Rasterization

Algorithmen und Anwendung

Tissen, Lars Semercio, Mustafa

Postscript Seminar
Theoretical Computer Science

Inhaltsverzeichnis

Inhaltsverzeichnis

Rasterization Definition

Rasterization

auch: Rendern, Scankonvertierung

Umrechnen einer Vektorgrafik zu einer Rastergrafik

[1]

[2]

Rasterization Definition

Rasterization

auch: Rendern, Scankonvertierung
Umrechnen einer Vektorgrafik zu einer Rastergrafik

- Input: kontinuierliche Vektorgrafik
- Output: diskrete Bitmap
- Berechnung durch Interpreter: Raster Image Processor (RIP)

[1]
[2]

Rasterization

User Space

Rasterization

User Space

- virtuelle Oberfläche, die von Benutzer modifiziert wird
- unabhängig von Gerätespezifikation
- (theoretisch) unendlich

Rasterization

User Space

- virtuelle Oberfläche, die von Benutzer modifiziert wird
- unabhängig von Gerätespezifikation
- (theoretisch) unendlich

Device Space

Rasterization

User Space

- virtuelle Oberfläche, die von Benutzer modifiziert wird
- unabhängig von Gerätespezifikation
- (theoretisch) unendlich

Device Space

- geräteabhängige Auflösung
- Bitmatrix mit (x,y) Koordinaten
- diskrete, endliche Auflösung

Inhaltsverzeichnis

PostScript Interpreter Poppler

- PDF rendering library
- Freie Software von freedesktop.org
- Genutzt von Evince, Okular, Inkscape, Xournal

[3]

PostScript Interpreter Poppler

- PDF rendering library
- Freie Software von freedesktop.org
- Genutzt von Evince, Okular, Inkscape, Xournal

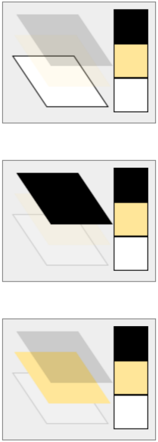
Grafikbibliothek Cairo

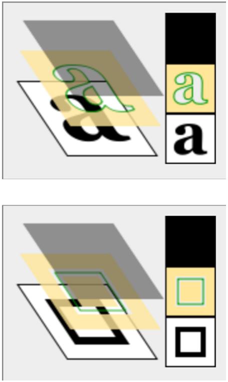
- geräteunabhängig
- Ausgabe auf Backends

[3]

Cairo - Drawing Model

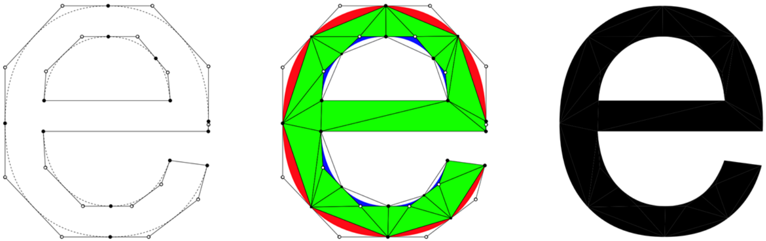
- Destination
- Source
- Mask





[4]

Zwischenergebnis



- Haben: Kontinuierliche Beschreibung (links)
- Wollen: Gerasterte Ausgabe (rechts)

[5]

Inhaltsverzeichnis

Primitiven als Ausgangsposition

Beispiele für Primitiven

Primitiven als Ausgangsposition

Beispiele für Primitiven

- Linien

Primitiven als Ausgangsposition

Beispiele für Primitiven

- Linien
- Kreise

Primitiven als Ausgangsposition

Beispiele für Primitiven

- Linien
- Kreise
- Kurven

Primitiven als Ausgangsposition

Beispiele für Primitiven

- Linien
- Kreise
- Kurven
- Dreiecke

Primitiven als Ausgangsposition

Beispiele für Primitiven

- Linien
- Kreise
- Kurven
- Dreiecke
- Kombination von Algorithmen für kompliziertere geometrische Objekte

Primitiven als Ausgangsposition

Eine Linie in PostScript

Primitiven als Ausgangsposition

Eine Linie in PostScript

```
newpath
15 25 moveto
70 90 lineto
stroke
```

Primitiven als Ausgangsposition

Eine Kurve in PostScript

Primitiven als Ausgangsposition

Eine Kurve in PostScript

```
newpath
144 144 moveto
288 144 288 288 144 288 curveto
stroke
```

Algorithmen für Linien: DDA-Methode

Annahmen des Algorithmus:

[6]

Algorithmen für Linien: DDA-Methode

Annahmen des Algorithmus:

- Zwei Punkte $(x_0, y_0), (x_1, y_1)$

[6]

Algorithmen für Linien: DDA-Methode

Annahmen des Algorithmus:

- Zwei Punkte $(x_0, y_0), (x_1, y_1)$
- $x_0 < x_1$

[6]

Algorithmen für Linien: DDA-Methode

Annahmen des Algorithmus:

- Zwei Punkte $(x_0, y_0), (x_1, y_1)$
- $x_0 < x_1$
- $0 \leq m \leq 1$

[6]

Algorithmen für Linien: DDA-Methode

Berechnung

Algorithmen für Linien: DDA-Methode

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0}$

Algorithmen für Linien: DDA-Methode

Berechnung

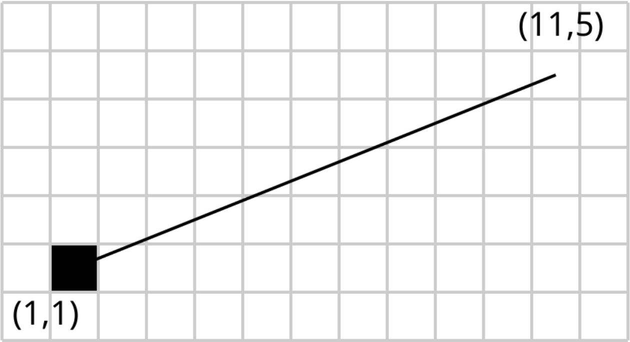
- $m = \frac{y_1 - y_0}{x_1 - x_0}$
- Schritt: $x_{current} \rightarrow x_{current} + 1$ und $y_{current} \rightarrow y_{current} + m$

Algorithmen für Linien: DDA-Methode

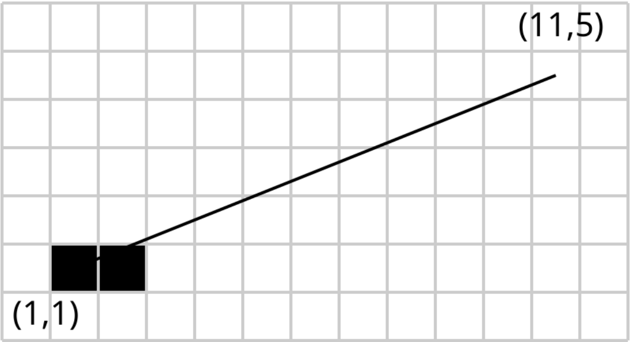
Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0}$
- Schritt: $x_{current} \to x_{current} + 1$ und $y_{current} \to y_{current} + m$
- Pixel: $(\lfloor x_{current} \rfloor, \lfloor y_{current} \rfloor)$

