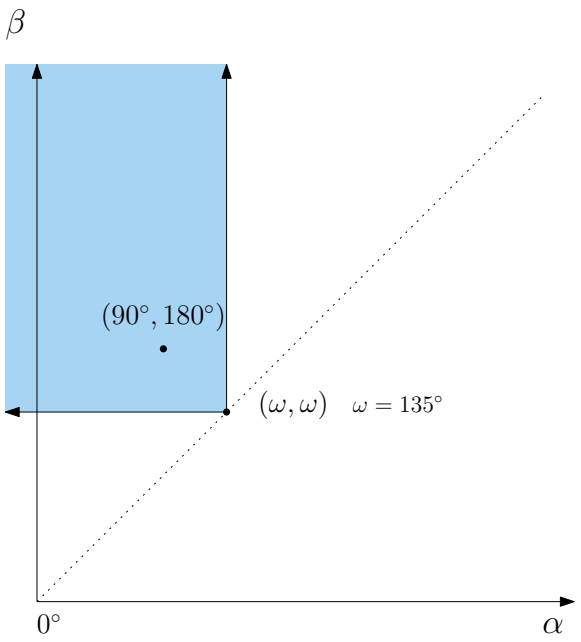
Aus der psychologischen Analyse seines Bruders *Brighton* konnte Ethan zwei Annahmen treffen: Erstens ist *Brighton* pendantisch und hat die Fallgrube exakt über dem Mittelpunkt installiert. Zweitens ist *Brighton* ein Superschurke mit Klasse und würde niemals eine Glaswand direkt unter der Fallgrube platzieren (Das wäre stillos und eine Riesensauerei). Aus seiner ALGO2-Ausbildung erinnert sich Ethan daran Probleme auf die essentiellen Elemente zu reduzieren.





Die erste Beobachtung ist dass die genaue Position der Scheiben irrelevant ist. Jede Scheibe lässt sich durch zwei Winkel α, β beschreiben. Als geschulter Algorithmiker erkennt Ethan sofort, dass jede Scheibe nur innerhalb des **Bereichs** $[\alpha, \beta]$ Einfluss auf seinen bescheidenen Laser trägt. Fasziniert von seiner Erkenntnis ignoriert Ethan für das Erste den Sonderfall, dass Scheiben auch seinen gedachten Nullmeridian kreuzen können und nimmt an dass $\alpha < \beta$

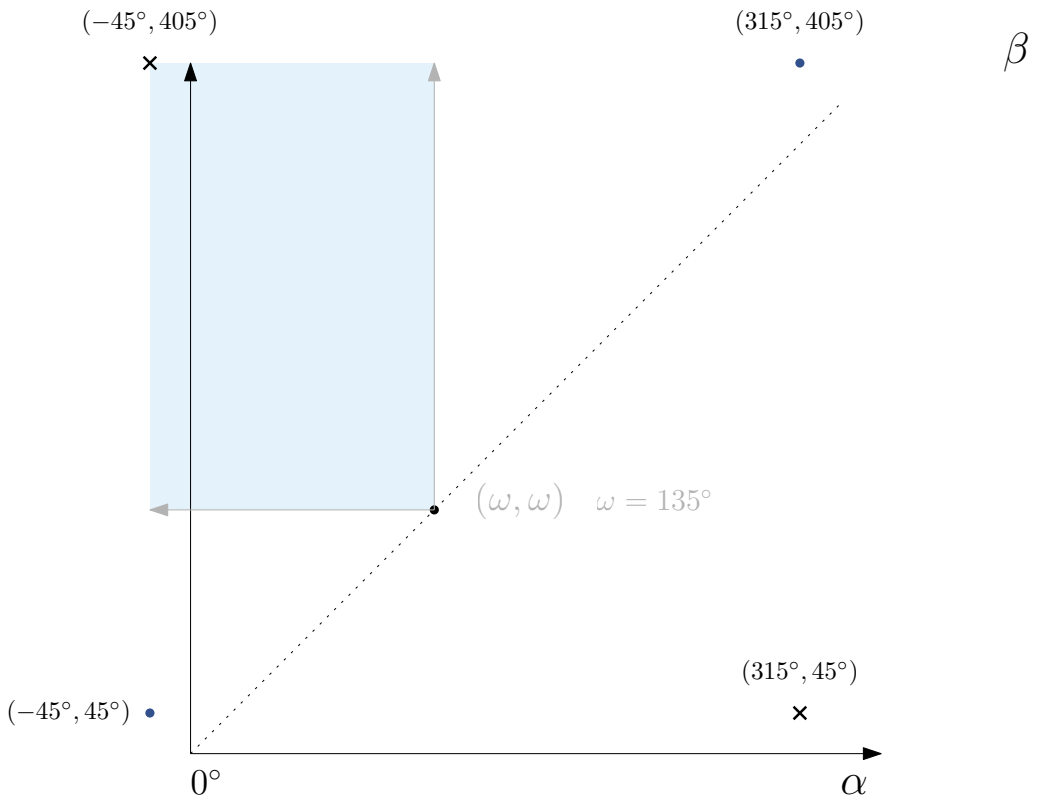
Bedauerlicherweise können die Scheiben trotzdem noch ziemlich willkürlich sein. Ethan zeichnet für eine Scheibe einen Punkt in ein 2D-Koordinatensystem. Ihn trifft die Erkenntnis: Ganz gleich welchen Winkel ω Ethan anpeilt, eine Scheibe hat nur dann Einfluss auf den Laser wenn $\alpha \leq \omega \leq \beta$.



