SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD

ELASTIČNI MODELI TEMELJENI NA SUSTAVU ČESTICA

Janko Sladović

Zagreb, lipanj 2012.

Sadržaj:

Uvod

Velik broj poznatih tijela posjeduje barem nekakva elastična svojstva. Nakon određene deformacije, elastična tijela se u svoj početni oblik vraćaju silom koja je proporcionalna udaljenosti od početnog položaja na kojoj se nalaze. Zakon koji opisuje ponašanje elastičnih tijela se zove Hookeov zakon, a otkrio ga je engleski znanstvenik Robert Hooke 1660. godine.

Pojam prikaza elastičnih tijela pomoću računalne grafike postoji još od kraja osamdesetih godina prošlog stoljeća, ali i dalje nije zastupljen u interaktivnim računalnim programima (prije svega računalnim igrama) u zadovoljavajućem postotku. Tamo dominiraju kruta tijela ili elastična tijela, ali napravljena oko krutog „kostura“.

Razlog takvom odabiru programera leži u činjenici da simulacija elastičnih tijela zahtjeva puno više parametara od krutih, koja ostaju stalno u početnom obliku. Veći broj parametara vodi do većeg broja operacija potrebnih za iscrtavanje svake scene, što vodi do sporijeg izvođenja ciljnog programa.

Uz problem očuvanja računalnih resursa, jedan od problema s kojim se prikazi elastičnih tijela suočavaju je nedostatak stabilnosti, jer je uz pogrešan „podražaj“ koji vodi do deformacije moguće doći u stanje iz kojeg je nemoguće vratiti tijelo u početan položaj.

Također je potrebno omogućiti smanjenje realističnosti u prikazu elastičnih tijela, ukoliko to programer zahtjeva, kako bi se dodatno ubrzalo izvođenje programa, a bez da konačan produkt izgleda nedovoljno realistično.

Hookeov zakon

Najosnovniji primjer elastičnog ponašanja je opruga. Kada se ona određenom silom pomakne iz svog početnog položaja, ona će prema svom početnom položaju krenuti silom koju određuje Hookeov zakon:

$$F = -k \cdot x$$

pri čemu je *k* koeficijent elastičnosti opruge, a *x* odmak od početnog položaja. Negativan predznak označava da će smjer sile biti suprotan smjeru u kojem je opruga rastegnuta.

To vrijedi ukoliko se ne preskoči granica elastičnosti opruge, u kojem slučaju će opruga ostati trajno deformirana, a njena elastična svojstva će prestati vrijediti. Elastična potencijalna energija koju ima opruga kada je rastegnuta odgovara formuli:

$$E_{ep} = \frac{1}{2} k \cdot x^2$$

Ta se energija, kako se smanjuje udaljenost od početnog položaja, postupno pretvara u kinetičku energiju, koja svoj maksimum doseže u početnom položaju. Nakon toga se tijelo giba ponovo prema svojoj maksimalnoj amplitudi te se taj proces odvija u krug, odnosno tijelo titra oko ciljnog položaja. Teoretski, proces bi mogao trajati beskonačno, ali zbog prigušenja se amplituda titranja smanjuje sa svakim prolaskom, dok se tijelo trajno ne zaustavi u početnom položaju.

Formula za elastičnu potencijalnu energiju je dobivena tako da se pretpostavilo da je u svakom beskonačno kratkom intervalu sila koja dijeluje na tijelo jednaka umnošku koeficijenta elastičnosti i odmaka iz početnog položaja. Stoga ukupna energija u tijelu odgovara integralu:

$$E = \int_0^{L-L_0} kx dx = \frac{1}{2} k (L - L_0)^2$$

Matematička osnova

Prema Hookeu, sila koja djeluje na rastegnutu oprugu glasi:

$$F = -k \cdot (x(t) - l_0)$$

Opruga se sastoji od dvije ključne točke: Jedne fiksne točke te jedne slobodne, na kojoj se nalazi tijelo mase *m*. Gore navedena sila vuče slobodan dio opruge prema fiksnom.

Za računanje brzine te promijene položaja tijela, trebat će se koristiti poluimplicitna Eulerova metoda integriranja (*semi-implicit Euler method*). Ta metoda služi za rješavanje dviju diferencijalnih jednadžbi zadanih u obliku:

$$\frac{dx}{dt} = f(t, v)$$

$$\frac{dv}{dt} = g(t, x)$$

Metoda daje rješenje preko slijedeće iteracije:

$$v_{n+1} = v_n + g(t_n, v_n) \cdot \Delta t$$

$$x_{n+1} = x_n + f(t_n, v_{n+1}) \cdot \Delta t$$

Razlika između poluimplicitne Eulerova metode te obične Eulerove metode je u tome što poluimplicitna pri računanju slijedeće vrijednost varijable *x* koristi brzinu u tom istom koraku (v_{n+1}), a obična koristi tranutačnu brzinu (v_n).

Koristeći prije navedenu formulu za računanje sile, moguće je dobiti brzinu i put u svakom slijedećem koraku, pod uvjetom da je poznata početna brzina te početni položaj:

$$v(t + h) = v(t) + h \cdot \frac{-k(v(t) - l_0)}{m}$$

$$x(t + h) = x(t) + h \cdot v(t + h)$$

Ali, kako bi rješenje bilo jednostavnije za implementirati, potrebno je razdvojiti x i y, vrijednosti brzine, odnosno položaja. Za potencijalnu primjenu programa u trodimenzionalnom prostoru u budućnost bi također trebalo dodati i z vrijednosti:

$$v_i(t + h) = v_i(t) + \alpha \cdot \frac{g_i - x_i(t)}{h}$$

$$x_i(t+h) = x_i(t) + h \cdot v_i(t+h)$$

gdje *i* označava os za koju se računa, α faktor krutosti, *g* ciljnu vrijednost položaja točke po osi, a x(t) trenutačnu lokaciju točke po istoj osi.

