Számítógépes Grafika

Klár Gergely
tremere@elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2010/2011. tavaszi félév

Mi az, amit változtatunk?

- Lehet: pozícó, orientáció, szín, normál, BRDF, stb.
- ”Értelme” van: pozíció, orientáció, kamera
- Animációt alkalmazunk:
  - a modellezési transzformációra
  - a kamera transzformációra

Animáció

- Állókép helyett képsorozat
- Objektumok/kamera/világ tulajdonságait változatjuk
- Egy kép $\equiv$ egy idő pillanat
- Képsorozat elég gyors $\rightarrow$ folyamatos mozgást érzékel az emberi szem

Animáció szintézis

- Legyen minden o objetumra, a modellezési trafó: $M_o \in \mathbb{R}^{4\times4}$
- Legyen a kamera trafó: $V \in \mathbb{R}^{4\times4}$
- *Megjegyzés: ha V a View mátrix, akkor csak a kamera pozicióját, orientációját tartalmazza; ha V a View és a Projection együtt, akkor tudjuk vele a látószöget is állítani*
- Legyen mindkettő idő függő!
- $M_o \in \mathbb{R} \rightarrow \mathbb{R}^{4\times4}$
- $V \in \mathbb{R} \rightarrow \mathbb{R}^{4\times4}$

Általános animációs program váza

```
while keep_running:
  t = get_time()
  for o in objects:
    M_o = M_o(t)
  V = V(t)
  render_scene()
```

Valósidejű animációs program váza

```
def update():
  # vagy idle, vagy onFrameMove
  t = get_time()
  for o in objects:
    M_o = M_o(t)
  V = V(t)

def render():
  # vagy display, vagy onFrameRender
  for o in objects:
    render_object(o)
```

Animáció szintézis

- Valósidejű/interaktív:
  - rögtön meg is jelenítjük a képkockákat
  - a számításnak elég gyorsnak kell lennie a folytonosság látszatához
  - a felhasználói eseményekre ragálni kell
  - $\Rightarrow$ olyan részletgazdag lehet a színtér, amit még így meg lehet jeleníteni
  - $\Rightarrow$ inkrementális képszintézist használunk

Animáció szintézis

- Nem valós idejű/*offline*:
  - ”Nem számít”, hogy mennyi ideig tart kiszámítani egy képkockát
  - Elkülönöl a szintézis és a visszajátszás
    - Először elmentjük a képkockákat
    - Aztán majd videóként lehet visszanézni
  - $\Rightarrow$ a felhasználó nem tud belenyúlni az animációba
  - $\Rightarrow$ olyan részletes színteret, és olyan bonyolult algoritmust használunk, amit ki tudunk várni

Offline rendering

```
Δt = 1/FPS
for (t=t_start; t<t_end; t+=Δt):
  for o in objects:
    M_o = M_o(t)
  V = V(t)
  render_scene_to_disk()

start_time = get_time()
for (t=t_start; t<t_end; t+=Δt):
  draw_frame(t)
  while (t-t_start+Δt > get_time()-start_time):
    wait()
```

Valósidejű animáció – rosszul

- Hogyal lehet a legkönnyebben elrontani az animáció számítást?
- Azzal, ha nem vesszük figyelembe, hogy mennyi idő telt el két képkocka között.
- Pl.: A középpontja körül akarjuk forgatni az objektumot állandó szögsebességgel.
  `model = rotation(phi, 0,1,0); phi += phi_step;`
- Az objektum olyan gyorsan fog forogni, amilyen gyakorisággal ez a kód részlet meghívódik.
- Gyorsabb gépen többször, lasabb gépen kevesebbszer.

Valósidejű animáció – jól

- Hogyal lehet a legkönnyebben ezt megelőzni?
- Sose azt tároljuk, hogy mennyivel kell változtatni, hanem, hogy mi a változás *sebessége*.
- Minden számítás előtt kérjük le, hogy mennyi idő telt el az előző képkocka óta, és ezel szorozzuk a sebességet.
- Pl.: `phi += phi_step;` helyett `phi += get_time_since_last_frame() * phi_speed;`
- Gyorsabb gépen kevesebb, lasabb gépen több idő telik el két képkocka között ⇒ gyors gépen kissebbeket lépünk, lassabban nagyobbakat.

Kamera animáció

- A kamera tulajdonságai:
  - szempozició (*eye*),
  - egy pont amire néz (*center*),
  - felfele irányt megadó vektor (*up*),
  - a képernyő/ablak oldal aránya (*aspect*),
  - nyílásszög (*fovy*).
- Ezek mind külön-külön változtathatók az animáció létrehozásához.

