Graphische Datenverarbeitung - Eine fruchtbare Verbindung von Mathematik und Informatik

Andreas Schwill, Universität Potsdam

1 Einleitung

Obwohl Mathematik und Informatik gemeinsame Wurzeln besitzen, gibt es nicht viele schuladäquate Teilbereiche, in denen sich Methoden der Mathematik und der Informatik organisch verknüpfen lassen. Die üblichen in der Mathematik vorhandenen Probleme sind aus informatischer Sicht nicht strukturreich genug, um die Modellierungs- und Beschreibungsmethoden der Informatik voll zur Wirkung kommen zu lassen. Umgekehrt erfordern die informatischen Probleme nur selten schultypische mathematische Vorgehensweisen[1].

Die graphische Datenverarbeitung scheint ein Kandidat zu sein, in dem sich Methoden beider Wissenschaften sinnvoll kombinieren lassen. Zum einen erfordert die Lösung der Probleme die Kenntnis typischer Verfahren der Schulmathematik (z.B. Geraden- und Kreisgleichungen, Projektionen, Spiegelungen), zum anderen treten Modellierungsfragen hinzu, wenn idealisierte mathematische Objekte, wie Geraden, Kreise, auf Raster- oder Vektorgraphikgeräten ausgegeben werden sollen. Wir wollen im folgenden einige zentrale Fragestellungen der graphischen Datenverarbeitung vorstellen, die in der Schule im Informatikunterricht, im Mathematikunterricht oder in interdisziplinären Projekten behandelt werden können.

2 Grundlagen

Unter graphischer Datenverarbeitung (kurz GDV) versteht man ein Teilgebiet der Informatik, das sich mit der Erfassung, Speicherung, Verarbeitung und Ausgabe graphischer Darstellungen befaßt. Das häufig in diesem Zusammenhang favorisierte Gebiet der Algorithmischen Geometrie ist *kein* originäres Teilgebiet der GDV, denn hier geht es weniger um die Darstellung geometrischer Objekte als um die Ermittlung von Zusatzinformationen, die aus der Darstellung erst ermittelt werden müssen.

Anwendungen der GDV findet man vor allem beim

- computerunterstützten Entwerfen (CAD)
- bei Präsentationsgraphiken
- bei medizinischen Anwendungen (z.B. Tomographie)
- bei der Kartographie
- bei Filmen
- bei Animationen
- und und und.

Der von der Informatik in diese vom Grundsatz her mathematisch geprägten Anwen-

[1] Man frage mal einen Informatiker, wann er zuletzt den Datentyp real sinnvolleingesetzt hat.

dungen eingebrachte Akzent betrifft vor allem die Effizienz und die Exaktheit, mit der die Darstellungen auf Ausgabegeräten produziert werden.
Nicht alle diese Anwendungen erscheinen schuladäquat, es sei denn in Projekten. Zur Behandlung im Unterricht – sei er mathematisch oder informatisch – bieten sich aber einige atomare Teilprobleme an, die in allen o.g. Anwendungen vorkommen:

- Wie zeichne ich Linien auf den Bildschirm?
- Wie zeichne ich Kreise auf den Bildschirm?
- Wie realisiere ich Fenster?
- Wie male ich eine Figur aus?
- Wie zeichne ich ein 3-dimensionales Objekt auf den Bildschirm?

Aus didaktischer Sicht bietet die GDV eine Reihe von Vorteilen. Zum einen scheint die Motivation der Schüler zur Behandlung dieser Gebiete gesichert, da man von hinreichend vielen Vorerfahrungen ausgehen kann: Spiele (z.B. Flugsimulatoren), 3D-Darstellungen, Filme (z.B. Terminator II, Jurassic Park), Stickereien, Zeichenprogramme usw. Zum anderen ist auch der Fortgang des Unterrichts gesichert, denn die o.g. GDV-Probleme sind in dem Sinne oft herausfordernd, daß mit jedem Schritt auf dem Lösungsweg weitere mögliche und für Schüler auch durchführbare Schritte sichtbar werden. Im Einzelnen besitzen die im weiteren behandelten Lösungen folgende Eigenschaften:

- Die Zielstellung jeder Problemlösung läßt sich an zwei Grundsätzen festmachen: Effizienz und Genauigkeit der Ausgabealgorithmen.
- GDV-Methoden beruhen auf wenigen fundamentalen Ideen, die in vielfältiger Kombination immer wieder Verwendung finden: Spiegelung, Reduktion auf ganzzahlige Arithmetik, Diskretisierung, Fehlerabschätzung, Approximation.
- Jedes Problem besitzt einfache und schwere Lösungen; die einfachen sind mit geringen mathematischen und informatischen Methoden erzielbar, die leistungsfähigeren ergeben sich häufig inkrementell unter schrittweiser Einbindung der o.g. fundamentalen Ideen durch organische Weiterentwicklung der naiveren. Geniale Ideen sind nur in wenigen Fällen erforderlich. Insbesondere sichert diese Eigenschaft, daß die Schüler stets eine Lösung finden.

Im folgenden wollen wir einige wichtige Algorithmen skizzieren und die jeweils wichtigsten Lerninhalte aus mathematischer und informatischer Sicht aufzeigen.

3 Liniendarstellung

Die generelle Vorgabe bei allen in der Folge in einer PASCAL-artigen Syntax vorgestellten Algorithmen sei ein Rasterausgabegerät (ein Bildschirm). Mittels der Prozedur

```
point (x,y: integer)
```

werde ein schwarzer Punkt auf die ganzzahligen Bildschirmkoordinaten (x,y) gesetzt. Punkte seien definiert durch

```
record
        x,y: integer
end.
```

Der einfachste Algorithmus zum Zeichnen einer Strecke zwischen den Punkten P und P' basiert auf der Standardgeradengleichung y=mx+b. Man setzt alle ganzzahligen x-Werte zwischen den x-Koordinaten von P und von P' ein und erhält die zugehörigen y-Werte der Zwischenpunkte, die jeweils anzuzeigen sind:

```
m:=(P.y-P'.y)/(P.x-P'.x);
b:=(P.y*P'.x-P'.y*P.x)/(P'.x-P.x);
for x:=P.x to P'.x do
        point(x,round(m*x+b)).
```

Dieser Algorithmus besitzt jedoch zwei entscheidende Nachteile, die ihn für die Praxis unbrauchbar machen: Er verwendet aufwendige Gleitpunktoperationen (Division und Multiplikation), die ihn ineffizient machen, und er ist sehr ungenau, wie Abb. 1 zeigt: Bei Geraden über 45° Steigung wird nur eine geringe Zahl von Punkten angezeigt. Diesen Nachteil kann man allerdings durch eine Fallunterscheidung bezgl. der Lage der Strecke in den Oktanten korrigieren.