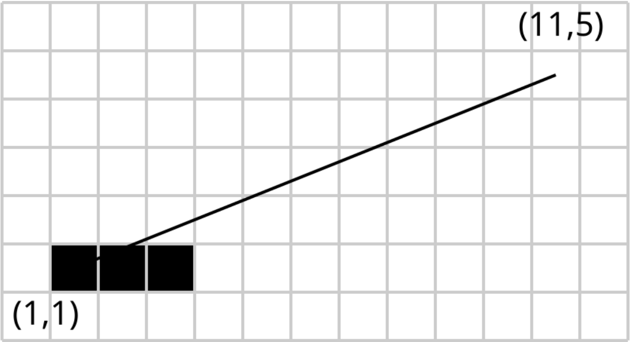
Algorithmen für Linien



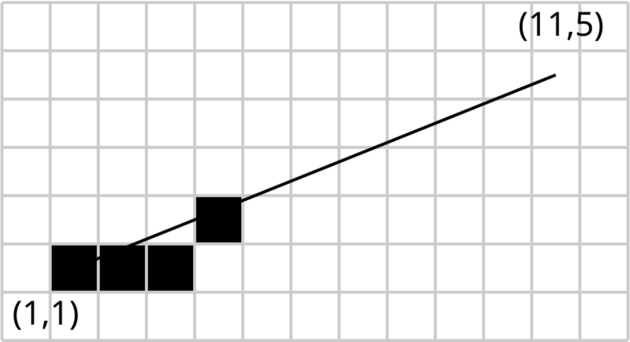
Algorithmen für Linien



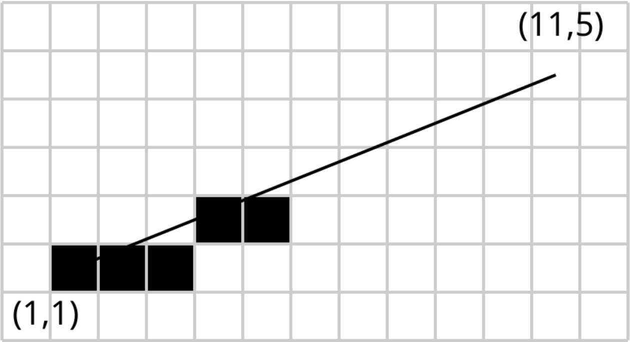
Algorithmen für Linien



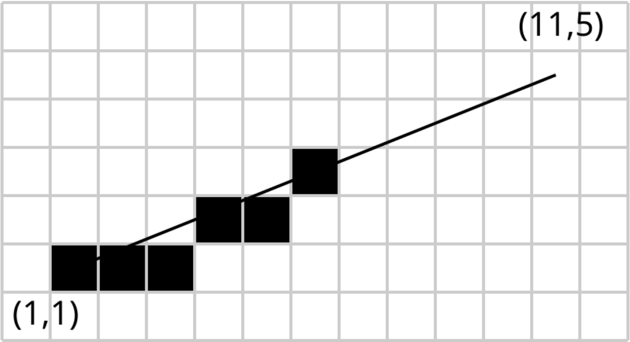
Algorithmen für Linien



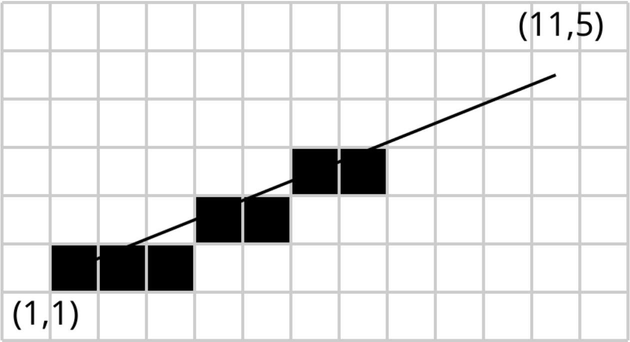
Algorithmen für Linien



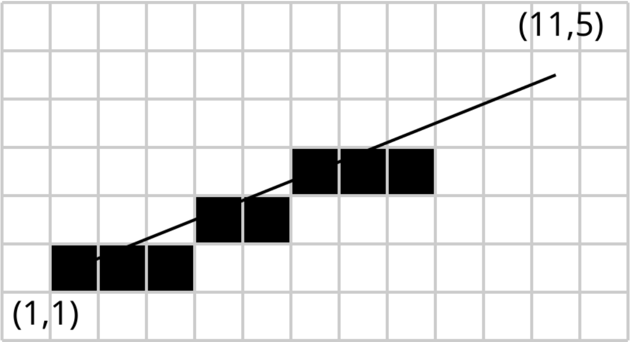
Algorithmen für Linien



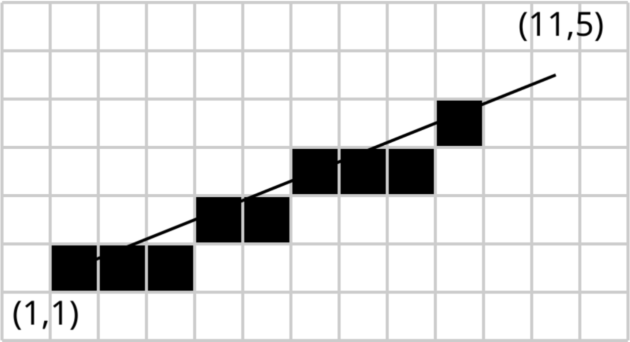
Algorithmen für Linien



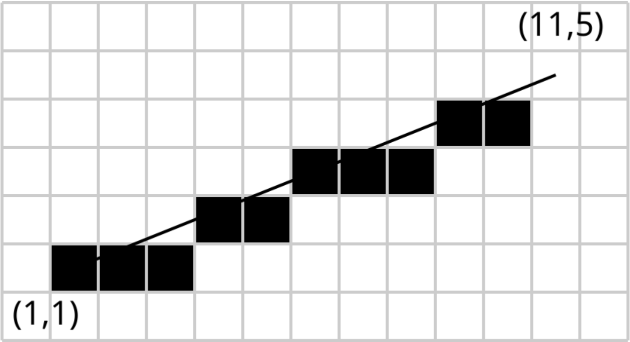
Algorithmen für Linien



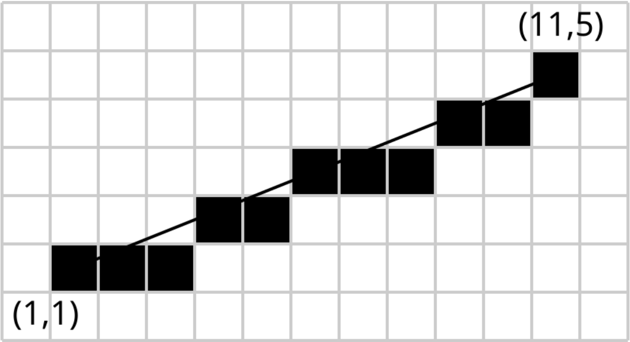
Algorithmen für Linien



Algorithmen für Linien



Algorithmen für Linien



Algorithmen für Linien: Bresenham-Algorithmus

Annahmen des Algorithmus

[7]

Algorithmen für Linien: Bresenham-Algorithmus

Annahmen des Algorithmus

- $(x_0, y_0), (x_1, y_1) \in \mathbb{Z} \times \mathbb{Z}$

[7]

Algorithmen für Linien: Bresenham-Algorithmus

Annahmen des Algorithmus

- $(x_0, y_0), (x_1, y_1) \in \mathbb{Z} \times \mathbb{Z}$

[7]

Algorithmen für Linien: Bresenham-Algorithmus

Annahmen des Algorithmus

- $(x_0, y_0), (x_1, y_1) \in \mathbb{Z} \times \mathbb{Z}$
- Rest wie vorher

[7]

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}$

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}$
- $b = y_0$ y-Achsenabschnitt

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}$
- $b = y_0$ y-Achsenabschnitt
- $y = m \cdot x + b$

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}$
- $b = y_0$ y-Achsenabschnitt
- $y = m \cdot x + b$
- $f(x, y) = m \cdot x - y + y_0 = \frac{\Delta y}{\Delta x} \cdot x - y + y_0$

Algorithmen für Linien: Bresenham-Algorithmus

Berechnung

- $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y}{\Delta x}$
- $b = y_0$ y-Achsenabschnitt
- $y = m \cdot x + b$
- $f(x, y) = m \cdot x - y + y_0 = \frac{\Delta y}{\Delta x} \cdot x - y + y_0$
- $f'(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
  - $f(x, y) \in \mathbb{Z}$, wenn $x, y \in \mathbb{Z}$

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
- Letztes Pixel: $(x_{current}, y_{current})$

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
- Letztes Pixel: $(x_{current}, y_{current})$
- Nächstes Pixel:

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
- Letztes Pixel: $(x_{current}, y_{current})$
- Nächstes Pixel:
- i. $(x_{current} + 1, y_{current})$ oder

Algorithmen für Linien: Bresenham-Algorithmus

Wahl des Pixels

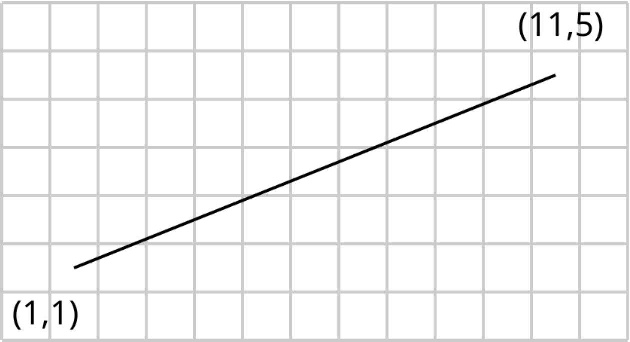
- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
- Letztes Pixel: $(x_{current}, y_{current})$
- Nächstes Pixel:
- i. $(x_{current} + 1, y_{current})$ oder
- ii. $(x_{current} + 1, y_{current} + 1)$

Algorithmen für Linien: Bresenham-Algorithmus

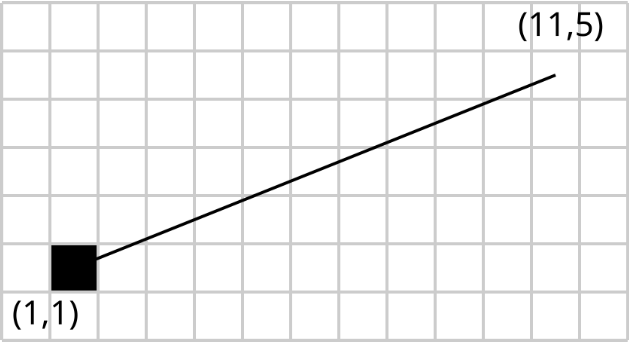
Wahl des Pixels

- $f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_0$
- Letztes Pixel: $(x_{current}, y_{current})$
- Nächstes Pixel:
- i. $(x_{current} + 1, y_{current})$ oder
- ii. $(x_{current} + 1, y_{current} + 1)$
- Berechnung: $f(x_{current} + 1, y_{current} + \frac{1}{2}) > 0$ ii., sonst i.