Osnovna ideja

Ideja ovog rada je napraviti program koji će prikazati korisniku dvodimenzionalan objekt. Taj objekt će biti prikazan preko točaka koje predstavljaju svojevrsne „atome“ od kojih je objekt građen. Korisnik će potom pomicanjem točaka objekt izbaciti iz početnog položaja, a objekt će se morati elastično vratiti natrag u početni položaj.

Pomoću konačnog rasporeda točaka te njihovog prvotnog položaja se mora odrediti koliko se tijelo pomaknulo u koordinatnom sustavu te koliko se zarotiralo oko svoje osi. Pomak će se određivati pomoću pomaka centra mase objekta u odnosu na početni objekt. Rotacija će se određivati kao prosjek kutova za koji je svaka od točaka zarotirana oko centra mase u odnosu na početni položaj objekta.

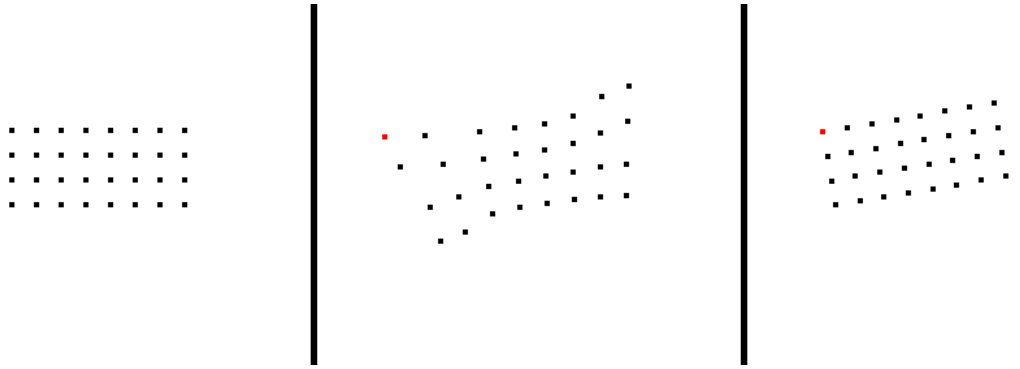
Rotaciju će biti mnogo jednostavnije odrediti u dvodimenzionalnom prostoru, pošto je moguća rotacija oko centra mase samo po jednoj osi. U trodimenzionalnom prostoru je potrebno odrediti matricu rotacije za svaku od tri osi te se time posao uvišestručuje, a brzina izvođenja programa na računalu usporava.

Nakon što se odredi konačan položaj svih točaka, za svaku točku se određuje njen odmak od položaja u koji se ona mora vratiti. Nakon što je to određeno, svakoj točci raste brzina kojom se kreće prema cilju, a paralelno s brzinom se smanjuje udaljenost od početnog položaja. Brzine i udaljenosti su razdvojene za *x* i *y* osi.

Kao što je opisano u poglavlju o Hookeovom zakonu, točke titraju oko ciljne pozicije dok se sve ne smire u ciljnom položaju, a s time i izvođenje programa prestaje.

Potrebno je osigurati konstantnu brzinu izvođenja programa, kako se ne bi dogodila situacija u kojoj isto tijelo titra različitom brzinom ovisno o snazi računala na kojem se program izvodi. Također, potrebno je korisniku omogućiti što jednostavnije i intuitivnije pomicanje točaka u položaj deformacije.

Sav kod za program će biti napisan u jeziku C++, uz korištenje grafičkog standarda OpenGL te biblioteke GLUT, koja će olakšati implementaciju komunikacije korisnika i programa.



Slika 1 - Primjer izvođenja programa: Početni ekran (lijevo), pomaknute toče (sredina), izgled tijela nakon titranja (desno)

Implementacija

Osnovne grafičke postavke

Kako je već napomenuto, korištena biblioteka za olakšan prikaz te komunikaciju s korisnikom će biti GLUT. Kao način prikaza će biti korištena metoda dvostrukog spremnika (*GLUT_DOUBLE*), dok će način prikaza boja biti *GLUT_RGB* (preko crvenih, zelenih i plavih komponenti). Veličina prozora se može ručno promijeniti, a kao početna postavka su zadane i širina i visina sa vrijednosti od 512 slikovnih elemenata.

Također, potrebno je GLUTu predati nazive metoda koje će biti korištene pri interakciji korisnika i računala. Metode koje su korištene za potrebe ovog programa su metode pozvane pri promijeni veličine prozora (*glutReshapeFunc*), pri prikazu (*glutDisplayFunc*), pritisku tipke na mišu (*glutMouseFunc*, služi za odrediti koja točka objekta je pritisnuta te kada je „puštena“), pomicanju miša (*glutMotionFunc*, služi za dinamički prikaz točaka dok se jedna od njih premješta), pritisku tipke na tipkovnici (*glutKeyboardFunc*, služi za započinjanje programa) te funkcija koja se poziva kada se ništa drugo ne događa (*glutIdleFunc*, služi za animiranje elastičnog kretanja točaka).

Primjena dvostrukog spremnika će povećati kvalitetu prikaza te uvelike umanjiti titranje koje bi se inače pri svakoj promjeni scene pojavilo na ekranu. Zato je na početku svakog iscrtavanja potrebno pozvati funkciju koja će zamijeniti trenutnu „aktivnu“ sliku (*glutSwapBuffers*) te onu sliku na kojoj će se crtati treba isprazniti (*glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)*). Nakon toga se može pozvati funkcija koja će crtati na ekranu.

Kako će se raditi isključivo u dvije dimenzije, potrebno je odrediti okomitu projekciju slike. Naredba *gluOrtho2D* određuje ravninu na koju će se projicirati slika.

Stvaranje prikaza tijela

Osnova svakog tijela koje će biti izrađeno za pokretanje i testiranje programa je nacrt. Nacrt će biti dvodimenzionalno polje čija visina i širina okvirno odgovaraju visini i širini predmeta, gledajući točke koje predstavljaju predmet, ne sam predmet. Svaki element polja poprima vrijednosti nula ili jedan, ovisno o tome da li se na tom mjestu nalazi točka ili praznina.

