

π Visualisierung

Svenson Han Göthel

(Lloyd Gymnasium Bremerhaven, 5c)

Inhaltsverzeichnis

Einleitung.....	1
Genutzte Software und Framework Implementation.....	2
Game-Loop für die Animation, frame by frame.....	2
Linien Malfunktion.....	2
Abgerollter Kreisumfang.....	2
Ergebnis.....	3
Kompletter Kreis.....	3
Abrollender Kreis.....	3
Abgerollter Kreis.....	4
Implementation.....	4
Kreis-Näherung.....	5
Ergebnis.....	5
3 Ecken.....	5
9 Ecken.....	6
57 Ecken.....	7
Implementation.....	7
Trigonometrische Verhältnisse.....	8
Ergebnis.....	10
Winkel = 127 Grad.....	10
Winkel = 225 Grad.....	11
Implementation: Sinus & Cosinus im Einheitskreis.....	11
Implementation Sinus & Cosinus Graph.....	12
Diskussion.....	12
Quellenangabe.....	13

Einleitung

In diesem Projekt wird π visualisiert:

1. Abrollender Kreis, Durchmesser, Umfang
2. Kreis-Näherung nach Archimedes
3. Trigonometrische Verhältnisse, Sinus und Cosinus

π wird mit dem Verhältnis von Umfang durch Durchmesser definiert.

Mit π kann man folgendes berechnen:

1. Kreisumfang = $2\pi r$
2. Kreisfläche = πr^2

3. Kugelvolumen = $\frac{4}{3} \pi r^3$

Der C++ Sourcecode ist unter [diesem Link einzusehen](#).

Ein Video ist unter [diesem Link veröffentlicht](#).

Genutzte Software und Framework Implementation

In meinem [Informatikkurs \(Programmierkurs\) mit meinem Vater](#) haben wir zusammen [eine Bibliothek \(Framework\)](#) erstellt um bequem Punkte auf dem Bildschirm zu malen.

Hier haben wir mathematische Elemente in C++ erstellt, unter anderem Klassen für Punkt, Vektor, Kreis, Rechteck und Linie. Die jeweilige Klasse ist in der Lage grundlegende mathematische Operationen auszuführen (Addition, Subtraktion...) als aber auch sich selber auf dem Bildschirm malen.

Diese Bibliothek wiederum verwendet SDL2, ein weiteres Framework welches grundlegende grafische Funktionen wie Fensteröffnen und Grafikspeicherzugriff zur Verfügung stellt.

In meinem Informatikkurs haben wir unter anderem folgende Grundlegende Funktionen erstellt welche auch in diesem Projekt genutzt werden.

Game-Loop für die Animation, frame by frame

- Nutzung von zwei Framebuffer (Bilder), eines sichtbar, das andere wird gemalt.
- Nach Beendigung des Malens werden die Framebuffer getauscht.
- Optional werden noch eingaben Verarbeitet (Tastatur)

Linien Malfunktion

- Eingabewerte sind Start- und Endpunkt
- Die X und Y Distanz zwischen den beiden Punkten wird bestimmt und eine Schleife über den größeren Abstand abgelaufen um Lücken zu vermeiden. Innerhalb der Schleife wird die Pixelfarbe pro Punkt gesetzt und dann die X- und Y-Position gegen den Endpunkt für den nächsten Schleifendurchlauf erhöht.

Und so weiter ...

Abgerollter Kreisumfang

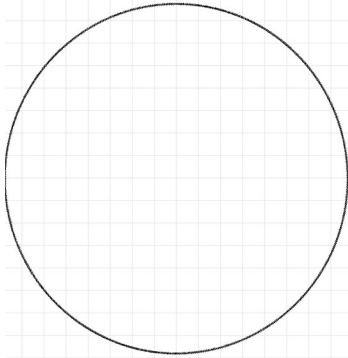
Meine erste Idee war einen Kreis zu einer Linie abzurollen und immer bei einem ganzen Durchmesser eine neue Farbe zu wählen. Das entspricht der Formel $\pi = \text{Umfang}(u) \text{ durch Durchmesser}(d)$. Die Linie ist dann dreimal so lang wie der Kreisdurchmesser plus ein paar zerquetschte :-)

Ergebnis

Kompletter Kreis

fps 49.792530, animating, manual

$2 \cdot \pi \cdot r$ Ausgerollt
 $u = \text{umfang}, r = \text{radius}, d = \text{durchmesser}$
 $u = 2 \cdot \pi \cdot \text{Radius}$
 $\pi = u / 2 \cdot r$
 $d = 2 \cdot r$