In seinem Koordinatensystem entspricht dies einer **halben Bereichsanfrage** $[\omega, \infty) \times (-\infty, \omega]$. Ethan ist begeistert: Dies entspricht exakt seinen Beschränkungen durch Vorberechnungszeit $\mathcal{O}(n \log n)$, Speicher $\mathcal{O}(n)$ und der äußerst exotischen Panikreaktion $\mathcal{O}(\log n + k)$.

Doch wie soll der Agent Scheiben in das Koordinatensystem eintragen die seinen Nullmeridian kreuzen (Bsp. $\alpha = 315^\circ, \beta = 45^\circ$). Der Punkt (α, β) funktioniert nicht da er niemals in einer Bereichsanfrage liegen würde. Analog ist der Punkt $(\alpha - 360^\circ, \beta - 360^\circ)$ in zu vielen Bereichsanfragen (komplett inverse Antwort zu dem was unser Agent erreichen möchte). Tatsächlich muss unser Agent für diese Glasscheibe zwei Punkte in sein Koordinatensystem eintragen: $(\alpha - 360^\circ, \beta)$ und $(\alpha, \beta + 360^\circ)$

Dies ist zum Glück nur ein konstanter Faktor. Und wenn unser Agent mit konstanter (aber hinreichender) Menge an Luft und konstanter (aber unzureichender) Menge an Laserpointern eine Sache in ALGO2 gelernt hat, dann dass konstante Faktoren irrelevant sind.

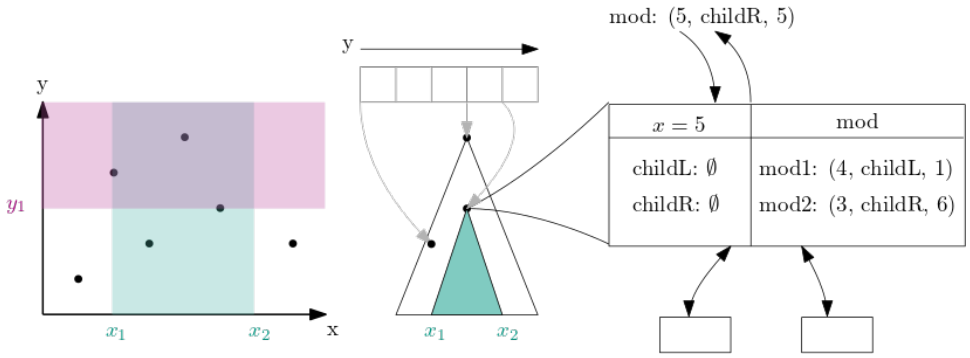


Wie kommt man drauf?