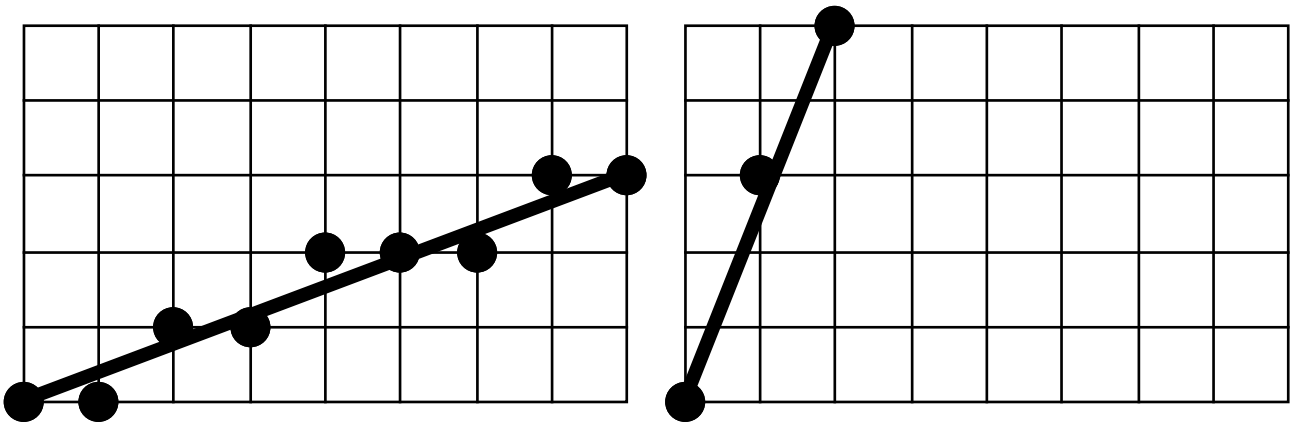

Abb. 1: Zeichnung einer Strecke durch einen naiven Algorithmus

Wesentlich effizienter und genauer ist der Bresenham-Algorithmus. Er beschränkt sich auf einfache arithmetische Operationen auf ganzen Zahlen und reduziert damit den Rechenaufwand erheblich. Die Idee besteht darin, statt jeden anzuzeigenden Punkt über die Geradengleichung explizit zu berechnen, schrittweise vorzugehen und einen anzuzeigenden Punkt (x_i,y_i) unter Vermeidung der Geradengleichung aus seinem Vorgänger (x_{i-1},y_{i-1}) zu berechnen. Man beschränkt sich dabei auf den 1. Oktanten und löst alle übrigen Fälle durch Spiegelung (Abb. 2).

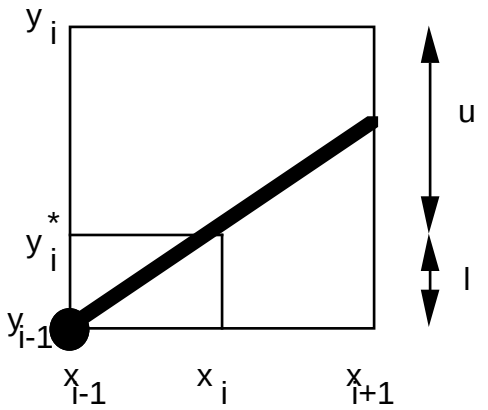

Abb. 2: Vorgehen beim Bresenham-Algorithmus für Strecken

Liegt der Punkt (x_{i-1},y_{i-1}) bereits vor, so gilt jedenfalls $x_i=x_{i-1}+1$. Ferner kommen für y_i nur die Koordinaten y_{i-1} oder $y_{i-1}+1$ in Frage, das heißt der nächste Punkt (x_i,y_i) liegt auf der gleichen Höhe wie (x_{i-1},y_{i-1}) oder aber einen Rasterpunkt höher. Sei $y_i^*=mx_i+b$ der exakte y-Wert an der Stelle x_i. Man berechnet nun die Abstände von y_i und y_{i-1} zu y_i^*:

$u_i=y_i-y_i^*$, $l_i=y_i^*-y_{i-1}$.

Das Vorzeichen der Differenz $d_i=l_i-u_i$ entscheidet nun, ob man für die y-Koordinate von x_i den Wert y_{i-1} (bei $d_i\leq 0$) oder den Wert $y_{i-1}+1$ (bei $d_i>0$) wählt.

Die hierzu insgesamt nötigen Rechenoperationen optimiert man folgendermaßen:

Fall 1: $y_i=y_{i-1}+1$ (d.h. man geht einen Rasterpunkt aufwärts). Dann gilt: $d_i=2y_i*2y_i+1=2(y_{i-1}*+m)-2y_{i-1}-1=2y_{i-1}*-2y_{i-1}+2m-1=d_{i-1}+2m-2$.

Fall 2: $y_i=y_{i-1}$ (d.h. man bleibt auf der y-Ebene des Vorgängerpunktes). Dann gilt: $d_i=2y_i*-2y_i=2(y_{i-1}*+m)-2\ y_{i-1}=2y_{i-1}*-2\ y_{i-1}+2m=d_{i-1}+2m$.

Die in beiden Fällen noch vorhandene Gleitpunktrechnung in m=(P.y-P'.y)/(P.x-P'.x) beseitigt man durch Übergang von d_i zu $d_i'=d_i$ (P.x-P'.x). Auf das Vorzeichen hat diese Transformation keine Auswirkung, da man sich im ersten Oktanten befindet.

Das gesamte Verfahren ergibt sich schließlich durch Wahl des Startwertes für d_1':

$d_1'=2(P.y-P'.y)-(P.x-P'.x)$.