$\pi = 3.141593$
 $u = 2440.074951$
 $r = 388.349976$
 $d = 776.699951$

Abrollender Kreis

fps 55.793194, animating, manual

$2 \cdot \pi \cdot r$ Ausgerollt
 $u = \text{umfang}, r = \text{radius}, d = \text{durchmesser}$
 $u = 2 \cdot \pi \cdot \text{Radius}$
 $\pi = u / 2 \cdot r$
 $d = 2 \cdot r$



$\pi = 3.141593$
 $u = 2440.074951$
 $r = 388.349976$
 $d = 776.699951$

Abgerollter Kreis

fps 57.681213, animating, manual

$2 \cdot \pi \cdot r$ Ausgerollt
 $u = \text{umfang}, r = \text{radius}, d = \text{durchmesser}$
 $u = 2 \cdot \pi \cdot \text{Radius}$
 $\pi = u / 2 \cdot r$
 $d = 2 \cdot r$

$\pi = 3.141593$
 $u = 2440.074951$
 $r = 388.349976$
 $d = 776.699951$

Implementation

- Animation
 - Abtragen vom Kreisumfang und Aufragen zur Linie pro Animations-Frame.
- Male einen Animations-Frame mit abgerolltem Kreisabschnitt
 - Eingabewerte sind Mittelpunkt, Radius, Anfangs- und Endwinkel
 - Zeichnen eines Kreissegments, wobei der vom vollen Kreis fehlende Umfang am Boden als Linie dargestellt wird.
 - Aufruf erfolgt innerhalb einer Animationsschleife mit wachsendem off_pct Wert, so das die Animation eines abrollenden Kreises entsteht. Die abgerollten Kreisanteile, d.h. der Umfang, wird als Linie am Boden gezeichnet.
 - Hier wird das Ergebnis U/d farblich nach ganzen Zahlen getrennt dargestellt, d.h. π .

```
void draw_circle_unroll(const point_t& pm, float r, const float thickness, float off_pct_in) {  
    const float off_pct = min(1.0f, off_pct_in); // Abzurollender Teil vom Umfang in Prozent  
    const float u_unit = 2.0f * M_PI; // Umfang Einheitskreis  
    const float off = (r * u_unit) * off_pct; // Abzurollender Teil vom Kreisumfang  
  
    // Male den Umfang als Linie,  
    // wobei pro Durchmesser eine eigene Farbe gewählt wird, d.h. U/d visualisiert.  
    point_t p0 = point_t(pm.x, pm.y - r);  
    int count = 0;  
    for(float off_left=off; off_left > 0; off_left -= r * 2, ++count) {  
        pixel::set_pixel_color(unroll_colors[count]); // Farbe pro Durchmesser  
        float off_done = off - off_left; // Schon gemalter Anteil  
        float off_now = min(off_left, r * 2); // Aktuelle Position  
        point_t tl = point_t(p0).add(off_done, 0); // top_left für Rechteck
```

```

    rect_t(tl, off_now, thickness).draw(true);
}
// Male Kreissegment
const float ul_a = u_unit * ( 1 - off_pct ); // Uebrigbleibender Winkel
pixel::set_pixel_color(0, 0, 0, 255);
float w = pixel::adeg_to_rad(270); // Anfangswinkel vom Kreissegment
// w + ul_a = Endwinkel vom Kreissegment
draw_circle_seg(point_t(pm.x + off, pm.y), r, thickness, w, w + ul_a);
}

