Dokumentation Computergrafik 1 - Aufgabe 1

Jan Rabe 766212

9.11.2010

Contents

Part I

Aufgabenstellung

Chapter 1

Pflichtaufgaben

Implementieren Sie zunächst die Rasterisierung von Linie, gefülltem Kreis, gefülltem Dreieck und gefülltem Polygon in einer einheitlichen Farbe Ihrer Wahl (außer Schwarz oder dunkles Grau). Implementieren Sie für das Polygon sowohl eine Rasterisierung als ganzes, als auch eine Triangulierung. Bei der Triangulierung ist es ausreichend, wenn Sie konvexe Polygone zerlegen können. Erstellen Sie anschließend einen zweiten Rasterisierer, der lediglich die Ränder der Primitiven in weißer Farbe zeichnet. Erweitern Sie nun den ersten Rasterisierer und verwenden Sie die aus der Datei geladenen Farben zur Füllung der Primitiven. Hierbei muss eine Interpolation der Farben zwischen den Eckpunkten stattfinden.

Chapter 2

Ausgefüllte Flächen

Im Rasterzier landen die Koordinaten und Farbwerte als 'Figure'. Dort wird jeweils getestet, ob die Figure ein Rechteck, eine Linie, ein Kreis, ein Dreieck oder ein Polygon ist. Falls es sich um eine Linie handelt, wird diese in das Objekt Line gecasted. Daraufhin füllt die Linie den FrameBuffer. Der FrameBuffer ist ein zwei-Dimensionaler Color Array, welcher am Ende auf ein Canvaz abgebildet wird. Das heißt, die Linie zeichnet sich selbst. So auch analog die anderen Flächen.

Um auf Sonderfälle einzugehen, habe ich unterschiedliche Test-Dateien mit Koordinaten zu den einzelnen Flächen angelegt.

2.1 Rechteck

Die funktionalität war bereits gegeben.

2.2 Linie

Anfangs habe ich die Linie mithilfe der Liniengleichung gezeichnet, jedoch haben lediglich zwei Fälle funktioniert. Nämlich von links nach rechts auf Horizontaler Richtung und von oben Links nach unten Rechts in einem 45° Winkel. Daraufhin habe ich etwas rechechiert und bin auf den Bresenham Algorythmus gestoßen. (http://www.cs.unc.edu/ mcmillan/comp136/Lecture6/Lines.html) Dieser geht auch elegant auf alle Möglichen Sonderfälle ein.

2.3 Kreis

Der Algorythmus zeichnet Zeile für Zeile, wobei jeder Punkt überprüft wird, ob er innerhalb des Kreises liegt. Wenn ja wird der FrameBuffer mit dem Farbwert des Kreises gefüllt.

2.4 Dreieck

Anfangs wird der minimale und maximale x-/y-Wert berechnet. Damit hat man das Grenz-Rechteck in welchem sich das Dreieck befindet. Dazu berechnet man Alpha, Beta und Gamma, welche zusammen 1 ergeben. Alpha, Beta und Gamma berechnet man mit Hilfe der Liniengleichung. Nun schaut man nur noch ob, Alpha, Beta und Gamma jeweils im Bereich zwischen 0 und 1 liegen. Wenn ja, liegt der Punkt im Dreieck und wird gezeichnet.

2.5 Polygon

Es gibt die Methode scanFill() für den Scan-Fill Algorythmus und fillConvex() für den Triangulierungs Algyrthmus.

2.5.1 Scan-Fill

Überprüft, ob ein Punkt im Polygon liegt, mit Hilfe des pointInPolygon-Algorythmus. (http://www.ecse.rpi.edu/Homepages/wrf/Research/Short$_N otes/pnpoly.html$)

2.5.2 Triangulation

Die Idee des Algorythmus von Kong (http://www.sunshine2k.de/stuff/Java/Polygon/Kong/Kong.html) ist es die nach außenstehenden Ecken (Ears) abzuschneiden. Diese Ecken werden aus der Liste der Polygonknoten entfernt und als Dreieck in die Dreick-Liste hinzugefügt. Der Kong Algorythmus wird im Konstruktor ausgefüht und muss daher nur einmal beim erstellen ausgeführt werden.

Chapter 3

Umrandung der Flächen

Um die Umrandungen zu testen, wurde die Klasse OutlineRasterizer erstellt, die von den Figuren die outline()-Methode anstelle der fill()-Methode aufruft.

3.1 Rechteck

Man nehme alle Eckpunkte und verbindet diese durch Linien in vorgegebener Reihenfolge.

3.2 Linie

Bei der Linie kommt nichts hinzu.

3.3 Kreis

Hier habe ich einfach eine weitere Abfrage eingebaut, welche überprüft, ob (-Radius * Math.PI) ¡= der Liniengleichung liegt. Optimierungsbedarf besteht.

3.4 Dreieck

Analog zum Viereck.

3.5 Polygon

Analog zum Viereck.

Chapter 4

Vorschläge zur Erweiterung

4.1 Suchen Sie nach optimierten Algorithmen und implementieren Sie diese.

Die Linie funktioniert nun interpoliert in alle Richtungen, dank des Jack Bresenham Line Algorythmus.

4.2 Implementieren Sie Antialiasing.

Aus Zeitmangel, habe ich dies nicht mehr geschafft.

4.3 Erweitern Sie die Triangulierung so, dass Sie beliebige Polygone zerlegen können.

Mit Hilfe des Kong-Algorythmus, können nun konkave Polygone rasterisiert werden. Jedoch Polygone, die sich überschneiden und somit Artefakte bilden, oder Polygone mit Löchern, funktionieren damit jedoch noch nicht.