Popunjavanje nacrta za jednostavnije oblike, poput pravokutnika, je jednostavno te se može riješiti pomoću jednostavnih petlji koje C++ omogućuje. Stvaranje oblika koji bi trebao predstavljati čovjeka je već zahtjevnije te je uglavnom takve oblike potrebno ručno upisati, uz poneku petlju radi lakše preglednosti.

Nakon što je stvoren nacrt, potrebno je iz nacrta dobiti točke koje predstavljaju tijelo. Točke tijela su prikazane pomoću razreda:

```
class Point {
public:
	float x, y;
	float k;
	float d;
	float vx, vy;
	Point(float x1, float y1) {
		x = x1; y = y1;
	}
	Point() {}
};
```

Razred Point (točka) sadrži x i y koordinate točaka, varijable vx i vy označavaju brzine kojima se točka kreće po x i y osi, što je korišteno kod animiranja, dok varijable k i d označavajudaljenost točke od početnog položaja te kut za koji je zaokrenuta, a obje se koriste pri određivanju pomaknutog i rotiranog oblika tijela. Konstruktor prima x i y vrijednosti točaka koje se upisuju na odgovarajuće mjesto, dok je prazan konstruktor služio uglavnom za testiranje.

Metoda za stvaranje točaka koristi polje nacrta te visinu i širinu nacrta. Koristeći te dvije varijable te dužinu i širinu ekrana, izračunava idealan početni položaj za objekt, kako ne bi bio preblizu nekom od rubova ekrana. U računu se koristi i veličina točke na ekranu (koja je veća od jedan kako bi bilo lakše vidjeti objekt), ali njen učinak je gotovo pa zanemariv.

```
startX = (sub_width / 2.) - pointSize * (width / 2.) - offset * ((width - 1) / 2.);
startY = (sub_height / 2.) - pointSize * (height / 2.) - offset * ((height - 1) / 2.);
```

Zatim se unutar dvostruke *for* petlje prolazi svakim članom nacrta te, ukoliko je njegova vrijednost jedan, stvara se odgovarajuća točka. Dio unutar petlje gdje se stvara nova točka glasi:

```
if (blueprint[i][j] == 1) {
```

```
        Point p(startX+j*(pointSize+offset), sub_height-(startY+i*(pointSize+ offset)));
        points.push_back(p);
}
```

Isti dio koda, preciznije onaj unutar *if* upita, se ponavlja kako bi se dodatno sačuvala originalna lokacija točke, pošto je vrlo izgledno da će ova lokacija biti promijenjena tijekom izvođenja programa. Naredba *push_back* dodaje točku na kraj vektora koji ih sadržava.

Dodatno se u tom dijelu koda računa početni centar mase tijela. On je jednostavna suma x i y koordinata točaka podijeljena s brojem točaka. Dodatno unaprijeđenje koda bi moglo uključivati specifične mase za svaku točku, ali se za potrebe ovog programa pretpostavilo da je masa pojedinih atoma u tijelu konstanta za jedno tijelo.

Kako bi prikaz tijela bio pregledniji, točke se podebljavaju prije crtanja. Veličina točke se pamti u varijabli *pointSize* te je potrebno to uračunavati pri stvaranju prikaza tijela na ekranu, jer bi bez toga moglo doći do situacije u kojoj bi točke bile bliže jedna drugoj nego što bismo htjeli.

Pomicanje točaka

Poželjno je da se deformiranje objekta čini na što intuitivniji način, stoga je najpraktičnije rješenje direktno pomicati točke objekta pritiskom na odgovarajuću točku te pomicanjem miša. GLUT poziva funkciju za odgovor na pritisak tipke, a kao parametre prenosi x i y koordinate lokacije gdje je tipka pritisnuta te koja je tipka i na koji način pritisnuta. Funkcija koja odgovara na pritisak lijeve tipke miša izgleda ovako:

```
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
	mouseDown = true;
	if(clickedX == -1 && clickedY == -1) {
		clickedX = x; clickedY = y;
		checkProximity(x,y);
	}
}
```

Ova funkcija prvo ispituje da li je tipka koja je pritisnuta lijeva tipka miša te da li je njeno stanje pritisnuto (alternativno može biti otpuštena). Ukoliko je uvjet zadovoljen, u globalnu varijablu se zapisuje da je miš pritisnut (što će biti korišteno pri animaciji micanja točaka). Također, funkcija *checkProximity* provjerava da li je miš pritisnut u blizini neke točke, kako nebi bilo potrebe da korisnik pritisne točno u središte tražene točke, već samo što bliže. Varijable *clickedX* i *clickedY* pamte originalnu lokaciju gdje je miš pritisnut.

Na analogan način je riješena situacija kada je otpuštena lijeva tipka miša:

```
if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
	mouseDown = false;
	clickedX = -1; clickedY = -1;
}
```

Samo pomicanje točaka nakon što je pritisnuta tipka miša se odvija dinamički, kako se povlači miš, tako se za svaki novi poziv funkcije gleda pomak miša u odnosu na prošli. Točka koju smo odabrali se pomiče za taj iznos, dok se svaka od preostalih točaka pomiče za sve manji iznos što joj je udaljenost od odabrane točke veća:

```
dragX = x - clickedX; dragY = y - clickedY;
clickedX = x; clickedY = y;
for(int i = 0; i < points.size(); i++) {
	if (i == index) {
		points.at(i).x += dragX;
		points.at(i).y += dragY;
	}
	else {
		points.at(i).x += dragX / udaljenostTocakaUPocetnomObliku(i);
		points.at(i).y += dragY / udaljenostTocakaUPocetnomObliku(i);
	}
}
```

Gdje se varijabla *index* koristi za pamćenje indeksa pritisnute točke u vektoru koji pamti sve točke, a funkcija *udaljenostTocakaUPocetnomObliku* vraća koliko je pojedina točka bila udaljena od pritisnute u originalnom obliku tijela.

