Kette mit gleich langen Gliedern

Rutel

11. November 2021

1 Einführung

In einigen Computergraphik-Modellen wünscht man sich eine Kette, die an zwei Punkten aufgehängt ist. Wenn diese Kette auch noch kurzfristig modifizierbar sein soll, helfen die meisten Programmpakete nicht weiter, da muß man selber ran. Das habe ich getan.

Im folgenden geht es mir darum, die Rechnungen, die ich dafür angestellt habe, nicht zu vergessen. Damit es auch für weniger mathematisch orientierte Leute noch ungefähr genießbar bleibt, werden die Rechnungen in die Anhänge verbannt. Als Bemerkung vorweg: Ketten sind FPS-Fresser. Ein einzelnes Kettenglied mit Volumen ist selbst unter größten Qualen nicht unter 24 Tris zu haben. Mit viereckigem Draht und und einer sechseckigen Grundform sind es 48 Tris, und das schmerzt noch, wenn die Glieder etwas größer sind … und wenn sie kleiner sind, braucht man mehr Kettenglieder. Bei einer ordentlichen Kette ist man also schnell bei tausenden Tris. Wer sich mit flachen Kettengliedern zufrieden gibt, kommt mit zwei Tris pro Kettenglied aus, wenn Transparenz ausgenutzt wird.

2 Kette modellieren

Die Wikipedia spuckt für die Kettenlinie die Formel

$$y(x) = a \cdot \cosh\left(\frac{x - x_0}{a}\right) + y_0$$

aus, mit einer Herleitung, die einige Leute sogar verstehen – die meisten Physikstudenten werden damit gequält (hoffentlich noch). x_0 gibt die Lage des tiefsten Punktes der Kette an, $y_0 + a$ dessen vertikale Position und a ist der Radius des Krümmungskreises im tiefsten Punkt. Da die Kette nach unten durchhängt und deshalb die Kurve von links nach rechts gesehen eine Linkskurve beschreibt, ist a auf jeden Fall positiv.

Wir wollen Anfangs- und Endpunkt (P_1 und P_2) sowie die Länge der L Kette vorgeben. Letzteres, weil die Kettenglieder eine feste Länge haben und ordentlich ineinander hängen sollen. Derjenige, dem das egal ist – oder wenn es um ein Seil oder einen Draht geht –, kann es sich wahrscheinlich einfacher machen, den Krümmungsradius a vorgeben und die entstehenden Gleichungen vollständig analytisch lösen.

Die Betrachtung soll zunächst innerhalb der Ebene verlaufen, die durch die Punkte P_1 und P_2 sowie die Senkrechte im Modell vorgegeben ist. Das läßt sich über eine Koordinatentransformation einfach ins Dreidimensionale übertragen. Der horizontale Abstand der Punkte sei ℓ, der vertikale h, wobei wir $\ell > 0$ voraussetzen. Wir setzen $P_1 = (0/0)$ in den Ursprung; dann hat P_2 die Koordinaten (ℓ/h).

Die Kurve schreiben wir zunächst als Weg in der Ebene:

$$c(t) = \left(t, a \cdot \cosh\left(\frac{t - x_0}{a}\right) + y_0\right) \tag{1}$$

Diesen Weg müssen wir nun nach der Bogenlänge parametrisieren. Das ermöglicht uns hinterher, die Kettenglieder in gleichen Abständen entlang der Kettenlinie zu verteilen; bei der alten Parametrisierung – oder wenn man sich um die Parametrisierung gar nicht kümmert – sind bei gleichen Abständen der Parameter t die Punkte um so weiter auseinander, je steiler die Kurve ist. In der Parametrisierung nach der Bogenlänge mit dem Parameter s ist der Abstand zweier Punkte auf der Kurve gleich dem Abstand der Parameter. Die Parametrisierung nach der Bogenlänge nimmt eine besonders einfache Form an, wenn $c(s = 0)$ der Scheitel der Kettenlinie ist. Diese neue Parametrisierung ist:

$$c(s) = \left(a \ln\left[\frac{1}{a}\left(s + \sqrt{s^2 + a^2}\right)\right] + x_0, \sqrt{s^2 + a^2} + y_0\right) \tag{2}$$

Die Berechnungen dazu stehen im Anhang A.

