SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 690

MODEL ELASTIČNE TKANINE

Janko Sladović

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 7. ožujka 2014.

DIPLOMSKI ZADATAK br. 690

Pristupnik: **Janko Sladović**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Model elastične tkanine**

Opis zadatka:

Proučiti modele kojima se opisuju elastične strukture. Proučiti modele tkanina. Razraditi model elastične tkanine te ostvariti simulaciju djelovanja sile na ovom modelu. Ostvariti programsku implementaciju koja omogućuje prikaz razrađenog modela i načiniti usporedbu sa stvarnom tkaninom. Na različitim primjerima prikazati ostvarene rezultate. Diskutirati utjecaj parametara. Načiniti ocjenu implementiranog algoritama i ostvarenih rezultata.
Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 30. lipnja 2014.

Mentor:
Prof.dr.sc. Željka Mihajlović

Djelovođa:
Doc.dr.sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Prof.dr.sc. Siniša Srbljić

Sadržaj

Popis slika

1. Uvod

Prikaz tkanina u računalnoj grafici predstavlja vrlo složen problem koji kombinira dva ključna zahtjeva: dobiti uvjerljivu reprezentaciju stvarne tkanine te napisati program koji se brzo izvršava. Glavni izazov je pojednostaviti brojne interakcije koje se odvijaju unutar tkanina na manji broj računalnih operacija koje program provodi.

Količina predmeta na kojima je potrebno simulirati izgled tkanina, kao na primjer odjeće, dovela je do brojnih pokušaja izrade modela prikaza tkanina. Jedna od mogućih podjela tih modela je na diskretne te kontinuirane modele.

Diskretni modeli rastavljaju tkaninu na niz čestica u pravilu povezanih oprugama te su jednostavniji za implementaciju, ali manje kvalitetni. Alternativan diskretan model je 1993. predstavio David E. Breen, u njemu je u obzir uzeo osnovne mehaničke interakcije između niti u tkaninama kako bi izračunao ukupnu energiju koja djeluje na svaku česticu u tkanini. Kontinuirani modeli koji prikazuju kako se tijelo deformira pod utjecajem sila, značajno su kompliciraniji za implementaciju, ali i daju kvalitetnije rezultate.

U ovom radu će biti prikazana tri načina za simulaciju tkanina: Prvi je klasični model masa i opruga, drugi prilagođava tkaninama već napravljeni model prikaza elastičnih tijela [1], dok treći predstavlja svojevrstan hibrid prethodne dvije metode. Ideja tog konačnog modela je pokušati uzeti poželjna svojstva iz prethodna dva, a u isto vrijeme ukloniti neke od problema koji će biti objašnjeni u narednim poglavljima.

2. Matematička i fizikalna osnova

2.1. Osnovno o tkaninama

Zbog kompliciranе unutarnje građe, odnosno isprepletenosti te elastičnih svojstava niti koje ih čine, tkanine prikazuju nelinearna, anizotropna svojstva koja je teško precizno simulirati. Dakako, moguće je sva ta svojstva rastaviti na pojedinačne parametre, za izračun kojih je potrebno koristiti kompliciranе strojeve koji se fokusiraju na specifična svojstva, ali čak i za pojednostavljene modele potrebno je računati nekoliko desetaka parametara.

Primjer jednostavnijeg modela možemo pronaći u radu Huamina Wanga, Jamesa O'Briena te Ravija Ramamoorthija s Berkeleyja [2]. Oni su u svom radu prikazali pojednostavljen model koji se sastoji od 24 parametra rastezanja tkanina te 15 parametara savijanja tkanina, odnosno ukupno 39 parametara. Tih 39 parametara su izračunali za desetak raznih materijala, sa po tri primjera svakog materijala, a za svaki primjer su provodili po 35 testova, odnosno 105 po materijalu.

Iz tog primjera je jasno zašto se danas preferira koristiti pojednostavljene modele koji koriste linearne elastične modele koji ovise o nekoliko parametara koji određuju krutost materijala. Za potrebe grafičkih aplikacija poput računalnih igara ili filmova nije potrebno imati najprecizniji mogući prikaz tkanina, dovoljno je predstaviti korisniku kvalitetnu aproksimaciju koja će se prikazivati potrebom brzinom.

2.2. Sustav masa i opruga

Sustav masa i opruga kojim će se prikazivati tkanine u narednim poglavljima ima svoju osnovu u Hookeovom zakonu. Hookeov zakon nam govori da je sila koja djeluje na tijelo na rastegnutoj opruzi proporcionalna udaljenosti tijela od položaja ravnoteže, odnosno formulom:

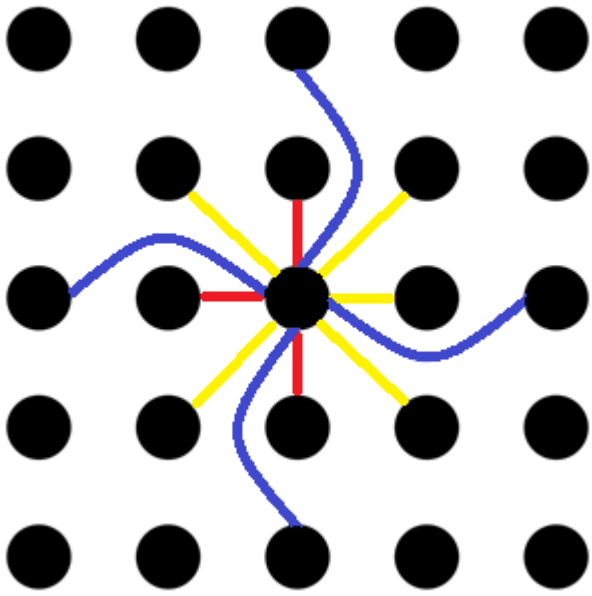
$$F = -k(l - l_0) \tag{1}$$

gdje l označava trenutnu duljinu opruge, l_0 početnu duljinu opruge (odnosno duljinu opruge u položaju ravnoteže), a k konstantu opruge. Očito je iz ovoga da će upravo ta konstanta određivati neka od svojstava tkanina u sustavu masa i

opruga. Negativan predznak označava da će sila uvijek djelovati u smjeru suprotnom od smjera rastezanja opruge.

U sustavu koji se koristi za prikaz tkanina koristimo nekoliko vrsta opruga, koje predstavljaju nekoliko vrsta sila koje djeluju na čestice. Strukturne opruge (eng. *structural springs*) povezuju najbliže susjedne čestice horizontalno i vertikalno te služe za svladavanje sila rastezanja i kompresije.

Opruge striženja (eng. *shearing springs*) povezuju najbliže dijagonalno susjedne čestice i služe za svladavanje istoimenih sila, dok opruge savijanja (eng. *bending springs*) služe za svladavanje sila savijanja te povezuju svaku drugu česticu horizontalno i vertikalno. Sve tri vrste opruga su prikazane na slici 1. Dodatno, moguće je oprugama savijanja povezivati i svaku drugu česticu dijagonalno.