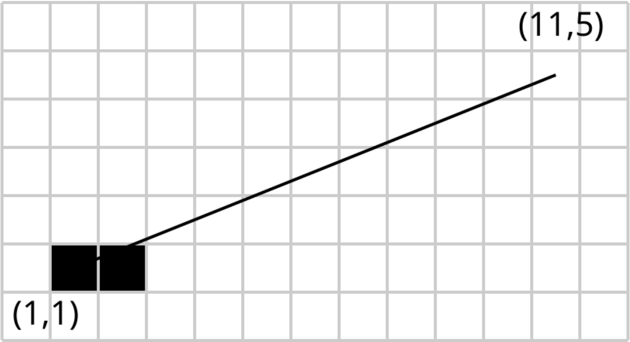
Algorithmen für Linien



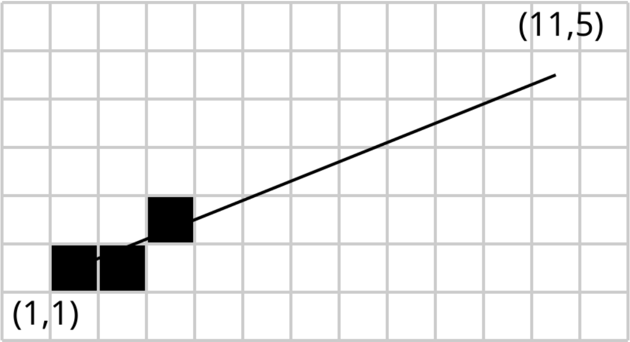
Algorithmen für Linien



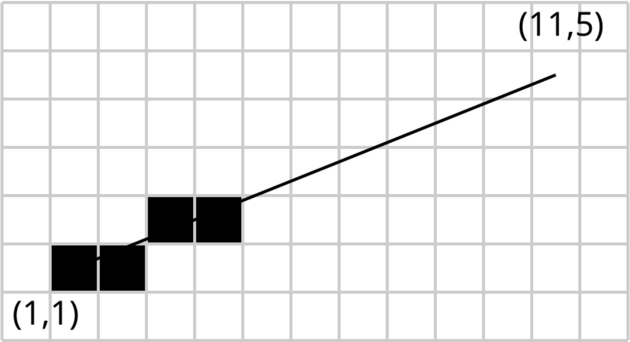
Algorithmen für Linien



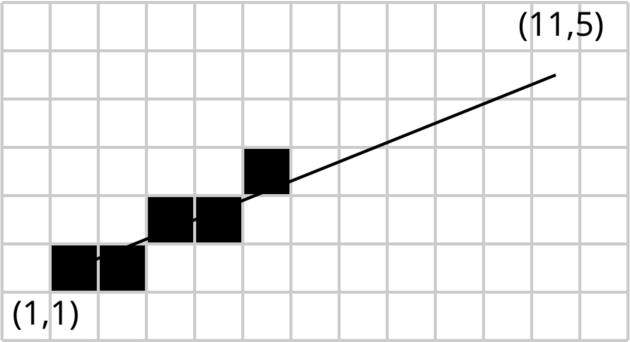
Algorithmen für Linien



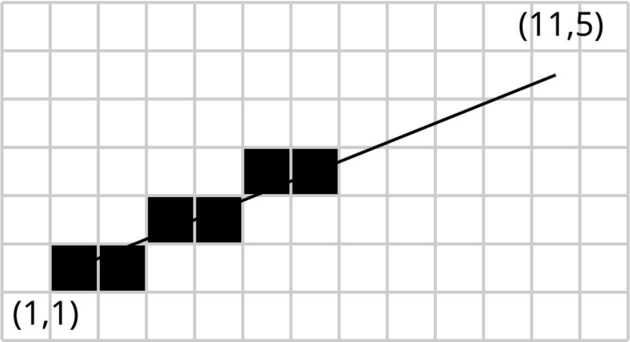
Algorithmen für Linien



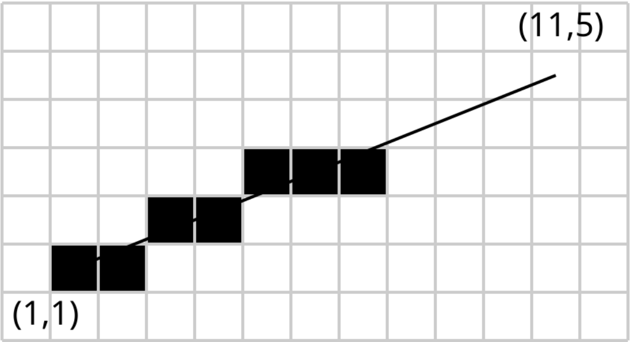
Algorithmen für Linien



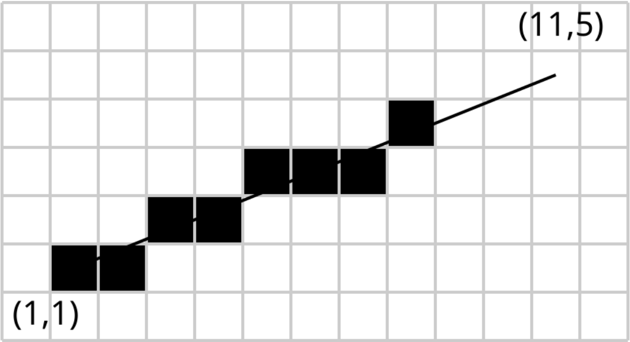
Algorithmen für Linien



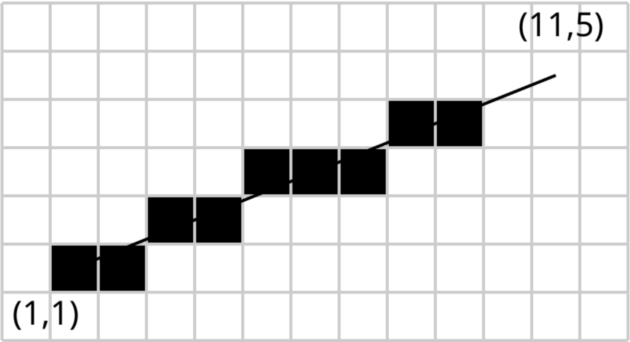
Algorithmen für Linien



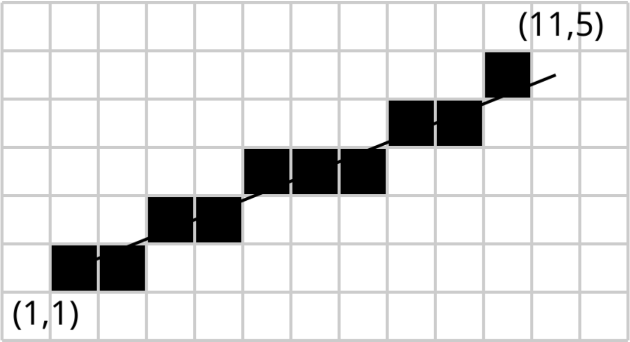
Algorithmen für Linien



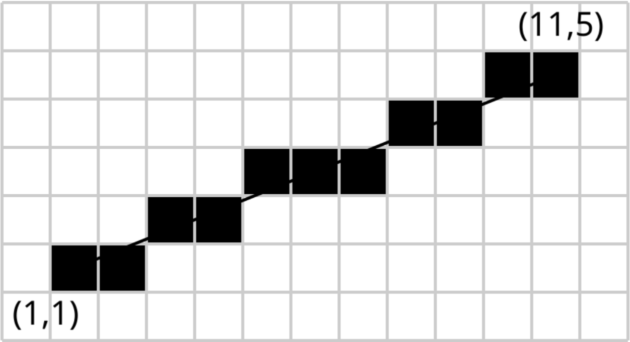
Algorithmen für Linien



Algorithmen für Linien



Algorithmen für Linien



Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus

Unterschied zum Algorithmus von Bresenham

[8]

Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus

Unterschied zum Algorithmus von Bresenham

- Vernachlässigt nicht das genaue Ergebnis von $f(x, y)$

[8]

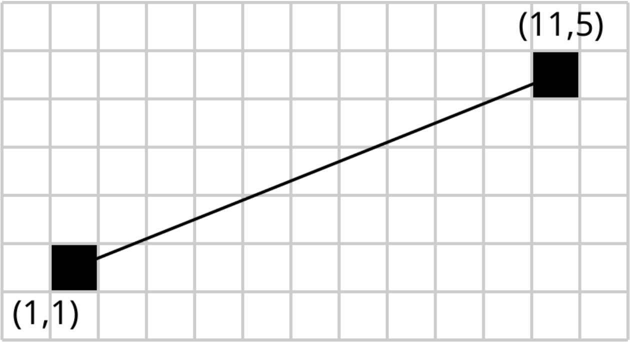
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus

Unterschied zum Algorithmus von Bresenham

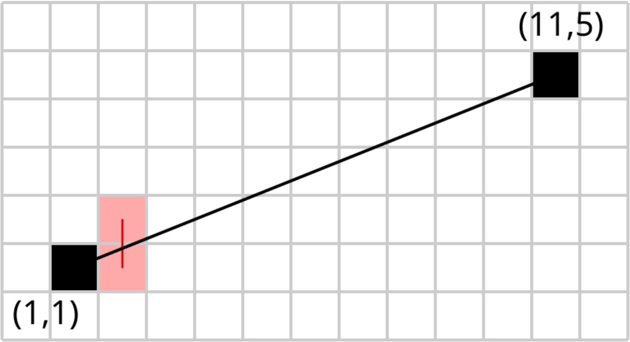
- Vernachlässigt nicht das genaue Ergebnis von $f(x, y)$
- Nutzt den resultierenden Error, um Transparenz beider Pixel festzulegen

[8]

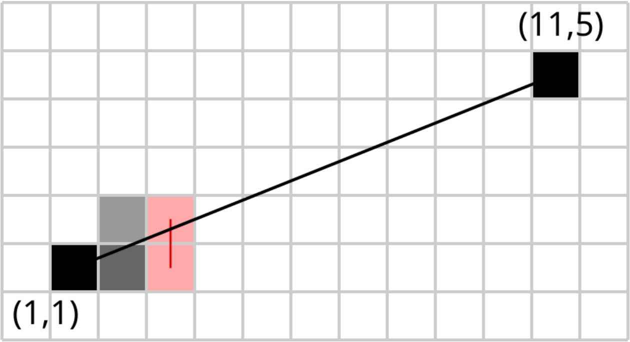
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



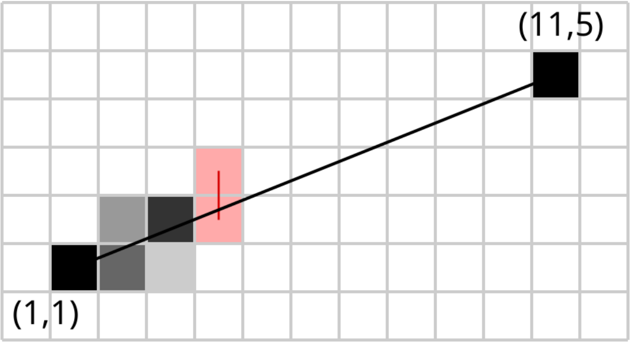
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



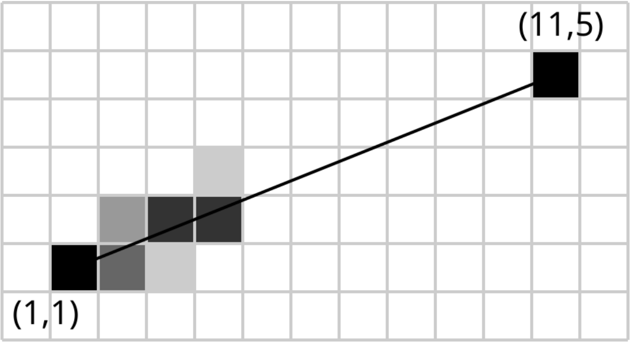
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



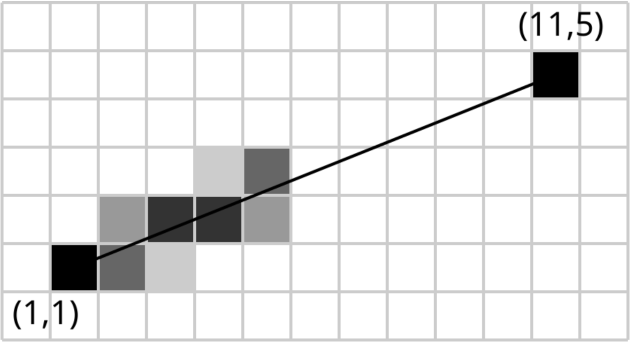
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



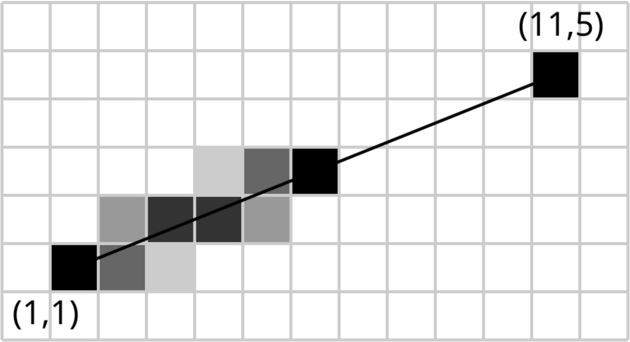
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



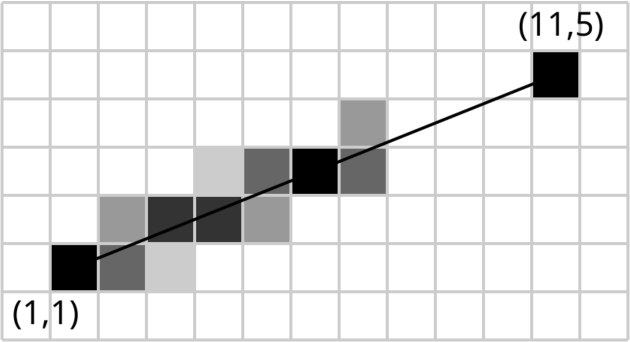
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



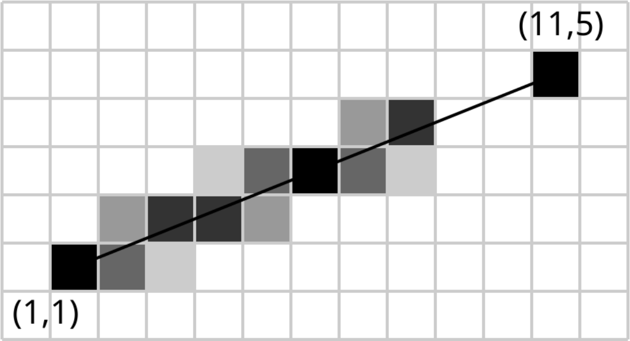
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



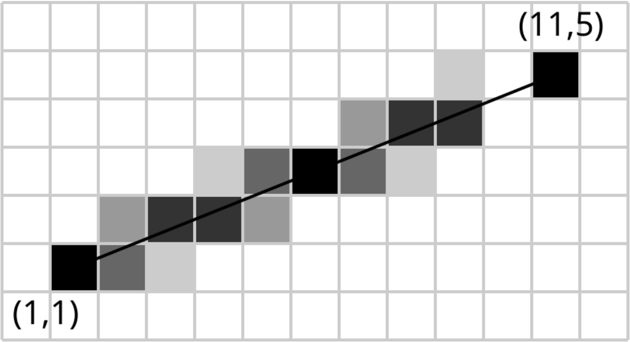
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