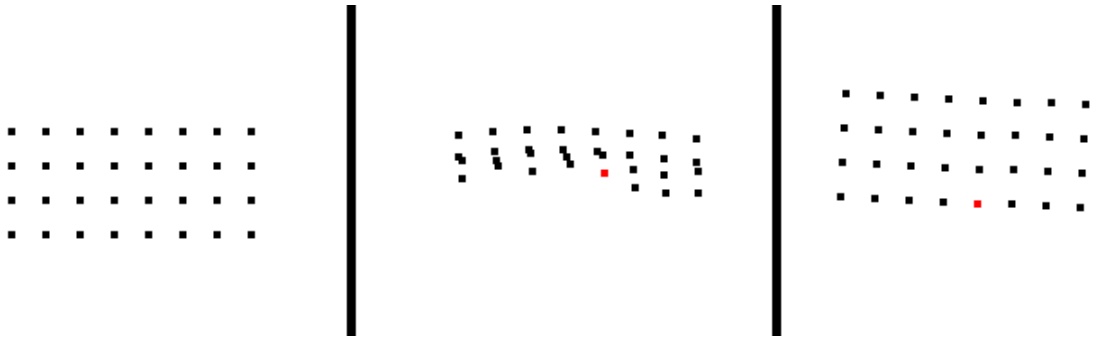
Određivanje pomaka tijela

Određivanje samog pomaka tijela u koordinatnom sustavu se radi preko promijene pozicije centra mase. Kako se već ranije izračunao početni položaj centra mase, samo je potrebno odrediti konačnu lokaciju centra mase koristeći konačne pozicije točaka:

```
pomX = 0; pomY = 0;
for (int i = 0; i < points.size(); i++) {
	pomX += points.at(i).x;
	pomY += points.at(i).y;
}
finalXcm = (GLdouble) pomX / points.size();
finalYcm = (GLdouble) pomY / points.size();
```

Zatim se pomak određuje kao jednostavna razlika početnih i konačnih x i y koordinata centara masa:

```
transXcm = finalXcm - originalXcm;
transYcm = finalYcm - originalYcm;
```



Slika 2 – Primjer određivanja pomaka tijela: početno stanje (lijevo), nakon deformacije (sredina) te konačan položaj nakon završenog titranja (desno)

Određivanje kuta za koji je konačno tijelo zarotirano u odnosu na početno je malo kompliciranije. Prvo se računa nagib pravca koji vodi od početnih točaka do početnog centra mase. Zatim se računa nagib pravca koji vodi od konačnih točaka do pomaknutog centra mase te se računa kut koji ta dva pravca zatvaraju po formuli:

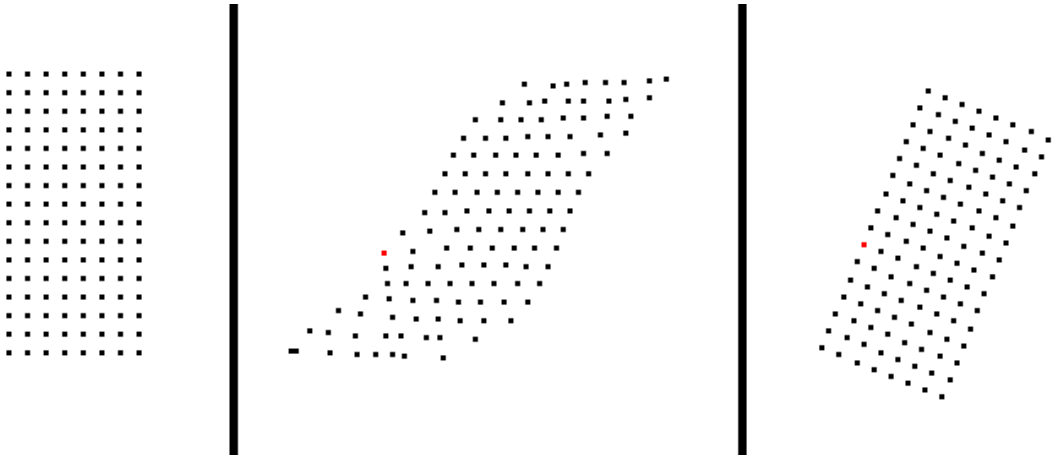
$$\alpha = \tan^{-1} \frac{k_2 - k_1}{1 + k_1 k_2}$$

Zatim se svi kutovi pamte u zasebnom polju kako bi se kasnije mogao izračunati prosječan kut za koji je tijelo zarotirano:

```
for (int i = 0; i < points.size(); i++) {
```

```
        k1 = (originalPoints.at(i).y-originalYcm)/(originalPoints.at(i).x-originalXcm);
        k2 = (points.at(i).y-finalYcm)/(points.at(i).x-finalXcm);
        angles.push_back(atan((k2 - k1) / (1 + k1 * k2)));
}

pom = 0;
for (int i = 0; i < angles.size(); i++) pom += angles.at(i);
avgAngle = pom / angles.size();
```



Slika 3 – Primjer određivanja kuta rotacije tijela: početni položaj (lijevo), deformiranje (sredina) te konačan položaj nakon što završi titranje (desno)