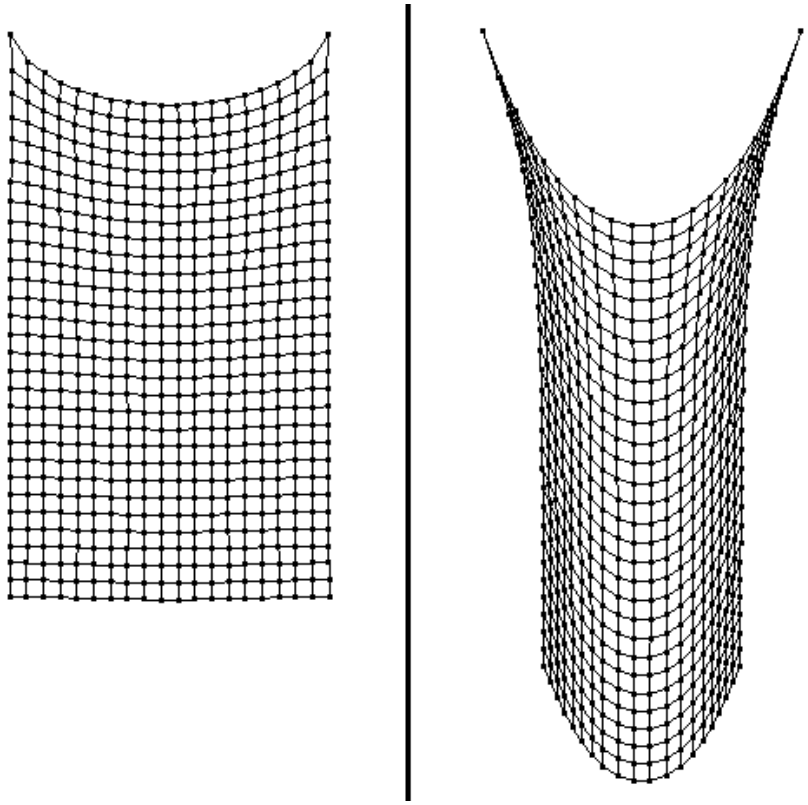
Slika 1 - Tri vrste opruga u sustavu masa i opruga – crvenom bojom su označene strukturne opruge, žutom opruge striženja, a plavom opruge savijanja

Važnost druge dvije vrste opruga može se vidjeti na slici 2, gdje je prikazan izgled materijala sa sve tri vrste opruga te samo sa strukturnim oprugama. Vidljivo je koliko stabilnije djeluje prva slika. Konstante koje imaju strukturne opruge su u pravilu nekoliko puta veće od onih koje imaju opruge striženja te savijanja.

Za svaku od opruga se računa sila kojom će ona privući ili odgurnuti svaku od čestica koje povezuje. Ukoliko je jedna od čestica teža od druge, na nju će djelovati manja sila. Skalar koji određuje na koju od čestica će djelovati kolika sila možemo izračunati kao:

$$s = \frac{\frac{1}{m_1}}{\frac{1}{m_1} + \frac{1}{m_2}} \tag{2}$$

iz čega je vidljivo da će skalar biti jednak 0.5 ukoliko su dvije mase jednake. Za drugu česticu bi u brojniku nazivnika bilo m_2 umjesto m_1.



Slika 2 - Razlika između korištenja sve tri vrste opruga (lijevo) te samo strukturnih opruga (desno)

Ta sila se računa prema Hookeovoj formuli navedenoj u ovom poglavlju. Nakon što su određene sve sile, bilo sile elastičnosti, bilo neki vanjski utjecaji poput gravitacije ili nekih drugih sila, računa se akceleracija koju ta sila daje čestici prema formuli:

$$a = {}^{F}/_{m} \tag{3}$$

Taj korak nije nužno potreban ukoliko sve čestice u tkanini imaju jednaku masu, što je uglavnom i slučaj.

2.3. Elastični modeli temeljeni na sustavu čestica

Drugi pristup izradi elastičnog modela svodi se na ideju opisanu u radu Meshless Deformations Based on Shape Matching [1]. Iako je ideja prvotno namjenjena za prikaz elastičnih tijela, a ne tkanina, ovdje će se razmotriti kako je moguće primijeniti tu ideju za prikaz tkanina, bilo samostalno, bilo kao hibrid između te ideje te sustava masa i opruga. Na slikama 3 i 4 je prikazana primjena tog postupka na elastične dvodimenzionalne i trodimenzionalne objekte.

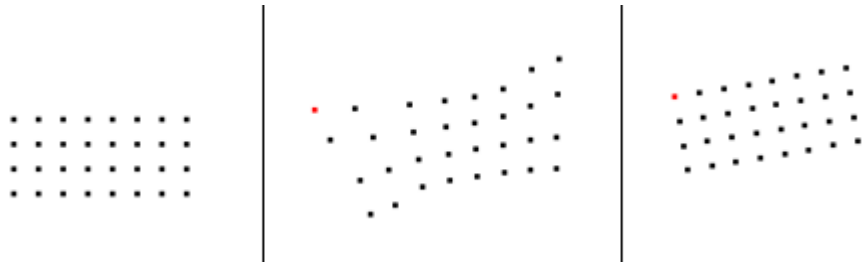
Osnovna ideja tih modela je aproksimacija konačnog izgleda elastičnog tijela, gledajući koliko svaka pojedinačna čestica odstupa od svog početnog položaja te tražeći prosječnu transformaciju gledajući sve čestice. Ta prosječna transformacija se najčešće dobiva kao prosječan pomak u prostoru (pomak centra mase) te prosječna razlika rotacije u prostoru, pri čemu se gleda kut koji zatvara čestica u početnom položaju s početnim centrom mase i kut koji zatvara čestica u trenutnom položaju s konačnim centrom mase.

Traženje pomaka centra mase je trivijalan zadatak koji se svodi na računanje prosječnog položaja centra mase, što je moguće jednom izračunati na početku pokretanja programa da se izbjegnu kasnija računanja, a radi se tako da se izračunaju prosječne x, y i z koordinate čestica. Taj postupak se ponavlja prilikom svakog ponovnog računanja položaja kako bi se dobio pomak centra mase. Matematički zapisano vrijedi:

$$x_{cm} = \frac{\sum_i m_i x_i}{\sum_i m_i} \tag{4}$$

$$x_{cm_0} = \frac{\sum_i m_i x_{i_0}}{\sum_i m_i} \tag{5}$$

gdje *x* označava vektor koji sadrži x, y i z koordinate čestica, a *m* predstavlja masu pojedinačne čestice. U pravilu su mase konstantne kroz cijelu tkaninu, ali ovakvo računanje ostavlja mogućnost za neke kompliciranije sustave. Potpisana nula označava početan položaj čestica koji je konstantan kroz cijeli program.