Der Algorithmus:

```
x:=P.x; y:=P.y;
dx:= P'.x-P.x; dy:=P'.y-P.y;
c:=2*dy; delta:=c-dx;
c':= delta-dx;
repeat
    point(x,y);
```

```
        x:=x+1;
        if delta≤0 then
            delta:=delta+c
        else
            y:=y+1;
            delta:=delta+c'
        fi;
    until x>P'.x.
```

Mit der Bearbeitung dieser Problemstellungen sind eine Reihe zentraler Tätigkeiten beider Wissenschaften verknüpft:

- aus der Mathematik: das Aufstellen von Geradengleichungen, die Behandlung von Steigungen und die entsprechende Klassifikation von Geraden, der Umgang mit Spiegelungen an Achsen und Vorzeichenwechseln.
- aus der Informatik: Verzweigungen, Schleifen mit while und repeat, der Datentyp Verbund, geschachtelte Anweisungen, Komplexitätsbetrachtungen über arithmetische Operationen.
- für beide Wissenschaften: Sehr ertragreiches schrittweises methodisches Erschließen eines kleinen Bezirks durch Lösung von Teilproblemen und durch Verallgemeinerung von Lösungen, Fertigwerden mit Rechen- und Darstellungsungenauigkeiten.

4 Kreisdarstellung

Die Erkenntnisse und Überlegungen aus der Entwicklung eines einfachen Algorithmus zum Zeichnen von Linien können z.B. in Form eines Projekts auf das Zeichnen von Kreisen transferiert werden. Zusätzliche Kenntnisse oder Techniken sind nicht erforderlich oder können von den Schülern selbst entwickelt werden.

Der Bresenham-Algorithmus für Kreise basiert ebenfalls auf der Idee, einen anzuzeigenden Punkt (x_i,y_i) unter Vermeidung der Kreisgleichung $x^2+y^2=r^2$ aus seinem Vorgänger (x_{i-1},y_{i-1}) zu berechnen. Man beschränkt sich diesmal auf den 2. Oktanten und auf Kreise um den Ursprung und löst alle übrigen Fälle durch Spiegelung und Translation (Abb. 3).

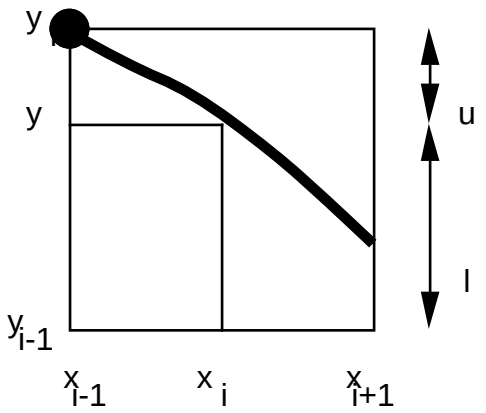

Abb. 3: Vorgehen beim Bresenham-Algorithmus für Kreise

Auf die einzelnen Schritte wollen wir hier nicht eingehen, es genügt die iterative Berechnungsvorschrift für die Werte d_i:

$d_{i+1} = d_i+4x_i+6+2(y_{i+1}^2-y_i^2)-2(y_{i+1}-y_i)$.

Für $d_i<0$ wählt man als y-Koordinate für x_i den Wert y_i-1 und für $d_i\geq 0$ den Wert y_i. Anschließend kann man die Gleichung noch operationell optimieren, denn es gilt:

$d_{i+1} = d_i+4(x_i+1)+2$, falls $d_i<0$

und

$d_{i+1} = d_i+4(x_i+1)+2-4(y_i-1)$, falls $d_i\geq 0$.

Mit dem Startwert

$d_1=3-2r$

ergibt sich dann folgender einfacher Algorithmus:

```
x:=0; y:=r;
d:=3-2*r;
while x≤y do
    point(x,y);
    if d≥0 then
        d:=d-4*(y-1);
        y:=y-1
    fi;
    d:=d+4*(x+1)+2;
    x:=x+1
od.
```

Gegenüber dem Bresenham-Algorithmus für Linien ergeben sich zusätzlich folgende unterrichtlichen Aspekte:

- für die Mathematik: die Vertiefung der Kreisgleichung, die Behandlung einfacher Translationen (wenn Kreismittelpunkte nicht im Ursprung liegen).

