Themengebiete „Geometrische Algorithmen“

- **Grundlagen**
  - **Pythagoras**
  - **Grundbegriffe der axiomatischen Geometrie**
    - Parallelen Axiom
- **Algorithmen**
  - **Konvexe Hülle**
    - Naiver Algorithmus
    - Inkrementeller Algorithmus
    - Graham Scan
  - **Liniensegmentschnittproblem**
    - Naiver Algorithmus
    - Sweepline Algorithmus
  - **Triangulation**
    - Art-Gallery Problem
    - Dualer Graph
    - Naiver Algorithmus
    - I-Monotone Polygone
  - **Lineare Programmierung**
    - Inkrementeller Algorithmus
  - **Orthogonale Bereichssuche**
  - **Point-Location-Problem**
    - Streifenalgorithmus
    - Trapezoidalgorithmus
    - Randomisierter inkrementeller Algorithmus
  - **Erneut Konvexe Hülle**
    - Konvexes Set/Extrem Punkte
    - Äquivalente Definitionen
    - Lower Bound zur Berechnung der CH
    - Links und Rechtkurven
    - Point Pruning
    - Edge Pruning
    - Jarvi’s March
    - Graham Scan
    - Quick Hull
    - Merge Hull
    - Inkrementelles randomisiertes Sortieren
    - Inkrementeller randomiserter Algorithmus zur Berechnung der CH
  - **Voronoi Diagramme**
    - D&C Algorithmus
    - Hierarchische Triangulation mittels VD
    - Closest Pair Problem
    - All nearest Neigbours Problem
    - MST nach Kruskal
  - **Duality Transformation**
    - Level von Punkten
    - Inkrementeller Algorithmus
    - Zonentheorem
  - **Halbebenendiskrepanz**
    - Raytracing
  - **Delaunay Triangulation**
    - Illegalitätstest auch graphisch
    - Inkrementeller Algorithmus
  - **Rechteckschnittproblem**
    - Segmentbäume
    - Intervallbäume
    - Priority Search Trees
      - Binärbäume/Minheaps
  - **Hidden Line Elimination**
    - 3 Klassen

        - Plane Sweep
        - Range Tree
        - Bedeckungszahl
        - Dynamic Countour Maintenance
        -
- **Datenstrukturen**
    - Doppelt verkettete Kantenliste
    - Binärer Suchbaum
    - Kd-Bäume
    - 2-Dimensionale Bereichssbäume (Range Trees)
    - Segment Bäume
    - Intervallbäume
    - Priority Search Trees
    - Min-Heaps
    - Range Trees

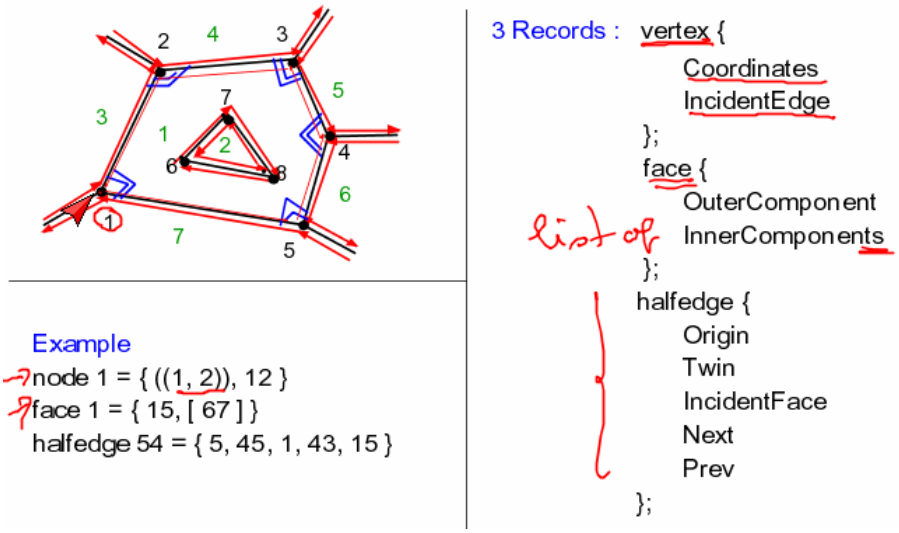
Fragenkatalog Geometrische Algorithmen

1. **Beweise den Satz des Pythagoras geometrisch!** [Stichwort Quadrat, Schiefes Quadrat, darin 4 rechtwinklige Dreiecke mit der Hypothenuse als Kanten des Quadrats. In der Mitt wieder ein Quadrat Kantenparallel zum äußersten. Die Hypothenuse des Dreiecks sei z, die eine Kathete x und die andere y. Die Gesamtfläche der 4 Dreiecke + das Quadrat in der Mitte ist z^2, eine Kante des Quadrate in der Mitte ist y-x und damit ist die Fläche des inneren Quadrates $(y-x)^2$. Die Flächen der 4 Dreiecke zusammen sind 4*((x*y)/2) also 2xy. ➔ $z^2=2xy+(y-x)^2$ ➔ $z^2=2xy+y^2-2xy+x^2$ ➔ $z^2=x^2+y^2$ qed.]
2. **Was sind die fundamentalen Begriffe der axiomatischen Geometrie?** [Punkt, Gerade, Ebenen und Inzidenzrelationen (wie z.B. „liegt auf", „geht durch")]
3. **Was sind die 3 Axiome der Geometrie aus den Elementen von Euklid?** [1. Für zwei Punkte Q und P gilt, es gibt genau eine gerade g die durch beide Punkte geht | 2. Für jede Gerade g gilt, es gibt einen Punkt der nicht auf g liegt | 3. Für jede Gerade g und jeden beliebigen Punkt P welcher nicht auf g liegt, gibt es genau eine Gerade h die keinen gemeinsamen Punkt mit g hat (also parallel zu ihr ist]
    a. **Beweise die Unabhängigkeit des Parallelen-Axioms (Axiom 3)!** [Man entwerfe ein Welt in der Axiom 1 und 2 gilt, aber Axiom 3 nicht. ➔ Man nehme einen Kreis und nur die Punkte im Kreis. Geraden sind Sehnen in dem Kreis. Man kann das nun so konstruieren, dass A1 und A2 gelten, A3 aber nicht, A1 ist einfach zu konstruieren 2 Punkte miteinander zu verbinden. A2 ist auch einfach, denn man kann einen beliebigen Punkt neben die Gerade eben legen. A3 Durch die Begrenzung des Kreises kann man nun eine gerade h auch so legen, dass sie nicht parallel ist zu g, denn der eigentliche Schnitt liegt ausserhalb des Kreises. Klein argumentiert nun so: Wenn wir aus A1 und A2 das Axiom A3 folgern könnten, dann dürfte es eine Welt, wie die eben beschriebene nicht geben.]
    b. **Welches Model wird für den Beweis benutzt?** [Das Modell von Klein]
4. **Was ist eine konvexe Hülle?** [Die Konvexe Hülle ist eine Teilmenge von Knoten von S in der Ebene. Man kann diese Punkte nun überschneidungsfrei so miteinander verbinden, dass alle anderen Punkt in S innerhalb dieser „Hülle" liegen. Diese Hülle hat die Eigenschaft, dass sie konvex ist, d.h. jeder Kantenzug beschreibt im Uhrzeigerseinn eine Rechtskurve]
5. **Wie kann man die Konvexe Hülle naiv berechnen? Laufzeit?** [Lösung mit der Rechts Regel: Ein Liniensegment ist dann Teil der konvexen Hülle, wenn alle Punkte von P rechts von dieser Linie liegen. Man nimmt also jedes Punktepaar ($O(n^2)$) und vergleicht mit allen anderen Nachbarn, ob diese rechts davon liegen ($O(n^2)*O(n)$➔ $O(n^3)$)]
6. **Wann nennt man einen Polygonzug „Simpel", wann „konvex**"? [Simpel: Wenn er sich nicht selber schneidet (kreuzt). Konvex: Wenn das eingeschlossene Gebiet Konvex ist]
7. **Nennen Sie einen inkrementellen Algorithmus zur Berechnung der konvexen Hülle?** [Erster Schritt: Aufteilen in Upper-Hull und Lower-Hull. Sortiere die Punkte in aufsteigender x Richtung. Zweiter Schritt Man beginne mit 2 Punkten und verbinde diese dann nehme man inkrementell immer einen Punkt hinzu und aktualisiere die berechnete obere Hülle. (im Prinzip Graham Scan -> Antwort nächste Frage).]
8. **Was ist der Graham Scan?** [Berechnung der oberen Hülle. Punkte in aufsteigender x-Richtung sortiert. Man nimmt sukzessive einen Punkt hinzu. Sobald man 3 Punkte hat, testet man, ob diese eine Rechtskurve bilden, ist dies der Fall, so wird ein neuer Punkt hinzugenommen und dieser und die letzen zwei Punkte auf Rechtskurve geprüft. Bilden die Punkte keine Rechtskurve, dann muss der erste und der dritte Punkt miteinander verbunden werden.Der 2. Punkt wird dann aus der Liste potenzieller Kandidaten gestrichen. Bevor wir nun mit einem neuen Punkt fortsetzen können, müssen wir die nun übriggebliebenen 3 letzten Punkte auch wieder auf Rechtskurve überprüfen und ggf. wieder den mittleren Streichen. Sobald das Rechtskurvenkriterium wieder erfüllt ist, können wir einen neuen Punkt hinzufügen. Das Ganze ist für die untere Hülle analog.]
9. **Wie schnell kann man die Konvexe Hülle damit berechnen?** [$O(n \log n)$ ➔ Sortieren der Punkte dauert $O(n \log n)$, Die For-Loops benötigen nur $O(n)$]
10. **Was versteht man unter dem Liniensegmentschnittproblem?** [Gegeben ist eine Menge von Liniensegmenten. Gesucht sind alle Schnittpunkte sich schneidender Liniensegmente. Der Schnitt zweier Linien kann jeweils in $O(1)$ berechnet werden]
11. **Anwendungsgebiete?** [z.B. Geo Informationssystem bei denen Flusskarten und Straßenkarten übereinander gelegt werden. Man finde Schnitt von Strassen und Flüssen, um z.B. herauszufinden wo man Brücken bauen muss.]

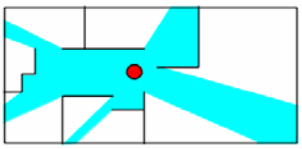
12. **Wie funktioniert der naive Algorithmus? Laufzeit?** [Beim naiven Algorithmus wird einfach jedes Liniensegment mit jedem anderen geschnitten, d.h. in $O(n^2)$. Das Ziel ist es aber einen Algorithmus zu finden der eine Laufzeit in Abhängigkeit der tatsächlichen Schnitte hat, der Output-Sensitiv ist]
13. **Wie funktioniert der Sweepline Algorithmus zur Berechung des Liniensegmentschnittproblems?** [Man sortiere alle Start und Endpunkte in absteigender y-Richtung des Sets Q, diese definieren dann die Haltepunkte der Sweep-Line. Desweiteren definiere man sich ein Set T, welches die jeweils aktiven Segmente enthält, das heisst diejenigen, die gerade von der Sweepline geschnitten werden. In unserem Set Q werden die Anfangspunkte jeweils mit einem Punkt vor dem Namen der Linie und die Endpunkte mit einem Punkt nach dem Namen markiert (z.B. H. für den Endpunkt von H). In die Statusstruktur T werden nun nach und nach die jeweils geschnittenen Punkte (mit der Sweep-Line) eingetragen. Dabei werden diese jeweils so eingefügt, dass die x-Reihenfolge auch in T erhalten bleibt. D.h. wenn die x Koordinate eines Elementes kleiner ist als ein bereits vorhandenes, wird es links von diesem abgelegt. Nachbarn die sich neuerdings in diesem Schritt sehen können, werden reportet und jeweils markiert durch ein „!“ das macht man in dem man schaut, an welcher Stelle das neue Element eingefügt wird. Wird es zwischen zwei Segmenten eingefügt, dann gibt es zwei neue Nachbarschaftspaar zu berichten. Entfernt man einen Endpunkt, betrachtet man vor dem Entfernen den jeweils linken und rechten Nachbarn dieses Punktes und berichtet diese Beiden als nun neue Nachbarn. Da man auch bei Schnittpunkten mit der Sweep-Line anhalten muss (=> Einträge in der Statusstruktur vertauschen), muss man diese in der Event-Queue an der richtigen Stelle (y-Koordinate ist entscheident) eintragen. Diese y-Koordinate ist aber immer größer als die aktuelle.]
    a. **Was wird in der Event-Queue bzw. in der Status-Struktur gespeichert?** [Event Queue: Alle Start (.H) und Endpunkte (H.) der Liniensegment in absteigender y-Richtung sortiert (während des Algorithmuses kommen dann noch die Schnittpunkte hinzu) | Status-Struktur: Die jeweils von der Sweepline geschnittenen Segmente. Jeweils neu hinzugekommene Nachbarn, die sich sehen können, werden berichtet und mit einem „!“ markiert.]
    b. **Welche Operationen benötigt man für die Event-Queue?** [Initialisierung (Sortieren) ➔ O(m log m) m=2n wobei n die Anzahl der Segmente ist| deletemin ➔ O(log m): Nächster zu betrachtender Haltepunkt | Einfügen ➔ O(log m) : Schnittpunkte]
    c. **Welche Datenstruktur kann man für die Event-Queue verwenden?** [Balancierter Suchbaum wobei ein eingefügter Punkt dann kleiner als ein anderer ist, wenn die y-Koordinate kleiner ist. Wenn die y-Koordinaten identisch sind, wird nach der x-Koordinate entschieden.]
    d. **Welche Operationen benötigt man für die Status-Struktur?** [Initialisierung | Einfügen | Entfernen | Nachbarn finden | (Reihenfolge-Änderung)]
    e. **Welche Datenstruktur kann man für die Status-Struktur verwenden?** [Balancierter Suchbaum➔ Blätter sind nach x Richtung der Start und Endpunkte gespeichert ➔ Alle Operationen gehen dann in O(log n)]
    f. **Wenn n die #segmente ist, und k die #schnitte wie lassen sich dann Operationen auf die Event-Queue und die Status-Struktur nach oben abschätzen?** [Operationen auf die Eventqueue bzw. Statusstruktur sind es höchstens 2n+k viele ➔ Jedes Segment hat einen Anfangs und einen Endpunkt (deswegen 2n) . Die k Schnittpunkte werden auch in der Eventqueue gespeichert,]
    g. **Von was hängt die Laufzeit ab?** [Von der Anzahl der Schnitte ➔ O((n+k)log n)]
    h. **Erkläre das Theorem des Linien-Segment-Schnitts (Laufzeit, Platz)** [Sei S ein Set von n Liniensegmenten in der Ebene. Alle Schnittpunkte von S die ein Segment mit allen anderen hat, können in O(n log n + k log n) Zeit und O(n) Platz berichtet werden, wenn k die Größe der Ausgabe ist.]
    i. **Welche Spezialfälle gibt es falls sich zwei Segmente schneiden?** [Der Schnittpunkt ist gleichzeitig Start- bzw. Endpunkt eines Segmentes | Der Schnittpunkt liegt innerhalb mehrerer sich schneidenen Segmente. In der Statusstruktur müssen dann ggf. die Reihenfolgen geändert werden.]
        i. **Was muss man in einem Schnitt tun, bei dem sich mehrere Segmente schneiden?** [Die Reihenfolge der beteiligten Segmente in der Statusstruktur invertieren]
14. **Was besagt das Nachbarschaftslemma?** [Wenn s_i und s_j zwei nicht horizontale Linien sind, die sich in einem Punkt p schneiden. Dann gibt es einen Eventpoint oberhalb von p ab welchem s_i und s_j benachbart (d.h. adjazent) werden und somit auf Schnitt getestet werden.