Slika 3 - Dvodimenzionalna primjena metode elastičnih modela temeljenih na česticama [9]

Traženje prosječne rotacije je znatno kompliciraniji postupak. U dvodimenzionalnim sustavima bilo je dovoljno gledati za koliko je stupnjeva točka bila okrenuta oko svoje osi (os bi prestavljala centar mase), što vrijedi i u trodimenzionalnim sustavima, ali je računanje znatno otežano. Kako bi se taj postupak olakšao, korišteni su kvaternioni za prikaz rotacija. Kvaternion između dva vektora (u ovom slučaju vektora koji povezuju čestice s početnim odnosno konačnim centrima masa) se sastoji od četiri vrijednosti, *w*, *x*, *y* i *z*, a one su dobivene na sljedeći način:

$$q.w = \cos\frac{kut}{2} \tag{6}$$

$$q.x = \sin\frac{kut}{2} * os.x \tag{7}$$

$$q.y = \sin\frac{kut}{2} * os.y \tag{8}$$

$$q.z = \sin\frac{kut}{2} * os.z \tag{9}$$

U tim formulama je *kut* dobiven kao skalarni produkt normaliziranih vrijednosti ta dva vektora, a *os* kao vektorski produkt normaliziranih vrijednosti ta dva vektora. Kako su svi kvaternioni relativno blizu, dovoljno je samo pronaći metodu koja aproksimira njihov prosjek. Metoda za izračun se može prikazati sljedećim pseudokodom:

```
za svaki kvaternion:
       brojObradjenihKvaterniona++;
       privremenaVrijednost = 1 / brojObradjenihKvaterniona;

       prosjecanKvaternion.W += q.W;
       w = prosjecanKvaternion.W * privremenaVrijednost;

       prosjecanKvaternion.X += q.X;
       x = prosjecanKvaternion.X * privremenaVrijednost;

       prosjecanKvaternion.Y += q.Y;
       y = prosjecanKvaternion.Y * privremenaVrijednost;

       prosjecanKvaternion.Z += q.Z;
       z = prosjecanKvaternion.Z * privremenaVrijednost;

       normaliziraj(prosjecanKvaternion);
```

Prosječan kvaternion na početku bio inicijaliziran s nulama kao sve četiri vrijednosti. Alternativno je moguće i preskočiti normalizaciju u svakom koraku te ju samo izvršiti na kraju, ukoliko je potrebno ubrzati rad programa. Nakon provedenog postupka dobili smo konačan kvaternion koji predstavlja prosječnu rotaciju svih pojedinačnih kvaterniona. Iz tog kvaterniona možemo dobiti os oko koje trebamo rotirati sve vektore kao i kut za koji je potrebno rotirati vektore. Koristeći matricu za rotaciju točke za kut φ oko osi v [3]:

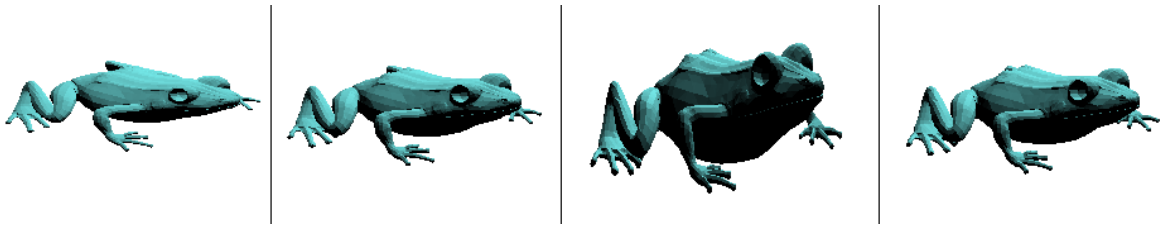
$$R = \begin{matrix} v_x^2(1-\cos\varphi)+\cos\varphi & v_x v_y(1-\cos\varphi)-v_z\sin\varphi & v_z v_x(1-\cos\varphi)+v_y\sin\varphi \\ v_x v_y(1-\cos\varphi)+v_z\sin\varphi & v_y^2(1-\cos\varphi)+\cos\varphi & v_y v_z(1-\cos\varphi)-v_x\sin\varphi \\ v_x v_z(1-\cos\varphi)-v_y\sin\varphi & v_y v_z(1-\cos\varphi)+v_x\sin\varphi & v_z^2(1-\cos\varphi)+\cos\varphi \end{matrix} \quad (10)$$

u kojoj potpisano slovo uz v označava na koju od tri vrijednosti vektora se odnosi te poznatu formulu za transformaciju:

$$x_{rotirano} = R x_{početno} \quad (11)$$

možemo dobiti konačnu formulu za ciljani položaj čestica:

$$x_{konačno} = R\left(x_{početno} - x_{cm_0}\right) + x_{cm} \quad (12)$$



Slika 4 - Primjer metode elastičnih modela temeljenih na sustavu čestica primijenjene na trodimenzionalni objekt [10]

Postoji još nekoliko načina izračuna prosječne rotacije čestica oko centara mase, ali ovaj se pokazao najjednostavnijim za implementirati. Druga mogućnost je traženje matrice A koja minimizira izraz:

$$\sum_i m_i(Aq_i - p_i) \tag{13}$$

gdje je q_i vektor koji vodi od početnog centra mase do početnog položaja čestice i, a p_i vektor koji vodi od trenutnog centra mase do trenutnog položaja čestice *i*, a m_i označava masu čestice *i*.

Derivacijom tog izraza, dolazimo do formule za matricu *A*:

$$A = (\sum_i m_i p_i q_i^T)(\sum_i m_i q_i q_i^T)^{-1} = A_{pq}A_{qq} \tag{14}$$

u kojoj je matrica A_{qq} simetrična matrica koja nije vezana uz rotaciju te se traženje matrice *R* odvija isključivo preko matrice A_{pq} uz formule:

$$A_{pq} = RS \tag{15}$$

$$S = \sqrt{A_{pq}^T A_{pq}} \tag{16}$$

Problem s ovim postupkom je vrijeme potrebno za računanje korijena matrice A_{pq} te računanje inverza matrice *S*, koje je u potpunosti usporilo rad programa. Moguće je primijeniti metode koje će olakšati izračun inverza matrica, ali postupak s kvaternionima je vrlo brz i daje rezultate koji su sasvim prihvatljivi.

