Proseminar Computergraphik SS2006

Vortrag Rastergraphikalgorithmen

Patrick Bergner

9. Mai 2006

Gliederung

1 **Einleitung** – Was ist eine Rastergraphik?

2 **Rasterkonvertierung** – Probleme, Effekte und Lösungsvorschläge für Linien und Kurven

3 **Füllen von Polygonen und geschlossenen Kurven**

4 **Clippen von Linien und Polygonen an Polygonen**

1 Einleitung – Was ist eine Rastergraphik?

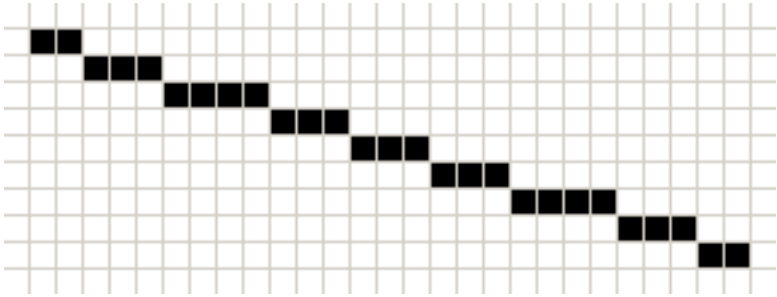
- matrixförmige Anordnung von Pixeln mit zugeordneter Farbigkeit und evtl. Alphakanal
- Merkmale: Breite, Höhe, Farbtiefe
- Ansteuerung eines Computerbildschirms geschieht mit Rastergraphik aus Framebuffer der Grafikkarte

Abbildung 1: Beispiel einer gerasterten Grafik

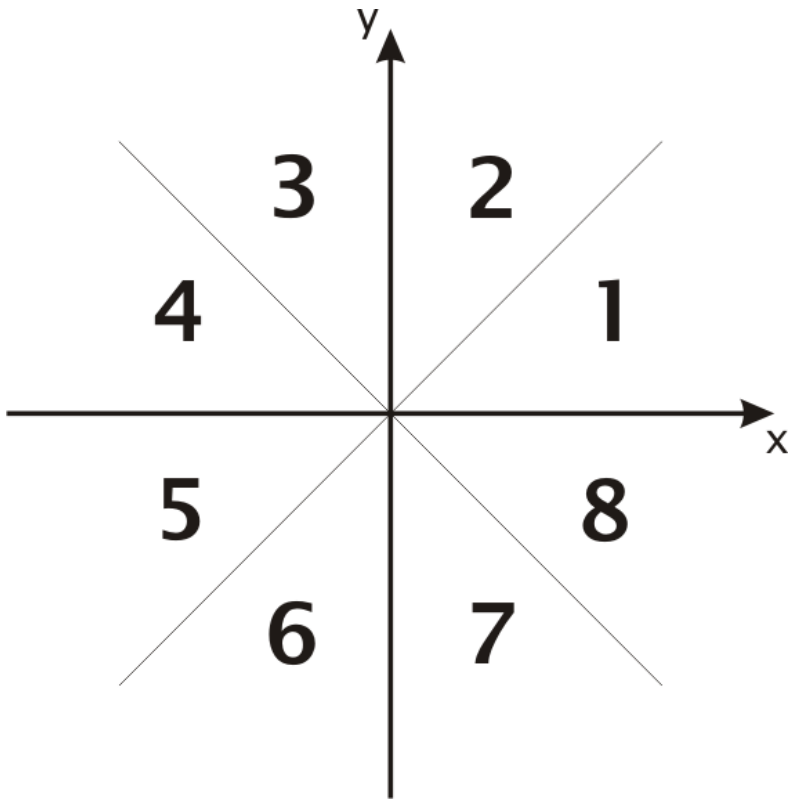
2 Rasterkonvertierung – Probleme, Effekte und Lösungsvorschläge für Linien und Kurven

- Problem: Darstellung stetiger Objekte in diskretem Raster
- Ziel: genaue und effiziente Darstellung

- Effekte:
  - ungleiche Intensität
  - Aliasing
  - Overstrike
  - Treppchenbildung



§ Oktanden des kartesischen Koordinatensystems:



- intuitive Lösung zur Rasterung von Linien:
  - zu jedem x-Wert den y-Wert mittels Geradengleichung $y = m \times x + n$ berechnen und runden
  - Problem: Gleitkomma-Multiplikation notwendig
- Überführung in inkrementelle Lösung:
  - Beobachtung: x-Wert ändert sich in jedem Schritt um 1 (treibende Achse), y-Wert um m
  - Vermeidung der Rundungsoperation durch Einführung einer Variablen error, die die Abweichung vom idealen Objekt bereithält
  - Resultat: eine Gleitkomma-Addition (y + m) und ein Gleitkomma-Vergleich (error)
  - nur für Linien im Oktand 1 geeignet, da x-Wert immer um 1 erhöht und y minimal um 0 und maximal um 1 erhöht wird $\rightarrow$ maximal 45° Steigung

inkrementeller Algorithmus zur Rasterung einer Linie im ersten Oktand

```
void line (int x1, int y1, int xn, int yn) {

  float m, error;
  int x, y;

  m = ((float) (yn - y1)) / (xn - x1);
  y = y1;
  error = 0.0;

  for (x = x1; x <= xn; x++) {

    setPixel(x, y);
    error += m;

    if (error >= 0.5) {

      y++; error--;

    }
  }
}
```

- **Bresenham-Algorithmus zur Rasterung von Linien:**
  - **Skalierung von m und error, so dass m und error immer ganzzahlig sind**
  - **leicht hardwareseitig implementierbar, da keine Gleitkomma-Operationen mehr notwendig**
  - **in erster Form ebenfalls nur für Linien im Oktand 1 geeignet**

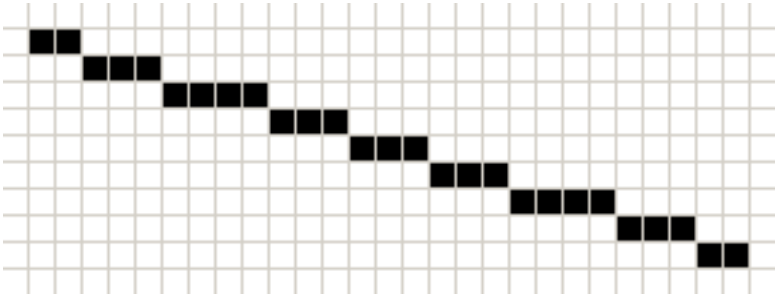


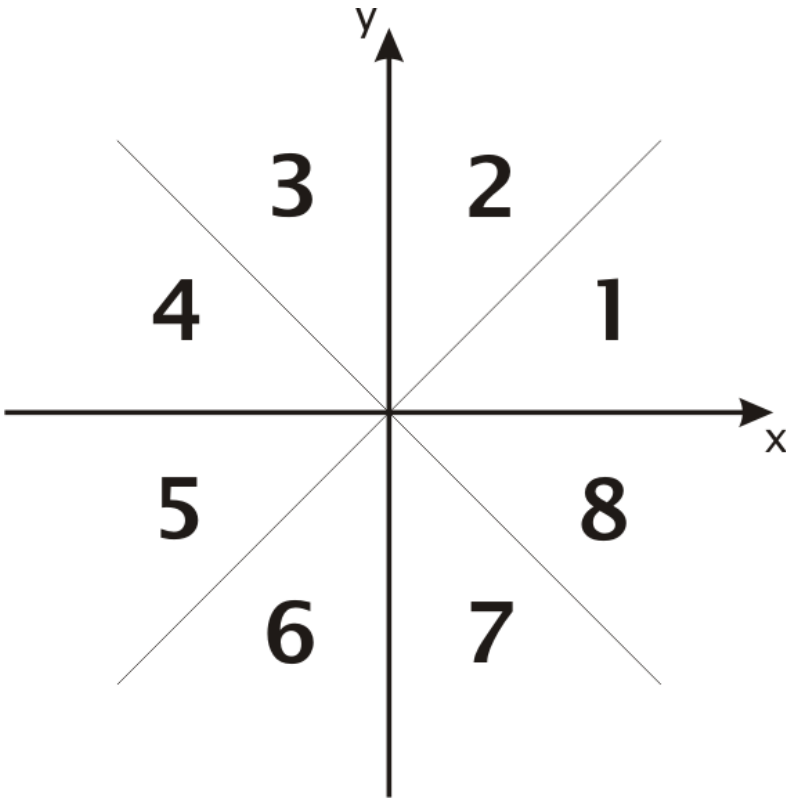
Abbildung 2: Beispiel einer mittels Bresenham-Algorithmus gerasterten Linie

- **algorithmische Realisierung mit Entscheidungsvariable d und rekursivem Formelsystem:**
  - **gegeben: Punkte (x_1,y_1) und (x_2,y_2)**
  - **$\Delta x = x_2 - x_1$**
  - **$\Delta y = y_2 - y_1$**
  - **$d_0 = 2 \times \Delta y - \Delta x$**
  - **$d_1 = 2 \times \Delta y$**
  - **$d_2 = 2 \times (\Delta y - \Delta x)$**
  - **$d_{i+1} = d_i + d_2$ für $d_i \geq 0$**
  - **$d_{i+1} = d_i + d_1$ für $d_i < 0$**
- **wenn $d_i \geq 0$ wird y_i inkrementiert, bleibt ansonsten gleich**