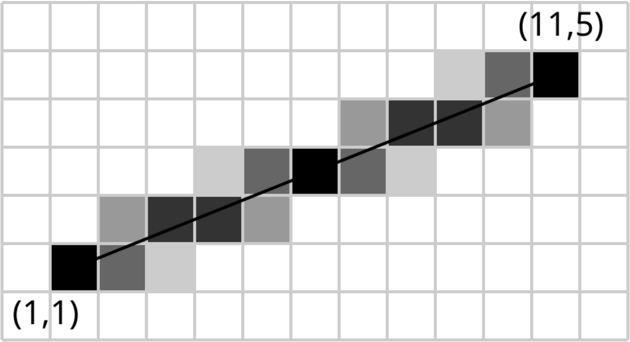
Algorithmen für Linien: Xiaolin Wu’s Linien-Algorithmus



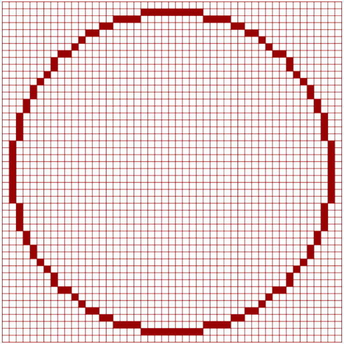
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



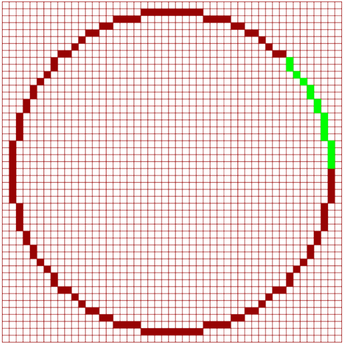
Algorithmen für Linien: Xiaolin Wu's Linien-Algorithmus



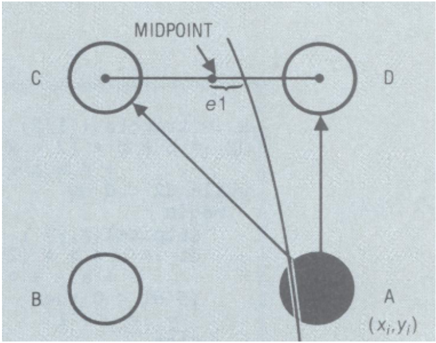
Algorithmus für Kreise: Midpoint-Algorithmus



Algorithmus für Kreise: Midpoint-Algorithmus

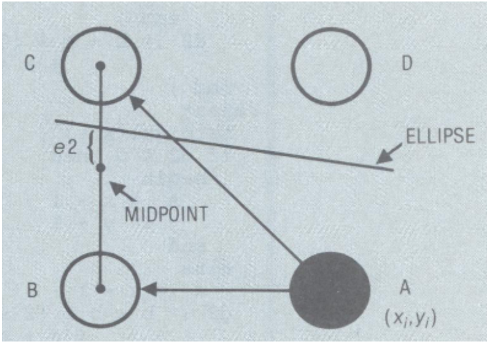


Algorithmus für Kreise: Midpoint-Algorithmus



[9]

Algorithmus für Kreise: Midpoint-Algorithmus



[9]

Algorithmus für allgemeine Kurven

Ausgangssituation

Schritte

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben

Schritte

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend

Schritte

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend
- Error eines Punktes (x, y) gegeben durch $e = f(x, y)$

Schritte

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend
- Error eines Punktes (x, y) gegeben durch $e = f(x, y)$

Schritte

- Momentanes Pixel (x, y)

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend
- Error eines Punktes (x, y) gegeben durch $e = f(x, y)$

Schritte

- Momentanes Pixel (x, y)
- Mögliche nächste Pixel $(x + 1, y), (x, y + 1), (x + 1, y + 1)$

Algorithmus für allgemeine Kurven

Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend
- Error eines Punktes (x, y) gegeben durch $e = f(x, y)$

Schritte

- Momentanes Pixel (x, y)
- Mögliche nächste Pixel $(x+1, y), (x, y+1), (x+1, y+1)$
- Vergleiche Errors von $(x, y+1), (x+1, y+1)$ für x-Schritt

Algorithmus für allgemeine Kurven

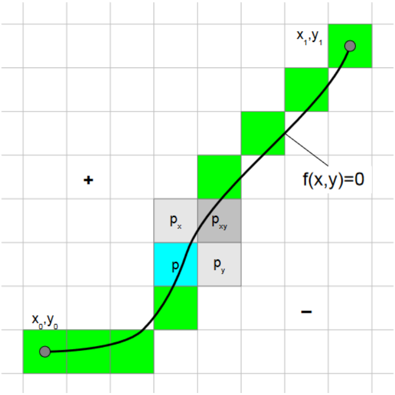
Ausgangssituation

- Implizite Funktion $f(x, y)$ gegeben
- Kurve monoton steigend
- Error eines Punktes (x, y) gegeben durch $e = f(x, y)$

Schritte

- Momentanes Pixel (x, y)
- Mögliche nächste Pixel $(x + 1, y), (x, y + 1), (x + 1, y + 1)$
- Vergleiche Errors von $(x, y + 1), (x + 1, y + 1)$ für x-Schritt
- Vergleiche Errors von $(x + 1, y), (x + 1, y + 1)$ für y-Schritt

Algorithmus für allgemeine Kurven



[10]

Kubische Bézier-Kurven

Recap: Bézier-Kurven

Kubische Bézier-Kurven

Recap: Bézier-Kurven

- Durch vier Punkte b_0, b_1, b_2, b_3 definiert

Kubische Bézier-Kurven

Recap: Bézier-Kurven

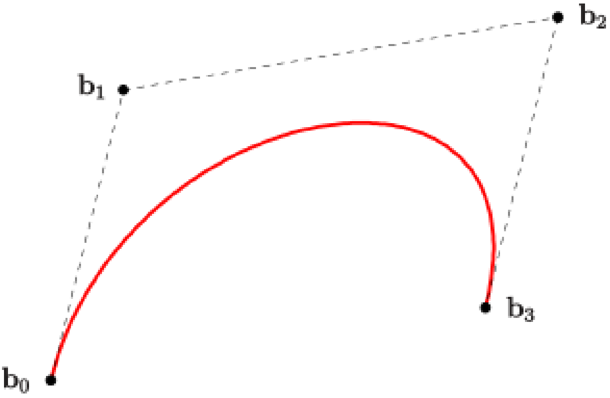
- Durch vier Punkte b_0, b_1, b_2, b_3 definiert
- Startpunkt b_0 und Endpunkt b_3

Kubische Bézier-Kurven

Recap: Bézier-Kurven

- Durch vier Punkte b_0, b_1, b_2, b_3 definiert
- Startpunkt b_0 und Endpunkt b_3
- Verschiedene Darstellungsformen und Eigenschaften, z.B. rekursive Definition

Beispiel für kubische Bézier-Kurve



[11]

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

Kurve durch Liniensegmente abschätzen

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert

Kurve durch Liniensegmente abschätzen

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert
- Wende Algorithmus für Kurven an (vorherige Folien)

Kurve durch Liniensegmente abschätzen

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert
- Wende Algorithmus für Kurven an (vorherige Folien)

Kurve durch Liniensegmente abschätzen

- Ermittle Punkte entlang der Kurve, die die Kurve approximieren

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert
- Wende Algorithmus für Kurven an (vorherige Folien)

Kurve durch Liniensegmente abschätzen

- Ermittle Punkte entlang der Kurve, die die Kurve approximieren
- z.B. Aufteilen der Kurve in mehrere Segmente

Ansätze für die Rasterization

Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert
- Wende Algorithmus für Kurven an (vorherige Folien)

Kurve durch Liniensegmente abschätzen

- Ermittle Punkte entlang der Kurve, die die Kurve approximieren
- z.B. Aufteilen der Kurve in mehrere Segmente
- Wiederhole bis Segmente ungefähr Linien entsprechen

Ansätze für die Rasterization

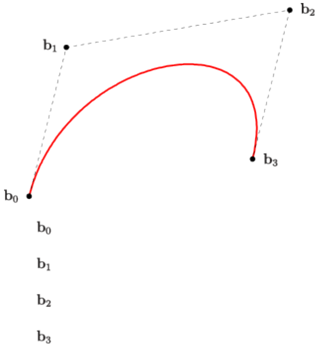
Kurve in monotone Abschnitte aufteilen

- Ermittle Punkte an denen sich Gradient vertikal/horizontal verändert
- Wende Algorithmus für Kurven an (vorherige Folien)

Kurve durch Liniensegmente abschätzen

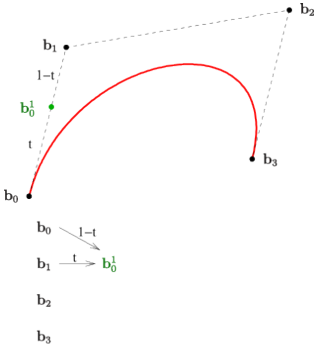
- Ermittle Punkte entlang der Kurve, die die Kurve approximieren
- z.B. Aufteilen der Kurve in mehrere Segmente
- Wiederhole bis Segmente ungefähr Linien entsprechen
- Wende Linienalgorithmus auf Segmente an

Aufteilung durch De-Casteljau-Algorithmus



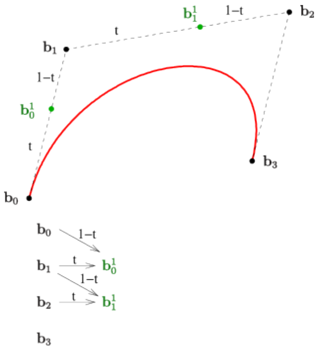
[12]
[11]

Aufteilung durch De-Casteljau-Algorithmus



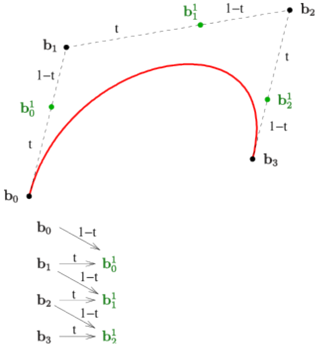
[12]
[11]

Aufteilung durch De-Casteljau-Algorithmus



[12]
[11]

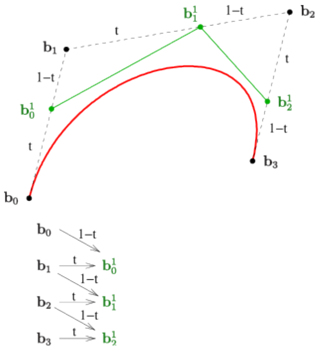
Aufteilung durch De-Casteljau-Algorithmus



[12]

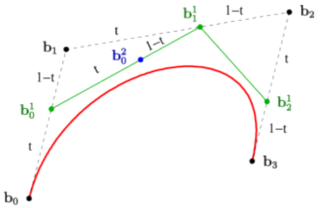
[11]

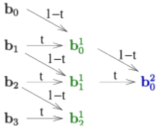
Aufteilung durch De-Casteljau-Algorithmus



[12]
[11]

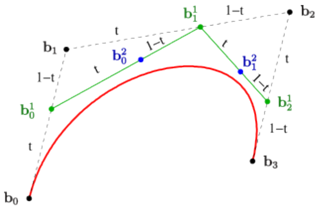
Aufteilung durch De-Casteljau-Algorithmus

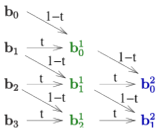




[12]
[11]