Pozíció és orientáció

- Pozícó: *"Hol van az objektum?"*
- Orientáció: *"Hogy áll, merrefele néz az objektum?"*
- Elsősorban ezt a kettőt szerenénk változtatni.
- $M_o(t)$ megadja mindkettőt.
- Normális esetben

$$M_o(t) = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix},$$

- $\vec{p} = (p_x, p_y, p_z)$ a pozíció.
- Az **A** mátrix tartalmazza az orientációt.

Orientációs paraméterek

- Tegyük $\vec{p}$-t és **A**-t is időfüggővé!
- $\vec{p}$ tagjait leírhatjuk külün-külön függvénnyel.
- *Pl.: Valami esik, elég p_y-t változtatni.*
- **A**-t tagjai összefüggenek, és csak az orentáció érdekel belőle minket (nem érdekel: méretezés, nyírás).
- Az orientáció megadható három tengely menti forgatással → három független függvénnyel.

Yaw, pitch, roll

- Egy objektum függőleges- (*yaw*), kereszt- (*pitch*) és hossztengelye (*roll*) menti elfordulásait egyszerre adjuk meg.
- Már találkoztunk vele, 3×3-as mátrixszal megadható, három forgatás szorzata.
- Három tengely menti elfordulás szögével megadható ⇒ megadja az orientációt is.

Képlet animáció

- Egy adott tulajdonság változását egy megfelelő függvénnyel írjuk le.
- Pl: Óra mutatói
  - Nagymutató: $yaw(t) = t/10$
  - Kismutató: $yaw(t) = t/240$
  - Ha t-t mp-ben adjum meg, a forgatásokat pedig fokban.
- Pl: Pattogó labda
  - $p_y(t) = |\sin(\omega t + \theta_0)| \cdot e^{-kt}$

Kulcskocka (*key frame*) animáció

- Egy bonyolult mozgást nehézkes lenne képlettel megadni.
- Inkább adjuk csak bizonyos időközönként, hogy *akkor* mit szeretnénk látni.
- Ezek a kulcskockák.
- Egy tulajdonságot két kulcskocka között *interpolációval* számolunk ki.

Kulcskocka (*key frame*) animáció

- Az interpolációval az objektum egyes paramétereire folytonos görbét illesztünk.
- Az animáció lejátszása/elmentése során a program minden képkockában a hozzá tartozó t értékkel kiértékeli az objektum paraméter-függvényeit.
- Ezekből számítja a transzformációs mátrixokat.
- A mátrixok felhasználásval előállítja a képet.

Lineáris interpoláció

- Legyen a két kulcskockánk időpontja t_0 és t_1.
- Legyen az interpolálandó tulajdonság g.
- Lineráis interpolációval $\forall t \in [t_0, t_1]$-re kapjuk

$$g(t) = \left(1 - \frac{t - t_0}{t_1 - t_0}\right) g(t_0) + \frac{t - t_0}{t_1 - t_0} g(t_1)$$

Esettanulmány

Interpoláció két kulcskocka között

K: Mi a baj a lineáris interpolációval?

V: Ritkán néz ki természetesen. Animáció során a sebesség konstans, előtte, utána nulla.

- Elgurított labda: folyamatosan lassul.
- Zuhanó zongora: folyamatosan gyorsul.
- Rakéta a Földről a Marsra: gyorul, halad, lassít.

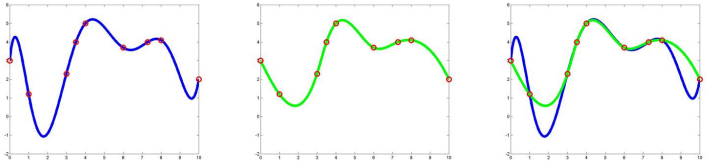
- Ilyen interpolációra használhatunk:
  - Gyök függvényt.
  - Másodfokú függvényt.
  - Logisztikus függvényt/görbét.
- Gyakorlatban: Bézier görbe

Polinom interpoláció

- n kulcspontra fel tudunk írni $n-1$-ed fokú polinomot.
- Interpolációs polinom: minden kulcskockában az előírt értéket veszi fel.
- Együtthatók számíthatók Lagrange intelpolációval.
- A lineáris interpoláció a Lagrange intelpoláció speciális esete $n = 2$-re.

Spline interpoláció

- A polinom interpolációval kapott fv. magas fokszám esetén a szomszédos pontok között ”hullámzik”, így elrontja az animációt.
- Spline interpoláció: használjunk több, egymáshoz kapcsolódó, alacsony fokszámú polinomot az interpolációhoz!



Pálya animáció

- Egy objektum mozgását megadhatjuk a bejárandó pálya megadásával is.
- A pályát egy 3D görbével adjuk meg.
- A model ezen a görbén halad végig.

Orientáció megadása

- Hogyan adjuk meg az objetumunk orentációját?
- Egy *előre*, és egy *felfele* irány egyértelműen meghatározza ezt.
- *Megjegyzés: v.ö. kamera esetén center-eye ill. up vektorok*
- Ha a pályagörbe differenciálható, akkor az megadja a sebességvektort minden időpillanatban.
- A sebességvektor mindig előre fele mutat.