D.h. man läuft mit der Sweep-Line nie an einem Schnittpunkt vorbei, ohne diesen zuvor erkannt zu haben.]

15. **Erkläre die Struktur der doppelt verketteten Kantenliste (Doubly connected edge list)**? [Man definiert sich 3 Records ➔ 1.: Node: Enthält die Koordinaten des Knoten und eine anliegende (inzidente) Kante ➔ 2.: face: Enthält die äußere Komponente einer Fläche, repräsentiert durch eine Kante dieser Fläche. Und es enthält die inneren Komponenten (Löcher dieser Fläche auch wieder repräsentiert durch jeweils eine Kante dieser Komponenten) ➔ 3.: Halfedge: Origin (Startknoten). Twin: die Kante die in die umgekehrte Richtung zeigt. Incident Face (diejenige Fläche, die an dieser Halbkante links davon liegt. Next (Nächste Halbkante) Prev (vorherige Halbkante) ➔ Man kann z.B. über den Nextpointer die Kanten einer Fläche von einem beliebigen Startpunkt aus gegen den Uhrzeigersinn durchlaufen (Prev analog in umgekehrter Richtung) Etc.]

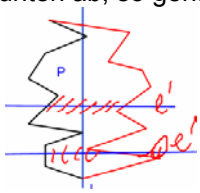


16. **Wie überlagert man zwei Doubly Connected Edge Lists?** [Sweepline Verfahren. Jeder Schnittpunkt wird nun ein neuer Datensatz in unserem Record.]
17. **Wie schnell kann man boolsche Operationen (Vereinigung, Schnitt, Subtraktion) für 2 gegebene Polygone berechnen?** [Wenn $n = |P_1| + |P_2|$ dann können alle 3 Operationen in $O(n \log n + k \log n)$ berechnet werden, wenn k die Größe der Ausgabe ist.]
18. **Was versteht man unter Triangulation?** [Das Einteilen eines Polygons in Dreiecke]
    a. **Ist die Triangulation eindeutig?** [Nicht eindeutig]
    b. **Beweis?** [Bei z.B. einem Viereck kann man auf 2 Arten triangulieren.]
19. **In exakt wie viele Dreiecke kann man ein beliebiges einfaches Polygon immer aufteilen?** [n-2 Dreiecke➔ n ist die Anzahl der Knoten]
    a. **Beweis?** [Ein Polygon mit 3 Knoten ist ein Dreieck, d.h. n-2 ist erfüllt. Nimmt man nun einen weiteren Knoten hinzu, kommt immer ein weiteres Dreieck mit dazu. Usw. qed]
20. **Was ist das Art-Gallery Problem?** [Das Bewachen einer Kunstausstellung, so dass jede Ecke einsehbar ist.]
    a. **Was ist dabei das Sichtbarkeits-Polygon (Visibility Polygon)?** [Es beschreibt diejenigen Bereiche die von einem Wächter eingesehen werden können.]

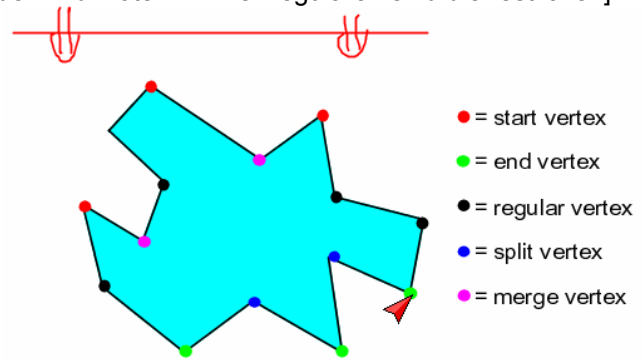


21. **Man benötigt also für jedes Dreieck des Polygons höchstens eine Wache, aber es geht auch mit weniger, wie?** [Das ist intuitiv schon klar, denn eine Wache aufgestellt an den Eckpunkten von zweien oder mehreren Dreiecken, kann mindestens diese auch mit einsehen. ➔ Dreifärbung des Triangulierten Polygons. Man nimmt dann nur noch die Wächter einer Farbe die minimal vorkommt➔ $\lfloor n/3 \rfloor$ Wächter erforderlich)]
22. **Was ist der Duale Graph eines Triangulierten Polygons?** [Ein Baum, der dadurch zustande kommt, dass man in jedes Dreieck einen Knoten legt und die Knoten benachbarter Dreiecke jeweils verbindet. Die o.g. Dreifärbung beginnt man nun bei dem Dreieck welches einen Blattknoten des Dualen Graphs besitzt]

23. **Was besagt das Theorem über die 3-Färbbarkeit von einfachen Polygonen? Beweis?** [Einfache Polygone sind immer 3-färbbar. Beweis: Der duale Graph ist ein Binärbaum, was heißt das wir eine 3-Färbung mit einem simplen DFS (Depth First Search: Tiefensuche) finden können.]
24. **Wieviele Wachen benötigt man nun für das Art-Gallery Problem höchstens (Worst-Case)?** [n/3 ➔ bei diesem Polygon]
25. **Wie schnell kann man naiv triangulieren?** [O(n²)]
26. **Wie kann man ein Konvexes Polygon triangulieren?** [Trivial: Man nehme einen beliebigen Knoten und verbinde diesen mit jedem anderen Knoten des Konvexen Polygons]
27. **Wann ist ein Polygon l-Monoton?** [Konvexe Polygone wären sehr einfach zu triangulieren, da das Aufteilen eines normalen Polygons in konvexe Teilpolygone aber genauso komplex ist, wie wenn man es direkt berechnen würde gibt es eine einfachere Variante, die sogenannten L-Monotonen Polygone ➔ Ein Polygon ist dann L-Monoton, wenn man eine Vertikale durch das Polygon legen kann, auf welcher man zu dieser Vertikalen senkrechte Linien ziehen kann, die in einem Stück in das Polygon passen, also nur 2 mal vom Polygon geschnitten werden. Läuft man ein solches geteiltes Polygon nun von oben nach unten ab, so geht man im Prinzip nur abwärts!]

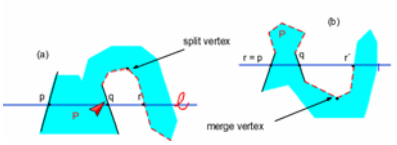


28. **Wann heisst ein Polygon Y-Monoton?** [Ein L-Monotones Polygon heisst Y-Monoton, wenn L die Y-Achse ist]
29. **Was ist ein Split-/Mergeknoten?** [Splitknoten: Die Stelle an der das Polygon auseinandergeht ➔ Analog dazu der Mergeknoten sind diejenigen, an denen zwei Teile eines Polygons zusammenlaufen]
    a. **Welche anderen Knotentypen gibt es?** [Startknoten: Diejenigen Knoten, an denen die Sweepline ein Y-Monotones Teilpolygon, das erste mal durchläuft ➔ Analog dazu der Endknoten ➔ Die Regulären sind die restlichen]



30. **Was ist der Helper**? [Der Helper einer Kante e_j ist der niedrigste Knoten oberhalb der Sweepline, der verbunden mit e_j eine Kante bildet, die vollständig im Polygon liegt
31. **Wie teilt man ein Polygon in y-monotone Teile auf?** [Die Split und Mergeknoten müssen eliminiert werden, in dem man Diagonalen nach oben bzw. nach unter zu einem anderen Knoten einzeichnet, wie geht das nun, ohne dass man das Polygon schneidet? Splitknoten:➔ Man nehme die linkeste Kante die an dieser Stelle von unserer Sweepline geschnitten wird. Man sucht sich nun den niedrigsten Punkt der noch im Bereich rechts dieser Kante liegt und nennen ihn Helper. Den Helper können wir nun bedenkenlos mit dem gefundenen Splitknoten verbinden. Mergeknoten: Entweder gleich in dem man einen Sweep von unten macht, geschickter wäre es wenn wir das gleich in unserem Downsweep erledigen könnten, das Problem ist aber, dass wir den unteren Bereich ja noch nicht kennen. ➔ Wir sagen also nun dieser Mergeknoten sei der Helper der gerade links geschnittenen Aussenkante. Entdeckt man nun einen neuen Knoten weiter unter der sich als Helperknoten für die noch geschnittenen linkeste Kante qualifiziert, so wird der Helper geupdatet und mit dem o.g. Mergeknoten verbunden ➔ Einen Schnitt mit einer anderen Kante kann man ausschließen!!!

32. **Welche Knotentypen besitzt ein y-monotones Polygon nicht?** [Split und Mergeknoten, diese werden ja eleminiert ➔ Umgekehrt heisst das, dass wenn ein Polygon nicht Y-Monoton ist, dann gibt es sicher einen Schnitt durch das Polygon, welcher aus mindestens zwei Teilen besteht]



33. **Welcher Zeit- und Speicherplatzbedarf ist erforderlich um ein einfaches Polygon in y-monotone Teile zu zerlegen?** [O(n log n) Zeit und O(n) Platz]
34. **Wie schnell kann man nun die unterschiedlichen Poylgone Triangulieren?** [Y-Monotones Polygon in O(n) Zeit | Ein einfaches Polygon in O(n log n) Zeit auf einem Algorithmus der O(n) Platz benötigt, wobei n die Anzahl der Knoten ist.]
35. **Was ist lineare Programmierung?** [Es geht dabei um die Optimierung einer so genannten Optimierungsfunktion, z.B. die Produktion bestimmter Güter unter der Verwendung bestimmter Rohstoffe. Jedes Gut hat nun bestimmte Bestandteile. Die verfügbaren Rohstoffe begrenzt durch die Reserve können wir nun durch Geradengleichungen darstellen. Die Geraden begrenzen Halbebenen, wir müssen nun jeweils Punkte berechnen, die auf der Halbebene liegen, Am einfachtsen berechnen wir jeweils die x, bzw. y-Nullstellen. Alle Geraden begrenzen nun eine bestimmte Fläche. Nun betrachten wir unseren Optimierungsvektor, welcher durch unsere Optimierungsfunktion bestimmt ist. Nun müssen wir also innerhalb unserer begrenzetn Fläche einen Wert finden, der in Optimierungsrichtung maximal wird. Diesen findet man, indem man eine Orthogonale an den Optimierungsvektor anlegt und die so weit in Optimierungsrichtung schiebts, dass diese die begrenzte Fläche gerade noch tangiert. Dieser Schnittpunkt ist dann unsere gesuchter Maximalwert.]
    a. **Was ist die Optimierungsfunktion?** [Diejenige Funktion die es gilt zu maximieren.]
    b. **Wie kann man diese maximieren?** [Man erzeugt aus den Randbedingungen Geradengleichungen. Diese begrenzen dann ein Gebiet in unserer Ebene. Mit Hilfe der Optimierungsfunktion und einer zu ihr orthogonalen finden wir nun in Maximierungsrichtung (Richtung des Optimierungsvektors) einen maximalen Wert]
    c. **Was sind die Bedingungen? Was beschreiben diese?** [Die Bedingungen sind z.B. die gegebenen Rohstoffe pro Produkt begrenzt durch die Reserve, diese kann man als Geradengleichungen beschreiben die dann die Ränder von Halbebenen beschreiben. Beispiel:

Production of two goods A and B using four raw materials
Value of A: 6 CU, value of B: 3 CU

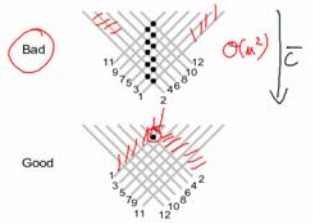
	Rm1	Rm2	Rm3	Rm4
Prod A	2	2	6	2
Prod B	4	1	2	2
Reserve	5	2	4	3

Maximize profit: $f_c(x) = 6x_A + 3x_B$ under the conditions:

$2x_A + 4x_B \leq 5$
$2x_A + 1x_B \leq 2$
$6x_A + 2x_B \leq 4$
$2x_A + 2x_B \leq 3$
$x_A, x_B \geq 0$

]
    d. **Wie findet man dann die maximale Lösung?** [Mit Hilfe der begrenzten Fläche, dem Optimierungsvektor und einer zu ihm orthogonalen]
    e. **Was ist die Dimension?** [Die Dimension in unserem war 2 da wir nur 2 Produkte hatten. Man kann das Problem aber theoretisch auf beliebig viele Produkte ausweiten. Der Unterschied ist dann nur das wir z.B. bei einer Dimension von 3 nicht Halbebenen, sondern Halbvolumen schneiden. Es entsteht somit beim Schnitt ein 3-dimensionaler Körper, bei dem es gilt mit dem Optimierungsvektor die richtige Ecke des Körpers zu finden, falls eine existiert. Im 4-dimensionalen lässt sich dies nicht mehr so einfach geometrisch beschreiben.]

f. **Was ist der Optimierungsvektor bzw. Optimierungsrichtung, und in welche Richtung geht sie?** [In Maximierungsrichtung]
g. **Wie sieht eine begrenzte, unbegrenzte und leere Region aus, und welche Auswirkungen haben diese auf die Optimierung bzw. Lösung?** [begrenzt: Genau eine Lösung, falls der Schnitt in einer Ecke ist bzw. beliebig viele, wenn der Schnitt auf einer Kante liegt (In diesem Fall nimmt man üblicherweise die Lexikographisch kleinste Kante. | unbegrenzt: beliebig viele Lösungen | Leere Region: Keine Lösung]
h. **Wie funktioniert der inkrementelle Algorithmus?** [Man nimmt die Halbebenen nach und nach hinzu. Wir berechnen nun jeweils die optimale Ecke bei Hinzunahme der i-ten Halbebene jedes mal. Es gibt nun 2 Fälle bei der Hinzunahme einer neuen Halbebene. Liegt die bisher berechnete optimale Ecke innerhalb der neuen Halbebene, dann ändert sich nichts! Liegt nun die bisherige optimale Ecke nicht innerhalb der neuen Halbebene, dann muss die optimale Ecke natürlich neu berechnet werden, da unsere Bedingungen jetzt restriktiver bzgl. der Ecke ist.]
    i. **Wieso sollte man diesen randomisieren?** [damit es nicht passiert, dass der optimale Punkt jedes Mal sich verändert bei der Hinzunahme einer neuen Halbebene. Dies geschieht dann, wenn jede neue Halbebene nur etwas restriktiver ist ➔ $O(n^2)$ | Best Case O(n). Deshalb Randomisierung in O(n)]
    ii. **Worst-Case Folge?**



i. **Mit welchem Zeit und Platzbedarf kann man das d-dimensionale Lineare Programmierungsproblem mit n Randbedingungen lösen?** [in O(d!n) Zeit und O(n) Platz)