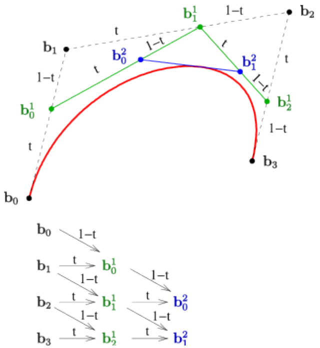
Aufteilung durch De-Casteljau-Algorithmus





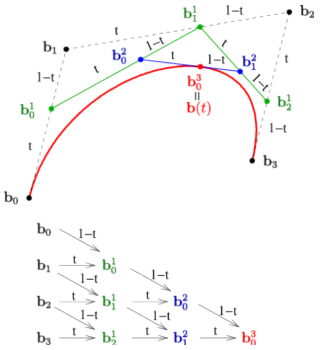
[12]
[11]

Aufteilung durch De-Casteljau-Algorithmus



[12]
[11]

Aufteilung durch De-Casteljau-Algorithmus



[12]
[11]

Inhaltsverzeichnis

Anti-Aliasing

Aliasing: Effekte

Anti-Aliasing

Aliasing: Effekte

- Allgemein: ungewollte Entstehung von Mustern

Anti-Aliasing

Aliasing: Effekte

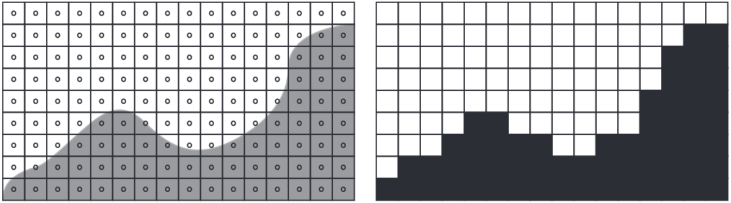
- Allgemein: ungewollte Entstehung von Mustern
- z.B. stufenartige Erscheinung

Anti-Aliasing

Aliasing: Effekte

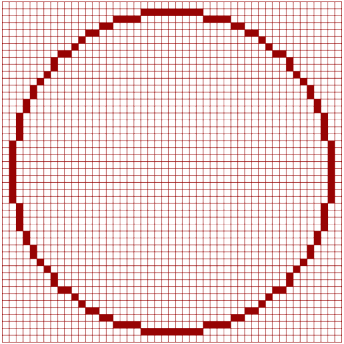
- Allgemein: ungewollte Entstehung von Mustern
- z.B. stufenartige Erscheinung
- "Moiré-Muster"

Anti-Aliasing (AA)

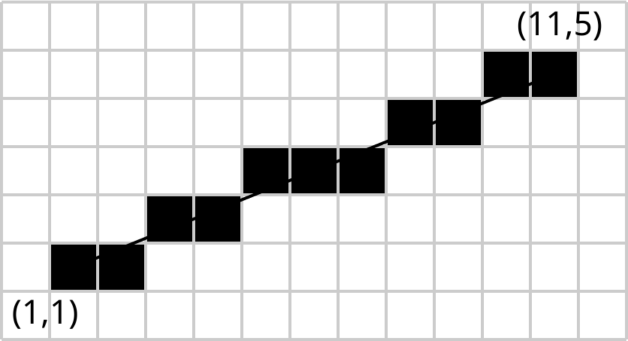


[13]

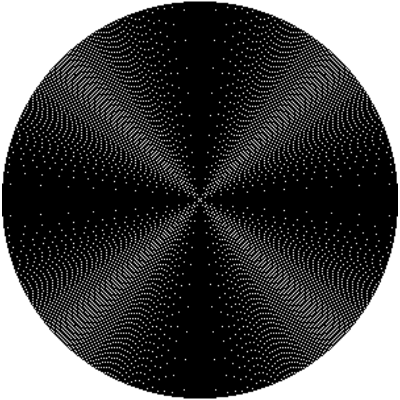
Anti-Aliasing (AA)



Anti-Aliasing (AA)



Anti-Aliasing (AA)



[14]

Supersampling-Methoden

Idee

Supersampling-Methoden

Idee

- Sampling, aber mit höherer Auflösung als Ziel

Supersampling-Methoden

Idee

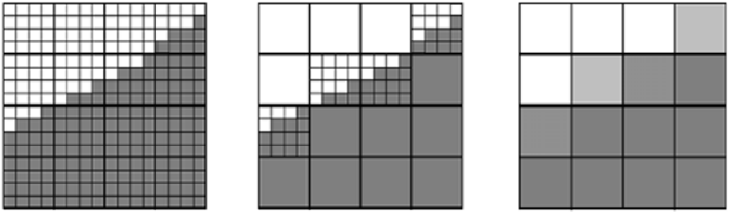
- Sampling, aber mit höherer Auflösung als Ziel
- z.B. 2x, 4x, 8x

Supersampling-Methoden

Idee

- Sampling, aber mit höherer Auflösung als Ziel
- z.B. 2x, 4x, 8x
- "Downsampling", um wieder zur ursprünglicher Auflösung zu kommen

Supersampling



[15]

AA Methoden

Alternativen zum SSAA und Tricks

AA Methoden

Alternativen zum SSAA und Tricks

- MSAA: Nur an Kanten, nicht unbedingt im regelmäßigen Muster [16]

AA Methoden

Alternativen zum SSAA und Tricks

- MSAA: Nur an Kanten, nicht unbedingt im regelmäßigen Muster [16]
- Samples an randomisierten Orten innerhalb des Pixels

AA Methoden

Alternativen zum SSAA und Tricks

- MSAA: Nur an Kanten, nicht unbedingt im regelmäßigen Muster [16]
- Samples an randomisierten Orten innerhalb des Pixels
- Samples an Pixelgrenzen

Inhaltsverzeichnis

Polygon Rasterung

Problem

Polygon Rasterung

Problem

- Polygone sind komplexer als einfache Primitive
- Können Löcher und Überschneidungen haben
- Entscheidung, welche Pixel innerhalb und welche außerhalb liegen

Polygon Rasterung

Problem

- Polygone sind komplexer als einfache Primitive
- Können Löcher und Überschneidungen haben
- Entscheidung, welche Pixel innerhalb und welche außerhalb liegen

Vorteile

Polygon Rasterung

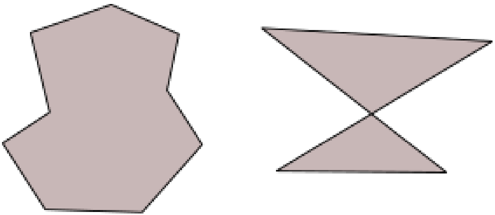
Problem

- Polygone sind komplexer als einfache Primitive
- Können Löcher und Überschneidungen haben
- Entscheidung, welche Pixel innerhalb und welche außerhalb liegen

Vorteile

- Haben Linien als Außenkanten
- RIP's sind auf Dreiecke spezialisiert

Einfache vs. überschlagende Polygone



einfach überschlagend

- Überschneidungen durch z.B. Sweep-Line-Algorithmus finden
  - Bspw. Bentley-Ottman [17]
- Komplexe Polygone in einfache Polygone aufteilen
  - Bspw. Subramaniam [18]

Rasterung von Polygonen

Einteilung in Dreiecke

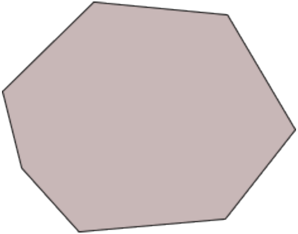
- RIP's sind auf Dreiecke spezialisiert
- Voraussetzung: Einfache Polygone
- Bei konvexen Polygonen trivial

[19]

Rasterung von Polygonen

Einteilung in Dreiecke

- RIP's sind auf Dreiecke spezialisiert
- Voraussetzung: Einfache Polygone
- Bei konvexen Polygonen trivial

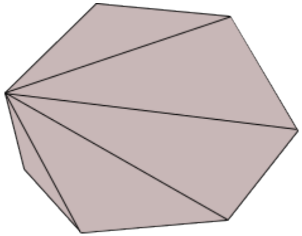


[19]

Rasterung von Polygonen

Einteilung in Dreiecke

- RIP's sind auf Dreiecke spezialisiert
- Voraussetzung: Einfache Polygone
- Bei konvexen Polygonen trivial



[19]

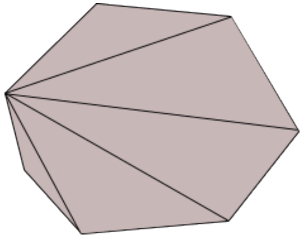
Rasterung von Polygonen

Einteilung in Dreiecke

- RIP's sind auf Dreiecke spezialisiert
- Voraussetzung: Einfache Polygone
- Bei konvexen Polygonen trivial

Nicht-konvexe Polygone

- Transformation in konvexe Polygone aufwendig
- Effizienter: y-monotone Polygone



[19]

y-monotone Polygonen

Monoton zu Linie l

- $\Leftrightarrow$ jede beliebige Linie l' orthogonal zu l das Polygon höchstens zwei mal schneidet

y-monotone Polygonen

Monoton zu Linie l

- $\Leftrightarrow$ jede beliebige Linie l' orthogonal zu l das Polygon höchstens zwei mal schneidet

y-monoton

- Jede Horizontale hat höchstens zwei Schnittpunkte mit dem Polygon
- Oder: Pfad vom obersten bis untersten Knoten führt nie bergauf

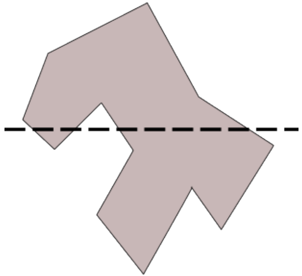
y-monotone Polygonen

Monoton zu Linie l

- $\Leftrightarrow$ jede beliebige Linie l' orthogonal zu l das Polygon höchstens zwei mal schneidet

y-monoton

- Jede Horizontale hat höchstens zwei Schnittpunkte mit dem Polygon
- Oder: Pfad vom obersten bis untersten Knoten führt nie bergauf



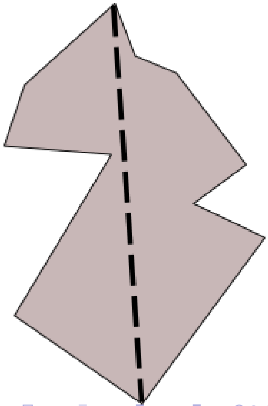
y-monotone Polygonen

Monoton zu Linie l

- $\Leftrightarrow$ jede beliebige Linie l' orthogonal zu l das Polygon höchstens zwei mal schneidet

y-monoton

- Jede Horizontale hat höchstens zwei Schnittpunkte mit dem Polygon
- Oder: Pfad vom obersten bis untersten Knoten führt nie bergauf



Warum y-monotone Polygone?

- Ear-clipping braucht Zeit $O(n^2)$ [20]
  - Funktioniert nur bei Polygonen ohne Löcher
- Transformation in y-monotone Polygone braucht Zeit O(n log n)
- Triangulierung von monotonen Polygonen in linearer Zeit

Warum y-monotone Polygone?

- Ear-clipping braucht Zeit $O(n^2)$ [20]
  - Funktioniert nur bei Polygonen ohne Löcher
- Transformation in y-monotone Polygone braucht Zeit O(n log n)
- Triangulierung von monotonen Polygonen in linearer Zeit

Vorgehen

- Nicht-monotone Teilstücke identifizieren
- Teilstücke abtrennen
- y-monotones Polygon triangulieren

Warum y-monotone Polygone?