```

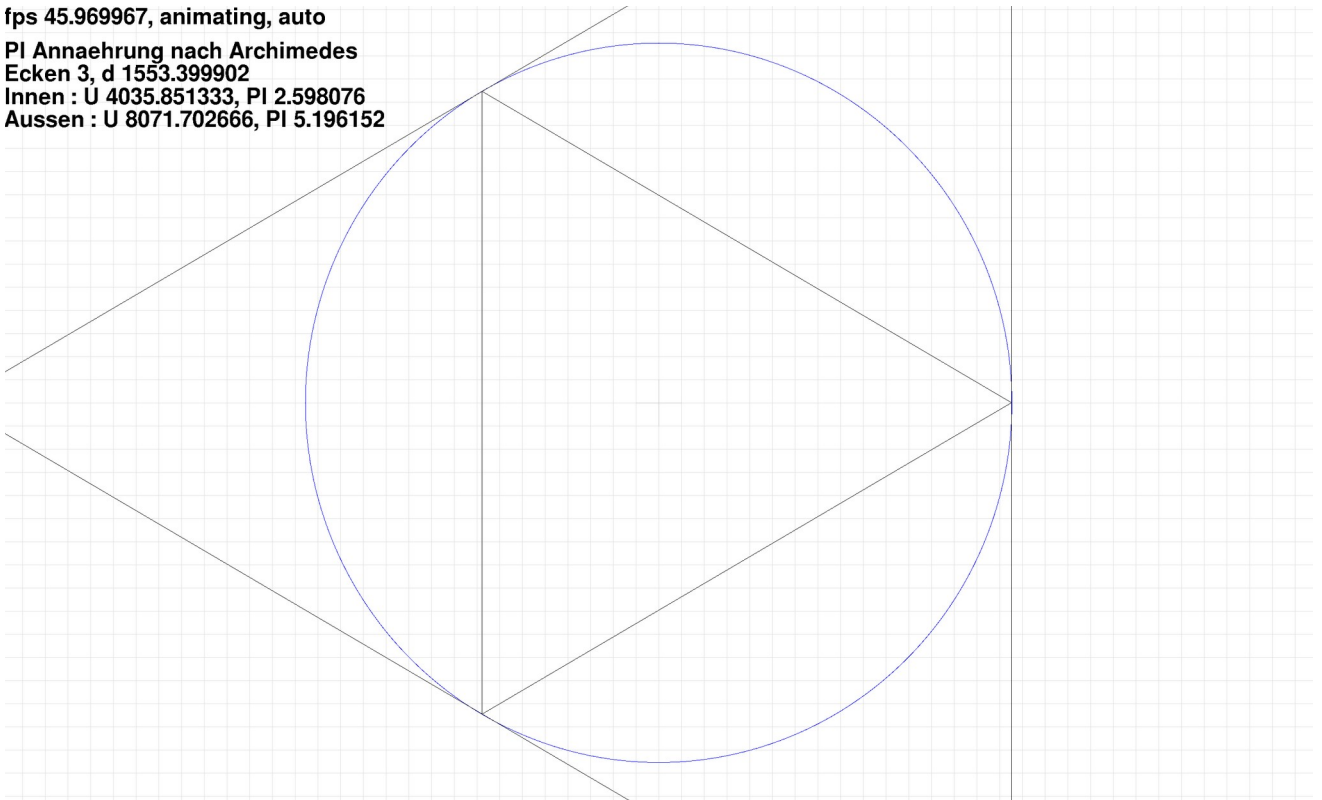
Kreis-Näherung

In der Kreis-Näherung geht es darum dem Kreisumfang (U) mit Durchmesser (d) nahe zu kommen, mit $\pi = U/d$. Dabei nimmt man n Ecken mit gleichem Winkelabstand $2\pi / n$ und Abstand zum Kreismittelpunkt $r=d/2$, damit das n-Eck sich genau in oder um dem Kreis befindet. Dann errechnet man die Kantenlänge (l) zwischen 2 beliebigen Ecken, so das der angenäherte Kreisumfang $n*l$ ergibt.

Ergebnis

3 Ecken

fps 45.969967, animating, auto
 PI Annaehrung nach Archimedes
 Ecken 3, d 1553.399902
 Innen : U 4035.851333, PI 2.598076
 Aussen : U 8071.702666, PI 5.196152