2.4. Polu-implicitna Eulerova metoda

Prilikom simulacije matematičkih sustava u stvarnom vremenu, često se moraju rješavati diferencijalne jednadžbe druge razine. Takve su jednadžbe često zadane u obliku:

$$x''(t) = f(t, x(t), x'(t)) \tag{17}$$

uz početne uvjete:

$$x(t_0) = x_0,\ x'(t_0) = v_0 \tag{18}$$

Alternativno, moguće je izolirati brzinu kao zasebnu varijablu te tako dobiti diferencijalnu jednadžbu prvog reda:

$$\frac{dx}{dt} = f(t, v) \tag{19}$$

$$\frac{dv}{dt} = g(t, x) \tag{20}$$

za koju također vrijede početni uvjeti navedeni u (18). Postoje brojne metode numeričke integracije koje je moguće primijeniti na rješavanje ovih diferencijalnih jednadžbi, kao što su implicitna i eksplicitna Eulerova metoda te Runge-Kutta metoda. U ovom radu će se koristiti polu-implicitna Eulerova metoda koja nalazi rješenje gore navedenog sustava u obliku:

$$v_{n+1} = v_n + g(t_n, v_n) \cdot \Delta t \tag{21}$$

$$x_{n+1} = x_n + f(t_n, v_{n+1}) \cdot \Delta t \tag{22}$$

Drugim riječima, brzina u trenutku *n+1* se računa pomoću brzine u trenutku *n*, dok se položaj u trenutku *n+1* računa pomoću položaja u trenutku *n* te brzine u trenutku *n+1*.

3. Implementacija

3.1. Osnovne postavke

Svi algoritmi su implementirani u programskom jeziku C#. Kao grafički API je korišten OpenGL. Postoji nekoliko biblioteka koje daju pristup OpenGL naredbama u C#u. Jedna od starijih biblioteka je Tao Framework, ali njegov razvoj je zaustavljen prije više od 5 godina tako da nije najkvalitetnija opcija.

Novija i kvalitetnija opcija, koja je u većini slučajeva zamijenila Tao Framework, je Open ToolKit, poznatiji kao OpenTK. Riječ je o redovito nadograđivanoj, kvalitetno optimiziranoj besplatnoj biblioteci za izradu grafičkih programa u jeziku C#. Dodatna pogodnost rada s OpenTK bibliotekama je što sadrže gotove klase za, između ostalog, vektore, matrice i kvaternione, zajedno s većinom potrebnih funkcija za rad s tim zapisima.

Prvi korak u izradi aplikacije je učitati potrebne biblioteke, za OpenTK je korištena njegova, u trenutku početka izrade rada, najnovija stabilna verzija, 1.1.0.0. Ta je verzija omogućila korištenje OpenGLa 4.4, proširenu matematičku biblioteku i niz drugih pogodnosti koje nisu imale direktan utjecaj na ovaj rad. U međuvremenu je izašla i stabilna verzija 1.1.1, ali nije donijela značajne napretke koji bi pridonijeli kvaliteti izrade rada. Uz biblioteku OpenTK, potrebno je bilo učitati i *System.Drawing* biblioteku, o kojoj OpenTK ovisi.

Za izradu grafičkih aplikacija koristeći OpenTK koristi se njegova klasa *GameWindow*. Nasljeđivanje klase *GameWindow* omogućava nam da sami definiramo kako će se ponašati njene ključne metode. Neke od glavnih korištenih metoda su: *OnLoad*, *OnResize*, *OnUpdateFrame*, *OnRenderFrame*. Dodatno je moguće definirati i metode *OnClosing*, *OnOnload* te *OnWindowInfoChanged*, *OnKeyPress* i mnoge druge, ali te nisu bile korištene tijekom izrade rada.

Metoda *OnLoad* služi kako bismo definirali početne uvjete, postavili početne parametre te definirali početan izgled ekrana. U kompliciranijim aplikacijama, ovdje bi se učitavale teksture i razni vanjski podaci poput glazbe, ali za potrebu ove aplikacije dovoljno je postaviti grafičke postavke. Metoda *OnResize* se poziva ukoliko dođe do promjene veličine prozora aplikacije i uglavnom služi kako bismo definirali novu projekcijsku matricu u tom slučaju.

Metode *OnLoad* i *OnResize* su bile jednake za sve napravljene implementacije prethodno opisanih modela:

```
protected override void OnLoad(EventArgs e)
{
     base.OnLoad(e);

     GL.ClearColor(1.0f, 1.0f, 1.0f, 0.0f);
     GL.Enable(EnableCap.DepthTest);
}

protected override void OnResize(EventArgs e)
{
    base.OnResize(e);

    GL.Viewport(ClientRectangle.X, ClientRectangle.Y, ClientRectangle.Width,
       ClientRectangle.Height);

    Matrix4 projection = Matrix4.CreatePerspectiveFieldOfView((float)Math.PI
       / 4, Width / (float)Height, 1.0f, 64.0f);
    GL.MatrixMode(MatrixMode.Projection);
    GL.LoadMatrix(ref projection);
}
```

Metoda *OnUpdateFrame* služi kako bi se pripremila iduća scena za iscrtavanje. U nju se stavlja sva potrebna programska logika, pozivaju sve potrebne metode za računanje novih položaja čestica te se po potrebi provjerava kolizija. Za samo iscrtavanje služi metoda *OnRenderFrame*, u njoj se koriste same OpenGL naredbe i pozivaju metode za iscrtavanje pojedinačnih elemenata.

U nastavku će biti prikazan izgled metode *OnRenderFrame* u kojoj varijable *pointmasses* označava listu koja sadrži individualne čestice u tkanini.

```
protected override void OnRenderFrame(FrameEventArgs e)
{
    base.OnRenderFrame(e);

    GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
    cam.Update();
    Matrix4d modelview = cam.ViewMatrix;
    GL.MatrixMode(MatrixMode.Modelview);
    GL.LoadIdentity();
    GL.LoadMatrix(ref modelview);

    GL.Color3(0.0, 0.0, 0.0);

    foreach (PointMass p in pointmasses) p.draw();

    SwapBuffers();
}
```

Interakcija korisnika s programom se odvija preko miša i tipkovnice, a moguće ju je implementirati na nekoliko načina. Interakcija s tipkovnicom se može odvijati preko posebne metode *OnKeyPress* ili preko *OnUpdateFrame* metode, unutar koje koristimo svojstvo *Keyboard* koju nam klasa *GameWindow* omogućava. Interakcija s mišem se odrađuje pomoću varijable dobivene pozivom na funkciju:

```
OpenTK.Input.Mouse.GetState();
```

Metoda *Run* pokreće grafički program, a prima nula, jedan ili dva argumenta. Prvi od potencijalno dva argumenta određuje broj poziva metodi *OnUpdateFrame* u sekundi, odnosno koliko često će se osvježavati položaji čestica u ovom slučaju. Drugi argument određuje koliko će se često iscrtavati scena na ekran, odnosno pozivati metoda *OnRenderFrame*. Oba parametra označavaju ciljani broj pozivanja u sekundi, ali zbog usporavanja programa može doći do manjeg zadanog broja. Ukoliko se neki od argumenata ostavi prazan ili postavi na vrijednost nula, program će ciljati na maksimalan mogući broj pozivanja u sekundi.

Radi kvalitete prikaza, koristi se metoda dvostrukog spremnika (double buffering). Ta metoda ne iscrtava sliku direktno na ekran, već ju priprema u memoriji, postavlja na ekran kad je gotova, a istovremeno počinje s pripremom druge slike. Metoda dvostrukog spremnika je podržana u OpenTKu pomoću metode *SwapBuffers* unutar *GameWindow* klase.