Vorgehen

- Nicht-monotone Teilstücke identifizieren
- Teilstücke abtrennen
- y-monotones Polygon triangulieren

Theorem

Jedes einfache Polygon mit n $\geq$ 3 Knoten kann in n-2 Dreiecke zerteilt werden [21]

Transformation in y-monotone Polygonen

Seien v,u,w $\in V$ und (v,w), (v,u) $\in E$.

Definition v oberhalb von u

- $v_y > u_y$ oder
- $v_y == u_y$ und $v_x < u_x$

Transformation in y-monotone Polygonen

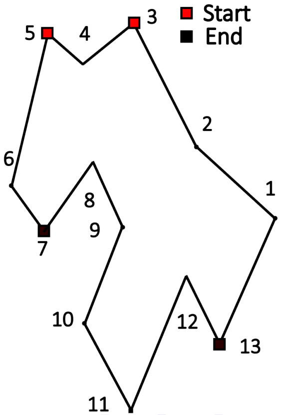
Seien v,u,w $\in V$ und (v,w), (v,u) $\in E$.

Definition v oberhalb von u

- $v_y > u_y$ oder
- $v_y == u_y$ und $v_x < u_x$

Klassifizierung von $v \in V$

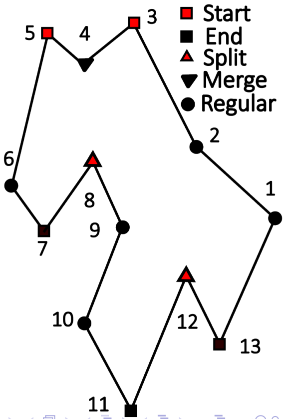
- Start: v oberhalb von u und w und Innenwinkel(v) $< \pi$
- End: v unterhalb von u und w Innenwinkel(v) $< \pi$



Transformation in y-monotone Polygonen

Klassifizierung von $v \in V$

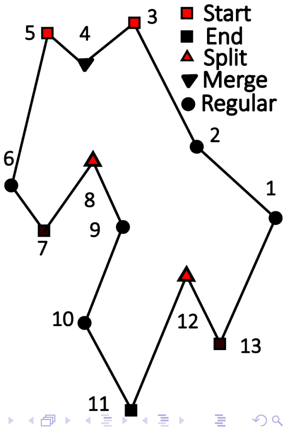
- Split: v oberhalb von u und w und Innenwinkel(v) $> \pi$



Transformation in y-monotone Polygonen

Klassifizierung von $v \in V$

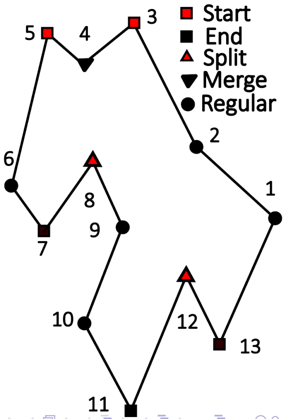
- Split: v oberhalb von u und w und Innenwinkel(v) $> \pi$
- Merge: v unterhalb von u und w Innenwinkel(v) $> \pi$



Transformation in y-monotone Polygonen

Klassifizierung von $v \in V$

- Split: v oberhalb von u und w und Innenwinkel(v) $> \pi$
- Merge: v unterhalb von u und w Innenwinkel(v) $> \pi$
- Regular: Sonst



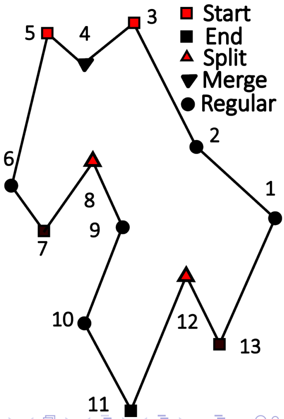
Transformation in y-monotone Polygonen

Klassifizierung von $v \in V$

- Split: v oberhalb von u und w und Innenwinkel(v) $> \pi$
- Merge: v unterhalb von u und w Innenwinkel(v) $> \pi$
- Regular: Sonst

Theorem

Ein Polygon ist y-monoton, wenn es weder Split- noch Merge-Knoten hat



Transformation in y-monotone Polygonen

Ziele

- nicht-monotone Teilstücke durch abtrennen monotonisieren
- Diagonalen dürfen keine andere Kanten schneiden

Transformation in y-monotone Polygonen

Ziele

- nicht-monotone Teilstücke durch abtrennen monotonisieren
- Diagonalen dürfen keine andere Kanten schneiden

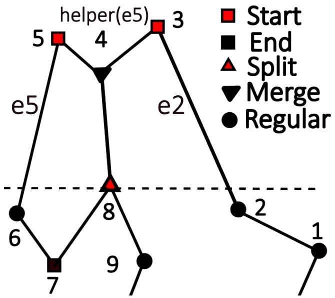
Nicht-monotone Teilstücke abtrennen

- Sweep-Line scannt Polygon
- Priority-Queue = Liste von Knoten
  - Split: Diagonale nach oben
  - Merge: Diagonale nach unten

Split-Knoten

Vorgehensweise

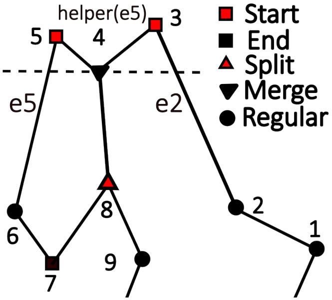
- Sei $v_i \in V$ ein Split-Knoten und seien $e_j, e_k \in E$ direkte linke bzw. rechte Kanten von v_i
- Helper(e_j) = niedrigster Knoten zwischen e_j und e_k, so dass Horizontale zu e_j innerhalb des Polygons liegt
- Falls es keinen gibt:
  oberer Endpunkt von e_j
- Verbinde v_i mit Helper(e_j) **oberhalb** von v_i



Merge-Knoten

Vorgehensweise

- Merge: Innenwinkel $> \pi$ und unterhalb seiner Nachbarn
- Sei $v_i \in V$ ein Merge-Knoten
- Seien $e_j, e_k \in E$ direkte linke bzw. rechte Kante von v_i
- Verbinde mit höchsten Knoten **unterhalb** von v_i
  - Problem: Knoten unterhalb noch nicht entdeckt
- Aber: v_i wird zu helper(e_j)

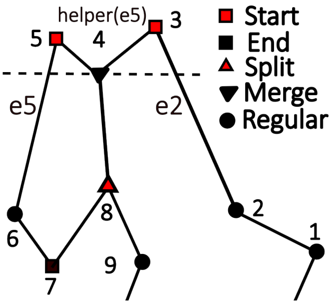


Merge-Knoten

Vorgehensweise

Wenn neuer helper(e_j)=v_l entdeckt wird:

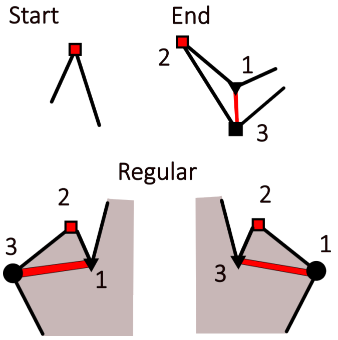
- Checke, ob alter Helper Merge-Knoten war
- Falls ja: Diagonale zwischen v_i und v_l ziehen
- Falls v_l Split-Knoten: Diagonale wird sowieso gezogen
- Falls kein neuer Helper gefunden wird: Diagonale zu unterstem Endpunkt von e_j ziehen



Weitere Knoten

Start-, End- und Regular-Knoten

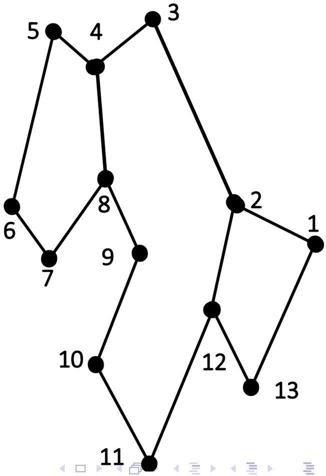
- Start: Setze Helper der linken Kante
- End: Diagonale, falls Helper der linken Kante Merge-Knoten war
- Regular: Diagonale, falls linker oder rechter Helper ein Merge-Knoten war



Ergebnis der Transformation

Zwischenergebnis

- Polygon ist nun y-monoton
- Kann nun in Dreiecke aufgeteilt werden



Triangulierung

Greedy-Algorithmus

- Durchlaufe Knoten in absteigender Reihenfolge
- Füge Diagonale zu so vielen Knoten wie möglich hinzu
- Falls keine Diagonale möglich: Lege Knoten auf Stack
- Jede Diagonale spaltet ein Dreieck ab und reduziert zu betrachtende Knoten

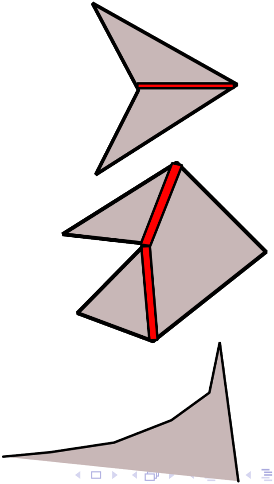
[19]

Triangulierung

Wann sind Diagonalen möglich?

1. Knoten liegt auf der anderen Seite
2. Innengrad eines Knotens zwischen zwei nicht-benachbarten Knoten auf der gleichen Seite ist $< \pi$

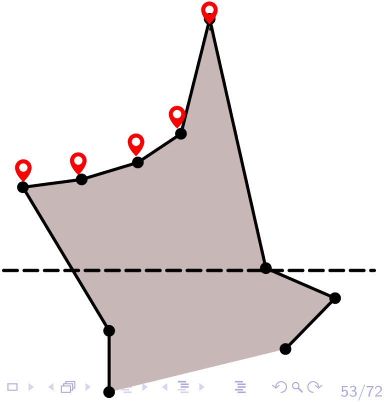
- Falls keine Diagonale möglich ist: Lege Knoten auf Stack



Triangulierung

Wann sind Diagonalen hinzuzufügen?

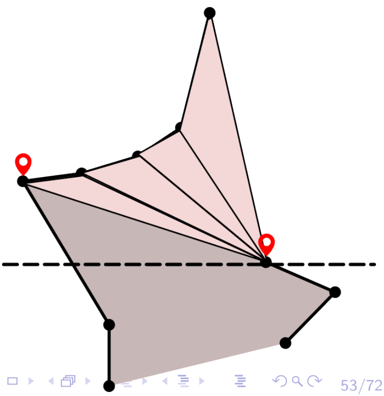
1. Knoten liegt auf der langen Seite des Trichters
   - Füge Diagonale zu allen Knoten auf dem Stack hinzu
   - Lösche alle Knoten auf dem Stack
   - Lege den zuletzt obersten und den aktuellen Knoten auf den Stack



Triangulierung

Wann sind Diagonalen hinzuzufügen?

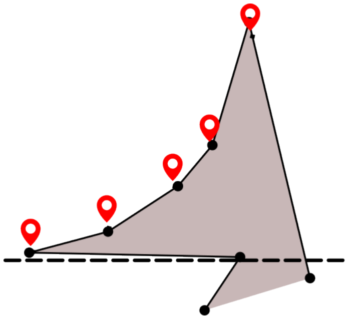
1. Knoten liegt auf der langen Seite des Trichters
   - Füge Diagonale zu allen Knoten auf dem Stack hinzu
   - Lösche alle Knoten auf dem Stack
   - Lege den zuletzt obersten und den aktuellen Knoten auf den Stack



Triangulierung

Wann sind Diagonalen hinzuzufügen?