9 Ecken

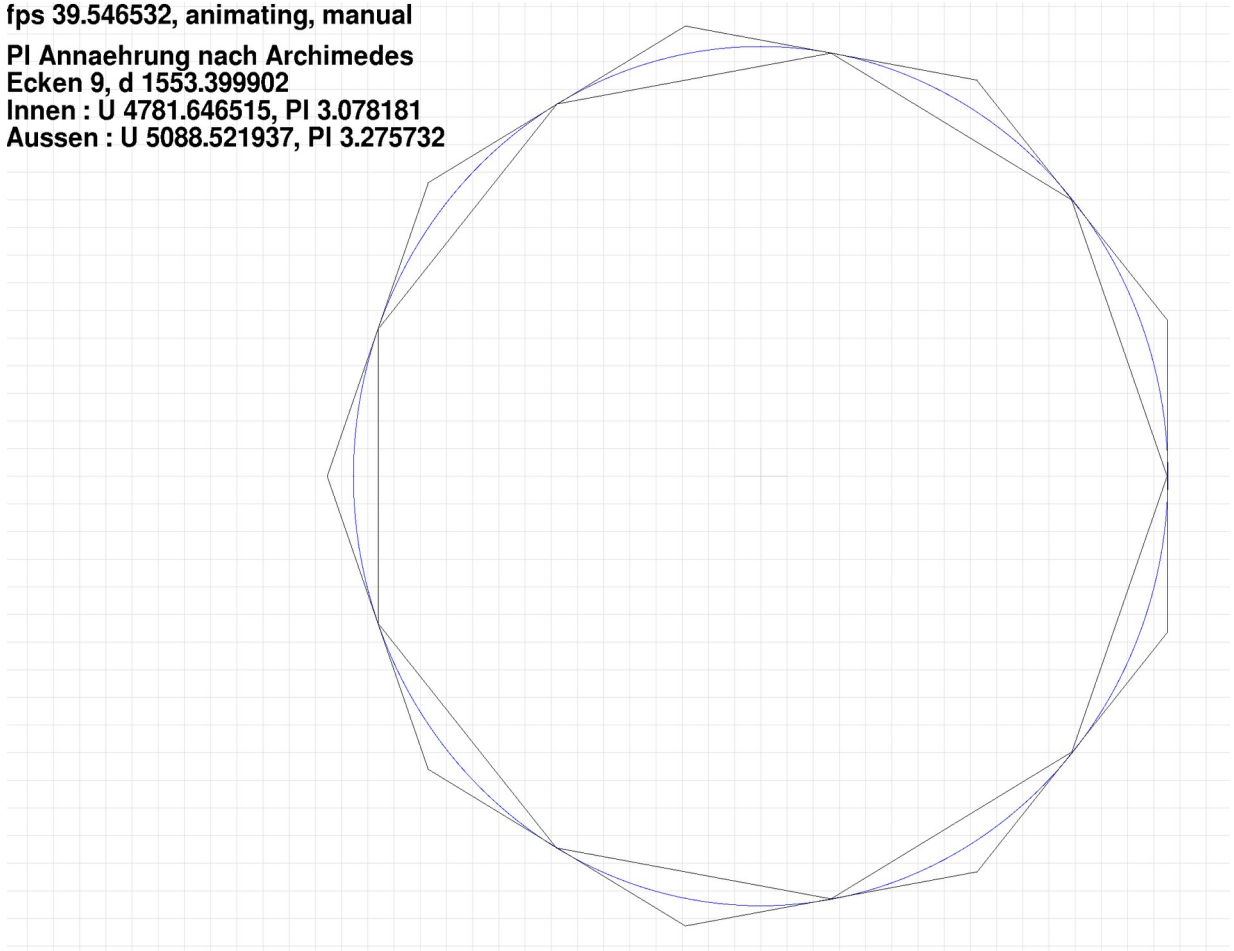
fps 39.546532, animating, manual

PI Annäherung nach Archimedes

Ecken 9, d 1553.399902

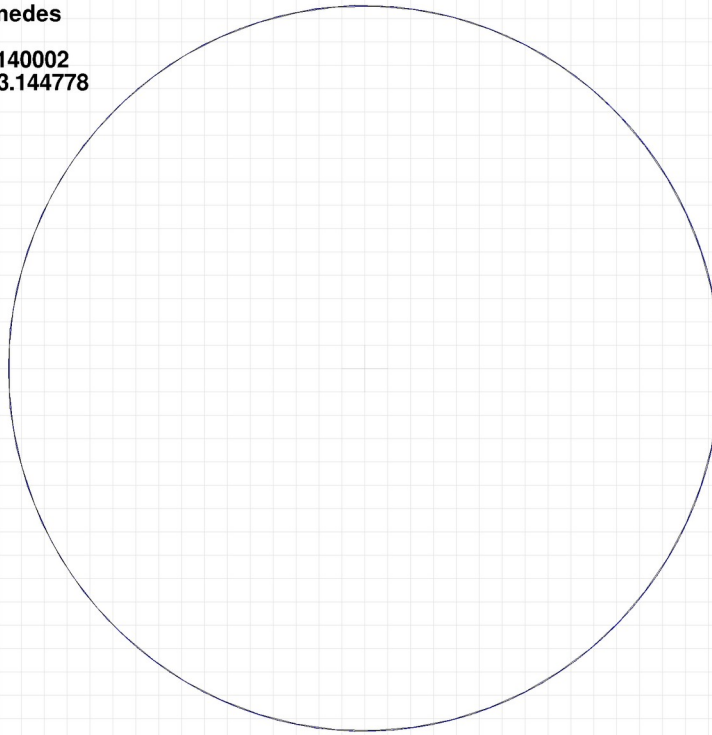
Innen : U 4781.646515, PI 3.078181

Aussen : U 5088.521937, PI 3.275732



57 Ecken

fps 39.452919, animating, manual
PI Annäherung nach Archimedes
Ecken 57, d 1553.399902
Innen : U 4877.679329, PI 3.140002
Aussen : U 4885.097269, PI 3.144778