3 Parameter

In der parametrisierten Form (2) stehen drei Parameter, die aus der Länge der Kette und den Endpunkten zu berechnen sind; ein vierter Parameter ist implizit hinzugekommen: Nämlich das Argument s_0, bei dem die Kurve durch den Punkt P_1 verläuft. a ist hierbei das Sorgenkind, weil sich die Gleichung nicht analytisch lösen läßt. Weiter unten mehr dazu. Die Gleichungen werden in Anhang B hergeleitet und sind

$$s_0 = \frac{h}{e^{\frac{\ell}{a}} - 1} - \frac{L - h}{2} \tag{3}$$

$$x_0 = -a \operatorname{arsinh}\left(\frac{s_0}{a}\right) \tag{4}$$

$$y_0 = -\sqrt{s_0^2 + a^2} \tag{5}$$

Für a läßt sich eine Funktion mit Ableitung angeben, aus deren Nullstelle größer 0 es sich bestimmt:

$$f(\alpha) = -\lambda \operatorname{arsinh}(\alpha) + \alpha \tag{6}$$

$$f'(\alpha) = -\frac{\lambda}{\sqrt{\alpha^2 + 1}} + 1 \tag{7}$$

Die Werte α und λ stehen zu den eigentlichen Parametern in folgender Beziehung:

$$\alpha = \frac{\sqrt{L^2 - h^2}}{2a}$$

$$\lambda = \frac{\sqrt{L^2 - h^2}}{\ell}$$

Die Nullstelle läßt sich mittels des Newton-Verfahrens bestimmen. Als Startwert ist ein Wert $\alpha_0 = \lambda$ eine mögliche Wahl. Der Nachweis der Konvergenz in Anhang C.

4 Gebrauch

In diesem Abschnitt wird erklärt, wie man die bisher entwickelten Formeln unter lua benutzt, eine Kette aufzubauen.

4.1 Vor data()

Die folgenden Zeilen werden alle vor der Definition der data()-Funktion untergebracht. Sie stellen einige Helfer-Funktionen zur Verfügung.

Ich hasse es, mathematische Funktionen mit einem Vorsatz zu versehen. Deshalb wird das math-Paket in den Standard-Namespace geholt.

```
function openpackage (ns)
    for n,v in pairs(ns) do _G[n] = v end
end

openpackage (math)
```

Der Areasinus hyperbolicus existiert in lua nicht, also wird er definiert:

```
function arsinh (x)
    return log(x+sqrt(x^2+1))
end
```

Als nächstes brauchen wir die Funktionen, die die x- und y-Koordinaten der Kette berechnen. Die Ankerpunkte findet man an den Argumenten $s = 0$ und $s = L$. Da wir die Kurvenparameter a, $x0$ und $y0$ noch nicht kennen, werden sie hier erst einmal als Argumente mitgeliefert. Sie können erst innerhalb der data()-Funktion berechnet werden, damit der Nutzer unterschiedliche Kettenlängen zur Verfügung hat.

```
function catenX (s, a, x0, y0)
    return a*log (s/a+sqrt((s/a)^2+1)) + x0
end

function catenY (s, a, x0, y0)
    return a*sqrt((s/a)^2 + 1) + y0
end
```

Die nächsten Zeilen stellen die Funktion f aus (6) und ihre Ableitung dar, sowie eine Implementation der Newton-Iteration für diese Funktion.

```
function f (xi, lambda)
    return -lambda*arsinh(xi) + xi
end

function fdash (xi, lambda)
    return -lambda/sqrt(xi^2+1) + 1
end

function iterate_newton (xi_in, lambda)
    local xi_new = xi_in
```