Kako je, kad je u pitanju tkanina, riječ o esencijalno dvodimenzionalnom objektu, vrlo je važno s koje strane promatramo isti, dok je kod raznih elastičnih objekata, poput gumenih igračaka, znatno manje važno s koje strane gledamo. Zato je za potrebu izrade ovog rada korištena kamera koja se nalazi na sferi koja okružuje centar ekrana. Koliko god taj način prikaza smetao prilikom interakcije miša i programa, on omogućuje da vrlo jednostavno, korištenjem nekoliko tipki na tipkovnici, promijenimo kut iz kojega gledamo na objekt.

U nastavku je prikazana skraćena verzija *OnUpdateFrame* metode u kojoj je prikazano kako korisnik može promijeniti položaj kamere. Varijabla *physics* označava klasu koja se bavi izračunom novih položaja čestica, varijabla *camera* je

instanca klase *Camera* koja služi za upravljanje pogledom na tkaninu, a njene varijable *Rotation* i *ArcBallRadius* su zadužene za pamćenje rotacije, odnosno radijusa sfere koja okružuje središte ekrana.

```
protected override void OnUpdateFrame(FrameEventArgs e)
{
   base.OnUpdateFrame(e);

   physics.update(pointmasses, gravity);

   if (Keyboard[Key.Escape]) Exit();

   if (Keyboard[Key.G]) toogleGravity();

   if (Keyboard[Key.W]) camera.Rotation += new Vector3d(1.0, 0.0, 0.0);
   if (Keyboard[Key.S]) camera.Rotation += new Vector3d(-1.0, 0.0, 0.0);
   if (Keyboard[Key.A]) camera.Rotation += new Vector3d(0.0, 1.0, 0.0);
   if (Keyboard[Key.D]) camera.Rotation += new Vector3d(0.0, -1.0, 0.0);
   if (Keyboard[Key.Q]) camera.Rotation += new Vector3d(0.0, 0.0, -1.0);
   if (Keyboard[Key.E]) camera.Rotation += new Vector3d(0.0, 0.0, 1.0);

   if (Keyboard[Key.KeypadPlus]) camera.ArcBallRadius += 0.1;
   if (Keyboard[Key.KeypadMinus]) camera.ArcBallRadius -= 0.1;

}
```

3.2. Implementacija modela masa i opruga

Prvi korak u izradi algoritma bilo je stvaranje tkanine koja će služiti u simulaciji. U stvarnim primjenama algoritma, bilo bi moguće koristiti već gotove objekte, ali za potrebu ovog rada je korišten jednostavan primjer dvodimenzionalne tkanine koja visi pričvršćena na dva mjesta. Primjer rada tog algoritma prikazan je na slici 5.

Metoda za izradu primjera tkanine kao argumente prima početne *x*, *y* i *z* koordinate, odnosno koordinate na kojima počinje izrada tkanine i udaljenost između čestica, odnosno duljinu opruge na kojoj ona miruje. Sama izrada svodi se na dvostruku petlju u kojoj se prolazi kroz tkaninu kao kroz 2D matricu te se *x* i *y* vrijednost povećavaju ovisno o udaljenosti od početnog položaja.

Također, kao je riječ o sustavu masa i čestica, potrebno je povezati čestice sa svojim susjedima. Duljine opruga koje povezuju te čestice ovise o tipu. Za strukturne opruge duljina je jednaka duljini razmaka između čestica, za opruge savijanja ona je dvostruko veća, dok je za opruge striženja ona $\sqrt{2}$ puta veća od

početne udaljenosti. Pseudokod koji predstavlja izradu tkanine sa visinom *visina* i širinom *sirina* za prvi algoritam glasi:

```
for (j = 0; j < visina; j++)
 {
     for (i = 0; i < sirina; i++)
      {
           x = xPocetno  + i * duljinaOpruge;
           y = yPocetno - (j * duljinaOpruge);
           z = zPocetno;

           cestica = new Cestica(x, y, z);
           N = duljina(cestice);
           diagDuljina = duljinaOpruge * sqrt(2);

           // strukturne opruge
           if (i != 0)
             cestica.povezi(cestice[N – 1], duljinaOpruge);
           if (j != 0)
             cestica.povezi(cestice[(j - 1) * sirina + i], duljinaOpruge);

           // opruge strizenja
           if (i != 0 && j != 0)
             cestica.povezi(cestice[(j - 1) * sirina + i – 1], diagDuljina);
           if (i != (sirina - 1) && j != 0)
             cestica.povezi(cestice[(j - 1) * sirina + i + 1], diagDuljina);

           // opruge savijanja
           if (i > 1)
             cestica.povezi(cestice[N – 2], 2 * duljinaOpruge);
           if (j > 1)
             cestica.povezi(cestice[(j - 2) * sirina + i], 2 * duljinaOpruge);

           cestice.dodaj(cestica);
       }
 }
```

Metoda koja spaja čestice s oprugama može ili biti rekurzivna, tako da zapravo u sustavu imamo 2N opruga gdje za svaku točku možemo pamtiti njene opruge ili imati listu s N opruga koje obuhvaćaju sve čestice u tkanini.

U sustavu masa i opruga, opruge koje povezuju čestice su zadužene za većinu interakcija. Prilikom računanja sila koje djeluju na svaku od čestica potrebno je u obzir uzeti gravitacijsku silu, zatim vanjske sile poput vjetra ili sile dobivene putem moguće interakcije s korisnikom te za svaku od veza s tom česticom potrebno je izračunati ukupnu silu djelovanja na česticu.

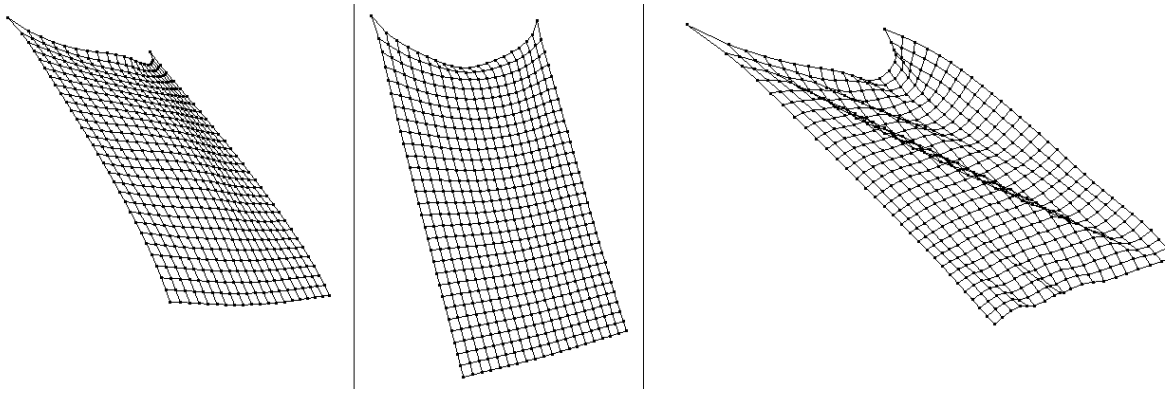
Potrebno je razdvojiti metode za izračun sila i metode za izračun novog položaja. Ukoliko bi te dvije metode bile povezane u jednu, sila na prvu čestice u tkanini (najčešće gornju lijevu) bi se računala isključivo pomoću starih vrijednosti

položaja, dok bi se sila za zadnju (odnosno donju desnu) česticu računala isključivo pomoću novih vrijednosti položaja, što bi moglo dovesti do manjih nepravilnosti u ponašanju programa. Ovako, ukoliko se prvo izračunaju sile za sve čestice, a zatim položaji za sve čestice imamo donekle stabilniji sustav.