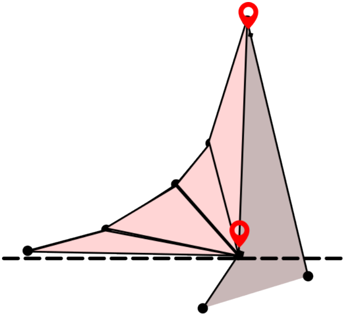
2. Innengrad eines Knotens zwischen zwei nicht-benachbarten Knoten auf der gleichen Seite ist $< \pi$
   - Lösche obersten Knoten vom Stack
   - Solange Diagonalen möglich sind, füge diese hinzu
   - Lege aktuellen und den zuletzt gelöschten Knoten auf den Stack



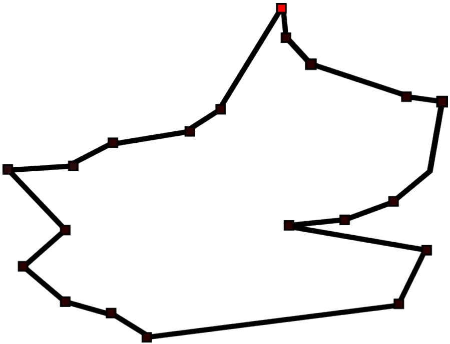
Triangulierung

Wann sind Diagonalen hinzuzufügen?

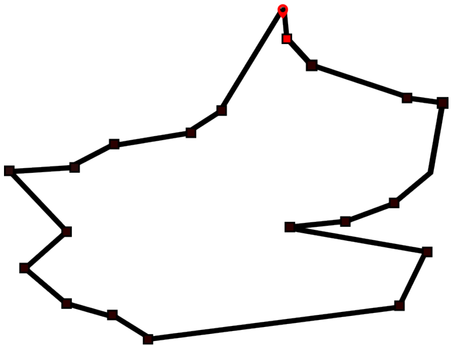
2. Innengrad eines Knotens zwischen zwei nicht-benachbarten Knoten auf der gleichen Seite ist $< \pi$
   - Lösche obersten Knoten vom Stack
   - Solange Diagonalen möglich sind, füge diese hinzu
   - Lege aktuellen und den zuletzt gelöschten Knoten auf den Stack