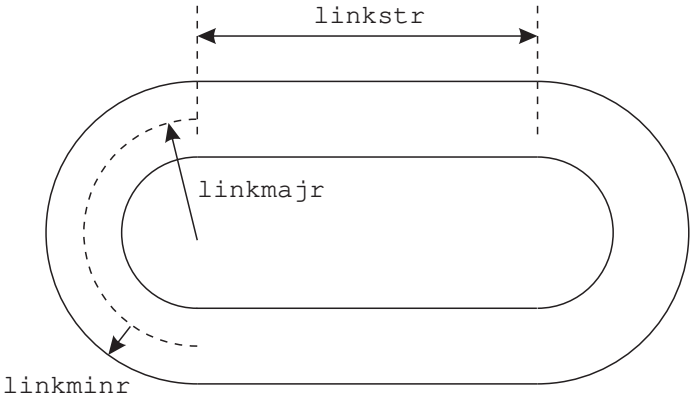

Abbildung 1: Die charakteristischen Längen eines Kettengliedes

```
    local xi_old = 0
    local i = 0
    repeat
        xi_old = xi_new
        xi_new = xi_old - f (xi_old, lambda) / fdash (xi_old, lambda)
        i = i+1
    until (abs(xi_new/xi_old-1) < 1E-6)
    return xi_new
end
```

4.2 Innerhalb updateFn()

Ein Kettenglied besteht aus einem aufgeschnittenen Torus mit Radien `linkmajr` und `linkminr`, deren Hälften durch Zylinder der Länge `linkstr` verbunden sind (Abb. 1).

Als Vorbereitung, die ich nicht gezeigt habe, wurden zwei fixe Kettenglieder so gesetzt, daß sie die Befestigung der Kette darstellen. Sie liegen beide horizontal. Nun werden die Startpunkte `p1` und `p2` so gesetzt, daß sie im Mittelpunkt desjenigen Halbtorus dieser Kettenglieder sind, an dem die Kette eingehängt wird. Die genaue Rechnung hierzu hängt vom umgebenden Modell ab.

Der Einheitsvektor `xdir` zeigt horizontal von `p1` in Richtung `p2` und `zdir` nach oben, so daß die beiden Vektoren das zweidimensionale Koordinatensystem aufspannen, in denen die beiden Punkte liegen. `zangle` wird benötigt, um hinterher die Kettenglieder in die richtige Position zu drehen.

```
updateFn = function(params)
    ...
    p1 = ...
    p2 = ...
    local chdir = vec3.new (p2.x-p1.x, p2.y-p1.y, 0)
    local xdir = vec3.normalize (chdir)
    local zdir = vec3.new (0, 0, 1)
    local zangle = -atan (xdir.y, xdir.x)
```

Die beiden Punkte `p1` und `p2` liegen aber nicht auf der Kettenlinie selber; wir müssen sie

deswegen verschieben, und zwar in den Schnittpunkt der Mittellinien der Tori mit der neu gefundenen Ebene:

```
p1 = vec3.add (p1, vec3.mul(linkmajr, xdir) )
p2 = vec3.add (p2, vec3.mul(-linkmajr, xdir) )
```

Wir bestimmen nun die Länge der Kette. `linkdist` ist der Abstand zweier Kettenglieder, l und h die horizontalen und vertikalen Abstände der beiden (neuen) Aufhängepunkte. Die Kette wird (hier als konstanter Wert, kann aber geändert werdne) um 1 m über die unbedingt nötige Länge einer gespannten Kette verlängert und dann so weit gekürzt, daß sie eine gerade Anzahl Kettenglieder lang ist. Die gerade Anzahl macht es absichtlich etwas komplizierter. Warum, steht weiter unten.

```
local linkdist = linkstr + 2*linkmajr - 2*linkminr
local l = sqrt ( (p2.x-p1.x)^2 + (p2.y-p1.y)^2  )
local h = p2.z-p1.z
local L = sqrt (l*l + h*h) + 1.0
L = floor (L/linkdist/2.0)*linkdist*2 - 2*linkminr
```

Jetzt können die Parameter der Kettenlinie ausgerechnet werden. Der Ankerpunkt eines Kettengliedes ist in seiner Mitte, so daß `soff` die Verschiebung der Mitte des ersten Kettengliedes entlang der Kettenlinie bezogen auf `s0` ist.

```
local Lh = sqrt (L*L - h*h)
local a = 0.5*Lh/iterate_newton (Lh/l, Lh/l)
local s0 = h/(exp(l/a) - 1) - (L - h)/2
local s1 = s0 + L
local x0 = -a*arsinh(s0/a)
local y0 = -sqrt(s0^2+a^2)
local soff = linkdist/2 - linkminr
```