- für die Informatik: die Optimierung von Rechenoperationen, auch wenn mathematische Gleichungen dadurch ein ungewohntes Aussehen bekommen.
- für beide Wissenschaften: die organische Erweiterung und Übertragung von Kenntnissen auf ein neues Problem in Form einer Haus- oder Projektarbeit.

5 Fenstertechnik

Alle Computersysteme ermöglichen die Aufteilung des physikalischen Bildschirms in logische Bildschirme (Fenster) (Abb. 4). Von einer Zeichnung dürfen nur diejenigen Bestandteile angezeigt werden, die im Fenster liegen, die übrigen werden unterdrückt.

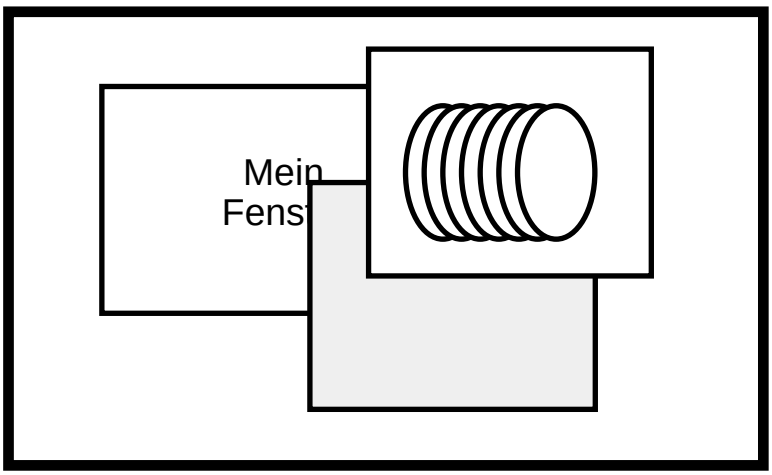

Abb. 4: Fenstertechnik

In der Schule kann man sich zunächst auf Linien beschränken.
Wir gehen von folgenden Grundannahmen aus. Ein Punkt sei jetzt definiert als

```
record
        x,y: real
end;
```

ein Fenster durch seine vier Fenstergrenzen

```
record
        left,right,low,high: real
end.
```

Man beachte, daß nun mit Welt- statt mit Bildschirmkoordinaten gearbeitet wird.

Ein einfacher Algorithmus zeichnet eine Strecke in der üblichen Form, wobei er für jeden Punkt zunächst überprüft, ob er im Fenster liegt oder nicht und angezeigt werden muß oder nicht. Dieser Algorithmus ist in hohem Maße ineffizient, da je Punkt

vier Abfragen bezüglich der Fenstergrenzen erforderlich sind.

Alternativ berechnet man wesentlich effizienter zunächst die Schnittpunkte jeder zu zeichnenden Strecke mit den Fenstergrenzen, aktualisiert die Endpunkte der Strekke so, daß das verbleibende Segment voll im Fenster liegt und zeichnet die Strecke.