Nakon što je određen pomak i rotacija, potrebno je napraviti vektor koji će sadržavati ciljne pozicije točaka, to jest one pozicije u kojima će se naći točke deformiranog tijela nakon što završe sa „titranjem“. Pomak točaka će biti jednostavan, samo je potrebno dodati pomak centara mase svakoj od točaka. Rotacija za kut α će se odvijati po slijedećoj formuli:

$$p'_x = \cos\alpha \cdot (p_x - o_x) - \sin\alpha \cdot (p_y - o_y) + o_x$$

$$p'_y = \sin\alpha \cdot (p_x - o_x) - \cos\alpha \cdot (p_y - o_y) + o_y$$

Gdje *p* predstavlja originalnu poziciju točke, *p'* rotiranu, a *o* centar rotacije. Primjena tih dviju formula u kodu glasi:

```
for (int i = 0; i < originalPoints.size(); i++) {
        pX = originalPoints.at(i).x + transXcm;
        pY = originalPoints.at(i).y + transYcm;
        pX2 = cos(avgAngle)*(pX-finalXcm) - sin(avgAngle)*(pY-finalYcm) + finalXcm;
        pY2 = sin(avgAngle)*(pX-finalXcm) + cos(avgAngle)*(pY-finalYcm) + finalYcm;
        Point p(pX2, pY2);
        transPoints.push_back(p);
}
```

Gdje *transPoints* označava vektor koji sadrži ciljne točke. Nakon što su određene konačne točke, nije potreban daljni unos podataka korisnika u program te se može krenuti s animiranjem titranja točaka do konačnog položaja. Za početak je svakoj točki potrebno dodijeliti početnu brzinu, to jest, postaviti x i y brzine na nulu kao početno stanje iz kojeg se kreće:

```
for (int i = 0; i < points.size(); i++) {
	points.at(i).vx = 0.0; points.at(i).vy = 0.0;
}
```

Animiranje

Animiranje će se odvijati unutar Idle funkcije GLUTa. Ta će funkcija svaki određeni period vremena računati novu brzinu svake od točaka te ih pomaknuti u odgovarajućem smjeru za iznos ovisan o periodu te brzini.

Prioritet kod animiranja kretanja točaka je osigurati provjeru koja će se pobrinuti da se cijeli kod za animiranje ne izvodi dok korisnik ne pritisne odgovarajuću tipku ili na drugi način ne da do znanja programu da krene s izvođenjem. To ćemo pratiti preko jednostavne globalne varijable čija je vrijednost potvrdna kada se treba provoditi animiranje.

Glut je u stanju javiti korisniku koliko je vremena prošlo u milisekundama preko slijedeće funkcije:

```
currentTime = glutGet(GLUT_ELAPSED_TIME);
timeInterval = currentTime - previousTime;
	...

previousTime = currentTime;
```

Kako bi se dobila konstantna brzina izvođenja programa, potrebno je provoditi računanja samo svaki određeni broj milisekundi. Taj broj će se pamtiti u globalnoj varijabli *interval*, a za uobičajenu verziju će se uzeti vrijednost od 100 milisekundi. Kada varijabla *timeInterval* preraste varijablu *interval*, funkcija će računati nove brzine i položaje točaka.

Određivanje nove brzine tijela je ovisno o trenutačnoj brzini te o udaljenosti od ciljne pozicije. Potrebno je račun provoditi za brzine po obje osi, x i y. Također, u jednadžbu je potrebno uračunati i faktor krutosti tijela, koji određuje brzinu povratka tijela u početni oblik, kao i faktor prigušenja, koji umanjuje amplitudu titranja do konačnog zaustavljanja u ciljnom položaju.

Pomak u prostoru se određuje koristeći novoodređenu brzinu pomoću najosnovnije formule za brzinu

$$\Delta s = v \cdot \Delta t$$

Takav postupak radimo za pomak po obje osi te cjelokupan postupak ponavljamo za svaku točku u tijelu:

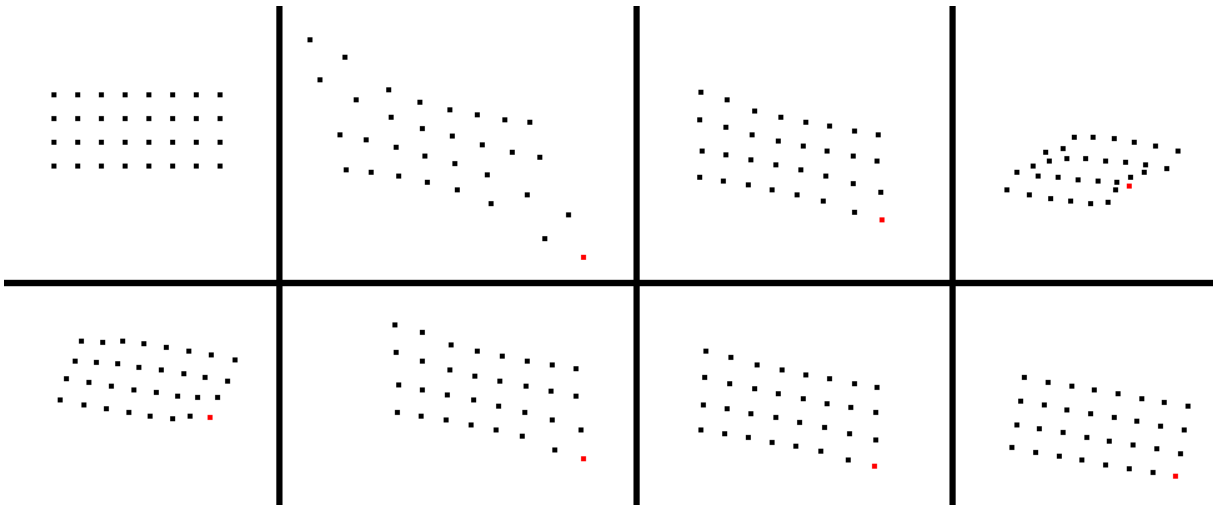
```
for (int i = 0; i < points.size(); i++) {
points.at(i).vx=fp*(points.at(i).vx+a*(transPoints.at(i).x-points.at(i).x)/interval);
points.at(i).vy=fp*(points.at(i).vy+a*(transPoints.at(i).y-points.at(i).y)/interval);

points.at(i).x += points.at(i).vx * interval;
points.at(i).y += points.at(i).vy * interval;
```

```
}
```

Kako bi se provelo zaustavljanje animacije nakon što sve točke dođu u ciljani položaj, potrebno je na kraju svake iteracije provjeriti jesu li sve točke u točnom položaju te je li njihova brzina jednaka nuli. Provjera brzine služi kako se program ne bi slučajno zastavio u trenutku kada točka prolazi kroz središnju točku titranja, ali posjeduje brzinu kojom bi nastavila put prema svojoj amplitudi.