Za svaku česticu se sila računa tako da se prvo poveća njena *y* komponenta ovisno o tome djeluje li na nju gravitacija. Zatim se prolazi kroz sve opruge koje povezuju tu česticu s drugima, a sila se povećava ili smanjuje ovisno o tome koliko je ta opruga daleko od svog položaja ravnoteže.

Moguće je dodatno definirati granicu izdržljivosti opruge, koja bi označavala maksimalno produljenje koje opruga može doživjeti prije nego ona pukne ili izgubi elastična svojstva. U matematici bi takva opruga jednostavno ostala u izduženom položaju, ali za potrebu prikaza elastičnih tijela, praktičnije je da veza pukne, a tkanina se razdvoji na dva dijela oko te veze. Najjednostavnije je provjeru duljine opruge izvršiti prije nego se izračuna sila za pojedinu oprugu.

Dodatno, sila se povećava ovisno o tome djeluje li vjetar na česticu. Ukoliko je to slučaj, vektor sile se povećava za vrijednost vektora smjera vjetra (koji mora biti normaliziran) pomnoženu sa snagom vjetra.



Slika 5 - Prikaz tri različita položaja tkanina prilikom izvođenja prvog algoritma

Nakon što je izračunata ukupna sila koja djeluje na sve čestice, računa se njihov pomak. Prvo se računa trenutna brzina čestice kao razlika između položaja od prije dva koraka te položaja iz prošlog koraka. Ukoliko je potrebno, moguće je dodati faktor prigušenja kako bi se brzina umanjila, odnosno kako bi se tkanina nakon nekog vremena prestala kretati ukoliko na nju ne djeluju vanjske sile.

Nakon što je izračunata brzina, računa se akceleracija kao omjer sile i mase čestice pomoću brzine i akceleracije prema formuli:

$$x = x_0 + vt + at^2 \tag{23}$$

te se računa novi položaj čestica. Neke čestice se računaju kao pričvršćene, odnosno nepomične, tako da za njih ne moramo računati nove položaje. Takve čestice mogu biti i vezane za neki objekt. Ako, na primjer, imamo zastavu kojom mašemo, postoji niz čestica koje su pričvršćene na štap koji drži zastavu te se ne miču od njega.

3.3. Implementacija modela temeljenog na sustavu čestica

Za razliku od prvog algoritma, u drugom algoritmu nije potrebno uopće koristiti veze između čestica, pošto svaka čestica ima svoj početni, trenutni i ciljani položaj. Moguće je stvoriti neke veze koje bi se iscrtavale kako bi bilo očitije kako tkanina izgleda, ali te veze se uopće ne koriste za računanje sila ili položaja. Dodatno, veze između čestica će pomoći pri dodavanju sjenčanja u algoritam kako bi on bolje izgledao, iako je to moguće riješiti jednostavnim pamćenjem susjednih čestica (dovoljno je pamtiti samo najbližu lijevu te gornju česticu).

Algoitam koji koristimo za izradu tkanine je pojednostavljena verzija algoritma iz prvog koraka, jedino je u konstruktoru koji se poziva prilikom stvaranja nove čestice potrebno dodatno zapamtiti početan položaj čestice u tkanini. Zbog nedostatka veza, nije moguće jednostavno implementirati trganje tkanine kao što je bio slučaj u prvom algoritmu. Moguće je, ukoliko se razmak između dva dijela tkanine poveća preko nekog praga, razdvojiti tkaninu na dva dijela, tako da svaki dobije svoje vlastito polje čestica, ali rezultati toga su u pravilu znatno slabiji od trganja tkanine u prvom algoritmu.

Kako, za razliku od prethodne metode, u ovom algoritmu nema potrebe razmatrati utjecaj do 12 različitih opruga na svaku od čestica, već se sile računaju isključivo na temelju trenutne i ciljane pozicije svake od čestica, za očekivati je da će ovaj algoritam funkcionirati nešto brže što se tiče brzine. Pseudokod koji

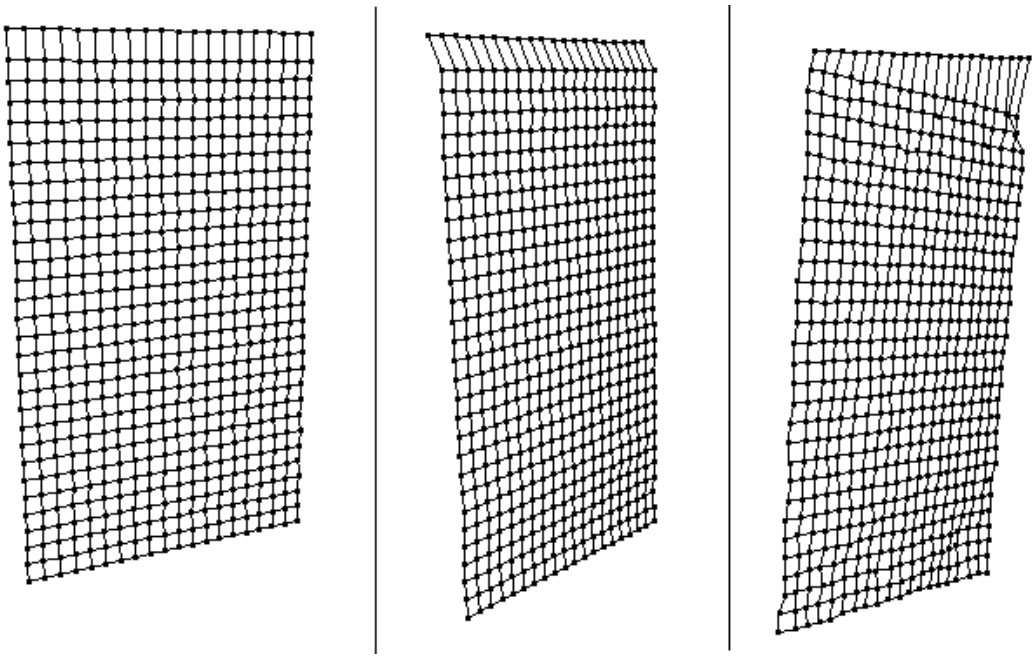
prikazuje računanje osi oko koje treba rotirati čestice te kuta za koji ih treba rotirati glasi:

```
t = nadjiTrenutniCentarMase(cestice);
t0 = nadjiPocetniCentarMase(cestice);

za svaku cesticu c u cestiacma
   {
       qi = c.x0 - t0;
       pi = c.x - t;

       kvaternioni.dodaj(stvoiKvaternion(pi, qi));
   }

Kvaternion q = nadjiProsjecanKvaternion(kvaternioni);

kut = q.vratiKut();
os = q.vratiOs();
```