Diese Aufgabe lässt sich mit vielen unterschiedlichen Ansätzen lösen. Wichtig ist zu erkennen dass bei zyklischen Aufgabestellungen üblicherweise eine Winkelrepräsentation sich anbietet. Der nächste Schritt ist es einen Lösungsansatz für das Problem zu bestimmen. Wir haben uns für eine halbe Bereichsanfrage entschieden da der Anfangs- und Endwinkel einer Scheibe völlig willkürlich sein können, aber alle Anfangswinkel sich exakt gleich verhalten und alle Endwinkel sich exakt gleich verhalten. Dies legt nahe die Anfangswinkel als eigene Dimension zu betrachten und die Endwinkel ebenso als eigene Dimension zu betrachten.

Aufgabe 2: Anderthalbe Bereichsanfragen **6 Punkte**

In einem binären Suchbaum an x-Werten können alle k Punkte im Intervall $[x_1, x_2]$ gefunden werden, indem der "linke" minimale Knoten $\geq x_1$ und der "rechte" maximale Knoten $\leq x_2$ in jeweils $\mathcal{O}(\log n)$ bestimmt werden. Die relevanten k Knoten können dadurch gefunden werden, dass man an jedem der bereits bei der Suche traversierten Knoten (linke und rechte Beschränkung) bis zum höchsten gemeinsamen Knoten "nach innen läuft". Jeder Knoten, den man dabei findet, liegt im angefragten Intervall. Diese finale Suche ist somit linear in der Ausgabegröße k. Damit kostet die



$[x_1, x_2]$-Anfrage insgesamt $\mathcal{O}(\log n + k)$ Zeit.

Eine Anfrage auf $[x_1, x_2] \times [y_1, +\infty]$ lässt sich als Anfrage $[x_1, x_2]$ in einem partiell persistenetem Suchbaum, in den die Punkte nach absteigender y-Koordinate sortiert eingefügt worden sind, verstehen. Dabei interpretiert man die y-Koordinate als Zeitpunkt. Punkte $p = (p_x, p_y) : p < y_1$ werden bei der Suche nach dem linken und rechten x-Wert im partiell persisten Baum nicht gefunden, da sie zum Zeitpunkt y_1 nicht "existiert" haben.

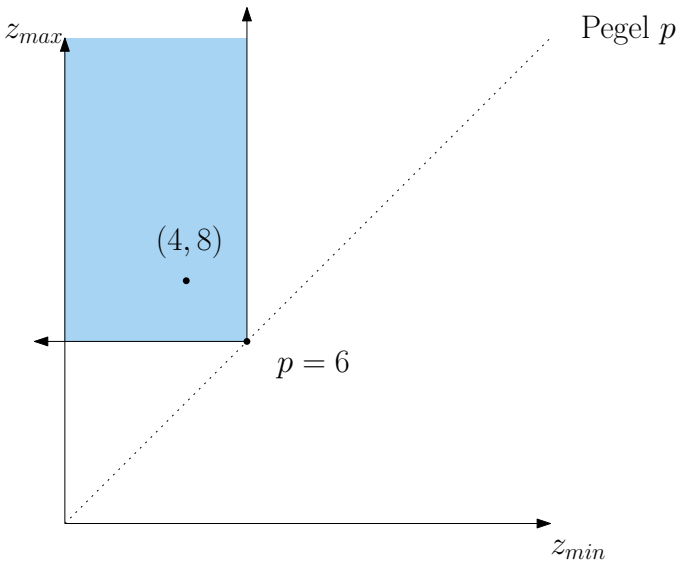
Die partielle Persistenz eines binären Suchbaums benötigt pro Suche amortisiert nur konstanten Zeit- und Platz-Overhead und kann somit vernachlässigt werden. Die Laufzeit beschränkt sich damit insgesamt auf $\mathcal{O}(\log n + k)$.

Wie kommt man drauf?