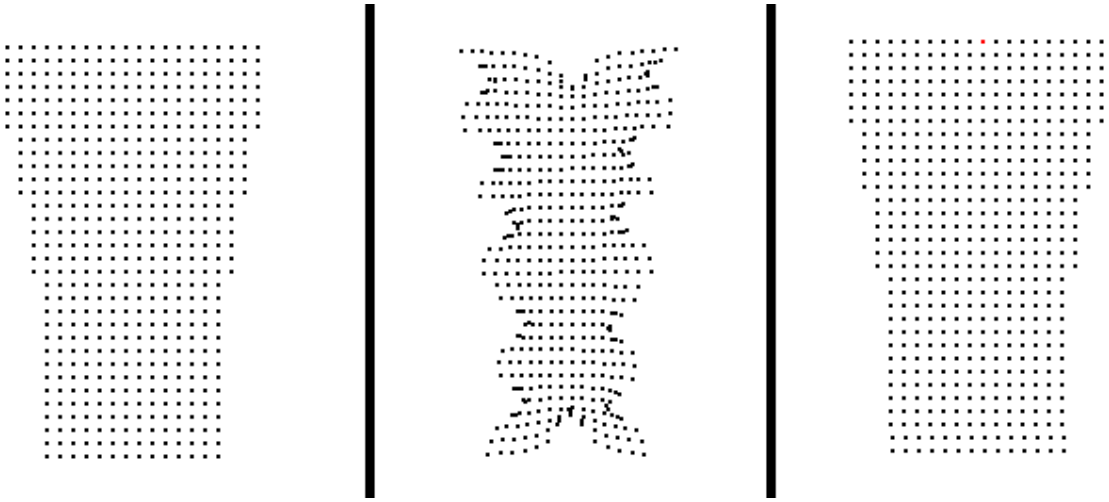
Također, ukoliko će se oblik više puta deformirati te vraćati u početni položaj u istom pozivu programa, potrebno je paziti da se vektor koji sadrži ciljane točke tijela isprazni prije pokretanja svake animacije, kako se nebi naknadne točke spremale u vektor koji već sadrži njihove lokacije. Za to služi jednostavna *clear* naredba.



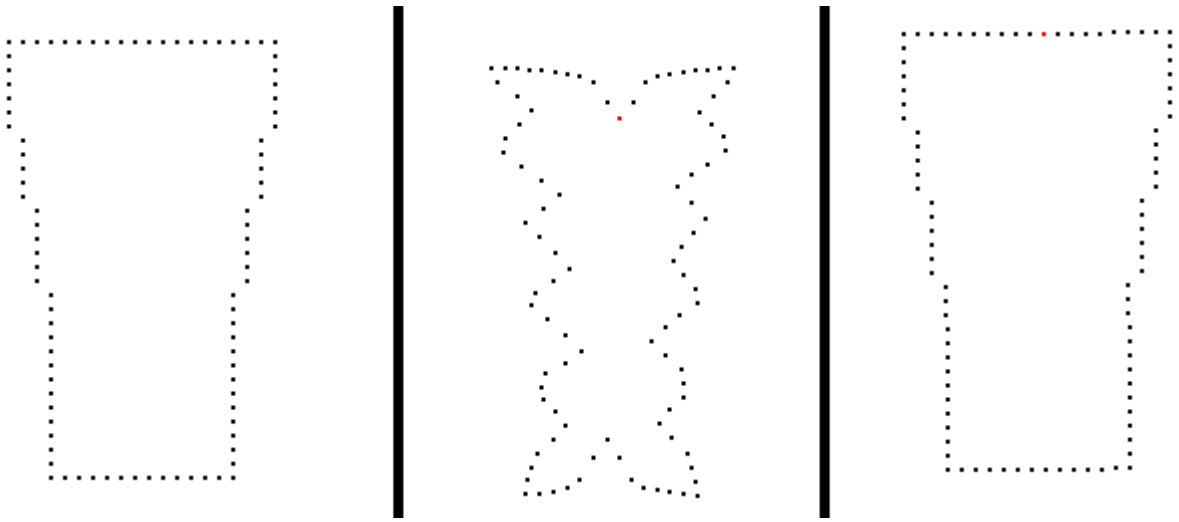
Slika 4 – Primjer izvođenja animacije titranja tijela, od početnog položaja, preko deformacije te titranja do konačnog položaja

Također je potrebno analizirati ponašanja različitih vrsta tijela kod programa. Za tu potrebu je načinjen model koji reprezentira čašu u dva različita oblika: Prvi, koji je puno tijelo kao i sva prikazana dosad te drugi, koji se sastoji samo od vanjskih obrisa.

Primjer rada programa za oba primjera će biti prikazan u slikama 5 i 6.



Slika 5 – Primjer rada sa „punim“ tijelom



Slika 6 – Primjer rada programa sa šupljim objektom

Test performansi

Za potrebe testiranja, postavljena je osnovna vrijednost intervala u kojem se slika iscrtava na ekranu na 10 milisekundi. Vrijednosti broja točaka će se svoditi na potencije broja dva, kako bi bilo lakše praćenje.

Za sve vrijednosti broja točaka do 1024 (2^{10}), program radi konstantnom brzinom od 100 sličica u sekundi, dok pri 2048 (2^{11}) točaka se već osjeća lagano usporenje, iako se i dalje iscrtava približno 100 sličica u sekundi.