Implementation

- Animation
 - Näherungsschritte: 3 bis 128 Ecken, 1 wird pro Schritt dazu addiert
 - Eingabewerte sind Mittelpunkt, Radius und die Anzahl der Ecken welche auf dem Kreisumfang liegen und somit den vollen Kreis 2π gleichmäßig aufteilen.
- Innen n-Eck (CircumferenceInner)
 - Zeichnen eines n-Ecks das genau in dem Kreis passt wo der Radius r und der Mittelpunkt pm ist.
 - Damit kommt man immer näher an den Kreisumfang und an π

```
float draw_circumferenceInner(const point_t& pm, float r, int n){
    const float alpha = 2 * M_PI / n; // Den Mittelpunkt verbindet man mit 2 benachbarten
    // so bildet sich ein Dreieck und der Winkel aus dem Mittelpunkt ist alpha
    float res;
    point_t p0 = pm + vec_t::from_length_angle(r, 2 * M_PI - alpha); // 1. Anfangspunkt
    {
        point_t p1 = pm + vec_t::from_length_angle(r, 0); // 2. Anfangspunkt
        res = n * (p0 - p1).length(); // res ist der Umfang von dem Multi-Eck
    }
    for(float i = 0.0f; i < 2 * M_PI; i += alpha){ // i = jetzige Winkel wo wir grade malen
        point_t p1 = pm + vec_t::from_length_angle(r, i);
        lineseg_t::draw(p0, p1); // Eine Linie vom Multi-Eck
        p0 = p1;
    }
    return res;
}
```

- Die Funktion „point_t::from_length_angle“ liefert ein Punkt von dem Sinus und dem Cosinus Wert zurück. Die X-Achse ist der Cosinus Wert und die Y-Achse ist der Sinus Wert. Die Funktion „length“ ist eine Funktion der die Vektorlänge festgelegt.
- Eine Schleife wird über alle Ecken abgelaufen. Die n Ecken haben einen gleichen Winkel Abstand von $2 * \pi / n$. Innerhalb der Schleife zeichnen wir eine Linie vom vorangegangenen Punkt zum aktuellen Punkt. Für den nächsten Schleifendurchlauf wird der aktuelle Punkt zum vorangegangenen Punkt. Außerdem wird der angenäherte Kreisumfang $n * l$ berechnet und die Differenz zum eigentlichen Umfang als auch π angezeigt.
- Aussen n-Eck (CircumferenceOutter)
 - Näherungsschritte: 3 bis 128 Ecken, 1 wird pro Schritt dazu addiert
 - Eingabewerte sind Mittelpunkt, Radius, Anzahl der Ecken

```
float draw_circumferenceOutter(const point_t& pm, float r, int n){
    const float alpha = 2 * M_PI / n; // Den Mittelpunkt verbindet man mit 2 benachbarten
    Ecken,
    // so bildet sich ein Dreieck und der Winkel aus dem Mittelpunkt ist alpha
    float res, l_2, l;
    {
        l = tan(alpha / 2) * 2 * r; // Seitenlaenge
        l_2 = l / 2; // Halbe Seitenlaenge
        res = n * l; // Umfang von dem Multi-Eck
    }
    for(float i = 0.0f; i < 2 * M_PI; i += alpha){ // i = jetzige Winkel wo wir grade malen
        point_t p1 = pm + point_t::from_length_angle(r, i);
        point_t pla = p1 + vec_t::from_length_angle(l_2, i + M_PI_2);
        point_t plb = p1 + vec_t::from_length_angle(l_2, i - M_PI_2);
        lineseg_t::draw(pla, plb); // Eine Linie vom Multi-Eck
    }
    return res;
}
```

- Die Funktion „tan“ berechnet den Tangens Wert. Die Funktion „point_t::from_length_angle“ hatte ich bereits erklärt.
- Es wird zuerst die Länge und der Winkel zwischen den Punkten ausgerechnet. Dann geht eine Schleife von 0° bis 360° . Im Schleifenkörper wird ein Punkt ausgerechnet und mit dem vorherigen Punkt verbunden.