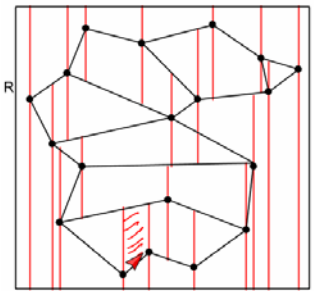
36. **Was ist orthogonale Bereichssuche?** [Rangesearch im d-dimensionalen Bereich, d.h. man sucht z.B. nach allen Leuten aus einem bestimmten Gehaltsbereich, einer bestimmten Bereichsmenge von Kindern und in einem bestimmten Altersbereich. Im 2-dimensionalen hat man also nur 2 Bereiche in denen man suchen will, im d-dimenionalen d.]
37. **Wie sieht ein binärer Suchbaum aus?** [Die Knoten beinhalten die Schlüssel, die Blätter begrenzen nun Intervalle zwischen den Schlüsseln]
38. **Wie sieht ein binärer Suchbaum aus?** [Schlüssel in den Blättern und „Wegweiser“ in den Knoten ➔ Beide Repräsentationen sind äquivalent!]
39. **Wie funktioniert die 1-dimensionale Bereichssuche in einem binären Blattsuchbaum?** [Wir suchen jeweils die 2 Intervallgrenzen. Bei der Suche nach den Intervallen ist man zunächst für beide Intervallgrenzen auf dem gleichen Pfad. Irgendwann teilen sich die Pfade. Diesen Punkt nennt man Split-Knoten. Wir müssen dann noch die (log n) Teilbäume betrachten um die Werte innerhalb des Intervalls zu finden]
    a. **Wie ist die Laufzeit?** [O(log n)]
40. **Gib den Inhalt des Theorems zur 1-dimensionalen Bereichsuche mit Benutzung von 1-dimensionalen Bereichsbäumen wieder!** [Ein 1-dimensionale Bereichssuche in einem Set von n Punkten, kann man mit einem 1-dimensionalen Range Tree in O(log n + k) Zeit beantworten, wobei k die Anzahl der Punkte im Bereich ist]
    a. Wie lange braucht man für ...
        i. **.. das Finden eines Splitt-Knoten?** [O(log n)]
        ii. **.. das Suchen eines Blattes?** [O(log n)]
    b. **Was ist die Laufzeit somit insgesamt, wenn k Punkte in dem entsprechenden Bereich liegen?** [O((log n) +k)]
41. **Zusammenfassung (Einsetzen):** Sei P eine Menge von Punkten im 1-dimensionalen Raum. Die Menge P kann in einem Balancierten Binärsuchbaum gespeichert werden, der O(n) Platz benötigt und in O(n log n) Zeit konstruiert werden kann, so dass eine Bereichsanfrage in Zeit O(k + log n) berichtet werden kann, wobei k die Anzahl der berichteten Punkte ist.
42. **Wann sind Punkte in allgemeiner Lage?** [Antwort: keine gleichen X bzw. Y-Koordinaten]

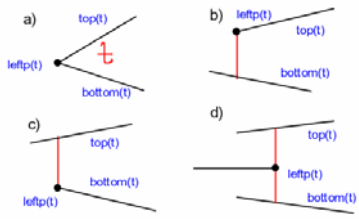
43. **Was sind kd-Bäume, und nach welchem Prinzip konstruiert man sie?** [Im Prinzip sind sie wie die 1-dimensionales Bereichssuchbäume, nur erweitert auf k dimensionen. Wir teilen nun im ersten Schritt bei der Konstruktion unseres Baumes die Menge von Punkten in der Ebene (!) in der Mitte nach X-Koordinate auf und schreiben den Median an die Wurzel und weisen, die Werte kleiner als der Median dem linken Blatt zu und die Werte größer als der Median dem rechten Blatt]. Nun setzen wir rekursiv fort, im nächsten schritt teilen wir nun aber jeweils nach der y-Koordinate auf! ➔ Abbruchbedingung, wenn nur noch ein Punkt in der Ebene ist.]
44. **Wie viel Platz braucht man bei der Konstruktion eines 2d-Baum (k=2)?** [O(n)]
45. **In welcher Zeit lassen sich 2d-Bäume konstruieren?** [O(n log n]
    a. **Beweis?** [Aufteilen nach Median geht jeweils O(n), die Tiefe ist log n ➔ O(n log n)]
46. **Was repräsentieren die Knoten eines kd-Baums?** [Sie repräsentieren Regionen. Der Wurzelknoten repräsentiert z.B. die gesamte Ebene. Der linke Sohn der Wurzel repräsentiert zum Beispiel die Halbebene links vom x-Wert des Wurzelknotens usw.]
47. **Wie funktioniert die Bereichssuche in einem kd-Baum?** [Suche der jeweiligen Regionen, die die Gesuchte Punktmenge enthalten ➔ Problematisch sobald die gesuchte Menge von mehreren Knoten teilweise repräsentiert wird]
48. **In welcher Zeit kann man nach einem Achsenparalleln Rechteckbereich in einem 2d-Baum suchen?** [O(SQRT(n) + k), wobei n die Gesamtzahl der gespeicherten Punkte, und k die Anzahl der berichteten Punkte ist.]
49. **Zusammenfassend 2d-Baum: Platz?** [O(n)] **Zeit?** [O(n log n) **Rechteck-Anfrage?** [O(SQRT(n) + k)]
50. **Was sind 2-Dimensionale Bereichs-Bäume (Range-Trees)?** [Wir teilen die Bereichssuche auf in 2 1-Dimensionale Bereichssuchen. Zunächst nach x, dann nach y-Bereich. Den Y-Bereich durchsucht man nun aber in einem sogenannten Assoziierten Baum, der zu jedem Knoten des Baumes existiert. Der Assoziierte Baum ist also ein Bereichssuchbaum für die y-Intervalle. Alle verwendeten Bäume sind balanzierte Binärsuchbäume.]
51. **Wieviel Platz benötigt ein Baum mit n Punkten in der Ebene?** [O(n log n)]
    a. **Beweise dies mit Hilfe des 1-dimensionalen Baumes.** [Ein 1-Dimensionaler Bereichssuchbaum benötigt linearen Speicherplatz. Die Tiefe ist log n ➔ Speicherplatz O(n log n)]
52. **Wie funktioniert die Suche in einem 2-dimensionalen Bereichsbaum?** [Erst eine Bereichssuche in X-Richtung und dann für alle ausgegebenen Bäume eine Bereichssuche in Y-Richtung]
53. **In welcher Zeit kann man nach einem Achsenparalleln Rechteckbereich in einem Bereichsbaum mit n Punkten suchen wenn k die Anzahl der auszugebenden Punkte ist?** [$O(\log^2 n + k)$ ➔ Die 1-Dimensionale Bereichssuche geht ja in O(log n +k) ➔ Das ganze wird log n mal gemacht]
54. **Wie sind höher-dimensionale (d>2) Bäume aufgebaut?** [Jeweils weitere assoziierte Bäume, d.h. die Knoten eines assozierten Baumes zeigen wieder auf weitere assozierte Bäume.]
    a. **Wieviel Zeit benötigt man für die Konstruktion?** [$O(n \log^{d-1} n]$
    b. **Wieviel Zeit benötigt man für eine Bereichsanfrage?** [$O(\log^d n)$]
    c. **Was ist Fractional Cascading und um wieviel läßt sich die Laufzeit durch dieses verbessern?** [Die Idee ist das wenn man z.B. in 2 Arrays bei denen das Zweite eine Teilmenge der Ersten ist und auf beiden den gleichen Bereich anfragen will, dass man dann das Erste mit dem Zweiten geschickt verzeigert. Somit ist es möglich, wenn man den Bereich im Ersten gefunden hat, im Zweiten den Bereich nicht erneut suchen muss. Genau das lässt sich irgendwie auf die Bereichssuchbäume übertragen. Die Verbesserung beträgt: Suchzeit für 2-dimensionale Range Trees auf O(log n + k), im d-dimensionalen lässt sich hiermit ein log-Faktor bei der Suche sparen.]
55. **Was ist das Point-Location Problem?** [Man Suche einen Punkt in einem Gebiet, also z.B. mein aktueller Standpunkt in einer Landkarte➔ Navigationssystem. Man finde also für ein Gebiet diejenige planare Fläche, die den Punkt enthält.]
    a. **Wie funktioniert der „Streifen-Algorithmus“?** [Zunächst legt man, damit alle Flächen beschränkt sind, ganz aussen herum eine sogenannte „Bounding Box“. Man legt durch jeden Endpunkt eine Vertikale, welche die Ganze Ebene in Streifen zerlegt. Bei n Segmenten hat man nun also 2n solche Streifen. Man Suche also nun den Punkt, welcher durch Seine Koordinaten gegeben ist. Diesen Punkt kann man nun mit

binärer Suche schnell finden, da die streifen geordnet sind nach x-Koordinaten. Jeder dieser Streifen wird aber durch die Segmente nun auch vertikal aufgeteilt. Speichert man diese Teile nun auch ab in y-Koordinaten geordnet, so kann man das in log n Schritten mit binärer suche ebenfalls finden. Merkt man sich nun noch, zu welchem Planar der Streifenabschnitt gehört, hat man das Problem gelöst. Somit wird sowohl in x- als auch in y-Richtung mit binärer Suche gesucht.]

i. **In welcher Zeit kann eine Anfrage gelöst werden?** [O(log n) ➔ GUT]
ii. **Was ist der Platzbedarf?** [O(n²) ➔ SCHLECHT]

b. **Wie funktioniert der „Trapezoid-Algorithmus“?** [Man teilt dabei nicht in vertikale Linien, die komplett von oben bis unten gehen, wir zeichnen die Geraden nun nur noch von Punkt bis zum nächsten Segment ➔ Keine Schnitte. Die Segmente sollen in allgemeiner Lage sein (keine 2 gleichen X- & Y-Werte).]



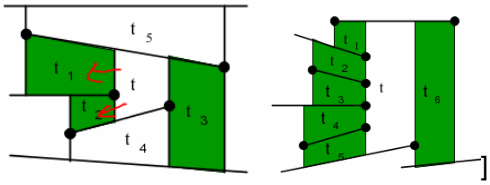
i. **Welche Eigenschaften haben die entstandenen Trapeze?** [konvex, begrenzt, die nicht vertikalen Seiten des Trapezes sind alle Abschnitte von Segmenten oder der Bounding Box| Jede Fläche hat 1 oder 2 vertikale Seiten und genau 2 nicht-vertikale Seiten]
ii. **Welche 5 Fälle für die Entstehung von linken Kanten von Trapezen gibt es?** [Es gibt 5 Möglichkeiten wie die Endpunkte die linke Seite eines Trapezes definieren können ➔ 

Zusätzlich kann noch passieren, dass der linke Rand durch die Bounding-Box definiert wird, da könnte man einen Endpunkt der Box definieren als Endpunkt. Für die rechten Seiten eines Trapezes gilt analoges.]
iii. **Wie viele Ecken und Trapezoide enthält eine „Trapezoiden-Karte“ mit n Liniensegmenten in allgemeiner Lage höchstens?** [6n+4 Ecken, 3n+1 Trapeze]
iv. **Beweise beides?** [Eine Ecke ist entweder eine Ecke der Bounding Box (4), oder ein Endpunkt eines Segmentes (2n) oder es ist ein Punkt wo eine vertikale Linie die durch einen Endpunkt geht ein anderes Segment trifft oder den Rand der Bounding Box (2*(2n)) ➔ alles aufsummiert: 6n+4 | Man kann statt die Anzahl der Trapeze zu bestimmten auch die linken Endpunkte des Trapezes berechnen. Ein Endpunkt könnte nun die linke untere Ecke der Bounding Box sein (1), oder ein linker Endpunkt eines Segmentes kann für höchstens 2 Trapeze die linke Seite bestimmen (2n). Ein rechter Endpunkt eines Segmentes kann jeweils nur für ein Trapez eine linke Seite sein (Fall d oben) (n) ➔ aufsummiert: 3n + 1, also beides viel besser als O(n²) wie beim Streifenalgorithmus]

56. **Was sind adjazente Trapezoiden?** [Benachbarte Trapeze, das heisst diejenigen Trapeze die links bzw. rechts vom Trapez liegen.]

a. **Wieviele gibt es bei Segmenten in allgemeiner Lage höchstens?** [4 Stück, 2 links und 2 rechts]
b. **Wieviele gibt es bei Segmenten die nicht in allgemeiner Lage sind?** [beliebig viele]

c. **Zeichne für beide Fälle ein Beispiel!** [links: allg. Lage | rechts: nicht

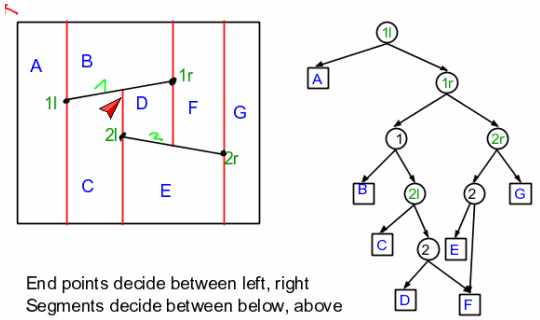


]

57. **Was sind vertikale Nachbarn?** [obere und untere Trapeze]

58. **Wie repräsentiert man Trapezoiden-Karten?** [Wir haben einen Record der zu jedem Trapez die Punkte speichert, die die obere bzw. untere Seite des Trapezes begrenzen und die Punkte die die linke bzw. rechte Seite des Trapezes begrenzen, also insgesamt 4 Punkte, durch diese 4 Punkte ist das Trapez genau definiert, des weiteren speichert man Punkte zu den maximal 4 Nachbarn.]

59. **Wie könnte eine Suchstruktur aussehen?**

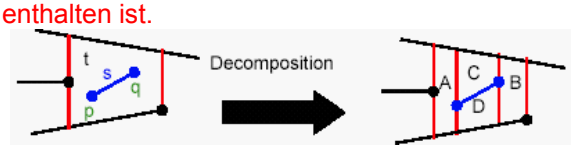


[Wir nehmen einen linken Segmentendpunkt und dessen Vertikale, der die Bereiche in links und rechts von der vertikalen aufteilt. Dann nehmen wir den rechten Endpunkt dieses Segmentes und verfahren gleich. Jetzt ist es aber so, dass der Bereich zwischen linkem und rechtem Endknoten im Prinzip genau das Segment selber ist, wir können nun also aufteilen zwischen „über“ und „unter“ dem Segment. Genauso kann man das in einem Binärbaum speichern. Die Punkte bzw. Segmente dann als Knoten. Die Blätter repräsentieren die Bereiche]

a. **Welche Rolle spielen dabei die Endpunkte?** [Teilen die Bereiche auf in links und rechts des Binärbaumes]
b. **Welche Rolle spielen dabei die Segmente?** [Teilen zwischen oben und unten auf]
c. **Ist die in der Vorlesung vorgestellte Suchstruktur eindeutig, oder kann man nach diesem Prinzip mehrere verschiedene Strukturen aufbauen?** [Nein, man kann ja beliebig anfangen]

60. Wie funktioniert der randomisierte inkrementelle Algorithmus der gleichzeitig die Trapezoiden-Karte und die Such-Struktur aufbaut? [Es wird zunächst eine Bounding-Box initialisiert in der garantiert alle Segmente Platz haben. Dann wird Segment für Segment hinzugefügt.]

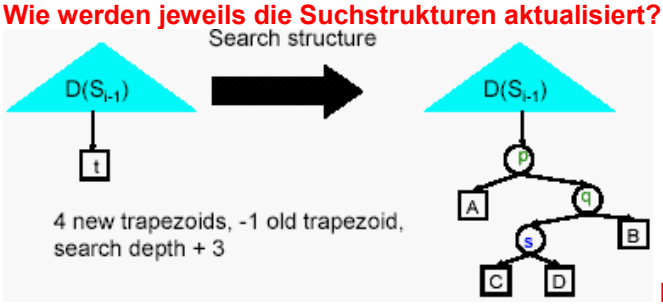
a. Welche 2 Fälle gibt es beim Einfügen eines neuen Segments? [Der einfache Fall ist, dass ein neues Segment komplett in einem bereits bestehendem Trapezoid enthalten ist.



