Kameramozgások

Bordé Sándor

2011. május 11.

Az alábbiakban a Fejlett grafikai algoritmusok gyakorlatra készített projektmunk ánkban, a billiárd játékban megvalósított kameramozgásokat és az azokhoz kapcsolódó algoritmusokat, elméleti hátteret fogom ismertetni.

Mozgatás a tengelyek mentén

Számítógépes grafikában a kamera mozgáskor valójában nem a kamera helyzete változik, hanem a színtér transzformálódik el az ellentétes irányba. Így a tengelyek mentén történő mozgást a megfelelő tengelyek mentén történő eltolással lehet megvalósítani.

A billiárd játékunkban mindhárom tengely mentén lehet haladni az FPS[1] játékokból ismert billentyűkkel: A és D vízszintesen balra-jobbra, W és S előre hátra. Függőleges irányban a T és G gombokkal tudunk fel illetve le menni. A mozgásokra be van állítva egy-egy ellenőrző feltétel, ami megakadályozza, hogy kimenjünk a szobából. Ezeket a feltételeket a szoba dimenziói adják.

Automatikus mozgatás

A programban megvalósítottam a folyamatos mozgatást. Ez valójában egy paraméterezhető függvény, amely paraméterben a kívánt kamerahelyzet koordinátáit várja (x, y és z). A függvény ezután odamozgatja a kamerát. Mindhárom koordináta egyszerre változik, így valójában a légvonalban legrövidebb úton közlekedik. Ha valamelyik koordináta már megegyezik a kívánt értékkel, (pl. függőlegesen már egy vonalban vele), akkor az onnantól kezdve változatlan marad.

A szebb látvány érdekében ez a mozgás időzített: adott időintervallum alatt teszi meg a távot, köbös csillapítással számítva a pillanatnyi sebességet. A köbös csillapítás képlete:[2]

$$v = 2 \cdot t^3 + 2, \text{ ha } t < 1$$
$$v = 2 \cdot (t-2)^2 \cdot t + 2 \text{ , ha } 1 < t < 2$$

A v a pillanatnyi sebesség, t az aktuális időpillanat, a 2 pedig a kezdősebesség illetve a növekmény mértéke is egyben (változtatható).

[1] First Person Shooter

[2] Az alábbi képleten alapul: http://www.gizma.com/easing/

A demo programban a z billentyűre van kötve ez a funkció, ami alapértelmezetten az origóba mozgatja a kamerát.

Forgatás hagyományos módon

A programban az összehasonlítás miatt két féle forgatást is implementáltam. Az egyik a hagyományos forgatás, a másik pedig a kvaterniókkal való forgatás. Hagyományos forgatás esetén egyszerűen csak megváltoztatom az adott tengely körül való elforgatás szögét, majd a színtért ennek megfelelően transzformálom.

Manuális forgatás

A forgatóműveletek elérhetők a billentyűzetről is: a $\leftarrow$ és a $\rightarrow$ gombokkal a függőleges tengely mentén lehet forgatni, a $\uparrow$ és $\downarrow$ billentyűkkel pedig előre-hátra lehet dönteni a kamerát. A harmadik tengely mentén szándékosan nem lehet forgatni.

Időzített forgatás

A forgatás is meg lett valósítva időzítve, hasonlóan a mozgatáshoz. A forgás sebességét itt is a fent bemutatott csillapító függvény számítja ki. A megvalósító függvény három paramétert vár: a forgatás szögét, a forgatás sebességét (adott időegység alatt hány fokkal forduljon) valamint a forgatás tengelyét.

A demo programban az r billentyűre van kötve a funkció, ez 90 fokkal forgatja el a kamerát.

Forgatás kvaterniók használatával

A kamera forgatásának másik módja a kvaterniókkal történő forgatás. Látványban nem különbözik a hagyományos forgatástól, viszont könnyebben és olvashatóbban adható meg a művelet. Hogy a programban használni tudjuk a kvaterniókat, implementáltam egy 3 dimenizós vektor osztályt (*Vektor3*) és egy Kvaternió (*Kvaternio*) osztályt.

(A kvaternió elméleti hátterének leírásától most eltekintek, figyelembe véve, hogy az előadásanyagban illetve az interneten fellelhető számos forrásban le van írva.)

Kvaternió osztály

A Kvaternió osztályom implementálja a kvaternióval kapcsolatos alapműveleteket (konjugálás, normalizálás, két kvaternió szorzata), vektor forgatását kvaternióval, valamint a kvaternió paramétereinek beállítását forgatási tengelyből és forgatási szögből. Lehetőség van a kvaternió mátrixának [3] lekérésére. Ez utóbbit majd a színtér transzformálásánál használjuk fel.