Orientáció megadása

- A *felfele* irány megadására két lehetőségünk is van.
- Ha van egy természetes *felfele*, akkor használjuk azt. (Mindennél, ami nem ből be a kanyarban.)
- Ha ez az irány is változik, akkor ez megegyezik a gyorsulás irányával, azaz a pályagörbe második deriváltjának irányával.

Ívhossz szerinti paraméterezés – motiváció

- Nézzük meg a ez a két görbét!

$$g_1(t) = \begin{bmatrix} t \\ t \end{bmatrix}$$

$$g_2(t) = \begin{bmatrix} \sin t \\ \sin t \end{bmatrix}$$

- Képük ugyan az.
- $g_1(t) \stackrel{?}{=} g_2(t)$? Nem!
- Ha t-vel egyenletesen lépünk, akkor g_1 szerint egyenletes mozgást kapunk, de g_2 szerint hol lassabban, hol gyorsabban haladunk majd a görbén.
- *(De mindkét esetben ugyan azon a pályán!)*

Ívhossz szerinti paraméterezés

- A görbéket úgy kell megadnunk, hogy adott t-szerinti lépéshez mindig azonos ívhosszhoz tartozzon.
- A görbe hossza egy t_0 ponttól t – *ig*:

$$s(t) = \int_{t_0}^{t} |g'(x)| dx$$

- Ekkor g ívhossz szerinti paraméterezése:

$$\bar{g}(r) = g(s^{-1}(r))$$

- Sajnos s ritkán számítható kényelmesen. (Még Bézier görbékre sem! Ld. *püthagoraszi hodográfok*)s
- Numerikus módszereket kell használni, ha általános görbéket meg akarunk engedni.

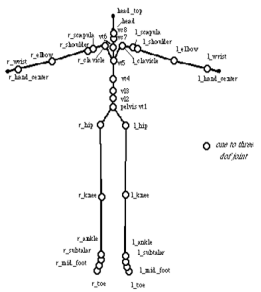
Állandó sebességű görbék

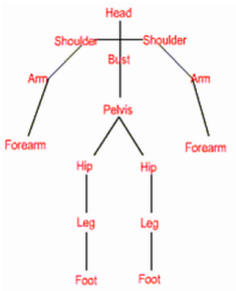
- Hogyan lehetne ellenőrizni, hogy legalább állandó sebességet ad-e meg a görbe?
- $g(t) = (x(t), y(t), z(t))$
- A sebességvektor t pillanatban: $g'(t)$.
- Ha $g'(t)$ nagysága állandó, akkor a sebesség is állandó.
- Azaz $|g'(t)| = \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2}$ értéke független t-től.

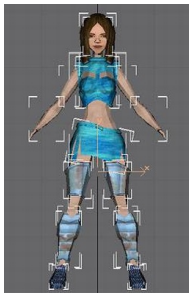
Hierachikus rendszerek

- Színtér gráfoknál már találkoztunk ilyenekkel.
- Egy gyerek objektum mozgását a szülőhöz viszonyítva adjuk meg.
- Gyerekeknek lehetnek újabb gyerekei, s.i.t.
- Hierachikus rendszert – fát – kapunk.

Példa: Emberi test







Kényszerek (*constraints*)

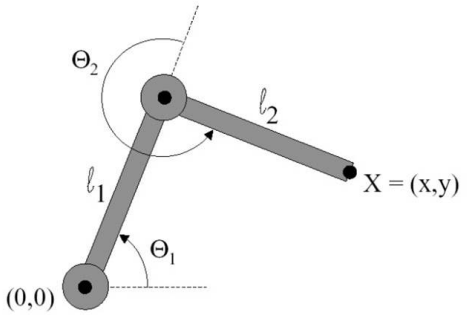
- Nem minden mozgást szeretnénk megengedni a szülőhöz képest.
- Ezeket a megszorításokat írhatjuk le *kényszerekkel*.
- Korlátozhatjuk a szabadságfokokat: pl. könyök csak egy tengely mentén tud forogni, de a csukló kettő
- Vagy a tartományokat: kevesen bírják, ha a fejük 90°-nél többet fordul.

Előrehaladó kinematika

- Végállapotot határozunk meg az állapotváltozók függvényében.
- Szimulációkhoz jól használható.
- Minden elemre megadjuk, a hozzá tartozó transzformációt.
- Ezeket a hierarchiában felülről lefele haladva értékeljük ki.
- Az adott elemhez tartozó transzformáció az összes ős és a saját transzformáció szorzata.