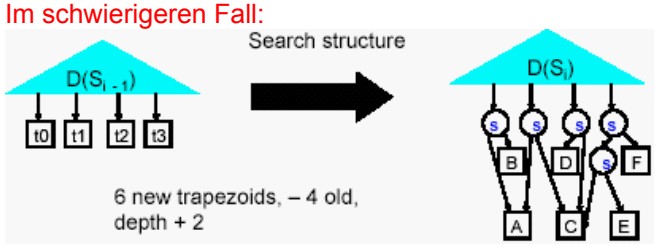
Der schwierige Fall ist, wenn das neu hinzugenommene Segment mehrere Trapezoide schneidet.

]

i. **Wie werden jeweils die Suchstrukturen aktualisiert?** [Im einfachen Fall:



Die Tiefe der Suchstruktur inkrementiert sich um 3.
Im schwierigeren Fall:



In diesem Beispiel erhöht sich die Tiefe nur um 2.]

ii. **Wie werden jeweils die Trapezoidenkarten aktualisiert?** []

b. **Wie findet man die sich schneidenden Trapeze nach dem Einfügen eines neuen Segments?** [Das linkeste Trapez kann einfach durch eine Anfrage in der Suchstruktur gefunden werden, indem man nach dem linken Endpunkt des neuen einzufügenden Segmentes sucht. Die anderen sich schneidenden Trapeze findet man, in dem man in der Trapezoidenkarte solange nach rechts läuft bis man den rechten Endpunkt des neuen Segmentes erreicht hat. Hierbei durchquert man alle schneidenden Trapeze, und kann sich diese merken.]

c. **Was ist der Worst-Case bzw. Average-Case für die Tiefe der Suchstruktur?** [Der worst-case ergibt sich durch den einfachen Fall: 3n. Das dieser Fall auftrifft ist allerdings sehr unwahrscheinlich, da ja jedes neu einzufügende Segment immer komplett in einem bestehenden Trapezoid enthalten sein müsste. Der average-case ist O(log n).]

61. **Gib die Länge des Suchpfades für die Anfrage eines willkürlichen Punktes an (worst-case, erwartete Länge)?** [O(n), O(log n)]
62. **Gib die Größe der Suchstruktur D(S) an (worst-case, erwartete Größe)?** [O(n²), O(n)]
63. **Gib die Laufzeit für die Konstruktion der Suchstruktur an (worst-case, erwartete Laufzeit)?** [O(n²), O(n log n)]
64. **Zusammenfassend Erwartungswerte für: Konstruktion, Platz und Anfrage?** [O(n log n) Konstruktionszeit in O(n) Platz mit einer Anfragezeit von O(log n)]
65. **Welche 4 Algorithmen zur Berechnung der konvexen Hülle haben wir besprochen?** [Point Pruning | Edge Pruning | Jarvis March | Graham Scan]
66. **Definieren Sie die Begriffe Konvexes Set und Extrem Punkt!** [Convex Set: Ein Set S von Kanten aus der Gesamtmenge aller möglichen Kanten ist konvex, gdw. für jedes Punktepaar p1,p2 aus S gilt, dass das Segment p1p2 komplett in S liegt | Extrem Punkt: Ein Punkt p in einem konvexen Set S nennt man extrem, gdw. es kein Segment aus S gibt welches p enthält]
67. **Wie ist die konvexe Kombination definiert?** [Die konvexe Kombination von n Punkten $p_n=(x_n,y_n)$ ist ein Punkt q, der sich durch Multiplikation eines Faktors α_i mit jedem anderen Punkt i, darstellen lässt. Hierbei muss jedes $\alpha_i \geq 0$ sein, und alle α_i aufaddiert müssen 1 ergeben. Formal: $\alpha_1(x_1,y_1) + \alpha_2(x_2,y_2) + \ldots + \alpha_n(x_n,y_n)$; $\sum_{i=1}^{n} \alpha_i = 1$.]
68. **Welche äquivalenten Definitionen der konvexen Hülle gibt es?** [1. Mit den Punkten auf der konvexen Hülle kann man die konvexe Kombination jedes Punktes innerhalb der konvexen Hülle bestimmen | 2. Die Konvexe Hülle einer Punktemenge P ist der Schnitt aller konvexen Sets die P enthalten | 3. Der Schnitt aller Halbebenen die die Punktemenge P enthält ist die konvexe Hülle von P]
69. **Was ist die untere Grenze zur Berechnung der konvexen Hülle? Wie wurde diese bewiesen?** [Die Berechnung der konvexen Hülle lässt sich aus dem Sortieren von Zahlen herleiten. Die Herleitung benötigt lineare komplexität. Die Berechnung der Anordnung der

Punkte um die konvexe Hülle entspricht eben dem Sortieren dieser Punkte. Sortieren geht wie wir wissen in mindestens O(n log n) Zeit, weshalb das auch die untere Schranke für das Berechnen der konvexen Hülle sein muss. Die Transformation die im Beweis verwendet wird ist: x_i -> (x_i,x_i^2). Das Gebilde was dabei entsteht ist eine Parabel die zumindest nach unten konvex ist. Somit ist das Aufzählen der Extrempunkte um die konvexe Hülle äquivalent mit dem Sortieren von Zahlen.]

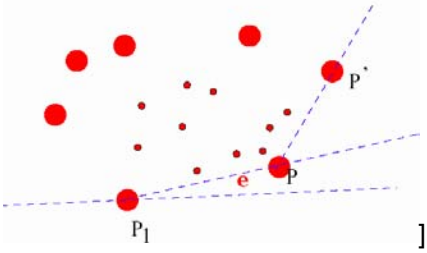
70. **Was ist eine Links- bzw. eine Rechts-Kurve? Wie lässt sie sich berechnen?** [Wenn wir wissen wollen, ob 3 Punkte eine rechts oder eine linkskurve beschreiben, müssen wir die ersten beiden Punkte durch eine gerade repräsentieren und feststellen ob der 3. Punkt links oder recht der Geraden ist | Sehr einfach lässt sich das berechnen, wenn man die 3 Punkte in eine 3x3 Matrix schreibt ergänz um den Vektor (111) => 3 Zeilen der Form: x1,y1,1 – x2,y2,1 – x3,y3,1. Berechnet man nun die Determinante dieser Matrix, erhält man die Information, ob die Kurve nach rechts oder links geht. Bei negativer Det. Ist es eine Rechtskurve, bei positiver eine Linkskurve und bei Det.=0 liegen alle 3 Punkte auf einer Geraden]

71. **Wie funktioniert Point-Pruning?** [Man nimmt 3 beliebige Punkte und verbindet diese zu einem Dreieck. Nun sucht man alle Punkte innerhalb und ausserhalb des dreiecks in O(n) und löscht die innerhalb des Dreiecks, diese gehören sicher nicht zur konvexen Hülle. Jetzt betrachten wir ein noch nicht betrachtetes Dreieck und löschen wieder alle Punkte innerhalb. Wenn wir alle Dreiecksmöglichkeiten durchgetestet haben, haben wir die konvexe Hülle.]
    a. **Was ist die Laufzeit von Point-Pruning?** [$O(n^4)$ ➔ Alleine das betrachten der Dreiecke kann schon $O(n^3)$ benötigen, nämlich genau dann, wenn alle Punkte auf der konvexen Hülle liegen dann noch O(n) vergleiche pro Dreieck. Und damit haben wir nur die Extrempunkte die konvexe Hülle haben wir erst noch nicht, denn diese beinhaltet auch die Reihenfolge der Punkte].
    b. **Wie kann man es verbessern?** [Wir halten den linkesten und rechtesten Punkt der Punktmenge fest, und ziehen Dreiecke zu den übrigen Punkten. Den linkesten und rechtesten Punkt findet man in O(n). Es müssen nur noch n Dreiecke betrachtet werden. Das Verfahren lässt sich damit auf $O(n^2)$ verbessern]
    c. **Was muss noch getan werden, wenn man die Extrempunkte hat?** [Man muss diese sortieren]
    d. **Welche 2 Methoden für das Sortieren der Extrempunkte haben wir besprochen?** [1. Rotieren einer halbgerade mit Ursprung innerhalb der konvexen Hülle. Wir laufen damit alle Punkt der Reihe nach ab | Aufteilen der Extreme Punkte in obere und untere. Die Punkte im oberen Teil werden absteigend sortiert, die Punkte unterhalb aufsteigend]
        i. **Welche Laufzeiten haben sie?** [Beide O(n log n)]

72. **Wie funktioniert Edge-Pruning?** [Man nimmt eine beliebige Kante aus zwei Punkten aus P und testet ob alle restlichen Punkte auf einer Seite der Kante sind. Ist dies der Fall, dann haben wir ein Segment der konvexen Hülle gefunden. Danach muss man die Segmente noch sortieren.]
    a. **Welche Laufzeit hat dieses Verfahren?** [Es gibt $O(n^2)$ Paare, der Test auf welcher Seite der Rest der Punkte liegt geht in O(n), das Sortieren geht in O(nlogn), d.h. die Gesamtlaufzeit ist in $O(n^3)$]

73. **Wie funktioniert Jarvis´s-March (gift wrapping method)?** [Es ist eine Verbesserung des Edge-Pruning Algorithmus, wann immer man ein Segment gefunden hat, weiss man ja, dass der aktuelle Endpunkt des Segmentes auf der konvexen Hülle liegt, man muss also nur noch einen weiteren Punkt finden. Anfangs beginnt man mit dem untersten Punkt (niedrigste y-koordinate), welcher auf jeden Fall auf der konvexen Hülle liegt. Da der Algorithmus die Extrempunkte sogar in der richtigen Reihenfolge ausgibt, muss man diese nicht einmal mehr sortieren. Man läuft dazu gegen den Uhrzeigersinn und gibt jeweils denjenigen Punkt als nächsten aus, welcher den kleinsten Winkel bezüglich der Vorgängerkante hat (in O(n)) Das muss man solange fortsetzen, bis man am höchsten Punkt (größte y-koordinate)

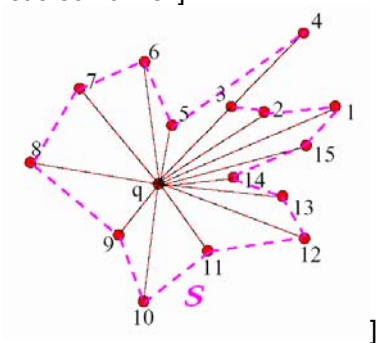
angekommen ist, danach geht es umgekehrt weiter.:



]

a. **In welcher Reihenfolge werden die Extrempunkte ausgegeben**? [gegen den Uhrzeiger sortiert]
b. **Welche Laufzeit hat das Verfahren?** [Das Finden der ersten 2 Punkte sowie die weiteren Punkte geht in O(n). Da wir k Extrempunkte haben ist der Algorithmus Output Sensitiv ➔ O(nk)]
c. **Unter welchen Umständen ist die Laufzeit linear?** [Wenn die Anzahl der Punkte auf der konvexen Hülle sehr klein im Vergleich zu n ist, oder die Anzahl der Extrempunkte durch eine Konstante begrenzt ist.]
d. **Wie sieht es bei diesem Verfahren mit höheren Dimensionen aus?** [Kann auch auf höhere Dimensionen skaliert werden]
e. **Wie kann das Verfahren verbessert werden?** [Man sortiert die Punkte in einer gewissen Art&Weise so dass man die Punkte die sich als Extrempunkte disqualifizieren für die nachfolgenden Berechnungen ausschliessen kann.]

74. **Wie funktioniert Graham´s Scan?** [Graham Scan ist eine Modifikation von Point Pruning: Man sucht sich einen Punkt q innerhalb der Punktemenge (z.B. innerhalb eines beliebigen Dreiecks). Nun sortiert man die Punkte aus P rund um q bzgl. Ihres Polarwinkels. S ist nun das Polygon, welches den Kantenzug rund um q beschreibt. Nun sucht man den linkesten Punkt und nennt ihn p. Den Vorgänger nennt man p^- und den Nachfolger p^+. Nun prüft man ob die Punkte p^-pp^+ eine Rechts- oder Linkskurve beschreiben. Falls ja wird p^+ zu p (man springt also eines weiter). Falls nein, wird p entfernt und p^- mit p^+ verbunden. Dann muss man einmal zurückspringen und erneut testen, falls es dann eine Linkskurve ist, dann wird p^+ zu p und es geht weiter. Ansonsten muss man erneut entfernen und „überbrücken“. Wenn man dann irgendwann den Ausgangspunkt erreicht hat, besteht P nur noch aus linkskurven, ist also konvex]



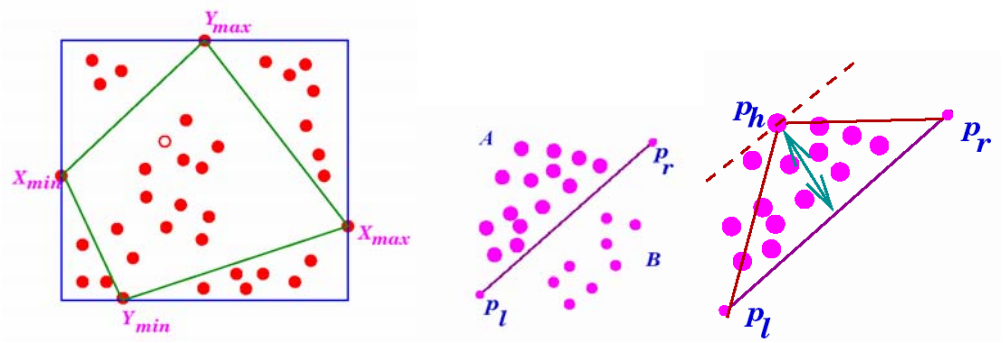
]

a. **Was ist die Laufzeit?** [Wenn die Punkte mal sortiert (O(nlogn)) sind benötigt Graham Scan O(n) Zeit => O(n log n)]

75. **Nenne noch mal alle Laufzeiten der 4 Verfahren zur Berechnung der konvexen Hülle!** [Point Pruning: $O(n^4)$ bzw $O(n^2)$ | Edge Pruning: $O(n^3)$ | Jarvi's March: O(nh) | Graham's Scan: O(n log n)]

76. **Welche 3 Divide-and-Conquer Algorithmen zur Berechnung der konvexen Hüllen haben wir besprochen?** [Quickhull, Mergehull, Randomized incremental Convex Hull]]

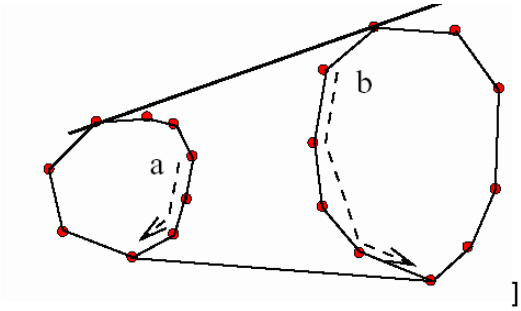
77. **Wie funktioniert die Quick-Hull Technik/Algorithmus?** [Er ist das Analogon zum Quick-Sort und ähnlich wie Point Pruning. Die Idee ist es Punkte, die sicher nicht zur konvexen Hülle gehören, so schnell wie möglich zu entfernen. Am Anfang inititalisiert man mit ein Quadrilateral bestehend aus den 4 Extrempunkten der x und y Richtung, d.h. der linkeste, der rechteste, der oberste und der unterste Punkt. Dies geht in O(n). Nun hat man in der Regel sehr viele Punkte die innerhalb dieses Quadrilaterals liegen und kann diese in O(n) entfernen



Nun hat man noch jeweils Punkt, die in den 4 Eck-Dreiecken liegen. Innerhalb dieser Dreiecke sucht man nun den linkesten und den rechtesten Punkt und teilt die restliche Punkte bezüglich dieser zwei Punkte auf. Wenn die Obere Punktemenge leer ist, dann sind die beiden Punkte P_l und P_r auf der konvexen Hülle. Falls die obere Menge nicht leer ist, finde einen Punkt p_h der das Dreieck $p_l p_r p_h$ maximal macht bzgl. der Senkrechten zu p_l und p_r. Gibt es mehrere Kandidaten, nimmt man den Linkesten. P_h ist auf jeden Fall ein Punkt der konvexen Hülle! Die Punkte innerhalb des Dreiecks werden wieder entfernt. Die nun wieder übrig gebliebenen Punkte über den Linien $p_l p_h$ bzw. $p_r p_h$ kann man wieder so behandeln. Mit den weiteren Ecken geht das natürlich analog.]

a. **Welche Laufzeit hat dieser? Worst-case?** [Bei annähernder gleichverteilung der Punkte ist die average Laufzeit O(n log n). Der Worst Case ist leider O(n²) wenn die Punkte z.B. alle auf der Hülle liegen (vgl. Quick-Sort, => Aufteilung der Mengen schlecht).]