Funkcije za traženje početnog centra mase i trenutnog centra mase su ekvivalentne onima opisanim u poglavlju 2.3. Računanje početnog centra mase te vektora q_i je dovoljno napraviti jednom na početku izvođenja programa kako bi se ubrzala svaka od iteracija, ali ovdje je to stavljeno u sklop petlje kako bi bilo jasno o kojem vektoru se radi. Funkcija za traženje prosječnog kvaterniona je ekvivalentna onoj opisanoj u poglavlju 2.3, a unutar OpenTKa je definirana potrebna funkcija za dohvaćanje kuta i osi rotacije iz kvaterniona.

Nakon što su se izračunali kut te os rotacije, koriste se matrica rotacije navedena u (10) i formula navedena u (12) kako bi se našao konačan položaj svake od čestica. Koristeći trenutan te ciljan položaj čestica, računa se pojedinačna sila za svaku od njih. Ta sila je proporcionalna udaljenosti, a karakteristike tkanina će se mijenjati ovisno o tome hoće li ta zavisnost biti linearna, kvadratna ili s nekom drugom potencijom. Uz izračunatu silu, koriste se iste formule opisane u poglavlju 3.2 kako bi se iz sile i trenutačne brzine izračunala brzina u idućem koraku te novi položaj čestica.

Zbog činjenice da je algoritam prije svega prilagođen simulaciji trodimenzionalnih objekata, a ne tkanina koje imaju drugačija svojstva, prije svega savijanja, rezultati dobiveni ovim algoritmom su značajno manje kvalitetni od dobivenih u poglavlju 3.2. Tijelo se više ponaša kao komad kartona nego kao tkanina te djeluje pretjerano kruto, kao što se može vidjeti na slici 6.



Slika 6 - Primjena elastičnih modela temeljenih na sustavu čestica na prikaz tkanina

Cilj je zato modela koji će biti opisan u idućem poglavlju kombinirati kvalitetu prikaza koju pruža sustav masa i opruga sa brzinom izvođenja te jednostavnijim aproksimacijama koje nam omogućava model opisan u ovom poglavlju.

3.4. Implementacija hibridnog modela

Ideja hibridnog algoritma je kombinirati kvalitetan prikaz sustava masa i opruga sa pojednostavljenom aproksimacijom položaja koju nam pruža model temeljen na sustavu čestica. Osnova tog algoritma je rastaviti tkaninu na manje dijelove koji se sastoje od konstantnog broja čestica. Moguće je napraviti verziju u kojoj dijelovi nemaju nužno isti broj čestica, ali za potrebu ovog rada je napravljena pojednostavljena verzija koja bi demonstrirala kako algoritam funkcionira.

Svaki od tih dijelova, koji možemo promatrati kao centar mase niza čestica, funkcionirao bi kao što funkcioniraju čestice u sustavu masa i opruga, bio bi povezan sa sve tri vrste veza sa susjednim dijelovima te bi na njega djelovale sve potrebne sile. Unutar tih dijelova, ovisno o položaju tog i susjednih centara masa

aproksimiramo konačan položaj svih čestica te one titraju prema tom konačnom položaju kao što rade u drugom algoritmu.

Očita prednost u odnosu na sustav masa i opruga je što se, na primjer u slučaju kad se dio sastoji od 4 čestice po širini te 4 čestice po visini, razmatra 16 puta manje veza, odnosno promatramo po 12 veza za centar mase, ali ne i za pojedinačne čestice. Prednost u odnosu na drugi algoritam jest što ne tretiramo tkaninu kao cjelinu, svaki od njenih dijelova i dalje djeluje više „ukočeno“ od čistog sustava masa i opruga, ali kvalitetnim aproksimiranjem možemo dobiti tkaninu koja u dovoljnoj mjeri simulira većinu svojstva sustava masa i opruga.

Algoritam za izradu tkanine je vrlo sličan onom iz prvog algoritma, jedino što se umjesto nove čestice stvara dio koji se sastoji od x čestica po duljini te y čestica po visini. Zatim se poziva konstruktor za svaki pojedinačni dio kojem se šalju potrebni parametri (položaj centra mase te duljina i visina dijela, uz masu pojedinačne čestice te razmak između čestica) koji raspoređuje čestice po stvorenom dijelu. Pseudokod koji prikazuje taj dio programa glasi:

```
xPocetno = centarMase.X - (sirinaDijela - 1) * razmak * 0.5;
yPocetno = centarMase.Y + (visinaDijela - 1) * razmak * 0.5;
zPocetno = centarMase.Z;

for (j = 0; j < visinaDijela; j++)
{
   for (i = 0; i < sirinaDijela; i++)
   {
      x = xPocetno + i * razmak;
      y = yPocetno - (j * razmak);
      z = zPocetno;

      Cestica c = new Cestica(x, y, z, masa);
      cestice.dodaj(c);
   }
}
```

Kao i u prethodnom algoritmu, moguće je pamtiti lokacije susjednih čestice ili simulirati veze između istih kako bi se poboljšao izgled programa te olakšala implementacija sjenčanja u kasnijim fazama dorade algoritama. Nešto je zahtjevnije simulirati vezu između čestica koje se nalaze na rubovima svojih dijelova, ali to se može riješiti tako da se za svaki dio pamte i njegovi susjedi, što će se pokazati važnim u idućim koracima izrade ovog algoritma.

Određivanje najbolje metode za predviđanje ciljnog položaja pojedinačnih čestica unutar dijela je bilo zahtjevnije nego u prošloj metodi. Primjenjivanje metode s kvaternionima nije dalo kvalitetne rezultate, pošto je svaki dio imao drugačiju orijentaciju te je dobiven „stepeničast“ izgled tkanine s naglim skokovima između dijelova.