Bresenham – Algorithmus zur Rasterung einer Linie im ersten Oktand

```
void bresenham (int x1, int y1, int xn, int yn) {

  int error, x, y, dx, dy;

  dx = xn - x1;
  dy = yn - y1;
  error = -dx/2;
  y = y1;

  for (x = x1; x <= xn; x++) {

    setPixel(x, y);
    error += dy;
    if (error >= 0) {

      y++;
      error -= dx;

    }
  }
}
```

§ **für Linien in anderen Oktanden gilt:**



Oktand	treibende Achse	andere Achse
1	x	inkrementieren
2	y	inkrementieren
3	y	dekrementieren
4	x	dekrementieren
5	x	inkrementieren
6	y	inkrementieren
7	y	dekrementieren
8	x	dekrementieren

Abbildung 3: die 8 Oktanden des kartesischen Koordinatensystems (links) und deren Berechnungsvorschriften für den Bresenham-Algorithmus (rechts)

- **Bresenham-Algorithmus zur Rasterung von Kreisen:**
  - **Kreis wird nur im Oktand 2 berechnet**
  - **alle anderen Punkte werden durch Spiegelung hinzugefügt**
  - **Formelsystem:**
    - **gegeben: Mittelpunkt (x_1,y_1) und Radius r**
    - **$d_1 = 1 - r$**
    - **$d_{i+1} = d_i + 2 \times x_i - 2 \times y_i + 5$ für $d_i \geq 0$**
    - **$d_{i+1} = d_i + 2 \times x_i + 3$ für $d_i < 0$**
  - **wenn $d_i \geq 0$ wird y_i dekrementiert, bleibt ansonsten gleich**
  - **x startet bei 0, y startet bei r und terminiert, wenn $x > y$**

Bresenham – Algorithmus zur Rasterung eines kompletten Kreises

```
void kreis_bresenham(int x1, int y1, int radius) {


    int f = 1 - radius;
    int x = 0;
    int y = radius;
    int dx = 0;
    int dy = -2 * radius;


    setPixel(x1, y1 + radius);
    setPixel(x1, y1 - radius);
    setPixel(x1 + radius, y1);
    setPixel(x1 - radius, y1);
```

Fortsetzung à

```
    while(x < y) {

        if(f >= 0) { y--;  dy += 2; f += dy; }

        x++;
        dx += 2;
        f += dx + 1;


        setPixel(x1 + x, y1 + y);
        setPixel(x1 - x, y1 + y);
        setPixel(x1 + x, y1 - y);
        setPixel(x1 - x, y1 - y);
        setPixel(x1 + y, y1 + x);
        setPixel(x1 - y, y1 + x);
        setPixel(x1 + y, y1 - x);
        setPixel(x1 - y, y1 - x);

    }
}
```

3 Füllen von Polygonen und geschlossenen Kurven

- zu füllender Bereich durch geometrische Beschreibung oder Grenzen von Bildschirmobjekten gegeben

- Scangeraden-Methode:
  - Scangerade wird von unten nach oben über Objekt gezogen und Schnittepunkte werden berechnet
  - gerade Anzahl Schnittpunkte S für konkave Polygone $\rightarrow$ allgemein müssen Segmente zwischen S_{2i-1} und S_{2i} $(1 \leq i \leq n)$ eingefärbt werden ($\rightarrow$ zum Bsp. mittels Bresenham-Algorithmus für Linien)
  - Vorsicht beim Füllen von Polygonen die über den Bildschirm hinausragen

    $\rightarrow$ Bildschirmkanten müssen als Polygonkanten kenntlich gemacht werden

- **Saatfüllen (floodfill):**
  - **Darstellung des Objektes durch bestimmte Grenzpixel gegeben (zum Beispiel andersfarbig)**
  - **Idee: ausgehend von einem Pixel werden rekursiv alle angrenzenden Pixel gefüllt, wenn sie nicht zum Grenzbereich gehören**
  - **Achtung: keine diagonalen Aufrufe**
  - **sehr hoher Aufwand für Rekursionskeller $\rightarrow$ Pufferüberläufe möglich und sehr wahrscheinlich**
  - **Abhilfe: Pixelläufe**
    - **Gruppe horizontal benachbarter Pixel, die an Hand ihres rechtesten Pixels unterschieden werden**
    - **nur Untersuchung der nach oben und unten benachbarten Pixel und Speicherung aller Pixelläufe**
    - **Füllen der Pixelläufe durch iterative Abarbeitung**