78. **Wie funktioniert der Merge-Hull Algorithmus?** [Auch „Convex Hull by D&C“ genannt. Merge-Hull deshalb, weil er auf einer generalisierung von Merge-Sort beruht.
1. Divide: Teile die Punktemenge in 2 gleich große Teile P_1 und P_2(falls |P| kleiner gleich 2 kann man direkt lösen. P_1enthält nun alle Punkte deren x-Koordinaten kleiner sind als die der Punkte aus P_2
2. Merge: Man berechnet die obere und untere Tangente der beiden Punktemengen und löscht alle Punkte die zwischen den beiden Tangenten liegen



]

a. **Wie funktioniert die Berechnung der unteren Tangente zweier konvexer Hüllen?** [Man nimmt den rechtesten Punkt der linken Hülle und den linkesten Punkt der rechten Hülle und verbindet Sie. Nun lässt man diese Gerade sukzessive nach unten wanderen, bis sie auf einer Seite die konvexe Hülle nicht mehr schneidet, das Gleiche macht man nun auf der anderen Seite. Schneidet Sie nun auf der anderen Seite wieder die Hülle (bedingt durch die Drehung) muss man dort auch wieder runterwandern usw.]
b. **Wie testet man ob die Gerade wirklich eine Tangente zur konvexen Hülle ist?** [Wenn beide Nachbarpunkte auf der selben Seite der Geraden liegen.]
c. **Wie lange benötigt man diese untere Tangente zu finden?** [O(n)]
d. **Welche Laufzeit hat der Merge-Hull Algorithmus?** [O(n log n)]

79. **Wie funktioniert der in der Vorlesung vorgestellte randomisierte Sortieralgorithmus?** [inkrementelles randomisiertes Sortieren von Punkten (oder auch Zahlen). Durch zufällige Permutation der Eingabe erhalten wir einen Las Vegas Algorithmus.
1. zufälliges Wählen eines Punktes aus der Punktemenge
2. Es entstehen nun Intervalle, welchen wir die übriggebliebenen n-1 Punkte mit Pointern zuordnen können. Wir halten eine sogenannte conflict List, die alle Punkte im ersten Intervall bzw. eine zweite conflict List, die die Punkte im zweiten Intervall beinhalten.
3. der nächste Punkt wird zufällig hinzugefügt und das betreffende Intervall (bzw. die conflict List) aufgeteilt. Mit der anderen conflict List müssen wir nichts machen. Wenn ein Punkt neu

eingefügt wird, müssen wir die Pointer der unsortierten Punkte bzgl. der neuen Intervalle updaten und den Pointer des Punktes x entfernen (der ist ja jetzt sortiert).
4. Wenn alle Punkte eingefügt wurden, haben wir ein sortiertes Set.]

i. **Was erlaubt ein effizientes Hinzufügen einer neuen Zahl?** [Wir wählen zufällig und können in Zeit O(1) feststellen wo die Zahl eingefügt werden muss, da wir ja einen Zeiger auf die Einfügestelle haben. Außerdem wissen wir was alles auf ein bestimmtes Intervall zeigt (Bidirektionale Verzeigerung), was wichtig für das Updaten dieser Zeiger ist, wenn eine Zahl hinzugefügt wird. Genau dies ist der schwierigste Analysepunkt, und wird mittels backward-analysis gelöst => Im Schnitt beträgt die Einfügezeit somit O(logn).]

ii. **Was sind die conflict lists und wie werden diese verwaltet?** [Die conflict Listes beinhalten alle noch unsortierten Punkte welche sich in einem bestimmten Intervall befinden. Verwaltet werden diese in Form von Pointern, die von den Punkte in das Intervall zeigen und umgekehrt]

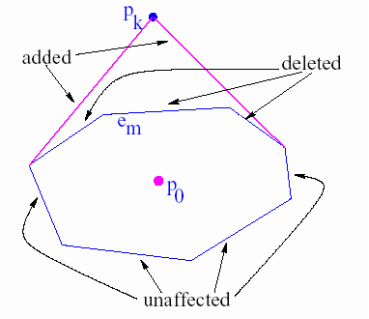
iii. **Wie wird mittels backward-analysis die Komplexität berechnet?** [Wir berechnen den Aufwand rückwärts. D.h. wir berechnen die Kosten wenn ein Punkt aus der sortierten Liste entfernt wird. Hierzu benötigen wir die Anzahl der Punkte deren Zeiger auf das Intervall zeigen, das durch die Entfernung entsteht. Diese entsteht durch *Anzahl der unsortierten Punkte* dividiert durch die *Anzahl der Intervalle* => (n-1)/(i+1) => O(n/i). Somit müssen wir O(n/i) Zeiger updaten, wenn wir den i+1-ten Punkt entfernt haben. Wenn wir dies nun n-mal machen (n Zahlen Entfernen) ergibt sich:

$$O(\sum_{i=1}^{n} n/i) = O(n \sum_{i=1}^{n} 1/i) = O(n \log n)$$

]

iv. **Welche Komplexität hat der Algorithmus?** [O(n log n) -> siehe backward-analysis]

80. **Wie funktioniert der randomisierte inkrementelle Algorithmus zur Berechnung der konvexen Hülle?** [Man nehme 3 beliebige Punkte der Eingabemenge und konstruiert die konvexe Hülle dazu (3 Punkte bilden immer eine konvexe Hülle). Nun wird randomisiert ein Punkt hinzugefügt! Und die konvexe Hülle geupdatet, aber wie geht das?:
1. Erstellen der Conflict List, indem man einen Punkt p_0 innerhalb unseres Initialen Dreiecks wählt und diesen mit jedem der restlichen Punkte verbindet.
2. Jede dieser Linien von p_0 nach p_i schneidet eine der 3 Kanten des Dreiecks. Jeder Punkt p_i erhält einen Pointer auf seine „Schnittkante“. Wir starten also mit 3 conflict Lists!
3. Zufälliges Hinzufügen eines Punktes. Wenn dieser auf der gleichen Seite ist wie p_0 wird er verworfen, da er ohnehin nicht in der konvexen Hülle ist. Ist der Punkt ausserhalb, so müssen wir die conflict Lists updaten und neue Kanten einziehen. In unserem Beispiel müssen wir die 3 Conflict-Lists der Punkte updaten, die durch die 3 zu entfernenden Kanten definiert sind, und auf die 2 Conflict-Lists der zwei neuen Kanten verteilen.
4. Wir speichern unsere konvexe Hülle als doppelt verkettete Kantenliste, damit wir auf der Hülle hin und her laufen können. Zu unserem neuen Punkt p_k können wir nun diejenigen Knoten v_i und v_i finden, die verbunden mit p_k jeweils eine Tangente der Hülle bilden.



Die dazwischenliegenden conflict Lists müssen auf die beiden neuen Listen aufgeteilt werden in und zwar für jeden Punkt in O(1) Zeit. Wenn alle conflict Listes leer sind, dann sind wir fertig.]

a. **Wie wird mittels backward-analysis die Komplexität berechnet?** [Man rollt das Problem wie beim Inkrementellen Sortier-Algorithms von hinten her auf, d.h. man entfernt einen Punkt aus der bisherigen Konvexen Hülle, und berechnet wie viel arbeit man an dem Updaten der Conflict-Lists verbraucht. Beim Entfernen werden 2 Kanten

der konvexen Hülle entfernt. Wenn wir in der i+1-ten Hülle i+1 Punkte haben ist die Wahrscheinlichkeit zufällig einen Punkt fürs Entfernen zu wählen 1/i. Es müssen noch n-i Punkte zu der Hülle hinzugefügt werden. Somit ergibt sich für die erwartete Anzahl von Punkten, die in den Conflict-Lists der 2 zu entfernenden Kanten liegen müssen, (n-i)/i = O(n/i). O(n/i) ist also die erwartete Arbeit die wir ins Entfernen eines Punktes investieren müssen. Macht man dies nun n-Mal erhält man aufsummiert eine Laufzeit von O(n log n). (siehe inkrementeller Sortier-Algorithmus -> backward-analysis)]

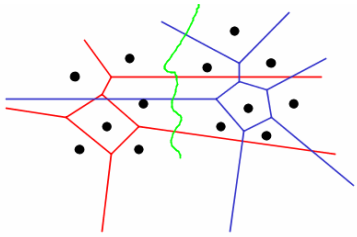
b. **Was wird randomisiert?** [Die Eingabepunkte]
c. **Welcher Typ von Algorithmus ist dies?** [Las Vegas ➔ Immer korrekt, wahrscheinlich schnell]
d. **Was ist die erwartete Laufzeit des randomisierten inkrementellen Algorithmus?** [Berechnet mit Hilfe Backward Analysis: O(n log n) erwartete Zeit]

81. **Was ist ein Voronoi Diagramm?** [Eine Einteilung von z.B. Postämtern in Bereiche die dem jeweiligen Postamt am nächsten liegen, damit man immer weiss, zu welchem Postamt man am schnellsten kommt, je nachdem in welchem Gebiet man sich befindet ➔ Sogenannte Gebiete gleicher nächster Nachbarn.]
    a. **Was ist die euklidische Distanz und wie berechnet sie sich aus zwei Punkten?** [Betrag der Abstandsvektoren (Luftlinie). Es gibt z.B auch die Manhattendistanz (Strassenaufteilung in Manhatten: B125
    b. erechnung: $dist(p,q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$]
    c. **Wie wird ein Voronoi Diagramm konstruiert?** [Man nimmt sich zwei Punkt und teilt diese über die Mittelsenkrechte in zwei Gebiete auf, so fährt man sukzessive durch Hinzunahme weiterer Punkte fort.]
    d. **Was passiert wenn die Punkte kolinear angeordnet sind?** [Nur n-1 Senkrechte Linien (Geraden) jeweils zwischen den Punkten. Die Voronoi Regionen sind dann in beide Richtungen unbeschränkt]
    e. **Wie sieht das Voronoi Diagramm sonst aus?** [Es ist ein zusammenhängender Graph. Die Kanten sind entweder Liniensegmente oder halbgeraden]
    f. **Warum ist das Voronoi Diagramm immer zusammenhängend?** [Wäre das nicht so, dann gäbe es eine Zelle, die die Ebene in zwei Teile teilen würde. Weil Voronoi-Zellen konvex sind, wäre diese Zelle aus 2 parallelen vollen Linien begrenzt. Wir wissen aber das die Kanten eines Voronoi-Diagramms keine vollen Linien sein können (Widerspruch) ausser bei kolinearen Anordnung.]
    g. **Welchen Ausgangsgrad hat jeder Knoten des Voronoi-Diagramms?** [3]
    h. **Welcher Spezialfall muss dabei aber ausgeschlossen werden?** [Die Punkte müssen in Allgemeiner Lage sein, was hier heisst, dass nicht mehr als 3 Punkte auf einem Kreis liegen dürfen]
        i. **Man zieht einen Kreis um eine Ecke des VD durch die 3 Nachbarschaftspunkte, ist es dann möglich, dass ein 4ter Punkt innerhalb dieses Kreises liegt?** [Nein]
        ii. **Welche Eigenschaft haben zwei nächste Nachbarn bzgl. des VD?** [Sie definieren eine Kante des Voronoi Diagramms dieses Punktes]
        iii. **Welche besondere Eigenschaft haben die Punkte die innerhalb einer unbegrenzten Fläche liegen?** [Die Punkte liegen auf der konvexen Hülle der Punktmenge. Umgekehrt gilt dies auch !]
    i. **Wie viele Ecken und Kanten hat ein Voronoi Diagramm mit n Punkten höchstens?** [Ecken: 2n-5 Kanten: 3n-6 wenn n die Anzahl der Punkte ist.]
        i. **Nennen Sie einen Beweisansatz** [Euler Formel (Ecken-Kanten+Flächen=2). Damit man aber nun auch die unbegrenzten Flächen damit berechnen kann nimmt man einen fiktiven Punkt hinzu, und verbindet alle halbgeraden mit diesem Punkt. | Anzahl der Ecken ist nun VD(P)+1 (wegen dem zusätzlichen Punkt) | Wir können nun sagen, dass (2* #Kanten des)VD(P) größergleich 3* #Knoten des VD(P) +1) ist. ➔ Einsetzen in Eulerformel]
    j. **In welcher besprochenen Datenstruktur kann ein Voronoi Diagramm gespeichert werden?** [Doubly Connected Edge List: Drei Records: Eine für die Knoten (Koordinaten) mit einer anliegenden inzidenten Kante, einer für die Flächen mit evtl. vorhandenen Löchern und einer für die Halbkanten mit den Informationen Origin, Twin, Incident Face, Next und Prev]
    k. **Wie funktioniert der D&C Ansatz zur Berechnung des VD?** [Aufteilen der Punktemengen in 2 Hälften und rekursive Berechnung. Rekursion bricht ab, wenn nur noch 1 Punkt in der Ebene ist, die Voronoi Region ist dann die ganze Ebene. Im

Merge-Schritt müssen dann 2 Voronoi-Diagramm durch einen gemeinsamen Kantenzug zusammengebracht werden.]

i. **Wie schnell geht das?** [O(n log n)]

ii. **Beim Mergeschritt muss man einen Kantenzug K berechnen, wie geht**



das genau? [Wir berechnen eine y-monotone Sequenz die die beiden hälften zusammenführt in der Form, dass man jeweils die Mittelsenkrechte einzeichnet und die überstehende Teile der Linien abschneidet. Hierzu fängt man mit den 2 obersten Punkten beider zu mergender VDs an, nimmt die Mittelsenkrechte, und läuft vom oberen unendlichen soweit bis man an eine Kante anstößt. Diese schneidet man dann ab. Je nachdem ob dies eine Kante des rechten oder linken VDs ist, nimmt man aus dem betreffenden VD den nächsten Punkt aus der konvexen Hülle des VDs der in Richtung der anderen VD zeigt. Jetzt wieder Mittelsenkrechte ... usw... (kann über eine Sweep-Line realisiert werden)]

iii. **Wie schnell geht die Berechnung des Kantenzuges?** [O(n)]

l. **Was versteht man unter hierarchischer Triangulation in einem VD zum Finden eines Punktes?** [Man sperrt das VD in ein großes Dreieck ein, die inneren Gebiete trianguliert man dann. ➔ Dann wird die Triangulation vergröbert, indem man Knoten entfernt, dann wird dieser Bereich neu trianguliert und Zeiger auf den entfernten Knoten gespeichert. Die hierarchische Triangulierung ist somit eine Baumstruktur welche eine schnelle und effiziente Auffindung von Daten ermöglicht. Die Wurzel ist das Ursprungsdreieck, welches durch weitere Dreiecke verfeinert wird. Gesucht wird somit hierarchisch, in dem man beim Durchlaufen des Baumes immer wieder prüft in welchen Nachfolgeknoten (=Dreieck) man gehen muss.]

