Midpoint circle Algorithmus

Kreisberechnung mit möglichst wenig Rechenaufwand.

Möglichkeit der Spiegelung an der X- und Y-Achse.

Spiegelung an der Y-Achse:

Wie wir in dieser Abbildung sehen, können wir jeden zu jedem berechneten Kreispunkt einen Anderen Spiegeln. Dies halbiert die Anzahl Kreispunkte, die wir berechnen müssen, um unseren Kreis zu zeichnen.

Um die Koordinaten des neuen Kreispunktes zu erhalte, multiplizieren wir die X-Koordinate unseres Originalpunktes mit -1.

P( x | y ) => P’( -x | y)  
P( 10 | 5 ) => P’( -10 | 5 )

Spiegelung an der X-Achse:

Dieselbe Spiegelung wie an der Y-Achse können wir auch an der X-Achse vornehmen. Wir haben nun die Menge an Kreispunkten, die wir berechnen müssen ein weiteres Mal halbiert.

Um die Koordinaten des neuen Kreispunktes zu erhalte, multiplizieren wir die Y-Koordinate unseres Originalpunktes mit -1.

P( x | y ) => P’( x | -y )  
P( 10 | 2 ) => P’( 10 | -2 )

Spiegelung an beiden Y=X Achse

Auch können wir unsere Bildpunkte an der Gerade Y = X spiegeln. Somit müssen wir nur noch die Hälfte eines Kreisviertels berechnen. Wir haben den Rechenaufwand ein weiteres Mal halbiert.

Um die Koordinaten des neuen Kreispunktes zu erhalte, tauschen wir die X und Y Koordinaten unseres berechneten Punktes.

P( x | y ) => P’( y | x )  
P( 10 | 2 ) => P’( 2 | 10 )

Spiegelung an beiden Y=-X Achse

Dieselbe Spiegelung wie an der   
Y = X Achse können wir auch an der Y = -X Achse vornehmen. Wir haben nun die Menge an Kreispunkten, die wir berechnen müssen ein weiteres Mal halbiert.

Um die Koordinaten des neuen Kreispunktes zu erhalte, tauschen wir die X und Y Koordinaten unseres berechneten Punktes und multiplizieren beide mit -1.

P( x | y ) => P’( -y | -x )  
P( 10 | 2 ) => P’( -2 | -10 )

Spiegelung an allen Achsen

( -y | x )

( y | x )

Wir haben nun unseren Kreis in acht Sektoren geteilt und demonstriert, dass wir mit den Kreispunkten einem dieser Sektoren unseren Kreis um jeden anderen Sektor erweitern können.

( -x | y )

( x | y )

Nun können wir die jedem Sektor eine Formel zuweisen. Wir müssen nur noch den ( x | y ) Sektor berechnen und können die restlichen Kreispunkte ableiten.

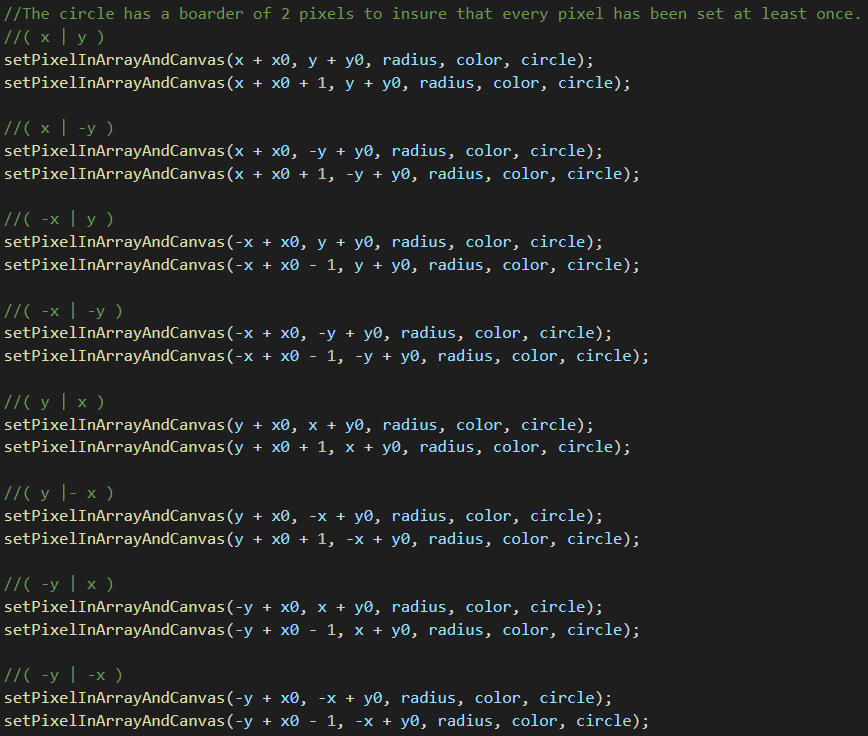
( -x | -y )

( x | -y )

( -y | -x )

( y | -x )

Wir sehen diese Ableitung gut in diesem Ausschnitt unseres Codes.



Nun müssen wir nur noch die Punkte dieses einen Sektors berechnen.

Was uns interessiert ist, welche Pixel wir einfärben sollen um unseren Kreis zu zeichnen.

Pixel im Array

Wir können die theoretische Kreislinie mit mathematischen Formeln Berechnen. Jedoch müssen wir eine Formel entwickeln, mit der wir bestimmen, ob wir einen Pixel einfärben.

Theoretischer Kreis

Nun werden wir bestimmen, wie wir unsere Kreispixel auswählen.

Wir berechnen den Theoretischen Mittelpunkt jedes Pixels. Mit diesen bilden wir ein Zweierpaar von Pixeln die sich berühren.

Was uns nun interessiert ist, ob die theoretische Kreislinie zwischen den Mittelpunkten eines Pares verläuft. Um dies zu erfahren berechnen wir die X-Koordinate des Kreises beim Y-Wert der beiden Pixelmittelpunkten berechnen und diese mit den X-Koordinaten der beiden Punkten vergleichen.

Verläuft die theoretische Kreislinie zwischen einem Mittelpunktpaar, so werden wir diese beiden Pixel einfärben

Nun bilden wir nochmals Pixelpaare. Diesmal mit allen Pixeld die übereinander sind. Auch bei ihnen führen wir den selben Prozess durch

Das Resultat

Nachdem wir dies für einen Sektor des Kreises gemacht habe, können wir die Restlichen Kreispixel mit der schon beschriebenen Formel berechnen.

Auf diese Art können wir mit dem Midpoint Circle Algorithmus schnell und effizient einen Kreis aus Pixeln berechnen. Perfekt für unser Projekt.