[3] http://en.wikipedia.org/wiki/Quaternion

Színtér transzformációja kvaterniókkal

A színtér kialakításában egy logikai változóval (a demo programban k billentyű lenyomásával váltható) szabályozható, hogy hagyományos vagy kvaterniókkal végzett forgatást szeretnénk használni. A hagyományos forgatásról volt szó az előző részben.

Forgatáskor megállapítjuk, hogy az egyes tengelyek mentén hány fokos forgatásra van szükség, majd ezekből létrehozzuk a két kvaterniót (a tengelyből és a szögből, ahol a tengelyt egy három dimenziós vektor reprezentálja). A két kvaterniót összeszorozzuk, ez jelenti a forgatás egymás utáni végrehajtását. A szorzat kvaternióból mátrixot készítünk. Ez lesz a modellnézeti mátrix, ami a *glMultMatrix()* függvénynek átadva a megfelelő irányba transzformálja a színteret.

Időzített forgatás kvaterniókkal

A hagyományos forgatás esetén a forgatás pillanatnyi sebességét egy csillapítási függvénnyel számítottam ki. Kvaterniókkal ez a művelet egyszerűsödött. Az úgynevezett SLERP[4] műveletet hívtam segítségül a végrehajtáshoz.[5]

Slerp

A spherical linear interpolation (gömbmenti lineáris interpoláció) egy kvaterniótól független geometriai művelet. Az alapelve az az, hogy egy körív minden pontja előáll az ív két végpontjának lineáris kombinációjaként. A Slerp képlete:

$$Slerp(p_0, p_1, t) = \frac{\sin[(1-t)*\Omega]}{\sin\Omega} p_0 + \frac{\sin t\Omega}{sin\Omega} p_1$$

,ahol Ω a forgatás szöge, és $0 \leq t \leq 1$

Forgatáskor létrehozunk két kvaterniót: az egyiket az aktuális szögelfordulásból, másikat pedig az elfordulás mértékével növelt értékből. Tengelynek azt kell választani, amely mentén forgatni szeretnénk (a demo programban a r gomb a függőleges tengely szerint forgat). Ezután az időzítő által meghívott függvényben alkalmazzuk a fent leírt képletet, ami meghatározza a q_m köztes kvaterniót. A kvaternió w koordinátájának koszinuszából kapjuk meg az új elfordulási szöget.

A forgatás sebességét az időpillanatok számával tudjuk szabályozni: minél kisebb a t változó, annál lassabb a forgatás.

[4] Spherical linear interpolation
[5] http://en.wikipedia.org/wiki/Slerp

A Slerp forráskódja a programomból

```
/*Kezdeti es vegkvaterniok letrehozasa*/
Vektor3 z(0.0f, 1.0f, 0.0f);
start = new Kvaternio();
start->kvaternioTengelybolEsSzogbol(z, (zRot));
start->normalizal();
end = new Kvaternio();
end->kvaternioTengelybolEsSzogbol(z, (zRot+withDeg));
end->normalizal();

/*Interpolacio*/
float gt = rotateTime / 50.0;

float cosQTheta = start->kvaternioCosTheta(*end);
float qTheta = std::acos(cosQTheta);
if (qTheta<0.0) qTheta=-qTheta;

float sinQTheta = std::sqrt( 1.0 - cosQTheta * cosQTheta );
float sinTQTheta = std::sin(gt * qTheta) / sinQTheta;
float sinMQTheta = std::sin((1.0 - gt) * qTheta) / sinQTheta;

Kvaternio qm;
qm.setX( start->getX()*sinMQTheta + end->getX()*sinTQTheta );
qm.setY( start->getY()*sinMQTheta + end->getY()*sinTQTheta );
qm.setZ( start->getZ()*sinMQTheta + end->getZ()*sinTQTheta );
qm.setW( start->getW()*sinMQTheta + end->getW()*sinTQTheta );

zRot = radianFokba( 2.0 * std::acos(qm.getW()) );
```

Források, felhasznált információk

- Csillapító függvény - http://www.gizma.com/easing/
- Kvaterniók általános leírása - http://en.wikipedia.org/wiki/Quaternion
- Kvaterniók leírása - http://www.geometrictools.com/Documentation/Quaternions.pdf
- Kvaterniók használata számítógépes grafikában - http://gpwiki.org/index.php/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation
- Slerp - http://en.wikipedia.org/wiki/Slerp
- Kvaterniók, kameramozgások - előadásjegyzet, kurzus jegyzet