m. Wie kann einem das VD helfen, wenn man z.B. die nächste Shell Tankstelle zu einem gegebenen Ausgangpunkt sucht? [Man macht eine Point-Location-Anfrage in einer geeigneten Flächenrepräsentation des VD (repräsentiert durch eine Baumstruktur wie z.B. durch hierarchische Triangulierung oder siehe Algorithmentheorie: Streifenkarte, Trapezoidenkarte, etc.). Weis man in welcher Fläche man sich befindet hat man auch die nächste Shell-Tankstelle.]

i. Wie schnell geht diese Anfrage bei bestehendem VD? [Da bei der hierarchischen Triangulierung bei Verwendung der Regeln von Kirkpatrick (triangulation refinement method) ein Suchbaum mit logarithmischer Tiefe entsteht: O(log n)]

n. **Wie kann man das Closest Pair und das „All nearest Neighbors“ Problem mit dem VD elegant lösen?** [Closest Pair: Man muss nicht mehr jeden mit jedem vergleichen, sondern nur Nachbarn! Das Voronoi Diagramm kann man ja in O(n log n) konstruieren, die Anzahl der Nachbarn ist dann nur noch linear, das heisst bei bestehendem VD kann man das Closest Pair Problem in O(n) lösen bzw. in O(n log n) wenn man das VD noch aufbauen muss | Das All nearest Neighbours Problem ist ganz ähnlich, denn man sieht sich einfach nur alle Nachbarn jedes Punktes an und nimmt den kleinsten]

o. **Wie kann man schnell einen Minimalspannenden Baum nach Kruskal aufbauen?** [Man betrachtet jeden Knoten der Punktemenge als Baum mit einem einzigen Knoten. Solange es noch mehr wie einen Baum in unserem Wald gibt, wird das folgende gemacht: Man sucht ein Punktepaar, welches zwei der Bäume verbindet und minimal ist und verbindet diese durch Vereinigen der beiden Bäume. Iteriere! $O(n^2 \log n)$]

p. **Wie geht das schneller bei bestehendem VD?** [Die linear vielen Nachbarn eines jeden Knoten kennen wir ja dann, also müssen wir nicht jeden mit jedem vergleich wie beim naiven Verfahren.]

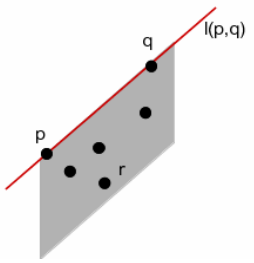
i. **Wie schnell bei bestehendem VD?** [O(n log n)]

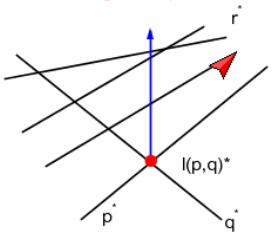
82. **Auf welche 4 besprochenen Arten kann man einen Punkt auf eine Linie abbilden bzw. umgekehrt? Nenne diese!** [Punkte bestehen üblicherweise aus Koordinaten und Linien werden durch eine Gleichung y=mx+b beschrieben. Beide sind durch genau 2 Angaben

eindeutig bestimmt, bei den Punkten eben die Koordinaten und bei den Linien durch m und b ➔ Durch diesen Zustand kann man Punkte auf geraden abbilden und umgekehrt. Dafür gibt es verschiedene Möglichkeiten u.a.: Slope Abbildung man bildet die Koordinaten x und y direkt auf m und b der geraden ab. Dann gibt es noch z.B. die Polar Abbildung. Ein Punkt P mit den Koordinaten a und b wird auf eine gerade abgebildet die nach ax + by = 1 aufgebaut ist. Die Polarabbildung bildet Punkte auf dem Einheitskreis direkt auf Tangenten ab. Die nächste besprochene Abbildung ist die so genannte Parabel Abbildung. Man bildet den Punkt P(a,b) ab auf y=2ax-b | Die Duality Transform Abbildung folgt nun genauer]

83. **Duality Transform wurde genauer beschrieben, wie geht das im Einzelnen?** [p(a,b) wird abgebildet auf p* y=ax-b und eine gerade l: y=mx + b wird abgebildet auf einen Punkt l*=(m,-b)]
    a. **Welche Eigenschaften bleiben trotz Transformation erhalten?** [Die Abbildung ist invers (p*)* ist p und (l*)* ist l➔ Beweis recht trivial durch Einsetzen | Die zweite Eigenschaft ist: Inzidenzerhaltung, d.h. Ein Punkt P der auf einer geraden l liegt, dann ist der Duale Punkt l* der Geraden l auch auf der dualen Geraden p* des Punktes p | Eine weitere Eigenschaft ist die Ordnungserhaltung, d.h. wenn ein Punkt p über einem Punkt l liegt, dann liegen auch die dualen Geraden p* und l* entsprechend übereinander]

84. **Was versteht man unter dem Level von Punkten?** [Wir betrachten eine Gerade die durch zwei Punkte p und q geht. Uns interessieren dabei drei Größen, nämlich die Anzahl der Punkte oberhalb der Geraden l(p,q) und die Menge aller Punkte unterhalb. Wenn man die Gesamtanzahl der Punkte kennt, braucht man natürlich nur einer dieser Größen berechnen. Wir berechnen die Größen für jedes beliebige Punktepaar. Naiv geht das in $O(n^3)$. Der Level eines Punktes ist die Anzahl der Geraden oberhalb des Punktes.]
    a. **Wie geht dies besonders elegant mit Hilfe der „Duality Transformation“?** [Man berechnet die Dualen Geraden zu den Punkten. Die Inzidenzerhaltungseigenschaft zeigt uns dann, dass wenn eine Gerade durch zwei Punkt geht, die dualen Geraden dieser beiden Punkte einen gemeinsamen Schnittpunkt haben. Wir transformieren also die beiden Punkte und erhalten in deren Schnittpunkt im Dualraum den Dualen Punkt der Geraden l(p,q). Die Ordnungseigenschaft bringt uns nun den Vorteil, dass die dualen Geraden aller Punkte die unterhalb unserer Geraden l(p,q) liegen im Dualraum überhalb unseres dualen Punktes l(p,q)* liegen. Das Ganze läuft nun auf einen Schnitt mit den Denjenigen Geraden oberhalb von l(p,q)* ➔ Naiv in $O(n^3)$, aber da man das ja für jedes Punktepaar berechnet, kann man die Information ja wieder verwenden, denn die Levels sind nicht unabhängig voneinander, sondern absteigend, d.h., wenn man den Level eines Punktes berechnet hat, dann hat der darüberliegende Punkt einen Level weniger.]



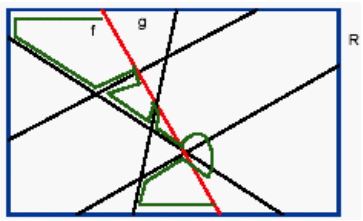


    b. **Wie geht der Algorithmus zu der o.g. Idee?** [Wir berechnen zu jeder Geraden den Level des linkesten Schnittpunktes. Man läuft dann entlang jeder dieser Linien und updatet die Levels jedes nächsten Schnittpunktes, je nachdem ob die schneidene Gerade von links unten oder von links oben kommt wird der Level dann um eins erhöht oder erniedrigt. Kommen zufälligerweise zwei Geraden, dann wird um 2 erhöht bzw. erniedrigt. ➔ Das Ganze geht nun in $O(n^2)$, da jeder der beiden Schritte nur O(n) geht.]
    c. **Wie geht denn das mit dem Ablaufen der Geradenschnitte genau und warum in O(n) Zeit?** [Sowohl die Schnittpunktberechnung zweier Geraden als auch die Ermittlung der Ausrichtung einer Geraden geht in O(1), da wir n Schnitte/Ausrichtungen zu berechnen haben folgt O(n)]

85. **Wie viele Knoten, Kanten und Flächen hat eine transformierte Punktmenge höchstens?** [Knoten: $n*(n-1)/2$ ➔ Jedes Paar von Geraden liefert eine Ecke| Kanten: n^2 ➔ Jede Gerade kann (n-1) mal geschnitten werden, daraus ergeben sich die Segmente also Kanten | Flächen: $n^2/2+n/2+1$ ➔ Eulerformel: Ecken-Kanten+Flächen = 2 ➔ Alle Zahlen gelten bei einfachen Arrangements (Ohne parallelen)]

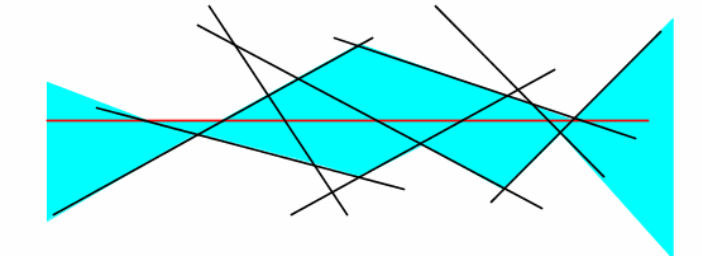
86. **Wie kann man ein solches Arrangement von transformierten Punkten speichern? Wie schnell geht das?** [Erstmal werden alle Schnittpunkte in eine Bounding Box gelegt indem man die Schnittpunkte berechnet und den höchsten, niedrigsten, linkesten und rechtesten sucht und die Box etwas größer macht ➔ in $O(n^2)$. Gespeichert wird z.B. in einer Doppelt verketteten Kantenliste. Man lässt eine Sweepline darüberlaufen ➔ Laufzeit $n^2 \log n$]

87. **Wie funktioniert der inkrementelle Algorithmus?** [Laufzeit: $O(n^2)$. Zerlegung der Bounding Box in Flächen und inkrementelles Einfügen der enthaltenen Geraden. Das gesamte Gebilde wird in einer doppelt-verketteten Kantenliste gespeichert. Nimmt man nun eine weitere Gerade hinzu, dann bestimmt man den Schnitt dieser neuen Geraden mit dem Rand (den linkesten Randpunkt). Da man die anderen Geraden ja schon in der Doppelt verketteten Kantenliste gespeichert hat, weiss man nun auch an welcher Stelle einer Fläche (Kantenzug) die neue Gerade beginnt. Jetzt muss man herausfinden, an welcher Stelle sie diese wieder verlässt, dazu betrachte man den die Fläche einschließenden Kantenzug und schneidet jede dieser Kanten mit der neuen Geraden. Hat man die betreffende Kante gefunden, dann weiss man, dass man zur benachbarten Fläche gehen muss, welche den weiteren Teil der Geraden enthält. Man muss nun also nur die betreffenden Fläche betrachten. Spezialfall: Wenn die Gerade genau an einem Schnittpunkt zweier Flächen diese verlässt, dann muss man wissen wo die neue Gerade weiter geht, da kann man eine weitere Eigenschaft der Doppeltverketteten Kantenliste nutzen, nämlich, dass man um einen Punkt herumlaufen kann in einer Zeit die abhängig vom Grad des Knotens ist. Wenn man nun die letzte Fläche die die neue Gerade enthält auch gefunden hat, dann weiss man auch die letzte Schnittkante (mit dem Rand). Die Laufzeit ist jetzt nur proportional zu dem durchlaufenen Gebilde. Das Splitten einer Fläche zum Update der Doppeltverketteten Liste geht in $O(1)$.]



88. **Was ist die Komplexität einer Zone einer Linie?** [Die Summe aller Anzahlen von Kanten und Ecken aller geschnittenen Flächen.]

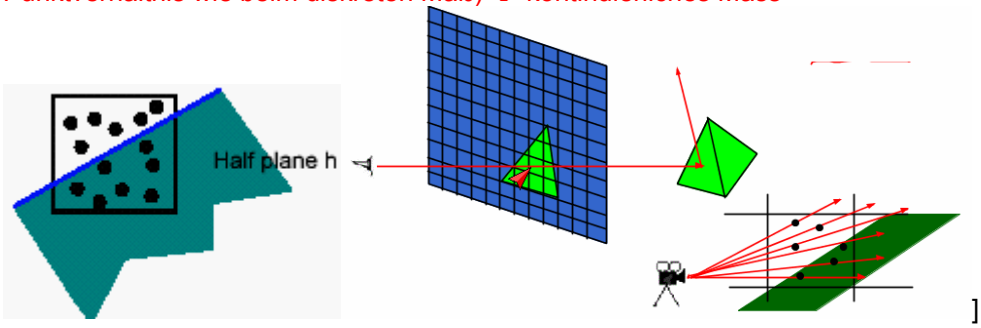
89. **Was besagt das Zonentheorem?** [Das Zonetheorem besagt, das die Komplexität der Zone einer Linie in einem Arrangment von m Linien in der Ebene $O(m)$ beträgt, oder anders ausgedrückt, dass wenn man eine neue Gerade wie oben in ein bestehendes Arrangement einfügt, die Laufzeit dafür proportional zu m ist, wobei m die Anzahl der bereits vorhanden Geraden ist. D.h. Die Zone hat eine Komplexität von $O(m)$. OHNE BEWEIS IN DER VORLESUNG WEIL ZU KOMPLIZIERT


]

90. **Wie schnell kann man nun mit Hilfe des Zonetheorems und einer Doppeltverketteten Kantenliste den Level einer Punktemenge berechnen?** [in $O(n^2)$, da es ja $O(n^2)$ Schnitte gibt und durch das Zonetheorem die Komplexität ja im Prinzip nicht steigt]

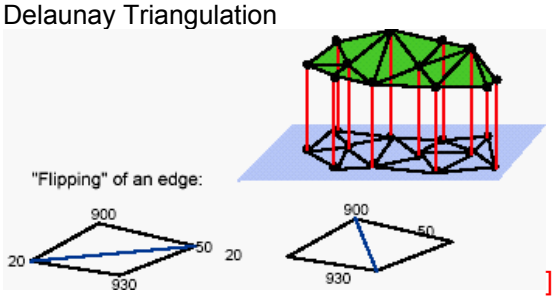
91. **Wie wird Halbebenen Diskrepanz beim Raytracing verwendet?** [Man möchte ja z.B. ein 3D Objekt auf ein Diskretes 2 Dimensionales Raster abbilden, wie z.B. eine Bitmapgrafik. Nun ist ein Pixel z.B. genau am Rand eines Objektes. Die Frage ist es nun, wie man dieses Pixel einfärbt. Beim Raytracing schickt man nun feine „Strahlen“ durch dieses Pixel. Für die Auswertung gibt es 2 Möglichkeiten, die dann aber bei der Halbebenen-Diskrepanz kombiniert werden:
1.Methode: Man zählt im Prinzip die Anzahl der Strahlen auf der einen Halbebene und die auf der Anderen und berechnet damit die resultierende Farbe ➔ diskretes Maß.
2. Methode: Man legt einen Schnitt durch die gemessenen Punkte (es kann mehrere Schnitte geben), der diese in 2 Teile teilt, und nimmt als Maß die Flächenverhältnisse (und nicht das

Punktverhältnis wie beim diskreten Maß) ➔ kontinuierliches Mass



]