In der nachfolgenden Schleife werden die Kettenglieder gesetzt. `n` zählt die Kettenglieder. `s` ist die Position auf der nach Bogenlänge parametrisierten Kettenlinie; wir ernten den Lohn der Mühe und haben (fast) gleiche Abstände zwischen den Kettengliedern. `x` und `z` sind die Koordinaten im Koordinatensystem, das wir oben durch `xdir` und `zdir` definiert haben. Die Winkel `xangle` und `yangle` beschreiben die Lage des Kettengliedes. Das Besondere bei `xangle` ist, daß die Kettenglieder von einem zum nächsten um 90° gedreht werden müssen, damit sie senkrecht aufeinander stehen. Dazu kommt noch eine kleine weitere Drehung, die sich hier über die ganze Kette zu 90° aufsummiert. Der Grund hierfür ist, daß an beiden Enden die Glieder (fast) senkrecht auf die verankerten Kettenglieder stehen soll. Das gibt der Kette ein natürlicheres Aussehen und kann in vernünftigen Grenzen modifiziert werden. Dazu hatte ich mich auf eine gerade Anzahl Kettenglieder festgelegt.

```
local n = 0
for s = s0, s1, linkdist do
    local x = catenX (s+soff, a, x0, y0)
    local z = catenY (s+soff, a, x0, y0)
    n = n+1
    local xangle = 3.1415926535/2*(n + (s-s0)/L)
    local yangle = atan (s/sqrt(s^2+a^2), a/sqrt(s^2+a^2))
    local pos = vec3.add (p1, vec3.add (vec3.mul(x,xdir), vec3.mul(z,zdir)) )
```

```
        table.insert(result.models, { id = "asset/ChainLinkA.mdl",
                     transf = transf.rotZYXTransl (vec3.new(zangle, yangle, xangle),
                     pos) })
    end
end      -- updateFn()
```

A Parametrisierung nach Bogenlänge

Die Kurve (1) soll nach der Bogenlänge parametrisiert werden. Dazu berechnen wir die Ableitung nach t:

$$\dot{c}(t) = \left(1, \sinh\left(\frac{t-x_0}{a}\right)\right)$$

und daraus die Norm

$$\|\dot{c}(t)\| = \sqrt{1+\sinh^2\left(\frac{t-x_0}{a}\right)} = \cosh\left(\frac{t-x_0}{a}\right)$$

. Die Umparametrisierung ist

$$s(t) = \int_{x_0}^{t} \cosh\left(\frac{\tau-x_0}{a}\right)\, d\tau = a\sinh\left(\frac{t-x_0}{a}\right)$$

mit der Umkehrung

$$t(s) = a\,\mathrm{arsinh}\left(\frac{s}{a}\right) + x_0$$

. Das in (1) einsetzen ergibt

$$\begin{aligned} c(t) &= \left(a\,\mathrm{arsinh}\left(\frac{s}{a}\right) + x_0, a\cdot\cosh\left(\frac{a\,\mathrm{arsinh}\left(\frac{s}{a}\right)+x_0-x_0}{a}\right) + y_0\right) \\ &= \left(a\,\mathrm{arsinh}\left(\frac{s}{a}\right) + x_0, a\cdot\cosh\left(\mathrm{arsinh}\left(\frac{s}{a}\right)\right) + y_0\right) \\ &= \left(a\ln\left[\frac{s}{a}+\sqrt{\left(\frac{s}{a}\right)^2+1}\right] + x_0, a\cdot\sqrt{\left(\frac{s}{a}\right)^2+1} + y_0\right) \end{aligned}$$

, woraus die behauptete Form (2) folgt.

B Berechnung der Parameter

Die Bedingung zu s_0 ist, daß dort die Kurve (2) durch den Startpunkt verläuft, also $c(s_0) = P_1$. Wegen der Parametrisierung nach der Bogenlänge ist dann klar, daß $c(s_0 + L) = P_2$ ist. Dies ergibt dann die Gleichungen

$$\begin{aligned} c(s_0) &= (0/0) \\ c(s_0+L) &= (\ell/h) \end{aligned}$$

Daraus zieht man die folgenden vier eindimensionalen Gleichungen:

$$0 = a\ln\left[\frac{1}{a}\left(s_0+\sqrt{s_0^2+a^2}\right)\right] + x_0 \tag{8}$$

$$
\begin{aligned}
0 &= \sqrt{s_0^2 + a^2} + y_0 && (9)\\
\ell &= a \ln\left[\frac{1}{a}\left((s_0 + L) + \sqrt{(s_0 + L)^2 + a^2}\right)\right] + x_0 && (10)\\
h &= \sqrt{(s_0 + L)^2 + a^2} + y_0 && (11)
\end{aligned}
$$