Za 4096 točaka se iscrtava između 70 i 80 sličica po sekundi, a za 8192 (2^{12}) točke tek 40 sličica po sekundi. Ako dodatno udvostručimo broj točakana 16384 (2^{13}), broj sličica koje su iscrtane po sekundi smanjuje se na dvadesetak ili manje. S narednim porastima broja točaka, brzina se smanjuje na svega nekoliko sličica po sekundi.

Iz dobivenog zaključujemo kako je program itekako efikasan, čak i na nešto sporijim računalima za računanje tijela koja se sastoje od čak i pet tisuća točaka. Naravno, ukoliko će se raditi sa tijelima u tri dimenzije, brzine će biti znatno sporije, ali i tada će biti moguće koristiti neka relativno detaljno napravljena tijela.

Moguća poboljšanja u programu

Osnovno moguće poboljšanje je prebacivanje tijela iz dvodimenzionalnog u trodimenzionalan prostor. Sam prikaz tijela će biti otežan, pošto će se umjesto jednostavnog dvodimenzionalnog polja koje je predstavljalo nacrt u originalnoj ideji morati koristiti trodimenzionalno polje.

Uz otežano „izrađivanje“ tijela, dodatan problem će predstavljati i deformiranje istog. Kako se u osnovnom programu radilo s dvodimenzionalnim prostorom na ekranu računala (koje je dvodimenzionalno), pomicanje točaka je bilo jednostavnije, pošto je pomak po x-osi računala bio jedna pomaku po x-osi u ravnini te analogno za y-os.

U trodimenzionalnom prostoru će biti potrebno za pomak pratiti i z-os, odnosno aproksimirati pomak po toj osi ovisno o načinu na koji korisnik pomiče miš. Alternativno rješenje bi bilo da korisnik upisuje pomake po sve tri osi u program, što bi bilo mnogo praktičnije za izvesti, ali bi se intuitivnost programa uvelike umanjila.

Nakon što je odrađeno deformiranje tijela, biti će potrebno unijeti određene promijene u algoritam za prepoznavanje ciljnog oblika tijela u koje će se vratiti. Određivanje pomaka u prostoru će biti vrlo slično kao u dvodimenzionalnom prostoru, odrađivat će se preko pomaka centra mase, ali će biti potrebno račun izvesti za sve tri koordinatne osi:

$$x_{cm}^0 = \frac{\sum_i m_i x_i}{\sum_i m_i}, y_{cm}^0 = \frac{\sum_i m_i y_i}{\sum_i m_i}, z_{cm}^0 = \frac{\sum_i m_i z_i}{\sum_i m_i}$$

$$x_{cm} = \frac{\sum_i m_i x_i}{\sum_i m_i}, y_{cm} = \frac{\sum_i m_i y_i}{\sum_i m_i}, z_{cm} = \frac{\sum_i m_i z_i}{\sum_i m_i}$$

gdje je x^0_{cm} x koordinata centra mase za početni oblik tijela, a x_{cm} x koordinata centra mase za konačni oblik tijela. Analogno vrijedi za y i z koordinate. Pomoću tih vrijednosti možemo izračunati pomak tijela kao:

$$\Delta x_{cm} = x_{cm} - x_{cm}^0, \Delta y_{cm} = y_{cm} - y_{cm}^0, \Delta z_{cm} = z_{cm} - z_{cm}^0$$

Određivanje kuta za koji je tijelo rotirano će biti višestruko kompliciranije: umjesto za jednu os, morati će se pratiti kut za koji je tijelo rotirano oko svake od svojih tri osi. Račun će biti vrlo sličan onome iz osnovnog dvodimenzioalnog primjera, ali će biti potrebno računati trostruko više informacija za svaku točku.

Velika prednost algoritma koji vraća točke u konačan položaj jest što, nakon što odredimo konačan položaj tijela, nema potrebe za kompliciranijim računanjem do kraja izvođenja programa, odnosno točke bi se trebale vratiti u ciljani položaj na potpuno isti način

za dvodimenzionalni prikaz kao i za trodimenzionalni, uz jedinu razliku što bi kod trodimenzionalnog bilo potrebno računati tri vrijednosti brzina te položaja.

$$v_i(t+h) = v_i(t) + \alpha \cdot \frac{g_i - x_i(t)}{h}$$

$$x_i(t+h) = x_i(t) + h \cdot v_i(t+h)$$

Drugim riječima, vrijednost *i* će u ovom slučaju poprimati vrijednosti *x, y* i *z*, za razliku od dvodimenzionalnog prostora gdje je poprimala samo *x* i *y* vrijednosti.

Osim prelaska u treću dimenziju, potencijalno poboljšanje u kodu može uključivati teksturiranje objekata. Kako sam prikaz preko točaka koje čine tijelo nije posebno atraktivan, potrebno bi bilo omogućiti da se preko objekta stavi određena tekstura kako bi korisnik stekao dojam da izobličava neko stvarnije tijelo.

Problem u tom slučaju bi predstavljalo realistično izobličavanje tekstura. Ako bismo samo pridijelili neku teksturu prostoru između tri točke, tako da teksturu razbijemo na okvirne trokute te svaki trokut dodijelimo jednom području, u nekim slučajevima (kao onaj osnovni koji se može vidjeti na slici 1.) bismo dobili jako grube pomake tekstura koji bi izgledali nerealistično.

Taj problem je donekle moguće umanjiti ukoliko povećamo broj točaka koje čine tijelo te smanjimo razmak između njih (odnosno, povećamo preciznost s kojom izrađujemo prikaz tijela), ali time automatski usporavamo rad računala te je potrebno naći optimalni kompromis između kvalitete prikaza te brzine izvođenja.

Još jedno moguće poboljšanje bi moglo predstavljati uvođenje „praga elastičnost“. U trenutačnoj verziji koda, ukoliko tijelo rastežemo s jedne na drugu stranu ekrana, ono će se uvijek vratiti u početni položaj, dok bi s pragom postojala određena sila kojom bismo djelovali na tijelo tako da ono ostane u trajno u svom položaju, odnosno izgubi svojstva elastičnosti.