Trigonometrische Verhältnisse

Eine Visualisierung von Sinus und Cosinus im 2π Einheitskreis.

$$r^2 = x^2 + y^2$$

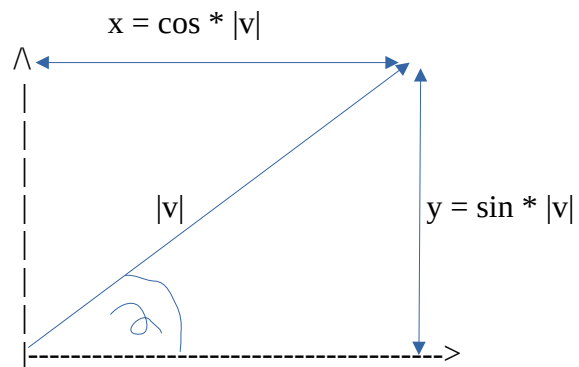
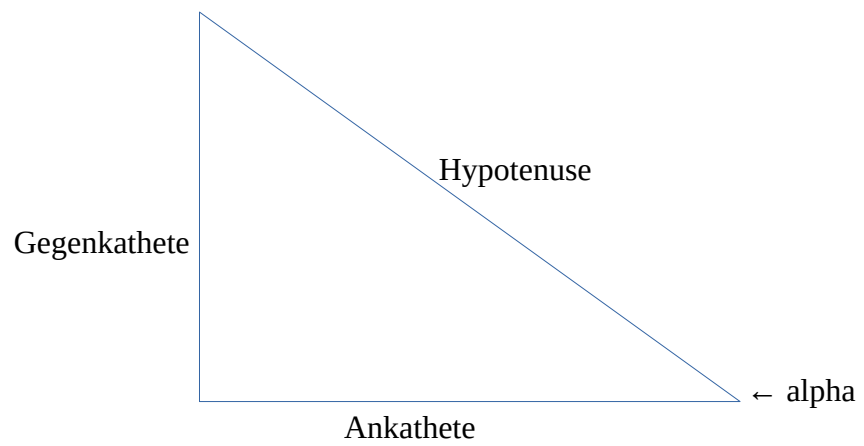
Cosinus(cos) = Ankathete durch Hypotenuse