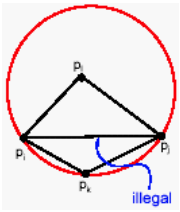
92. **Was versteht man unter Halbebenen Diskrepanz?** [Die Halbebenen Diskrepanz Ist das Supremum aller Unterschiede zwischen dem diskreten Maß und den kontinuierlichen Maßen für alle Halbebenen (Berechnung erfolgt also über alle möglichen Halbebenenschnitte des kontinuierlichen Maßes)]
    a. **Welche Eigenschaften haben Halbebenen mit maximaler Diskrepanz?** [Entweder Sie enthält 1. einen Punkt der Punktmenge direkt auf ihrem Rand oder 2. 2 oder mehr Punkte. Grund: Das kontinuierliche Maß kann ab diesem Punkt gesenkt werden, indem man die Halbebene bis zum nächsten diskreten Punkt parallel verschiebt, das diskrete Maß ändert sich dadurch aber nicht, die Diskrepanz ist also wenn überhaupt immer an diesen Übergängen maximal. Bei nur einem Punkt auf der Halbebene kann man die Ebene auch drehen um das kontinuierliche Maß zu senken. Um diese maximale Halbebenen Diskrepanz nun zu finden muss man zum Glück nur endlich viele Halbebenen ansehen, obwohl es ja unendlich viele Möglichkeiten gibt, aber diejenigen die die beiden Bedingungen oben erfüllen sind nur endlich viele. Das Ganze geht in $O(n^2)$]
93. **Was ist eine Delaunay Triangulation?** [Ein Winkeloptimale Triangulation eines Vierecks (das heißt man maximiert den minimalen Winkel). Bei Landkarten repräsentieren die Winkel z.B. Höheninformationen, es gibt nun immer zwei Möglichkeiten ein Viereck zu triangulieren, bei dem Bild unten links sieht es nun aus als wäre ein langes tiefes Tal zwischen zwei Bergen, wohingegen bei der rechten Triangulation ein leichtes Grat eingezeichnet ist. Die rechte Triangulation sieht zumindest was solche Karten angeht realistischer aus und ist auch eine Delaunay Triangulation



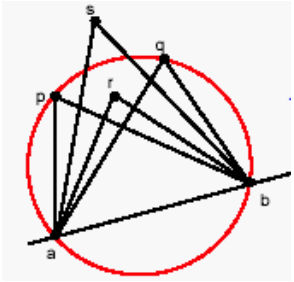
]
    a. **Anwendungsgebiete?** [Höhenlandkarten, man möchte also z.B. aus einer Menge von Punkten im 3-Dimensionalen eine möglichst realistische Landkarte erzeugen, um diese mit dem Computer darstellen zu können.]
94. **Auf was muss man bei der Delaunay Triangulation bzgl. der Winkel achten?** [Der kleinste Winkel soll maximiert werden, d.h. es ist immer die Triangulation eine Delaunay Triangulation deren kleinster Winkel größer ist]
95. **Wie viele Dreiecke und Kanten entstehen bei der Triangulation, von was hängt diese Anzahl ab?** [2n-2-k Dreiecke und 3n-3-k Kanten, wobei n die Anzahl der Punkte in der Ebene sind und k die Anzahl der Punkte auf der konvexen Hülle der Punktemenge ➔ Berechnet mit Euler Formel (n-e+f=2]
96. **Kann man auf beliebige weise triangulieren?** [ja aber nur endlich viele Kombinationen]
97. **Was ist der Winkelvektor?** [Alle Winkel aller Dreiecke (3m) aufsteigend geordnet. Es gibt insgesamt 3m Winkel, wenn m die Anzahl der Dreiecke ist.]
98. **Wann ist eine Triangulation Winkel-optimal?** [Wenn der Winkelvektor maximal ist bzgl. der anderen Triangulationsmöglichkeiten]
    a. **Wann ist eine Kante illegal?** [Wenn der minimale Winkel des Winkelvektors dieser Triangulation kleiner als der minimalste Winkel der anderen Triangulation ist.]

b. **Wann Legal?** [Wenn der minimalste Winkel des Winkelvektors durch flippen der Kante des Quadrilaterals nicht größer wird]
c. **Geht das immer?** [klar, es kann natürlich sein, dass beide Kanten legal sind.]

99. **Wie kann man den Illegalitätstest grafisch lösen?** [Kreiskriterium: Man zieht einen Umkreis um das entstandene Dreieck (d.h. einen Kreis der die 3 Punkte des Dreiecks enthält). Die Kante ist dann illegal, wenn der 4 Punkt des Quadrilaterals innerhalb des Kreises liegt. Wenn alle 4 Punkte auf dem, Kreis liegen, dann sind beide legal, geht aber eigentlich nicht, wenn die Punkte in allgemeiner Lage sind. Es gibt immer genau eine illegale Kante. Der Test ist symmetrisch]

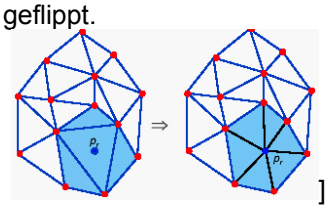


a. **Auf was beruht dieser Test?** [Satz von Thales ➔ Alle Winkel an Punkten auf dem Thaleskreis sind gleich. Die Winkel von Punkten ausserhalb des Kreises sind kleiner also die auf dem Kreis und die Winkel innerhalb des Kreises sind größer als die auf dem Kreis.



siehe Bild (rechts):
Hier sind die Winkel die von apb und aqb eingeschlossen werden gleich. Der Winkel asb ist kleiner als arb.]

100. **Was passiert, wenn alle 4 Punkte auf dem Kreis liegen?** [Beide legal]
101. **Kann man jede beliebige Triangulation in eine winkeloptimale Triangulation in endlich vielen Schritten umwandeln?** Ja, denn spätestens wenn man alle Kanten geflippt hat, dann ist es fertig. Eine einmal geflippte Kante muss nicht mehr geflippt werden.]
102. **Was hat die Delaunay Triangulation mit dem Voronoi Diagramm zu tun?** [Es gibt auch eine andere Definition der DT und zwar. Die DT ist das Straight Line dual des VD. D.h. benachbarte Punkte werden direkt verbunden | Eine andere Definition sagt: Wenn P eine Punktmenge in allgemeiner Lage ist und T eine Triangulation von P, dann ist T eine Delanay Triangulation, wenn der Umkreis jedes Dreiecks keinen weiteren Punkt enthält]
103. **Wie funktioniert der inkrementelle Triangulationsalgorithmus?** [Man fängt mit einem Um-Dreieck an, welches alle Punkte von P enthält und fügt nach und nach zufällig Punkte hinzu. Dann muss man suchen, in welchem Dreieck bzw. auf welcher Kante der neue Punkt liegt und entsprechende Kanten einfügen, und trianguliert in der Weise, das die Triangulation zu jedem Zeitpunkt eine DT ist. Das Außendreieck stört die innere Triangulation nicht (Das Kreiskriterium ist ja nicht erfüllt, das ist aber egal)!]
    a. **Beim Einfügen eines neuen Punktes gibt es 2 Möglichkeiten, welche?** [1.: Punkt liegt innerhalb eines Dreiecks, d.h. das Dreieck wird in 3 neue Dreiecke zerteilt. Durch das Einfügen kann natürlich eine Kante illegal werden, das heisst diese muss geflippt werden, zusätzlich muss man dann noch die direkt angrenzenden Kanten auf legalität testen, da sich dort auch illegale Kanten einschleichen können. Mehr als diese müssen aber nicht mehr betrachtet werden, da sich die Illegalität nicht durch den gesamten Graphen fortpflanzt. 2.: Der neue Punkt liegt direkt auf einer Kante, dann werden die beiden angrenzenden Dreiecke geteilt. Und die Kanten ggf. wieder geflippt.


]

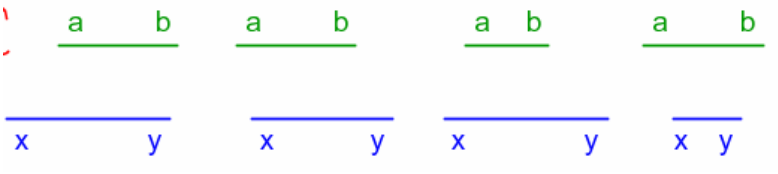
104. **Wie sieht die Datenstruktur für die Punktsuche in einem Delaunay Graphen aus?** [Hierarchisch aufgebaut während der Konstruktion. Wann immer ein neuer Knoten hinzukommt, wird der Knoten in 3 Söhne geteilt, d.h. die neuen Teildreiecke. Falls ein Flip stattfindet, werden ebenfalls neue Söhne generiert. Bei jedem Flipp und Einfügen erhöht sich die Höhe des Baumes um 1 ➔ Punktlokalisazion geht also immer Tiefer in die Baumstruktur
105. **Wie viele Dreiecke werden beim inkrementellen Algorithmus zur Delaunay Triangulation höchstens produziert?** [9n+1]
106. **Wie schnell und mit welchem Platzbedarf kann man Delaunay-triangulieren?** [O(n log n) erwartete Zeit und O(n) erwarteter Platz bei n Punkten in der Ebene.]

107. **Wie kann man beim Rechteckschnitt überlappende Kanten finden?** [Scanline Algorithmus, Event Queue: Obere und untere Begrenzungen der Rechtecke ➔ Wie bei dem Liniensegmentschnitt | Das geht in O(n log n) Zeit zu sortieren (Initialisieren der Eventqueue]

108. **Wie sieht die Status Struktur dazu aus?** [Wann immer man an einem Rechteckintervall vorbei kommt, wird dieses in die Statusstruktur geschrieben und verbleibt dort so lange, bis das Rechteck zu Ende ist. | Man muss also Intervalle einfügen und entfernen können. Desweiteren muss man die überlappenden Intervalle finden können. Die Intervalle bestehen alle aus diskreten Anfangs und Endpunkten, die im Prinzip vorher bekannt sind. Maximal 2n Anfangs-/Endpunkte]

109. **Wie funktioniert die Überlappungsanfrage zweier Intervalle der Statusstruktur?** [Bereichssuche eines Punktes in den Intervallen ➔ In der Statusstruktur wird dies gleich repräsentiert]

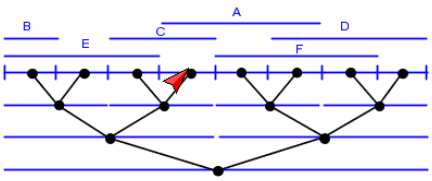
110. **Welche Möglichkeiten der Überlappung gibt es?** [4 Möglichkeiten



]

111. **Welche Baumstruktur ist zur Speicherung erforderlich?** [1. Bereichssuchbaum der die linken Endpunkte der gerade aktiven Intervalle enthält ➔ Realisiert mit einem balancierten Blattsuchbaum | 2. Struktur zur Speicherung der Intervalle, welche die Operationen Einfügen, entfernen und Aufspiessanfragen unterstützt ➔ Realisiert mit Segment Bäumen]

112. **Was ist ein Segment Baum?** [Die Blätter sind Elementarfragmente der Intervalle. Der Baum soll möglichst flach sein.]



   a. **Wie schnell kann man in die Segment Bäume ein Intervall einfügen?** [O(log n) wegen der Höhe des Baumes]
   b. **Wie schnell kann man den Segment Baum konstruieren?** [O(n log n) in O(n log n) Platz]
   c. **Wie schnell kann man zu einem bestimmten Punkt die Intervalle finden, die diesen enthalten?** [O(log n + k), wobei k die Anzahl der berichteten Intervalle ist ➔ Output-Sensitiv]
   d. **Wie schnell kann man löschen?** [O(log n) (gleich wie beim Einfügen)]

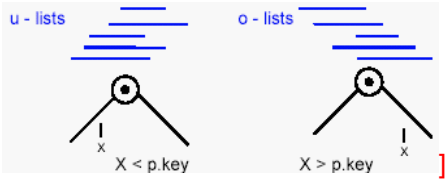
113. **Wie schnell kann man nun das gesamte Rechteckschnittproblem lösen?** [O(n log ² n + k)]

114. **Wie kann man das mit Hilfe eines globalen Wörterbuches verbessern?** [Alle Intervalle werden in einer doppelt verketteten Liste gespeichert ➔ Alle Operationen können dann in O(log n) ausgeführt werden ➔ Das Rechteckschnittproblem geht damit in O(n log n) ➔ Der Platzbedard kann noch auf O(n) gedrückt werden]

115. **Was sind Intervallbäume?** [Jeder Knoten ist mit einer Zahl beschriftet, und ist mit jeweils 2 Listen und zwar eine u-Liste und eine o-Liste beschriftet, in denen die Zahl des Knotens enthalten ist. In der u-Liste sind die aufsteigende unteren Endpunkte gespeichert, in der o-Liste die abnehmenden oberen Endpunkte. Der Baum kann als Skelett vorgegeben sein, was heisst, das er als kompletter Suchbaum für die Intervallgrenzen vorgegeben ist, wobei die Einträge in den Knoten leer sind.]

   a. **Wie schnell kann man hier einfügen bzw. löschen?** [in O(log n) Zeit, wobei das Finden des Knotens log n Schritte geht und das Finden des Intervalls dann auch noch mal log n Schritte, wobei der Baum O(n) Speicherplatz benötigt. Die Laufzeit bezieht sich hier auf einen Baum der schon als Skelett mit einer Größe von O(n) vorgegeben ist.]
   b. **Wie kann man nun diese Aufspiessanfragen machen?** [Will man zu einem Anfragepunkt x alle Intervalle suchen will, die diesen enthalten geht man folgendermaßen vor: Man durchläuft den Baum an der Wurzel beginnend. Hierbei gibt

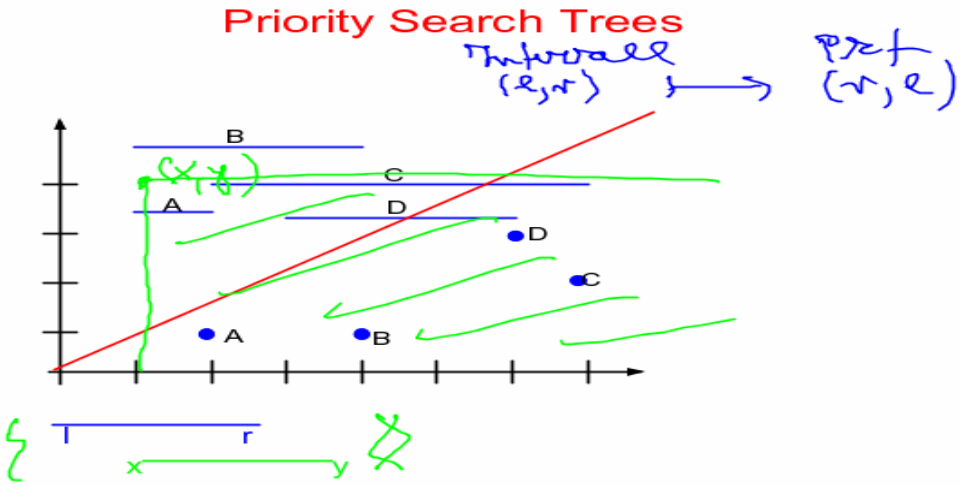
es 3 Fälle wenn man sich an einem Knoten p befindet:
1. x=p.key ➔ Anfragepunkt und Knotenwert identisch ➔ Alle Intervalle der u- und o-Liste zurückgeben. ➔ keine Rekursion
2. x<p.key ➔ Anfragepunkt kleiner als Knotenwert ➔ Anfang der u-Liste zurückgeben, d.h. alle Intervalle zurückgeben, deren unterer Endpunkt kleiner als der Anfragepunkt ist. Die oberen Endpunkte brauchen nicht betrachtet werden, da die Intervalle mindestens bis p.key reichen. ➔ Jetzt rekursiv mit dem linken Kind weitermachen und wieder die 3 Fälle prüfen.
3. x>p.key ➔ Anfragepunkt größer als Knotenwert ➔ Anfang der o-Liste zurückgeben. ➔ Jetzt mit dem rechten Kind weitermachen, und wieder die 3 Fälle prüfen.