Saatfüllen ohne Pixelläufe

```
void seedfill(int x, int y, int bcol, int fillcol) {

  if ((pixel_value(x,y) != bcol) && (pixel_value(x,y) != fillcol)) {

    setPixel(x, y, fillcol);

    seedfill(x+1, y, bcol, fillcol);
    seedfill(x-1, y, bcol, fillcol);
    seedfill(x, y+1, bcol, fillcol);
    seedfill(x, y-1, bcol, fillcol);

  }
}
```

4 Clippen von Linien und Polygonen an Polygonen

- häufige und grundlegende Operation der Computergraphik
- Möglichkeiten:
  - analytische Berechnung und anschließendes Clippen
  - Clippen während der Bildschirmdarstellung
- Clippen von Linien bezüglich einem rechteckigem Clip-Bereich:
  - verwendet um Linien zu clippen, die über Grenzen eines Bildschirmbereichs hinausragen
  - liegen beide Endpunkte der Linie innerhalb des Clip-Rechtecks liegt die ganze Linie innerhalb
  - liegt ein Endpunkt außerhalb des Clip-Rechtecks muss genauer analytisch untersucht werden

    $\rightarrow$ Schnittpunktberechnung von Linie und Clip-Rechteck $\rightarrow$ hoher Aufwand für sehr viele Linien

- **Algorithmus von Cohen & Sutherland:**
  - **Versuch der Vermeidung unnötiger Schnittpunktberechnungen**
  - **Aufteilung des Bereichs um das Clip-Rechteck in 9 Teile und Zuordnung eines 4-Bit-Wertes:**

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Zahlen von horizontal oder vertikal benachbarten Bereichen unterscheiden sich in einer Bitstelle
- Test auf komplett außerhalb: bitweise ODER-Verknüpfung zweier Bereichszahlen ist 0
- Test auf komplett innerhalb: bitweise UND-Verknüpfung ungleich 0 und eine Stelle mit gleicher 1
- wenn beide Tests nicht eindeutig sind, wird das Geradensegment an einer Fenstergeraden in zwei Teile geteilt und die Tests werden erneut durchgeführt
- wenn alle Geradensegmente auf einen der zwei Fälle zurückgeführt wurden, werden alle Segmente außerhalb des Clip-Rechtecks gelöscht und der Algorithmus terminiert

Cohen & Sutherland – Algorithmus zum Clippen von Linien an einem Rechteck

```
void clip (float x1, float y1, float x2, float y2, float xmin, float xmax,
float ymin, float ymax) {

  int c1, c2;
  float xs, ys;

  c1 = code(x1,y1);
  c2 = code(x2,y2);

  if (c1 | c2 == 0x0) line(x1, y1, x2, y2);

  else if (c1 & c2 != 0x0) return;

  else {

    intersect(xs, ys, x1, y1, x2, y2, xmin, xmax, ymin, ymax);

    if ( is_outside(x1,y1)) clip(xs, ys, x2, y2, xmin, xmax, ymin, ymax);
    else clip(x1, y1, xs, ys, xmin, xmax, ymin, ymax);

  }
}
```

- Clippen von Polygonen bezüglich rechteckigem Clip-Bereich - Algorithmus von Sutherland & Hodgman:
  - Divide & Conquer – Prinzip (Lösen eines komplexen Problems durch Lösen vieler einfacher)
  - einfaches Problem hier: Polygon an einzelner, unendlichen Geraden clippen
  - aufwendiger als Cohen & Sutherland, kann jedoch jedes konkave oder konvexe Polygon an jedem beliebigen Polygon clippen (auch im dreidimensionalen Raum)
  - Algorithmus kappt an unendlicher Geraden und gibt Koordinaten von Knoten zurück, die das geclippte Polygon beschreiben
  - rekursiver Aufruf für jeden neuen Knoten $\rightarrow$ Polygon durchläuft Pipeline von Kappungsfunktionen ohne Zwischenspeicherung $\rightarrow$ komplette Knotenliste des geclippten Polygons am Ende
  - kann ohne Pufferspeicher in Hardware implementiert werden

5 Quellen

§ „Algorithmen in der Computergraphik“, Dr. rer. nat. Thomas Rauber (1993)

§ „Grundlagen der Computergraphik. Einführung, Konzepte, Methoden.“, James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips (1994)

§ Vorlesungsskript Computergrafik, Prof. Dr.-Ing. F. Jaeger, HTWK Leipzig (2005)