Sinus(sin) = Gegenkathete durch Hypotenuse

Tangens(tan) = Gegenkathete durch Ankathete

$$\sin = y / r$$

$$\cos = x / r$$



$$|v| = r$$

$$r^2 = (\cos * r)^2 + (\sin * r)^2$$

$$r^2 = r^2 * (\cos^2 + \sin^2)$$

$$1 = \cos^2 + \sin^2$$

$$|v| = 1$$

$$x = \cos * |v|$$

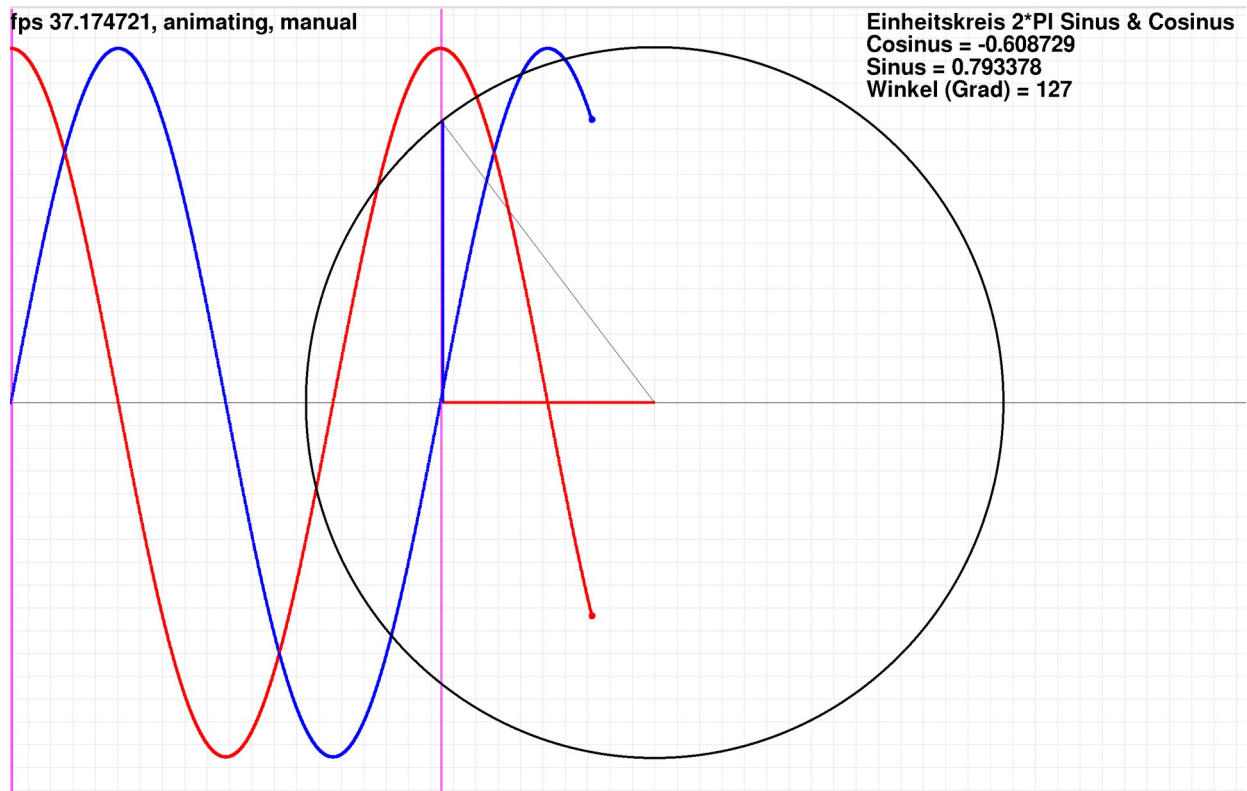
$$x = \cos$$

$$y = \sin * |v|$$

$$y = \sin$$

Ergebnis

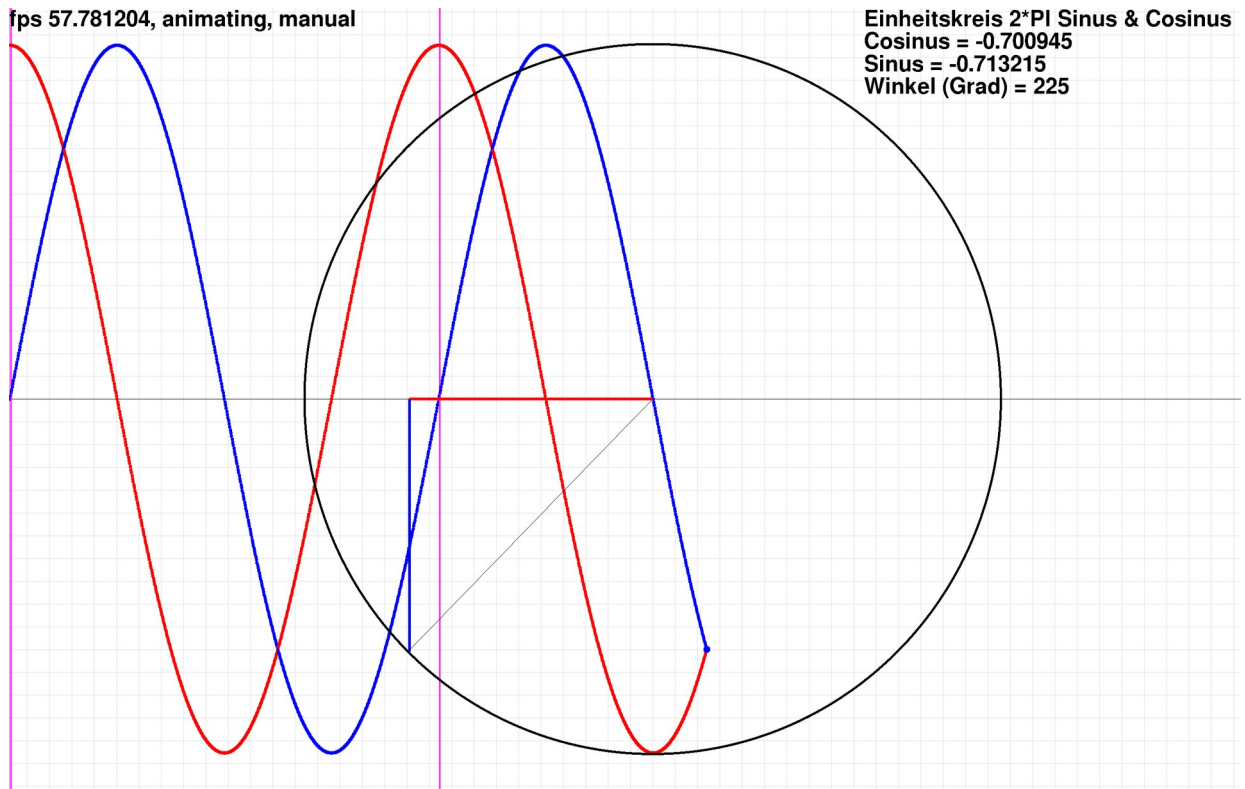
Winkel = 127 Grad



Winkel = 225 Grad

fps 57.781204, animating, manual

Einheitskreis 2*PI Sinus & Cosinus
Cosinus = -0.700945
Sinus = -0.713215
Winkel (Grad) = 225



Implementation: Sinus & Cosinus im Einheitskreis

- Ein Punkt auf dem Umfang wird durch seinen Winkel zur X-Achse übergeben (alpha).
- Gemalt werden:
 - Kreis

- Eine Linie vom Punkt zum Mittelpunkt (Radius)
- Eine Linie vom Punkt zur X-Achse (Sinus)
- Eine Linie vom Kreismittelpunkt zum Schnittpunkt mit o.g. Sinus-Linie

```
void draw_sin_cos(const point_t& pm, const float r, const float alpha, const float thickness)
{
    pixel::set_pixel_color(0 /* r */, 0 /* g */, 0 /* b */, 255 /* a */);
    draw_circle(pm, r, thickness, circle_t::line);

    const float x1 = cos(alpha) * r; // Cosinus Laenge
    const float y1 = sin(alpha) * r; // Sinus Laenge
    const point_t p0(pm.x + x1, pm.y);
    point_t p1(pm.x + x1, pm.y + y1);
    {
        // Cosinus
        point_t center2 = pm;
        pixel::set_pixel_color(255 /* r */, 0 /* g */, 0 /* b */, 255 /* a */);
        // lineseg_t::draw(pm, p0); // cosinus
        rect_t(center2.add(0, thickness / 2), x1, thickness).draw(true);
    }
    {
        // Sinus
        pixel::set_pixel_color(0 /* r */, 0 /* g */, 255 /* b */, 255 /* a */);
        // lineseg_t::draw(p0, p1); // sinus
        rect_t(p1.add(-thickness / 2, 0), thickness, y1).draw(true);