Eine $[x_1, x_2]$-Anfrage auf einem binären Suchbaum haben wir bereits in der Vorlesung gesehen. Das positiv offene y-Intervall lädt (vielleicht) dazu ein, y als Zeitdimension zu interpretieren. Der Rest ergibt sich über das Theorem zur partiellen Persistenz, wobei die Details gar nicht so relevant sind.

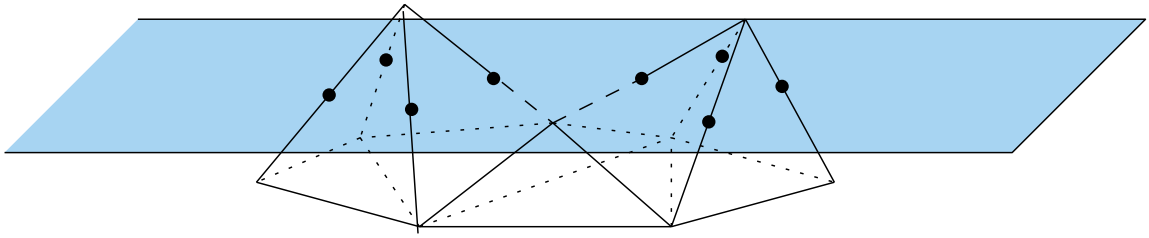
Aufgabe 3: Monster-Aufgabe **8 Punkte**

Für diese Aufgabe empfiehlt es sich wieder zu überlegen welche Informationen relevant sind. Für den Pegelstand ist lediglich die z-Koordinate interessant. Knoten eignen sich nur begrenzt als Informationsquelle. Tatäschlich sind die Kanten relevant: Jede Kante hat zwei z-Koordinaten. Seien diese nun z_{min} und z_{max} . Die Kante wird vom Wasserpegel p geschnitten genau dann wenn $p \in [z_{min}, z_{max}]$. Diese Information lässt sich auf ein 2D-Koordinatensystem übertragen.

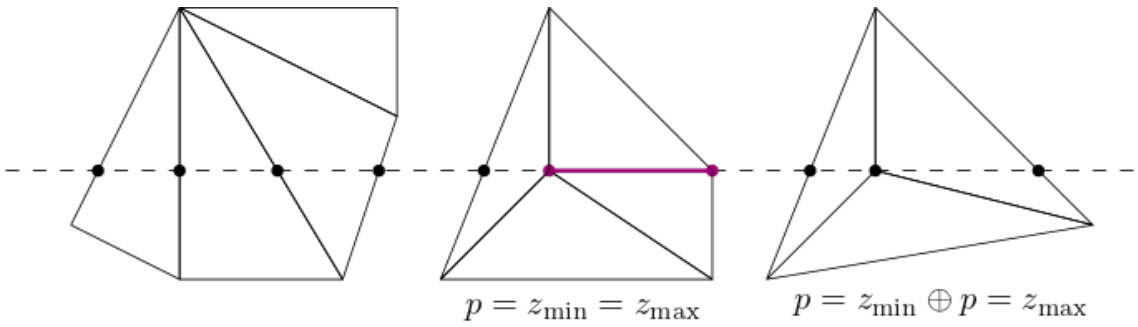


Mithilfe einer halben Bereichsanfrage lassen sich in $\mathcal{O}(\log n + k)$ durch Fractional Cascading alle Kanten finden die von dem aktuellen Pegel p geschnitten werden. Der Schnittpunkt lässt sich in jeweils konstanter Zeit berechnen.

Doch wie stellen wir fest, welche Schnittpunkte wie verbunden werden müssen um das Küstenlinienpolygon $\mathcal{P}$ zu vervollständigen?



Die Insel ist ein *triangulierter* geometrischer Graph. Im Fall, dass eine Kante des Graphen nicht in einem der Endpunkte geschnitten wird: Über die anliegenden dreieckigen Facette ist der Schnittpunkt (die geschnittene Kante) mit je einem weiteren Schnittpunkt gepaart. Gepaarte Schnittpunkte werden in $\mathcal{P}$ miteinander verbunden.