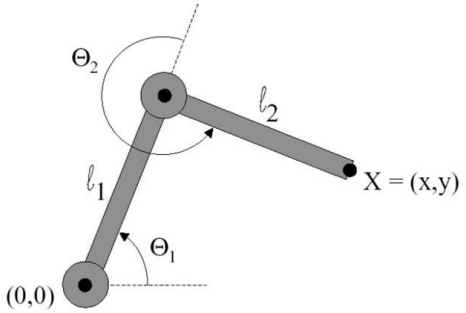
Példa

- Kétszabadságfokú, rotációs csuklókat tartalmazó rendszer.
- A csuklók csak a Z tengely körül fordulnak.



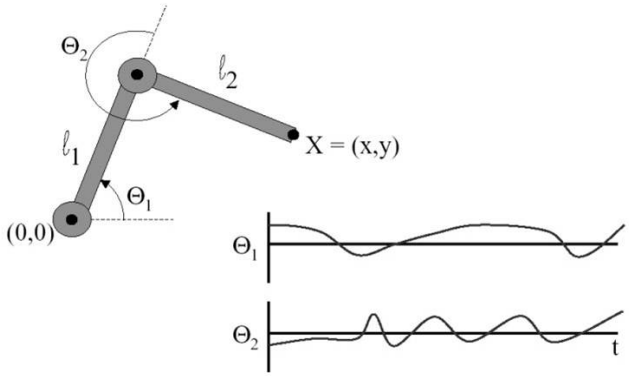
Példa – folyt.

- Állapotváltozók: Θ_1, Θ_2
- A végberendezés (X) pozícióját a gép számolja.
- $X = (l_1 \cos\Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin\Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$



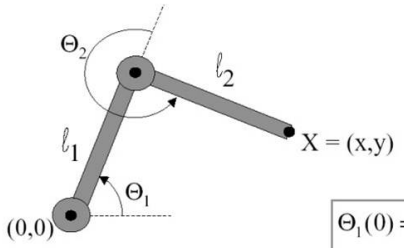
Példa – folyt.

- Az állapotváltozókat megadhatjuk (pl. spline) függvénnyel.



Példa – folyt.

- Az állapotváltozókat megadhatjuk kezdeti értékkel és sebességgel.



$$\Theta_1(0) = 60^\circ \qquad \Theta_2(0) = 250^\circ$$

$$\frac{d\Theta_1}{dt} = 1.2 \qquad \frac{d\Theta_2}{dt} = -0.1$$

Mit nem tud az előrehaladó kinematika?

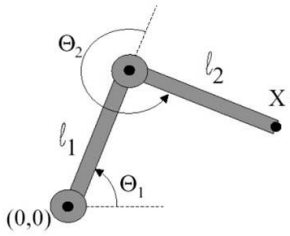
- Az előrehaladó kinematika nem használható, ha a strukturális összefüggés erősen nem lineáris
- Hiába interpolálunk egyenletesen az állapottérben, a végberendezés vadul kalimpálhat a kulcspontok között
- Probélmás esetek:
  - Láb mozgása a talajon
  - Végállapot jó, de menet közben a berendezés részei átmehetnek egymáson.

Inverz kinematika

- Az inverz kinematika a kritikus végberendezés helyzetét interpolálja, majd az állapotváltozók értékét végberendezés interpolált helyzetéből számítja vissza.
- Az inverz kinematika másik neve a cél-orientált animáció.
- *"Ezt szeretném megfogni, hogyan forgassam az izületeimet?"*

Példa

- A végberendezés helyzetéből visszaszámoljuk az állapotváltozók értékét.



$$\Theta_2 = \cos^{-1}\left(\frac{x^2 + x^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

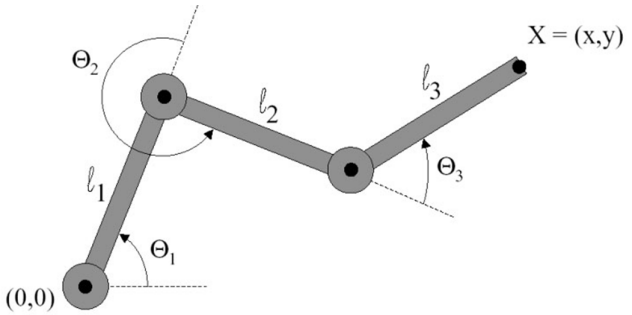
$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2))x + (l_1 + l_2 \cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2 \cos(\Theta_2))x}$$

Problémák

- Nehéz ”természetesnek látszó” mozgást leírni vele.
- Az inverz függvény kiszámítása nem triviális,
- és nem is egyértelmű (redundancia).

Példa

- Egyenletek száma: 2, ismeretlen változók száma: 3 $\Rightarrow$ Végtelen sok megoldás!
- Rendszer DOF $>$ végberendezés DOF
- Az emberi csontváz kb 70 DOF!



Példa

Nem egyértelmű, ill. nem létező megoldás