    }
    {
        // Radius
        pixel::set_pixel_color(0 /* r */, 0 /* g */, 0 /* b */, 255 /* a */);
        lineseg_t::draw(p1, pm); // radius
    }
}
```

Implementation Sinus & Cosinus Graph

- Die Funktion „cart_coord.min_x“ berechnet die kleinste X-Koordinate die man noch im Fenster sehen kann. Die Funktion „cart_coord.max_x“ berechnet die größte X-Koordinate die man noch im Fenster sehen kann. Die Funktion „cart_coord.width“ berechnet die Gesamtbreite vom Fenster. Die Funktion „set“ setzt ein Punkt auf die Koordinate die übergeben wurde. Die Funktion „get_fract“ berechnet die Zahl hinter dem Komma.
- Es geht erst eine Schleife von 0° bis zum Max. Winkel. In der Schleife werden die Pixelpositionen von dem Pixeln die gleich gemalt werden sollen und die Pixelfarbe festgelegt. Danach wird der Pixel gemalt. In der Schleife wird geguckt ob schon ein Kreis fertig ist oder ob der ganze Bildschirm schon voll von den ganzen Kurven ist. Wenn ein Kreis fertig ist dann wird an der Stelle wo grade ein Kreis fertig ist eine Linie von unten bis nach oben gemalt. Wenn der ganze Bildschirm voll ist dann soll er wider von vorne anfangen.

Diskussion

In diesem Projekt wusste ich zuerst nicht wie ich die Sinus und Cosinus Kurve hinbekomme. Aber dann hatte mein Vater eine Idee bekommen das ich zur X-Achse einfach den Sinus und Cosinus Wert dazu addiere.

Bei der n-Eck Kreis-Näherung hatte ich zuerst mit Linien und Vektoren die Kanten herausgefunden. Nach einer Diskussion mit meiner Mutter hat Sie die jetzt angewendete Tangens Lösung mir gezeigt.

Quellenangabe

- Grundlagenwissen der Mathematik meines Vaters, Sven Göthel, und meiner Mutter, Qun Göthel.
- Grundlagenwissen Programmiersprachen / Informatik meines Vaters, Sven Göthel.