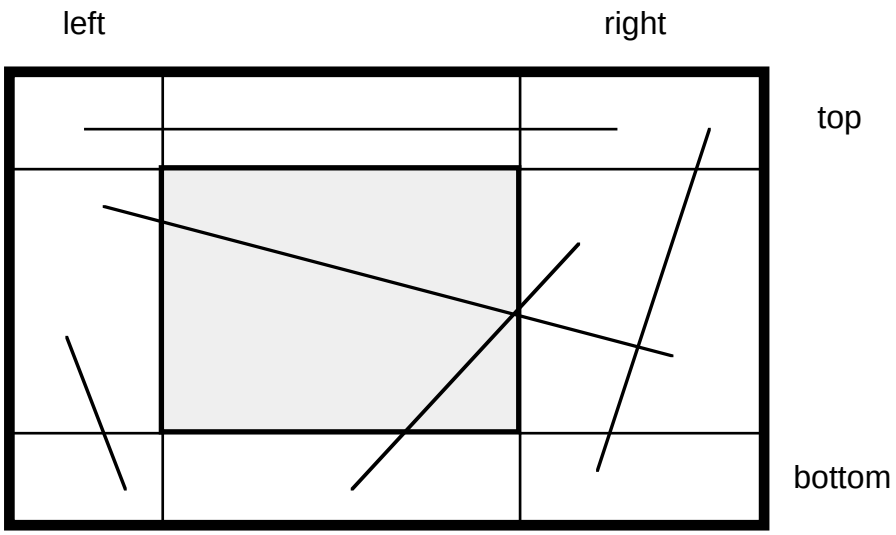

Abb. 5: Clipping

Der zugehörige effiziente Algorithmus stammt von Cohen und Sutherland. Er besteht in einem ersten Schritt darin, aus einer Menge von Strecken diejenigen auszuscheiden, die nicht angezeigt werden müssen. Dazu zerlegt man den Bildschirm in neun Rechtecke gem. Abb. 5 und vergibt Attribute $A(P) \in \{top,bottom,right,left\}$ an die Endpunkte jeder Strecke in Abhängigkeit davon, in welchem Rechteck sie liegen. Im zweiten Schritt werden Strecken, deren Endpunkte keine Attribute besitzen, vollständig angezeigt, und Strecken, deren Endpunkte ein Attribut gemeinsam haben, eliminiert, da sie nicht angezeigt zu werden brauchen. Die übrigen Fälle erfordern eine Sonderbehandlung: Sie werden an den Fenstergrenzen geclippt, d.h. mindestens einer ihrer Endpunkte wird durch den Schnittpunkt mit der Fenstergrenze ersetzt. Man durchsucht also alle übrigen Endpunkte P(x,y) und deren Attribute A(P):
Fall 1: Falls $left \in A(P)$ => clippe an der linken Fenstergrenze: y:=y+m*(left-x),x:=left
Fall 2: Falls $right \in A(P)$ => clippe an der rechten Fenstergrenze y:=y+m*(right-x), x:=right
Fall 3: Falls $top \in A(P)$ => clippe an der oberen Fenstergrenze x:=x+(high-y)/m, y:=high
Fall 4: Falls $bottom \in A(P)$ => clippe an der unteren Fenstergrenze x:=x+(low-y)/m, y:=low
Anschließend aktualisiert man A(P) und beginnt von vorn, bis alle $A(P)=\varnothing$ sind. Der

Fall m=0 erfährt eine Sonderbehandlung, auf die wir hier nicht eingehen.

Das obige Verfahren läßt sich im weiteren Verlauf des Unterrichts auf Polygone übertragen. Hier kommt das Problem hinzu, daß durch naives Clippen die Polygoneigenschaft verloren gehen kann und damit übliche Operationen, wie z.B. Färben nicht mehr zu korrekten Ergebnissen führt. Durch geschickte Erweiterung des Algorithmus von Cohen/Sutherland kann jedoch auch dieser Fall erfaßt werden, indem man die Polygonzüge zyklisch an den Fenstergrenzen clippt, damit die Fenstergrenzen als Polygonstrecken hinzu nimmt und folglich wiederum ein Polygon erhält. Auch in dieser Problemstellung besteht also für die Schüler die Möglichkeit, Mängel ihrer Überlegungen zu erkennen und mit relativ einfachen Erweiterungen selbständig zu lösen.

Auch das Problem der Realisierung von Fenstern führt zu einer Vielzahl miteinander verknüpfter mathematischer und informatischer Aktivitäten:

- aus der Mathematik: Geradengleichungen, Steigung, Schnittpunktberechnung
- aus der Informatik: Verzweigungen, Schleifen, case-Anweisung, Datentypen (vor allem der Datentyp Menge)
- wissenschaftsübergreifend: ein wirkungsvolles methodisches Vorgehen, die Vertiefung von Linien-Algorithmen, das Erkennen und Beheben von Mängeln durch Transfer bekannter Lösungsverfahren (beim Polygon-Clipping).

6 Füllalgorithmen

Zu dieser Klasse gehören alle Probleme, die sich mit dem Färben von allgemeinen Flächen befassen. In der Schule beschränkt man sich dabei zweckmäßigerweise auf eine Teilklasse, z.B. die Polygone. Bereits für diese Klasse sind effiziente und korrekte Lösungsalgorithmen (z.B. der Buffer-Fill-Algorithmus oder das Scan-Line Verfahren) recht anspruchsvoll und eignen sich eher für einen Intensivunterricht, z.B. in Form eines Projekts.