Također je moguće uvesti način na koji bi se simuliralo međudjelovanje različitih čestica tijela. Zasad program omogućava situaciju u kojoj dvije točke prolaze jedna kroz drugu bez ikakvih posljedica, a to ne doprinosi realističnosti prikaza te bi posebno neugodno bilo za vidjeti u situaciji u kojoj bismo stavili teksture preko objekata. Drugo moguće međudjelovanje bi uključivalo odbijanje točaka od čvrstih površina, na primjer rubova ekrana ili umjetno stvorenih krutih tijela na ekranu, pokretnih ili nepokretnih.

Što se intuitivnosti korištenja programa tiče, možda najnužnije rješenje bi bilo kvalitetno grafičko sučelje koje bi korisniku omogućavalo podešavanje izgleda tijela po svojoj želji. Ideja toga bi bila da se omogući korisniku sučelje u kojem će nacrtati obrise tijela, a program bi trebao automatski prema tom obrisu načiniti šuplje tijelo te ga, ukoliko je po želji korisnika, ispuniti.

Također bi bilo poželjno omogućiti korisniku da parametre poput krutosti tijela ne mora podešavati u samom kodu programa, već da u korisničkom sučelju postoje odgovarajuća mjesta u koja bi se ti podaci upisivali. Tako bi se omogućilo dinamičnije mijenjanje svojstva programa te bi se uvelike poboljšao dojam koji isti ostavlja na promatrače.

Zaključak

Programi koji simuliraju elastičnost objekata su kroz povijest bili vrlo složeni te su svojim nekvalitetnim performansama „poticali“ programere da umjesto elastičnih objekata koriste kruta tijela te pomoću elementarnog povezivanja istih ostavljaju dojam elastičnosti.

Nepotrebno je za naglasiti kako bi kvalitetna izvedba programa koji simulira elastičnost bila itekako tražena te korisna u raznim granama računarstva. No, da bi se program proglasilo kvalitetnim, on mora zadovoljiti cijeli niz zahtjeva, od brzih performansi i na limitiranijim računalima, preko stabilnosti koja jamči da se neće sustav rušiti pri svakoj manjoj iznimci te intuitivnosti koja bi omogućila programeru da razne parametre podesi ovisno o svojim potrebama ili mogućnostima.

Program koji je opisan u ovom radu omogućuje sve tri opcije, nije pretjerano zahtjevan za računala, relativno je stabilan (vrlo teško dolazi do prekida zbog pogreške u programu) te omogućuje precizno postavljanje raznih parametara kako bi se ostvarilo što realističnije ili manje realistično kretanje objekata.

Naravno, program otvara mogućnost cijelog niza poboljšanja, počevši od prelaska u treću dimenziju ili omogućavanja korisniku da preko jednostavnog grafičkog sučelja kreira svoj objekt po želji, ali i ostavlja par mjesta na kojima su nužne preinake u budućnosti, posebice na području izobličavanja tijela.

Literatura

[1] Meshless Deformations Based on Shape Matching – Müller, Heidelberger, Teschner, Gross

[2] Physically Based Deformable Models in Computer Graphics – Nealen, Müller, Keiser, Boxerman, Carlson

[3] Fast Lattice Shape Matching for Robust Real-Time Deformation – Rivers, James

[4] Interaktivna računalna grafika kroz primjere u OpenGL-u – Čupić, Mihajlović, 2011.

[5] Wikipedia, Elasticity - http://en.wikipedia.org/wiki/Elasticity_%28physics%29

Sažetak

U ovom radu je obrađena tema elastičnog ponašanja tijela predstavljenih kao skupine čestica. Glavni cilj rada je izrada programa koji simulira ponašanje elastičnog tijela u dvodimenzionalnom prostoru kada se na tijelo djeluje određenom vanjskom silom koja ga deformira. Osim programa, dan je detaljan pregled njegovog koda napisanog u programskom jeziku C++ te su priložene upute za korištenje istog programa.

Predstavljena je fizikalna osnova ponašanja elastičnih tijela te metode kojima se određuje konačan položaj u koji prelazi deformirano tijelo. Objašnjene su osnove matematičkih metoda kojima se dolazi do formula za kretanje tijela. Uz to, prikazane su i moguće promijene koje treba načiniti u kodu kako bi zadovoljio neke kompleksnije primjere, poput prebacivanja koda u trodimenzionalan prostor.

Ključne riječi: elastično tijelo, deformacija, sustav čestica

Abstract

This thesis presents the topic of the elastic behavior of bodies represented as collections of particles. The main goal of the thesis was to create a program that simulates the behavior of a body in a two-dimensional area when it's deformed by an outside force. Other than the program, the thesis contains a detailed description of the code used to make the program in the programming language C++ and the instructions on how to use the program.

The thesis also contains the physical base of the methods used to simulate the behavior of particles and which are used to determine the position of the body after the particles return to their original positions as well as the mathematical methods used to get all the necessary formulas. Other possible improvements to the program, such as converting it to a three-dimensional space are also examined.

Key words: elastic body, deformation, particle system

Upute za korištenje

Nakon pokretanja programa, korisnik odabire točku koju želi pomaknuti te pritiskom lijeve tipke miša dok je u njenoj blizini ju „prihvaća“ te pomicanjem miša seli na drugu poziciju. Taj postupak ponavlja za sve točke tijela koje želi pomaknuti. Ukoliko želi samo jednu točku pomaknuti, bez da to ima ikakav utjecaj na preostale, odabirom tipke „q“ prelazi u mod u kojem se točke pomiču individualno. Ponovnim pritiskom na tipku „q“ se vraća u normalan način pomicanja točaka.

Nakon što je korisnik doveo tijelo u odgovarajući deformirani oblik, pritiskom na tipku „s“ započinje titranje točaka prema konačnom položaju. Korisnik treba pričekati da se program izvrši do kraja (u konzoli ispiše da je gotov) prije nego što nanovo krene premještati točke tijela.