Die Verschiebungen x_0 und y_0 sind schnell eliminiert, indem man (8) und (10) sowie (9) und (11) ineinander einsetzt:

$$
\begin{aligned}
\ell/a &= \ln\left[\frac{1}{a}\left((s_0 + L) + \sqrt{(s_0 + L)^2 + a^2}\right)\right] - \ln\left[\frac{1}{a}\left(s_0 + \sqrt{s_0^2 + a^2}\right)\right]\\
&= \ln\left[\frac{(s_0 + L) + \sqrt{(s_0 + L)^2 + a^2}}{s_0 + \sqrt{s_0^2 + a^2}}\right] && (12)\\
h &= \sqrt{(s_0 + L)^2 + a^2} - \sqrt{s_0^2 + a^2} && (13)
\end{aligned}
$$

Gleichung (13) bringen wir in die Form

$$
\sqrt{s_0^2 + a^2} + h = \sqrt{(s_0 + L)^2 + a^2} \qquad (14)
$$

die man nachher auch verwendet, um (12) zu vereinfachen, quadriert und vereinfacht:

$$
\begin{aligned}
s_0^2 + a^2 + 2h\sqrt{s_0^2 + a^2} + h^2 &= s_0^2 + 2Ls_0 + L^2 + a^2\\
2h\sqrt{s_0^2 + a^2} &= 2Ls_0 + L^2 - h^2 && (15)
\end{aligned}
$$

Freundlicherweise hoben sich die in s_0 quadratischen Glieder gegenseitig auf. Dafür ist eigentlich eine ausführliche Dankprozession angebracht. Jetzt nochmal quadrieren, was eine zusätzliche Lösung beschert, die aber nicht stört:

$$
\begin{aligned}
4h^2\left(s_0^2 + a^2\right) &= 4L^2 s_0^2 + (L^2 - h^2)^2 + 4Ls_0(L^2 - h^2)\\
4h^2a^2 &= 4(L^2 - h^2)s_0^2 + (L^2 - h^2)^2 + 4Ls_0(L^2 - h^2)\\
\frac{4h^2a^2}{L^2 - h^2} &= 4s_0^2 + 4Ls_0 + (L^2 - h^2)\\
&= (2s_0 + L)^2 - h^2\\
(2s_0 + L)^2 &= \frac{4h^2a^2}{L^2 - h^2} + h^2\\
2s_0 + L &= \pm h\sqrt{\frac{4a^2}{L^2 - h^2} + 1} && (16)
\end{aligned}
$$

h ist vorzeichenbehaftet, aber das ist mit dem $\pm$ zusammen erst mal egal, daher sind keine Betragsstriche nötig.

Nun zu Gleichung (12). Mittels (14) und anschließend (15) wird sie zu

$$
\begin{aligned}
\ell/a &= \ln\left[\frac{s_0 + L + h + \sqrt{s_0^2 + a^2}}{s_0 + \sqrt{s_0^2 + a^2}}\right]\\
&= \ln\left[1 + \frac{L + h}{s_0 + \sqrt{s_0^2 + a^2}}\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \ln\left[1+\frac{2h(L+h)}{2hs_0+2s_0L+L^2-h^2}\right]\\
&= \ln\left[1+\frac{2h}{2s_0+L-h}\right] \qquad (17)
\end{aligned}
$$

(16) erlaubt nun, s_0 ganz rauszuwerfen:

$$
\begin{aligned}
\ell/a &= \ln\left[1+\frac{2h}{\pm h\sqrt{\frac{4a^2}{L^2-h^2}+1}-h}\right]\\
&= \ln\left[1+\frac{2}{\pm\sqrt{\frac{4a^2}{L^2-h^2}+1}-1}\right]\\
e^{\frac{\ell}{a}}-1 &= \frac{2}{\pm\sqrt{\frac{4a^2}{L^2-h^2}+1}-1}\\
\pm\sqrt{\frac{4a^2}{L^2-h^2}+1}-1 &= \frac{2}{e^{\frac{\ell}{a}}-1}\\
\pm\sqrt{\frac{4a^2}{L^2-h^2}+1} &= \frac{2}{e^{\frac{\ell}{a}}-1}+1\\
\frac{4a^2}{L^2-h^2} &= \left(\frac{2}{e^{\frac{\ell}{a}}-1}+1\right)^2-1\\
\frac{a}{\sqrt{L^2-h^2}} &= \frac{1}{2}\sqrt{\left(\frac{2}{e^{\frac{\ell}{a}}-1}+1\right)^2-1}
\end{aligned}
$$