Einfache Lösungsansätze gewinnt man aber bereits ohne große Vorarbeiten, und man kann ihre Mängel analysieren. Ein naheliegender Ansatz ist z.B. das rekursive Färben. Man startet bei einem Punkt innerhalb des Polygons und färbt rekursiv die umliegenden Nachbarn (Abb. 6).

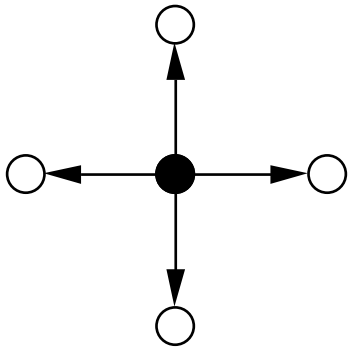

Abb. 6: Rekursives Färben

Hieran knüpfen sich zwei Überlegungen:

- Ist das Verfahren effizient? Der durch die Rekursion verursachte Effizienzverlust hinsichtlich Laufzeit und Speicher macht das Verfahren für die Praxis unbrauchbar.
- Ist das Verfahren korrekt? Färbt man die umliegenden vier Nachbarn und geht von dort rekursiv weiter, so wird in gewissen Fällen nicht das gesamte Polygon gefärbt, und man muß den Algorithmus mit anderen Startpunkten des Polygons neu beginnen. Wählt man die umliegenden acht Nachbarn, so läuft der Algorithmus aus dem Polygon heraus und färbt schließlich den gesamten Bildschirm (Abb. 6).

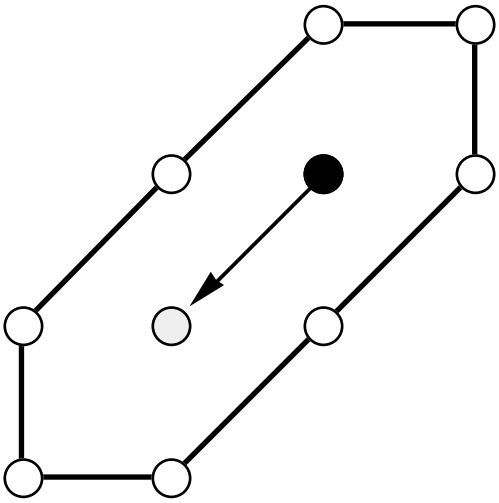

Abb. 7: Mehrfaches Ansetzen beim rekursiven Färben

Lerninhalte im Zusammenhang mit Füllalgorithmen:

- aus mathematischer Sicht: Geradengleichungen, Steigungen, Schnittpunktberechnungen, Extremwerte, Konvexität von Polygonen
- aus informatischer Sicht: Verzweigungen, Schleifen, Rekursion, Backtracking-Verfahren, lineare Listen, Sweep-line-Verfahren beim Scan-line-Algorithmus

- wissenschaftsübergreifend: Erkennen von Mängeln in (naiven und anspruchsvollen) Problemlösungen, methodisches Vorgehen.

7 Schlußbemerkungen

Wir haben anhand verschiedener Probleme der GDV gezeigt, welche Möglichkeiten es gibt, informatische und mathematische Denk- und Arbeitsweisen miteinander zu verbinden. Auch innerhalb einer Arbeitsgruppe (s. dazu den Bericht in diesem Band) war man sich darin einig, daß sich das Gebiet der GDV gut dazu eignet, denn es besitzt anspruchsvolle mathematische und informatische Inhalte, die im jeweils anderen, die Inhalte nutzenden Fach behandelt werden können. Im Detail hat sich dann aber ergeben, daß die behandelten Probleme oftmals am Rande von Mathematik und Informatik liegen, und sowohl Mathematiker als auch Informatiker Schwierigkeiten sehen, Dinge zu behandeln, die abseits des Lehrplans liegen und deren Behandlung keinen unmittelbaren fachspezifischen Nutzen liefert. Inwiefern dies auf die GDV tatsächlich zutrifft, ist in folgenden Treffen des AK Mathematik und Informatik genauer zu klären.