

## 2 Praxisaufgaben zu Grundlagen der Rasterisierung

In dieser Praxisaufgabe geht es um verschiedene grundlegende Standardalgorithmen der Rastergraphik. Die relevanten Algorithmen sollen dabei in Software nachimplementiert werden.

**Hinweise zur Implementierung:** Der Übung liegen verschiedene Materialien bei. Dabei handelt es sich um eine Musterlösung (für Windows), in denen Sie die Ergebnisse der einzelnen Aufgaben betrachten können, ein lauffähiges Programm als Quelltext, dem die von Ihnen im Rahmen der Aufgaben zu erstellenden Algorithmen fehlen und eine allgemeine Erklärung zum Programmaufbau. Die Programmstellen, denen Sie Quelltext hinzufügen müssen, sind gesondert markiert und enthalten weitere Hinweise zur Lösung der Aufgabe. Des Weiteren finden Sie den Quellcode der Musterlösung unter `src_solution`.

Die Bedienung der Musterlösung (und Ihres Arbeitsprogrammes) erfolgt mittels eines Kontextmenüs, das Sie über die rechte Maustaste erreichen. Neben den einzelnen Menüpunkten finden Sie in Klammern den entsprechenden Tastatur-Shortcut. Außerdem können Sie das Raster mit dem Mausekranz zoomen und durch Gedrückthalten des Rades und Bewegen der Maus verschieben.

Eine Projektdatei für Visual-Studio 2012 finden Sie in den entsprechenden Unterverzeichnissen des Ordners `build`. Falls Sie nicht mit Visual-Studio arbeiten, finden Sie im Unterordner `build/cmake` entsprechende CMake-Dateien.

In der Implementierung existiert eine Oberklasse `tool_base`, welche die gemeinsamen Funktionalitäten aller Zeichenwerkzeuge kapselt. Diese besitzt mehrere Versionen der Methode `draw` zum Zeichnen von Objekten, die über einen, beziehungsweise zwei Punkte definiert sind. Bei Interesse können Sie dem beiliegenden PDF-Dokument genauere Informationen zum Programmaufbau entnehmen.

### 2.1 Rasterisierung von Linien

In der Vorlesung wird auf spezielle Anforderungen an Algorithmen zur Rasterisierung von Linien eingegangen. In den folgenden Aufgaben sollen zwei etablierte Algorithmen, der DDA- und der Bresenham-Algorithmus, programmiert werden.

#### 2.1.1 Digital Differential Analyzer

Implementieren Sie den DDA-Algorithmus zum Rastern von Linien, indem Sie die `draw`-Methode der Klasse `dda_line_tool` aus der Datei `dda_line_tool.cpp` vervollständigen. Stellen Sie dabei zunächst den Standardfall für das Linienrastern her.

#### 2.1.2 Bresenham-Algorithmus

Implementieren Sie genauso den Bresenham-Algorithmus zum Rastern von Linien. Hierfür muss die `draw`-Methode der Klasse `bresenham_line_tool` aus der Datei `bresenham_line_tool.cpp` erweitert werden. Überlegen Sie für die Abgabe, welche Vor- und Nachteile der Bresenham-Algorithmus gegenüber dem DDA hat.

### 2.2 Füllalgorithmen

Zum Füllen existieren verschiedene Verfahren, die in der Vorlesung besprochen werden. Ihre Aufgabe besteht nun darin zwei grundlegende Füllalgorithmen nachzuvollziehen. Dabei handelt es sich zunächst

um die naive Implementierung in Form eines rekursiven Algorithmus' und anschließend um eine Erweiterung, welche keine Rekursion benötigt.

### 2.2.1 Rekursives Füllen

Implementieren Sie den rekursiven Füllalgorithmus wie in der Vorlesung gezeigt in der `draw`-Methode der Klasse `recursive_fill_tool`. Erstellen Sie außerdem eine Routine, um sich die maximale Rekursionstiefe auf der Ausgabekonsole ausgeben zu lassen. Wenn das zu füllende Feld zu groß wird, erzeugt das rekursive Füllen einen Programmabsturz (stack overflow).

*Hinweis:* Zur Abfrage der Farbe eines Pixels können Sie die Funktion `canvas_buffer::get_pixel(..)` verwenden.

### 2.2.2 Nichtrekursives Füllen

Implementieren Sie einen nichtrekursiven Füllalgorithmus, der auf dem FIFO-Prinzip (**first in, first out**) beruht. Dabei werden die noch zu verarbeitenden Pixel in einem STL-Kontainer zwischengespeichert. Der Kontainer `std::deque` (**d**ouble **e**nded **q**ueue) verfügt über die nötige Funktionalität. Mit `deque::push_back(..)` werden Elemente am Ende eingefügt. Mit `deque::front()` erhält man das vorderste Element des Kontainers und mit `deque::pop_front()` entfernt man es. Die Implementierung des nichtrekursiven Füllalgorithmus' erfolgt in der Klasse `non_recursive_fill_tool`.

*Hinweis:* Der Programmabsturz aus Aufgabe 2.2.1 darf beim nichtrekursiven Füllen nicht mehr auftreten.

## 2.3 Zusatzaufgaben

*Hinweis:* Die erreichbare Bonuspunktezahl für eine korrekt gelöste Zusatzaufgabe finden Sie am Ende der jeweiligen Aufgabenbeschreibung. Maximal können jedoch nicht mehr als 3 Bonuspunkte von diesem Übungsblatt eingebracht werden.

- Implementieren Sie die Funktionalität des Rechteck-Werkzeuges um per Mausinteraktion ein Rechteck zu zeichnen. Das Grundgerüst für diese Aufgabe stellt die Klasse `rectangle_tool` dar, dessen Implementierung Sie in der Datei `rectangle_tool.cpp` finden. (0,5 Pkt.)
- Implementieren Sie den Kreisrasterisierer nach Bresenham in `bresenham_circle_tool::draw(..)`. Recherchieren Sie selbstständig, um die genaue Funktionsweise dieses Rasterisierers zu verstehen. (1 Pkt.)
- Implementieren Sie einen Rasterisierer für eine Sternform mit per Kontextmenü einstellbarer Eckenzahl und Radius. Orientieren Sie sich für die Interaktion am Kreisrasterisierer. (1,5 Pkt.)
- Implementieren Sie den Pixellaufalgorithmus aus der Vorlesung zum effizienten Füllen. Verwenden Sie die Testfüllform, um Ihr Ergebnis zu validieren. Die Implementierung soll in der `draw`-Methode der Klasse `line_fill_tool` erfolgen. (1,5 Pkt.)
- Implementieren Sie den Sweepeline-Algorithmus aus der Vorlesung zum Füllen eines Dreiecks. Leiten Sie dazu eine eigene Klasse `sweep_line_tool` ab und definieren Sie die Koordinaten der 3 Eckpunkte fest im Code. Fügen Sie einen Menüpunkt im Kontextmenü hinzu. (1,5 Pkt.)