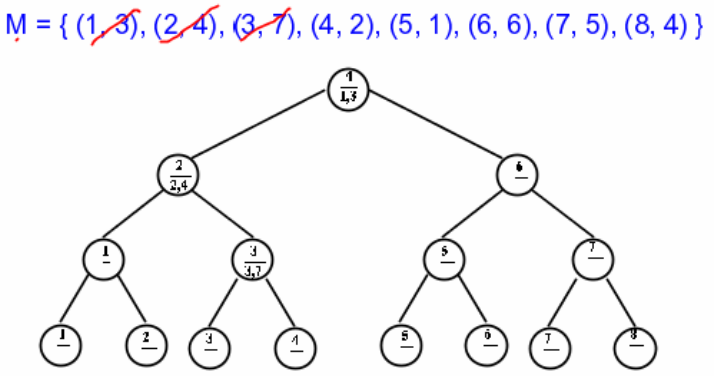
]

i. **Wie schnell gehen diese Anfragen?** [O(log n + k), wobei k die Anzahl der berichteten Intervalle ist ➔ Output-Sensitiv]

116. **Was sind Priority Search Trees?** [Zunächst bildet man die Intervalle auf Punkte in der Ebene ab, d.h. wenn ein Intervall von x1 bis x2 reicht, dann wird daraus ein Punkt mit den Koordinaten P(x2,x1). Alle diese Punkte liegen unter der Diagonalen durch das Koordinatensystem. Hat man nun eine Intervallanfrage y1,y2, transformiert man diese wieder in einen Punkt, diesmal aber andersherum also in einen P(y1,y2). Dieser Punkt liegt dann über der Diagonalen. Nun schneidet das Intervall all diejenigen Intervalle, deren transformierte Punkte rechts unterhalb des Punkte P(y1,y2) liegen.



Priority Search Trees von McCreight sind nun eine Hybrid Struktur mit im Prinzip Dimension 1,5 aus Suchbaum und Min-Heap ➔ Suchbaum für die x-Werte und ein Min-Heap für die y-Werte]

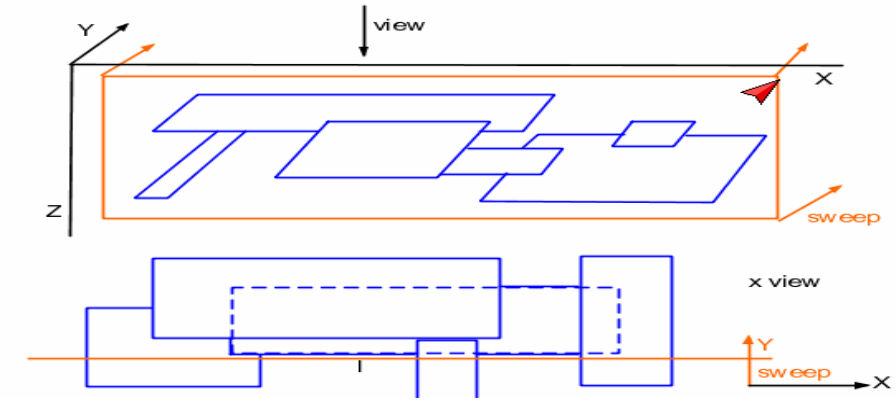


a. **Was passiert, wenn man nun den Punkt (4,2) einfügt?** [(4,2) wird in die Wurzel eingefügt und der bestehende Punkt (1,3) entfernt und neu eingefügt an den linken

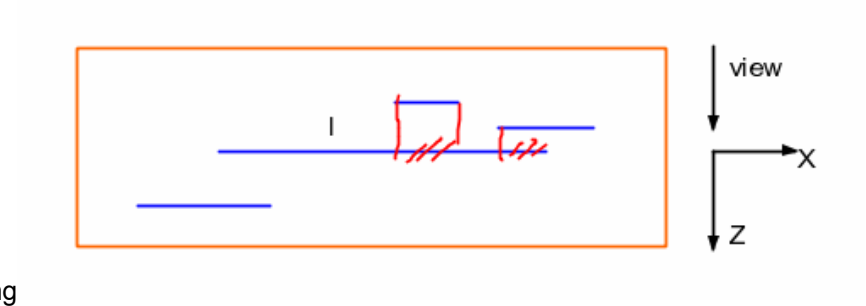
Sohn➔ der Knoten(2,4) wird nun entfernt und erneut eingefügt an den linken Sohn. Man verdrängt also immer die Punkte und fügt sie neu ein.]

b. **Wie sieht am Anfang das Skelett des Baumes aus?** [Raster mit den X-Werten an den Blättern]
c. **Warum sagt man ein Priority Search Tree hat Dimension 1.5?** [Weil man keine beliebige Bereichsanfrage machen kann, sondern nur eine südliche Bereichsanfrage, das heisst der gesuchte Rechteckbereich muss an der X-Achse anliegen]
d. **Welche Operationen kann man mit diesen ausführen?** [Einfügen eines Punktes, Löschen eines Punktes und eben die südliche Bereichsanfrage]
e. **Was haben diese mit Binärbäumen und Min-Heaps gemeinsam?** [Sie sind eine Hybrid Struktur. Der Baum ist ein Suchbaum für X-Koordinaten und ein Min-Heap bzgl der Y-Werte.]
f. **Wie schnell geht das Einfügen?** [O(log n)]
g. **Wie schnell kann man entfernen?** [O(log n) ➔ Hat man eine Lücke erzeugt muss man die unteren Punkte hochziehen und zwar immer den Punkt mit dem kleineren y-Wert, damit die Heapbedingung nicht verletzt wird]
h. **Wie schnell geht die südliche Bereichssuche?** [O(log n + k) z.B. Q(x,x',y) ➔ Man berichtet also alle Punkte die zwischen x und x' liegen und eine y-Koordinate haben die kleiner als y ist. Man benötigt log n Schritte um die X-Bereiche zu finden und k Schritte für die Y-Abgrenzung. Hierbei liegen die zu berichteten Punkte oben im Baum, also ev. nicht in den Blättern, den in den Blättern stehen ja die Knoten mit den größten y-Werten.]
i. **Wie kann man einen Priority Search Tree voll dynamisieren?** [Man trennt sich von der starren Skelettstruktur und nimmt einen natürlich Suchbaum, wann immer man ein neues Element einfügt, wird der Suchbaum erweitert]
j. **Wie verhindert man das Degenerieren?** [Man nimmt statt eines natürlichen Suchbaumes einen Balancierten Suchbaum, aber das wiederherstellten der Balancebedingung setzt Rotationen voraus. Unglücklicherweise erhält die Rotation zwar die X-Schlüssel, die Y-Anordnungen des Heaps gehen aber verloren. Dies muss man mit log n Schritten wieder „reparieren". Man kann dann also nur noch in O(log ² n Schritten einfügen]

117. **Was versteht man unter Hidden Line Elimination?** [Berechnung der sichtbaren Flächen in dem man die verborgenen/überdeckten Linien eliminiert.]
    a. **Anwendungsgebiete?** [3D-Computergrafik]
118. **Welche 3 spezielle Problemklassen gibt es dabei?** [Problemklasse A: isoorientierte rechteckige Flächen (alle Flächen sind parallel zur Projektionsebene, man schaut also immer rechtwinklig auf rechteckige Flächen), Problemklasse B: C-Orientierte Flächen, eine Erweiterung der Problemklasse A (höchstens c Richtungen, z.B. Dreiecke). Problemklasse C: C-Orientierte Körper im Raum (z.B. Pyramide)]
119. **Wie funktioniert Plane-Sweep?** [



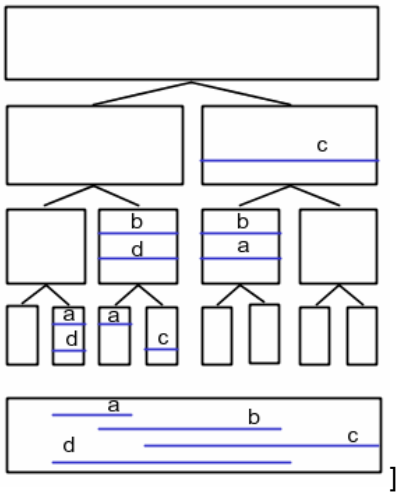
Wenn wir unsere Sweepplane über unsere Flächen hinwegschiebt, dann erscheinen Projektionen , bleiben eine weile und sie verschwinden wieder. Man betrachtet also die Rechtecke von der Seite her und ermöglicht somit eine Intervall-Darstellung der aktiven Rechtecke, allerdings gibt es mehrere Ebenen (z-Richtung) für die Intervalle.]
120. **Was bedeutet die Bedeckungszahl (Coverage Number)?** Die Anzahl derjenigen Fläche, die über unserem Liniensegment liegen. Man durchläuft in x-



Richtung
Die Berechnung der Bedeckungszahl gliedert sich in 2 Sub-Probleme. Wir müssen 1. zunächst die Bedeckungszahl des linke Endpunktes eines Liniensegmentes berechnen und 2. müssen wir für ein bestimmtes Intervall dieses Segmentes diejenigen Endpunkt finden, die innerhalb dieses Intervalls liegen und die Coverage Numbers updaten. Ein Intervall dessen Coverage Number 0 ist, ist somit von oben her sichtbar, da es von nichts überdeckt wird, und kann somit als sichtbar zurückgegeben werden.]

a. **Wie schnell kann man diese Berechnen bei bereits vorhandenem „Range Tree“ (Balanced Leaf Search Tree)?** [$O(\log^2 n)$]

121. **Wie werden die Intervalle im jeweils aktuellen Sweepzeitpunkt gespeichert?** [Segment Range Tree



]

122. **Wie kann man mit Hilfe des Segment Range Trees die Coverage Number bestimmen?** [Man muss nur den Baum bis zum Blatt durchlaufen. Man kann sogar nicht nur die Anzahl bestimmen, sondern sogar die betreffenden Segmente bestimmen. Dies geht in $O(\log^2 n + t)$ Tiefe des Baumes ist log n und in jedem Knoten muss man eine Bereichssuche machen in log n. t ist die Anzahl der berichteten Segmente]

123. **Was wäre ein Segment Rank tree?** [Man speichert in den Knoten die Coverage Numbers. Anfrage in $O(\log^2 n)$]

124. **Ein zweites Problem ist es die Endpunkte eines Segmentes und jeweils eingeschlossene Flächen zu berichten, wie schnell geht das?** [$O(\log n + r_e)$ wobei r_e die Anzahl der Intervalle sind, die geschnitten werden.]

125. **Wie schnell kann man also einfügen und entfernen in einen „Segment Range Tree“ und in einen „Range Tree“?** [$O(\log^2 n)$ bzw. $O(\log n)$]

126. **Wieviel Platz benötigt der „Segment Range Tree“?** [$O(n \log n)$, n = #Segmente]

127. **Wie schnell kann man nun das Problem lösen, bei dem alle Flächen Parallel zur x und y-Achse sind (Problemklasse A)?** [$O(n \log^2 n + k$ Zeit und $O(n \log n)$ Platz, wobei k die Anzahl der Kantenschnitte in der Projektionsebene bezeichnet.]

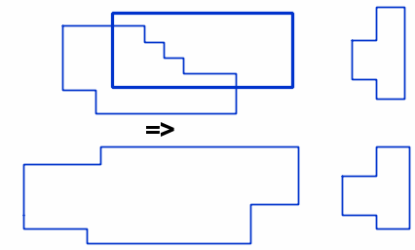
128. **Das Zweite Problem (Problemklasse B) sind zur z-Achse parallele Flächen, deren Kanten allerding in c verschiedene Richtungen zeigen (z.B. Dreiecke), wie kann man das Plane Sweep Verfahren hierbei anwenden?** [Man lässt die Sweepline in jede der c Richtungen laufen. D.h. c verschiedene Datenstrukturen]

a. **Wie schnell ist dieses Verfahren im Verhältnis zum vorherigen?** [gleich schnell wie Problem A. Der einzige Unterschied ist der Faktor c, aber der ist konstant]

129. **In der dritten Problemklasse sind nun alle Richtungen beliebig, wie schnell kann dieses gelöst werden?** [Auch gleich schnell, aber c^3]

130. **Es gibt einen Algorithmus für das HLE-Problem, der nicht output-Sensitiv ist, wie heisst dieser ggf.?** [Dynamic Contour Maintenance]]

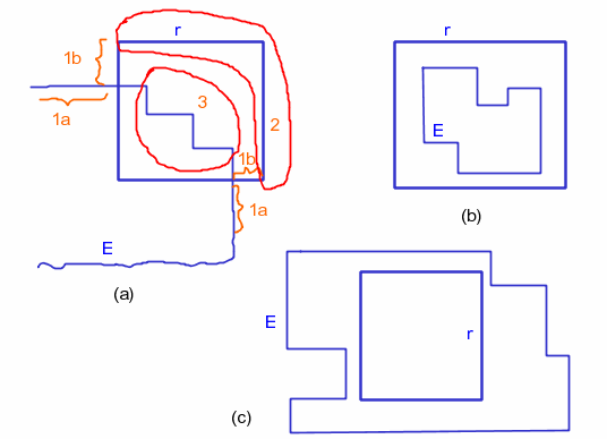
131. **Wie ist das generelle Prinzip dieses neuen Algorithmus?** [Man konstruiert die sichtbare Szene durch Einfügen von Objekten von vorne nach Hinten in die anfangs leere Szene. Wir erzeugen die sich ergebende Kontur dynamisch. Wenn ein neues Objekt hinzu kommt, wir berechnet ob dieses ganz oder teilweise durch die bisherige Kontur überdeckt wird ➔ Update der Kontur



Man berechnet dann alle Schnitte von r und c, alle Kanten, von r die komplett innerhalb oder ausserhalb von c liegen bzw. alle Kanten von c, die vollständig innerhalb von r liegen]

132. **Welche Fälle der Überdeckung kann es dabei geben und wie werden diese behandelt?** [
1. das neue Rechteck r liegt komplett innerhalb von der Kontur c ➔ das neue Rechteck ist nicht sichtbar ➔ kein Update der Kontur
2. das neue Rechteck r schneidet die Kontur c ➔ Kontur updaten ➔ dazu müssen die Schnitte der Kanten des Rechtecks mit der Kontur berechnet werden, und innerhalb liegende Segmente entfernt werden, und ausserhalb liegende Segmente, der Kontur hinzugefügt werden.
3. die Kontur c liegt vollständig innerhalb oder vollständig ausserhalb von r ➔ ggf. Kontur updaten, falls ausserhalb]

133. **Welche Teilprobleme treten bei diesem Algorithmus auf?**[Schnitt/Einzelpunkt in Kontur/Welche Punkte in einem bestimmten Rechteck]



a. **Welche Datenstruktur wird jeweils benötigt?** [Schnitt: Segment Range Tree | Einzelpunkt in der Kontur: Segment-Segment Tree | Punkte im Rechteck: Range-Range Tree]

134. **Wie schnell kann man die erste Problemklasse A nun mit dem nicht outputsensitiven Algorithmus lösen? Welcher Platz wird benötigt?** [In $O((n+q)\log^2 n)$ Zeit und $O((n+q)\log n)$ Platz, n=#Rechtecke, q=#sichtbarer Liniensegmente]

135. **Lässt sich diese Lösungsstrategie auch auf die Problemklassen B und C ausdehnen?** [Auf Problem B ja, auf Problem C nicht, da die Durchlaufrichtung so nicht machbar ist]