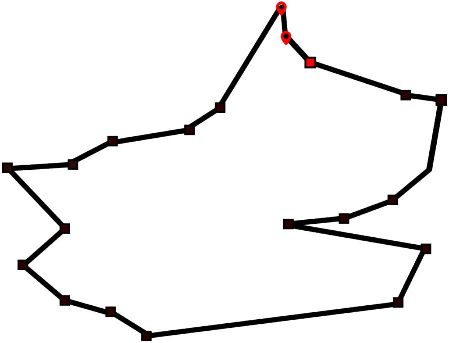
Triangulierung



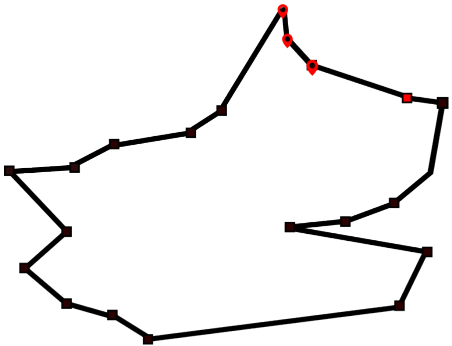
Triangulierung



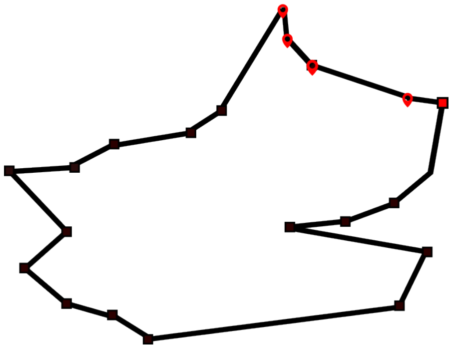
Triangulierung



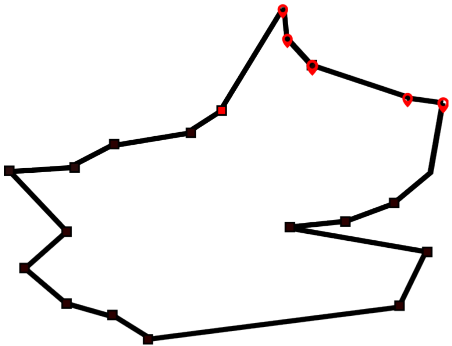
Triangulierung



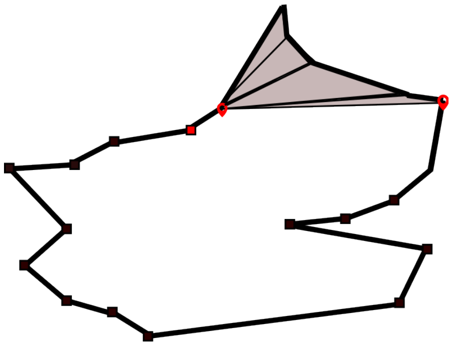
Triangulierung



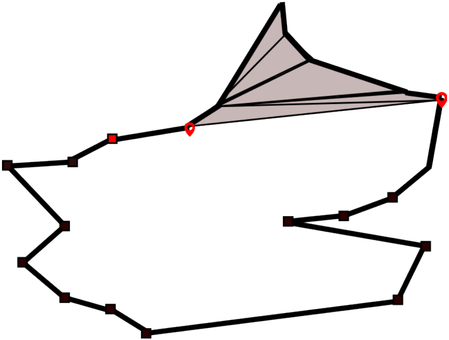
Triangulierung



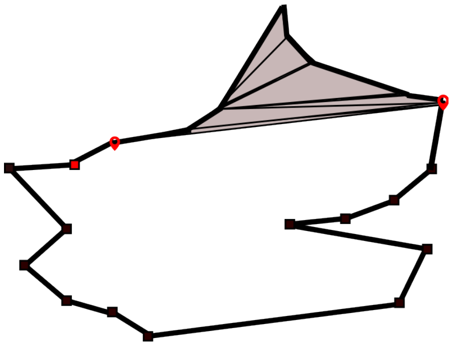
Triangulierung



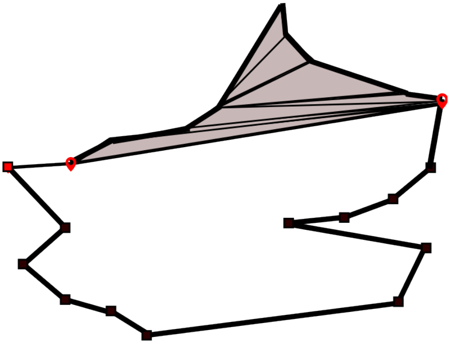
Triangulierung



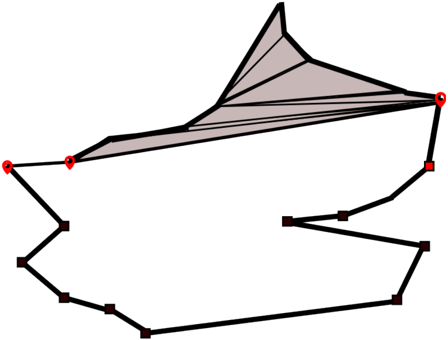
Triangulierung



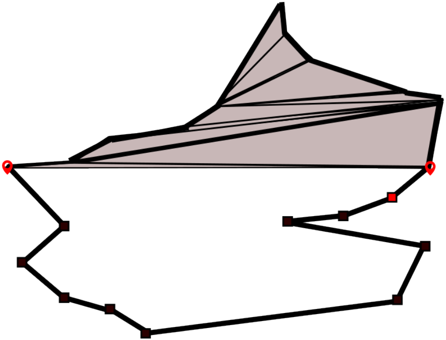
Triangulierung



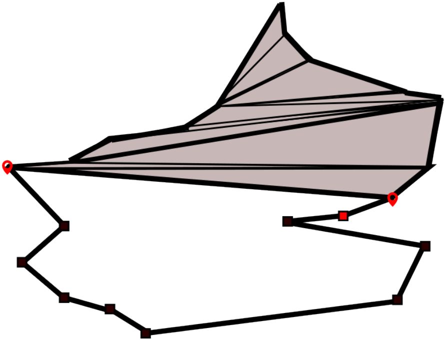
Triangulierung



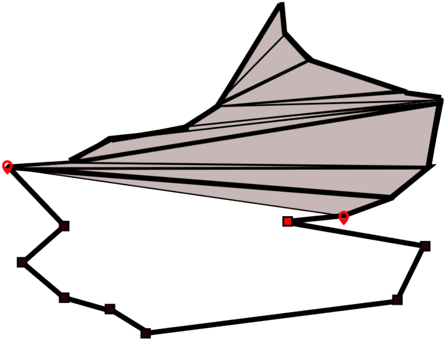
Triangulierung



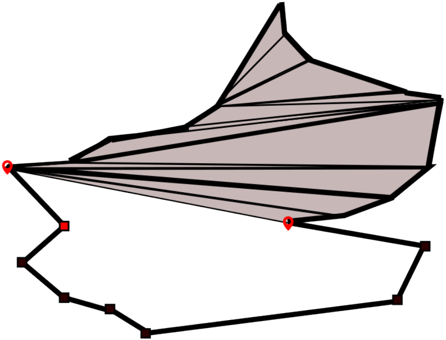
Triangulierung



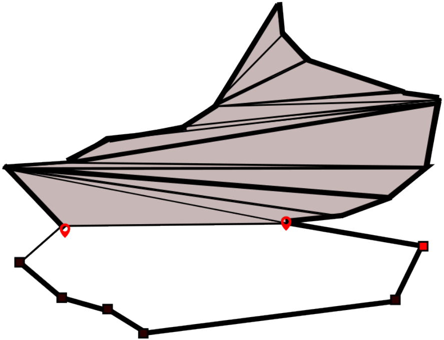
Triangulierung



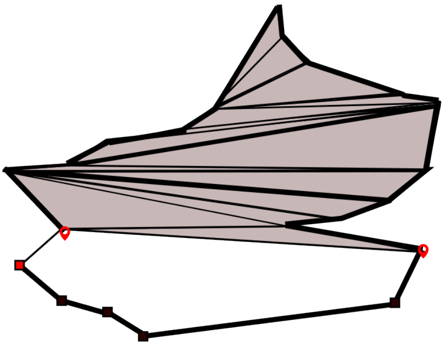
Triangulierung



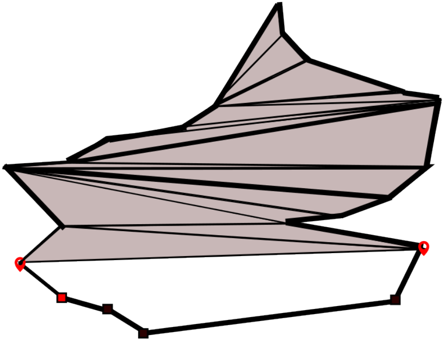
Triangulierung



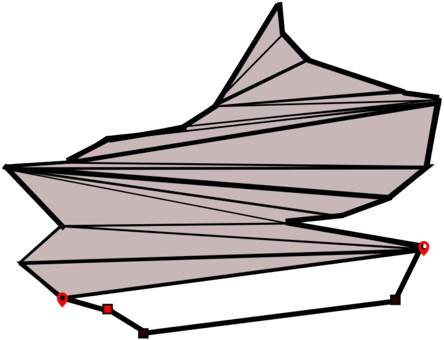
Triangulierung



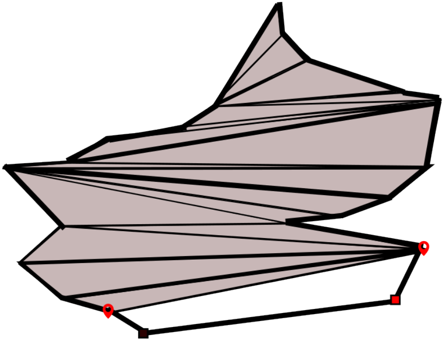
Triangulierung



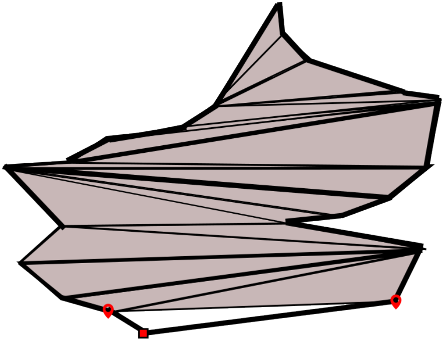
Triangulierung



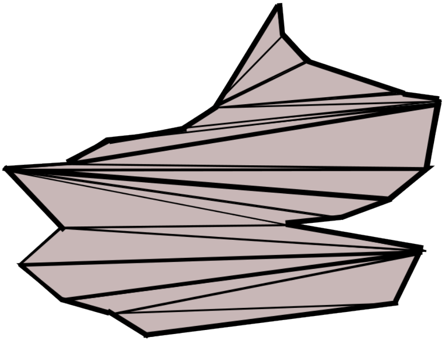
Triangulierung



Triangulierung



Triangulierung



Details

Laufzeit

- Jeder Knoten wird nur einmal betrachtet: O(n)
- Monotonisierung und Triangulierung damit insgesamt in O(n log n)

Und nun?

- Polygon muss gerastert werden
  - Linienalgorithmus für die Kanten
  - Füllregel, um innen festzulegen
- Alternative: Polygon in Trapeze rastern [22]
- Benachbarte Polygone nicht mehrfach einfärben

Quellen I

[1] M. F. Worboys and M. Duckham, *GIS: a computing perspective*. CRC press, 2004.

[2] W. Collins, “5.2 raster image processing,” accessed: 2023-01-03. [Online]. Available: https://opentextbc.ca/graphicdesign/chapter/5-2-raster-image-processing/

[3] “Poppler,” accessed: 2023-01-03. [Online]. Available: https://poppler.freedesktop.org/

[4] M. Urman, “Cairo tutorial,” 2007, accessed: 2023-01-03. [Online]. Available: https://www.cairographics.org/tutorial/

Quellen II

[5] C. Loop and J. Blinn, “Resolution independent curve rendering using programmable graphics hardware,” in *ACM SIGGRAPH 2005 Papers*, 2005, pp. 1000–1009.

[6] J. Trivedi, “Simulation of dda (digital differential analyzer) line generation algorithm,” *IJCSN International Journal of Computer Science and Network*, pp. 110–111, 2015.

[7] P. Koopman, “Bresenham line-drawing algorithm,” *Forth Dimensions*, vol. 8, no. 6, pp. 12–16, 1987.

[8] X. Wu, “An efficient antialiasing technique,” *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, p. 143–152, jul 1991. [Online]. Available: https://doi.org/10.1145/127719.122734

Quellen III

[9] J. R. Van Aken, “An efficient ellipse-drawing algorithm,” *IEEE Computer Graphics and Applications*, vol. 4, no. 9, pp. 24–35, 1984.

[10] A. Zingl, *A rasterizing algorithm for drawing curves.* na, 2012.

[11] Ag2gaeh, accessed: 2023-01-09. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=72458167

[12] P. De Casteljau, “Outillages méthodes calcul,” *Andr e Citro en Automobiles SA, Paris*, vol. 4, p. 25, 1959.

Quellen IV

[13] K. Beets and D. Barron, "Super-sampling anti-aliasing analyzed," 2000.

[14] Psiĥedelisto, 2007, accessed: 2023-01-09. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=74647088

[15] S. Lin, R. W. Lau, X. Lin, and P. Y. Cheung, "An anti-aliasing method for parallel rendering," in *Proceedings. Computer Graphics International (Cat. No. 98EX149)*. IEEE, 1998, pp. 228–235.

Quellen V

[16] Xu-dong, Jiang, B. Sheng, W. yao Lin, W. Lu, and L. zhuang Ma, “Image anti-aliasing techniques for internet visual media processing: a review,” *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)*, 2014.

[17] J. L. Bentley and T. A. Ottmann, “Algorithms for reporting and counting geometric intersections,” *IEEE Transactions on computers*, vol. 28, no. 09, pp. 643–647, 1979.

[18] L. Subramaniam, *Partition of a non-simple polygon into simple polygons.* University of South Alabama, 2003.

Quellen VI

[19] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational geometry: algorithms and applications, 2nd edition*. Springer, 1998.

[20] H. ElGindy, H. Everett, and G. Toussaint, "Slicing an ear using prune-and-search," *Pattern Recognition Letters*, vol. 14, no. 9, pp. 719–722, 1993.

[21] G. H. Meisters, "Polygons have ears," *The American Mathematical Monthly*, vol. 82, no. 06, pp. 648–651, 1975.

Quellen VII

[22] R. Seidel, “A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons,” *Computational Geometry*, vol. 1, no. 1, pp. 51–64, 1991.

Sweep-Line Algorithmus 1

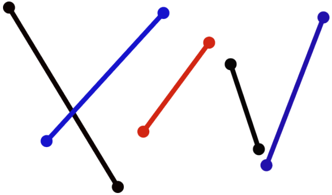
- Idee: Teste nur aktuell direkt benachbarte Kanten auf Schnittpunkte
- Status **T** (balanced BST): Geordnete Aktivenliste aller Kanten, die die Sweepline schneidet
- Eventpunkte = End- und Schnittpunkte der Kanten, gespeichert in priority-Queue **Q** (balanced BST)
- Aktionen am Eventpunkt:

  - oberer Endpunkt: Kante zu T hinzufügen, auf Schnitt zu linker/rechter Kante testen

  - unterer Endpunkt: Kante aus T löschen, auf Schnitt der alten Nachbarn testen

Sweep-Line Algorithmus 2

- Berechnete Schnittpunkte werden zu Eventpunkten
  - Beteiligte Kanten tauschen Reihenfolge
  - Müssen auf Schnitte mit neuen Nachbarn getestet werden
- Zeit: O(n log n + I log n) mit n = Anzahl der Kanten, I = Anzahl der Schnittpunkte



Beweis Polygon in n-2 Dreiecke 1

Beweis

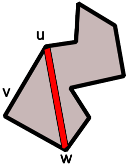
- Induktion über n. Für n = 3 trivial, da Dreieck.
- Sei nun n > 3 und das Theorem gelte für alle m < n.
- Sei P ein Polygon mit n Knoten. Beweis, dass eine Diagonale existiert:
- Seien v,u,w $\in V$, v der linkest-unterste Knoten, u,w die jeweiligen Endknoten der Segmente an v.
- Liegt die Kante (u,w) innerhalb von P, so ist es eine Diagonale
- Ansonsten: Es liegen Knoten innerhalb des Dreiecks u,v,w oder auf der Diagonalen (u,w)

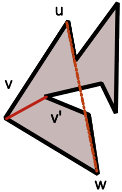
Beweis Polygon in n-2 Dreiecke 2

Beweis

- Von diesen sei v' der am weitest entfernte Knoten von (u,w) innerhalb des Dreiecks
- (v,v') ist nun eine Diagonale ohne Schnittpunkt, da sonst ein Endpunkt der geschnittenen Kante innerhalb des Dreiecks und weiter weg von (u,w) liegen würde, was aber Widerspruch zu v' ist
- $\Rightarrow$ Diagonale existiert

Anzahl der Dreiecke?





Beweis Polygon in n-2 Dreiecke 3

Beweis

- Jede Diagonale schneidet P in zwei Sub-Polygone P_1 und P_2
- Sub-Polygone haben weniger Knoten, können also nach IV auch trianguliert werden
- Jeder Knoten von P wird entweder zu P_1 oder P_2 zugeteilt, außer die beiden Knoten der Diagonale
- Also: $P_1 n + P_2 n =$ n + 2
- Durch Induktion: Jede Triangulierung von P_i besteht aus m_i -2 Dreiecken und somit die Triangulierung von P aus $(P_1 n$ - 2) + ($P_2 n$ - 2) = n -2 Dreiecken

Details Transformation y-monoton 1

- Knoten werden in einer priority-queue **Q** mit y-Koordinate als Priorität gespeichert (zusätzlich nach x-Koordinate)
- Aktive Kanten werden in einem binären Suchbaum **T** gespeichert, ebenfalls nach y-Koordinate sortiert
  - Da nur Kanten links von Split- und Merge-Knoten betrachtet werden, reicht es, die Kanten zu speichern, bei denen das innere des Polygon rechts von ihnen liegt
  - Zu jeder Kante wird der Helper gespeichert
- Status des Sweep-Line Algorithmus ist T und die Helper
- Sub-Polygone werden in doppelt verketteter Kantenliste **D** gespeichert (Kanten gegen den Uhrzeigersinn)

Details Transformation y-monoton 2

- Laufzeit: Erstellen von Q dauert O(n) und T dauert konstante Zeit

  \- An jedem Knoten (Event) wird höchstens einmal Q verändert (löschen), in T einmal gesucht, eingefügt und gelöscht und höchstens zwei Diagonalen in D hinzugefügt.

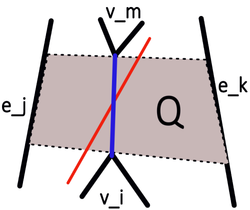
  \- Operationen in Q und T jeweils in O(log n) und hinzufügen in D in O(1)
- Bei höchstens n Knoten ergibt sich eine Laufzeit von O(n log n)
- Speicherbedarf: Jeder Knoten höchstens einmal in Q und jede Kante höchstens einmal in T, D gespeichert, also O(n)

Beweis Split- und Merge-Diagonale

Lemma

Der Algorithmus führt nicht-schneidende Diagonalen ein

- Beweis: Sei (v_i, v_m) die hinzugefügte Kante, wenn v_i erreicht wurde
- Seien e_j, e_k die jeweils direkte linke bzw. direkte rechte Kante
- Also ist bei erreichen von v_i helper$(e_j) = v_m$, also der niedrigste Knoten zwischen den Kanten
- Betrachte Fläche Q, die von den Horizontalen bei v_m, v_i und durch die Kanten e_j, e_k eingeschlossen wird



- Es gibt keine Knoten innerhalb von Q, sonst wäre v_m nicht der helper von e_j
- Angenommen es gäbe eine Kante, die die Diagonale schneidet
- Da der Endpunkt nicht in Q liegen kann und es sich um ein einfaches (nicht-überlappendes) Polygon handelt, muss die Diagonale die Horizontalen von v_m oder v_i zu e_j schneiden
- Widerspruch: Dann wäre e_j nicht direkt links von v_i