Sonderfall 1: $p = z_{\min} = z_{\max}$, d.h. der Pegel liegt auf einer Kante. Im Hilfsgraphen für die Bereichsanfragen liegen diese Kanten als Hilfspunkte auf der Diagonalen. Anliegende Facetten werden in diesem Fall überhaupt nicht geschnitten. Dieser Fall muss also separat behandelt werden: die Kante wird direkt in $\mathcal{P}$ übernommen.

Sonderfall 2: $p = z_{\min} \oplus p = z_{\max}$, d.h. der Schnittpunkt ist ein Punkt im Graph, die zugehörigen Kanten verlaufen jedoch nicht parallel zur x-Achse (wie in Sonderfall 1). Die Anzahl der am Punkt anliegenden Facetten kann beliebig groß sein. Es gilt also zu bestimmen welche der anliegenden Facetten geschnitten werden. Ausgeschlossen es handelt sich hier um Sonderfall 1, so existiert eine geschnittene Kante, die mit dem geschnittenen Punkt über die anliegende Facette gepaart ist. Tatsächlich lässt sich während der Vorberechnung für jeden Knoten die entsprechend zugehörigen Kanten berechnen, für welche der Punkt der mittlere Punkt des jeweils zugehörigen Dreiecks ist. Dies ist innerhalb der Speicherbegrenzung realisierbar da die Menge an Kanten $m \in O(n)$ für triangulierte Graphen ist und innerhalb der Zeitbegrenzung realisierbar da nur $O(m)$ Kanten betrachtet werden.

In allen Fällen können wir allerdings aufgrund der Triangulierung für jeden Knoten den Folgeknoten in $\mathcal{P}$ in $\mathcal{O}(1)$ mihilfe der bekannten Operationen `next(e)`, `face(e)`, `twin(e)` aus der Standard-Graphdatenstruktur bestimmen. Mittels der Folgeknoten lässt sich $\mathcal{P}$ eindeutig zeichnen.

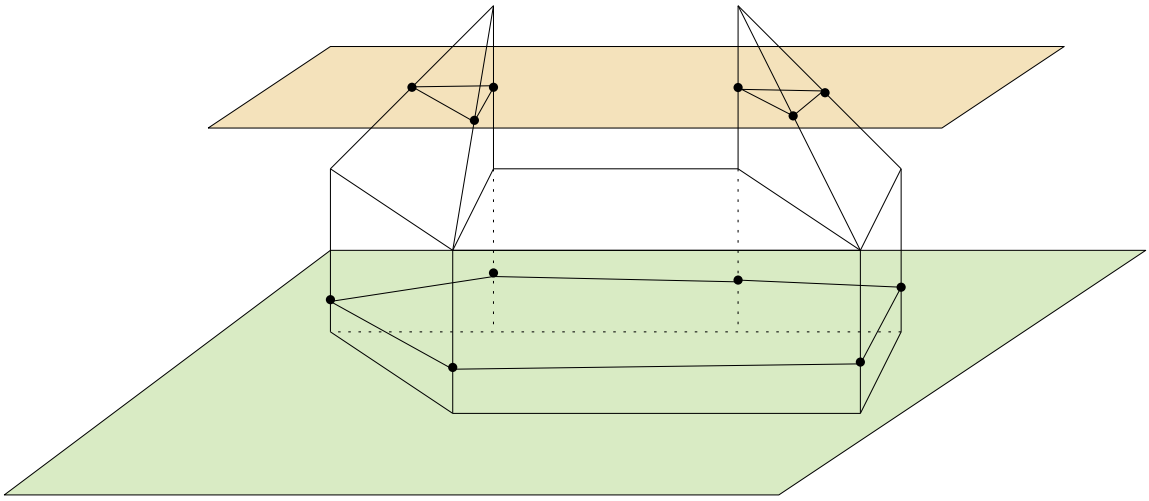


Abbildung 1: Beispielschnitte mit 6 Schnittpunkten aber 1 bzw. 2 Zusammenhangskomponenten