Bei der letzen Wurzel bestimmt sich das Vorzeichen daraus, daß $a>0$ sein muß und $L^2-h^2<0$ nur bei gerissener Kette erfüllt ist. Man kann weiter umformen:

$$
\begin{aligned}
\frac{2a}{\sqrt{L^2-h^2}} &= \sqrt{\left(\frac{1+e^{\frac{\ell}{a}}}{e^{\frac{\ell}{a}}-1}\right)^2-1}\\
&= \sqrt{\left(\frac{e^{\frac{\ell}{2a}}+e^{-\frac{\ell}{2a}}}{e^{\frac{\ell}{2a}}-e^{-\frac{\ell}{2a}}}\right)^2-1}\\
&= \sqrt{\left(\frac{\cosh\left(\frac{\ell}{2a}\right)}{\sinh\left(\frac{\ell}{2a}\right)}\right)^2-1}\\
&= \frac{1}{\sinh\left(\frac{\ell}{2a}\right)} \qquad (18)
\end{aligned}
$$

Gleichung (18) muß und kann jetzt durch ein Näherungsverfahren gelöst werden. Hierzu ersetzen a durch das dimensionslose $\alpha=\frac{\sqrt{L^2-h^2}}{2a}$ und haben nur noch einen weiteren Parameter $\lambda=\frac{\sqrt{L^2-h^2}}{\ell}$:

$$\alpha=\sinh\left(\frac{\alpha}{\lambda}\right)$$

woraus sofort die Nullstellenbedingung in (6) folgt. Eine schönere Darstellung für s_0 als (16) bekommt man aus (17):

$$\begin{aligned} e^{\ell}a - 1 &= \frac{2h}{2s_0 + L - h} \\ 2s_0 + L - h &= \frac{2h}{e^{\ell}a - 1} \\ s_0 &= \frac{1}{2}\left(\frac{2h}{e^{\frac{\ell}{a}} - 1} - L + h\right) \end{aligned}$$

, was sofort zu (3) führt. x_0 und y_0 bestimmt man zwanglos aus Gleichungen (8) und (9).

C Konvergenz des Newton-Verfahrens

Das Newton-Verfahren ist geeignet zur Bestimmung von α in (6). Hinreichende Konvergenzkriterien sind laut Wikipedia, daß in einem geeigneten Intervall $I = [a, b]$

- f konkav und
- $f(a) < 0$ und $f(b) > 0$ ist.

Die Ableitungen sind

$$\begin{aligned} f'(\alpha) &= -\frac{\lambda}{\sqrt{\alpha^2 + 1}} + 1 \\ f''(\alpha) &= \frac{\lambda\alpha}{\sqrt{\alpha^2 + 1}^3} \end{aligned}$$

Da $\lambda > 1$, ist die Konvexität für alle positiven Werte gegeben.

Es ist

$$\begin{aligned} f(\lambda) &= -\lambda \operatorname{arsinh}(\lambda) + \lambda \\ &= \lambda(1 - \operatorname{arsinh}(\lambda)) \quad < \quad 0 \end{aligned}$$

da $\lambda > 1$. Aufgrund des bekannten asymptotischen Verhaltens des natürlichen Logarithmus wird dieser von der linearen Funktion überholt, so daß es ein b geben muß, für welches $f(b) > 0$ gilt.

f' hat eine Nullstelle in

$$a = \sqrt{\lambda^2 - 1} < \lambda$$

Mit λ als Startwert konvergiert das Newton-Verfahren gegen den gewünschten Punkt. (Quelle des Satzes über Konvergenz: Eine Vorlesung Uni Saarland MFI0708, kap52, Urheber leider nicht im PDF genannt. Leider sagt sie nichts über $f'(x) = 0$)