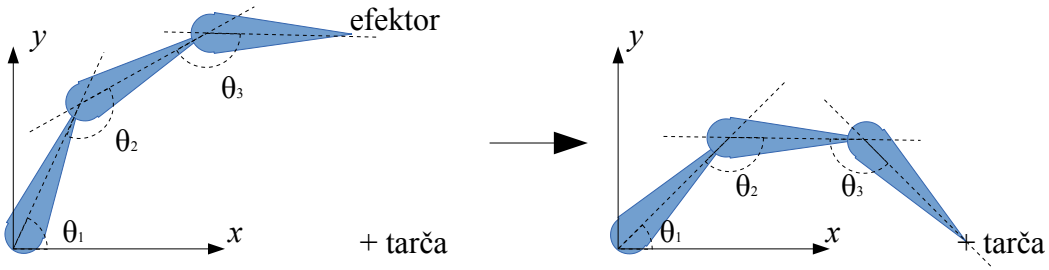
Vaja 5 – Inverzna kinematika

Ustvarite aplikacijo za interaktiven prikaz kinematične verige (lahko 2D), kjer naj zadnji člen kinematične verige (efektor) sledi miškinemu kazalcu (tarči). Ob premikanju miške se mora celotna kinematična veriga ustrezno spreminjati. V programu omogočite tudi nastavljanje števila členov in dolžine vseh členov kinematične verige. Osnovni del naloge je vreden **5 točk**. Uporabite lahko poljuben pristop za optimizacijo kinematične verige, najenostavneje pa je uporabiti v nadaljevanju opisan pristop. Implementacija pristopa mora biti lastna!

Za dodatnih **5 točk** omogočite tudi omejitev (dovoljeno območje gibanja oziroma kot gibanja posameznih členov) in obteževanje posameznih členov (koliko se lahko premikajo glede na ostale člene).

Dodatna pomoč

Kinematična veriga je skupek med seboj povezanih členov (kosti). Primer kinematične verige je prikazan na sliki 1. Iz slike 1 opazimo, da lahko kinematično verigo predstavimo s koti in začetnim položajem (položaj prvega člena). Cilj inverzne kinematike je ta, da kinematično verigo (leva stran slike 1) transformiramo tako, da efektor kinematične verige doseže tarčo (desna stran slike 1). Da to dosežemo, moramo kote θ_1, θ_2 in θ_3 ustrezno spremeniti.



Slika 1: Optimizacija kinematične verige

Za izračun kotov obstaja več metod, ki jih v grobem delimo na analitične in numerične metode. Analitične metode poskušajo kote kinematične verige izračunati na matematičen način z izpeljavo enačb. Ta metoda postane zelo zahtevna pri večjem številu členov kinematične verige. Numerične metode (ali tudi iterativne metode), po drugi strani, iterativno spreminjajo kote kinematične verige, dokler efektor ne doseže tarče. Med numberične metode prištevamo naslednje pristope: Jacobijeva inverzna metoda (Jacobian inversion method), Jacobijeva transpozicijska metoda, optimizacijske metode, ciklični koordinatni spust (Cyclic Coordinate Descend – CCD), genetski algoritmi itd.

Za našo aplikacijo bomo izbrali preprosto gradientno optimizacijsko metodo (https://web.archive.org/web/20160531030510/http://freespace.virgin.net/hugo.elias/models/m_ik2.htm). Ta metoda deluje tako, da kote kinematične verige spreminja iterativno glede na gradiente napake pri spremembah kotov posameznih členov. Opisan pristop lahko v grobem opišemo tudi kot minimizacijo napake oziroma funkcije *err* (izpis 1), ki jo lahko definiramo kot razdaljo med končno točko verige in tarčo.

```
function err (koti, tarča)
    efektor = glede na kote izračunaj lokacijo efektorja (glej izpis 3)
    return razdalja(efektor, tarča)
```

Izpis 1: Izračun napake

Da dodamo omejitve, funkcijo v izpisu 1 spremenimo tako, da vrača večjo napako v primeru prekoračitve omejitev. Člene pa obtežimo tako, da pri togih členih (ki se manj premikajo) napako povečamo ob rotaciji členov. Dober opis postopka implementacije omejitev je podan v članku od Power et al. 1999 [1].

Celoten pristop optimizacije kinematične verige je prikazan v izpisu 2. Vrednost *d* predstavlja dovoljeno napako. V primeru, da je napaka manjša od *d*, se zanka zaključi, saj je efektor kinematične verige dosegel svoj cilj. Lahko se tudi zgodi, da efektor nikoli ne doseže svojega cilja. V tem primeru zanko zaključimo po *max_st_iteracij*.

Koti kinematične verige so shranjeni v polju *koti* (v primeru iz slike 1 koti θ_1, θ_2 in θ_3). Pri implementaciji je potrebno paziti, da so koti členov kinematične verige podani relativno glede na starša.

```
function optimizacija_IK (koti, tarca, d, g, max_st_iteracij)
    // d, g in max_st_iteracij so uporabniško podani parametri
    koti = trenutno stanje verige
    while ( err(koti, tarča) > d and stevilka_iteracije < max_st_iteracij)
        for vsak kot θi na kinematični verigi:
            // izračunamo gradient napake glede na kot θi
            kotiA=koti
            kotiB=koti
            kotiA[i] = kotiA[i] + g
            kotiB[i] = kotiB[i] - g
            gradienti[i] = err( kotiA, tarča )  -  err( kotiB, tarča )
        koti = koti - gradienti
        //gradienti nam povejo, v katero smer moramo kote spremeniti, da zmanjšamo napako
```

Izpis 2: Optimizacija kinematične verige

Ob vsakem premiku miške se kliče funkcija za optimizacijo kinematične verige, zato je potrebno ob vsakem klicu optimizacije kinetične verige izhajati iz trenutne verige. V nasprotnem primeru bi veriga lahko nezvezno spreminjala svoje kote.

Izris

Za prikaz delovanja inverzne kinematike je potrebno implementirati izris, ki je prikazan v izpisu 3. Deluje tako, da se pomika od začetnega člena proti efektorju in pri tem izriše vsaki člen. Pri tem

moramo upoštevati kote in lokacijo predhodnih členov. S proceduro za izris lahko tudi izračunamo položaj efektorja.

```
function izris (koti)
    pos = začetni položaj verige
    accAngle=0 // predstavlja rotacijo i-tega člena verige
    for(float kot : koti)
        oldPos=pos;
        RM=create_rotation_matrix(accAngle+kot, vec3(0, 0, 1));
        pos += (RM * vec4(dolzina_clena_verige, 0, 0, 1)).xy;
        accAngle += kot;
        drawLine(oldPos, pos);
```

Izpis 3: Izris kinematične verige

Opisali smo delovanje inverzne kinematike nad dvodimenzionalnim problemom. Delovanje nad tridimenzionalnim problemom je podobno, vendar ga zaradi kompleksnosti ne bomo obravnavali. Takrat implementacijo ponavadi izvedemo s kvaternioni.

Dodatna literatura

[1] Power, Joanna L., A. J. Bernheim Brush, Przemyslaw Prusinkiewicz, and David H. Salesin. 1999. “Interactive Arrangement of Botanical L-System Models.” In *Proceedings of the 1999 Symposium on Interactive 3D Graphics - SI3D ’99*, 175–82. New York, New York, USA: ACM Press. https://doi.org/10.1145/300523.300548.