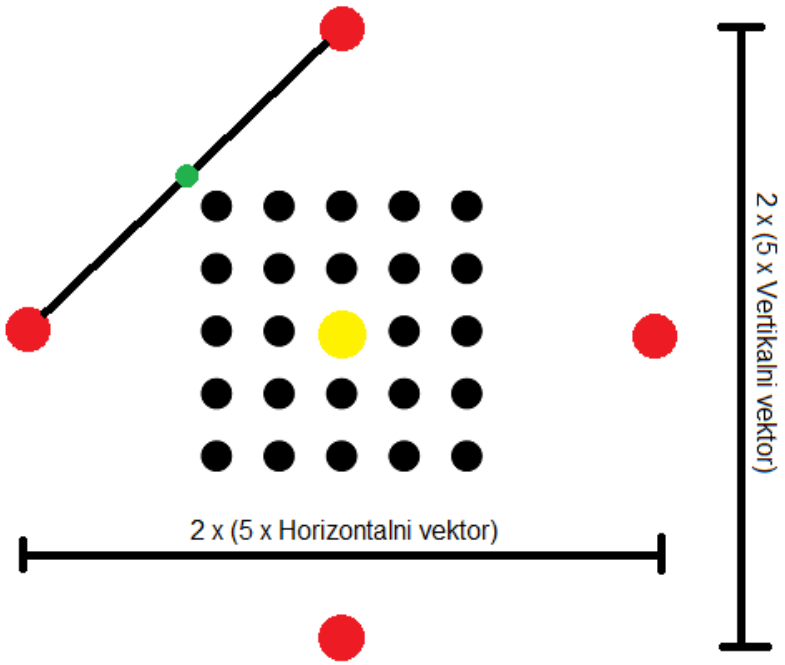
Iz tog razloga, bilo je potrebno odrediti metodu koja će u obzir uzeti i orijentacije susjednih dijelova unutar tkanine. Jedna od mogućnosti je bila prilikom određivanja kuta za koji je pojedinačan dio zakrenut uzeti u obzir i čestice susjednih dijelova. To je dovelo do nešto blažih skokova između dijelova, ali i dalje nije bilo idealno rješenje, a dovelo je do usporenog rada programa.

Najkvalitetnije pronađeno rješenje je u potpunosti zanemarilo bilo kakvo računanja kuteva između čestica, već se u potpunosti svodilo na traženje idealnog položaja za čestice između četiri susjedna dijela, gornjeg, donjeg, lijevog te desnog. To rješenje je davalo puno brže rezultate od alternativa, a kvaliteta prikaza je bila daleko najveća.

Iz tog razloga je dodatno potrebno pamtiti susjedne dijelove svakom od pojedinačnih dijelova. Neće biti potreban pristup pojedinačnim česticama unutar tih dijelova, potrebno će biti samo znati centar mase određenom dijelu. Dodatno, trebalo bi napomenuti da se dio programa u kojem se traže konačni položaji čestica unutar dijelova odvija nakon što se odrede novi položaji centara masa. Ukoliko je potrebno dodatno ubrzati rad programa, moguće je proces pojednostaviti tako da se koriste položaji gornjeg i lijevog dijela iz trenutne iteracije te donjeg i desnog dijela iz prošle iteracije, ali to će dovesti do određenog, iako ne dramatičnog, pada kvalitete prikaza.

Postupak funkcionira na sljedeći način: određujemo startnu poziciju u dijelu kao aritmetičku sredinu vektora gornjeg te lijevog susjednog dijela. Zatim određujemo vektor horizontalnog kretanja u dijelu kao razliku između desnog i lijevog susjednog centra mase podijeljenu sa širinom dijela te vektor vertikalnog kretanja u dijelu kao razliku između donjeg i gornjeg centra mase podijeljenu s visinom dijela.

Postupak je prikazan na slici 7. Tamo su crvenom bojom označeni centri masa susjednih dijelova tkanine. Crvenim krugom je označeno polovište dužine koja povezuje gornjeg i lijevog susjeda. Dodatno je prikazano kako pomoću udaljenosti između gornjeg i donjeg te lijevog i desnog susjeda možemo dobiti duljine vertikalnog i horizontalnog vektora.



Slika 7 - Prikaz postupka određivanja konačnog položaja čestica u tkanini

Koristeći ta dva dobivena vektora, vršimo postupak sličan inicijalizaciji same tkanine tako da prolazimo u dvostrukoj petlji kroz visini i širinu dijela te pomičemo vektor početnog položaja dijela za vektore horizontalnog i vertikalnog gibanja. Pseudokod koji opisuje taj postupak, u kojem *gornjiV*, *donjiV*, *lijeviV* te *desniV* predstavljaju gornji, donji, lijevi te desni vektor, glasi:

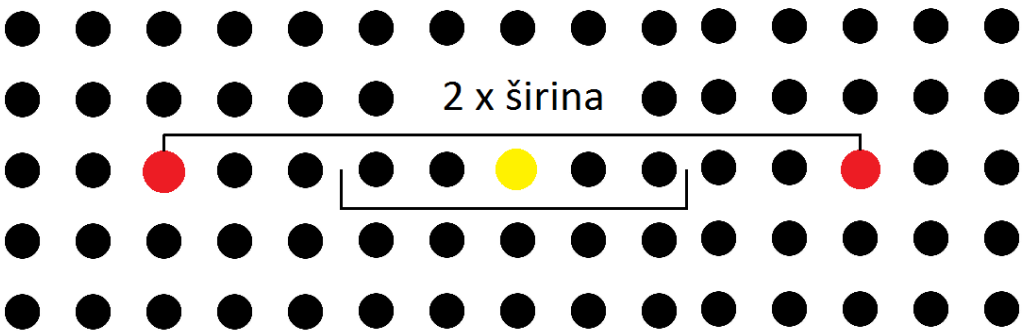
```
startnaPozicija = (gornjiV + lijeviV) / 2.0;
horizontalniVektor = (desniV - lijeviV) / (2.0 * sirinaDijela);
vertikalniVektor = (donjiV - gornjiV) / (2.0 * visinaDijela);

startnaPozicija += horizontalniVektor / 2.0;
startnaPozicija += vertikalniVektor / 2.0;

za (j = 0; j < visinaDijela; j++)
{
   za (i = 0; i < sirinaDijela; i++)
   {
      cestice[j * sirinaDijela + i].cilj = startnaPozicija +
                        (j) * vertikalniVektor + (i) * horizontalniVektor;
   }
```

```
}

za svaku cesticu c u cesticama
	c.odrediNoviPolozaj();
```

pri čemu funkcija *odrediNoviPolozaj* određuje sile koje djeluju na česticu ovisno o udaljenosti od ciljnog položaja te pomoću tih sila određuje novi položaj. Kao i u prethodnom algoritmu, to je moguće izvršiti unutar jedne funkcije, pošto položaj jedne čestice ovisi isključivo o njenom ciljnom položaju, a ne o položaju drugih čestica.

Kod traženja horizontalnog i vertikalnog vektora, dodatno je bilo potrebno svesti taj vektor na polovinu svoje početne vrijednosti, pošto udaljenost između gornjeg i donjeg te lijevog i desnog centra mase zapravo obuhvaća dvostruko više čestica od jednog dijela, kao što se može vidjeti na slici 8, gdje je prikazano kako je širina jednog dijela dvostruko kraća od udaljenosti njenih susjednih centara mase.

Također je potrebno početan položaj u svakom dijelu pomaknuti za polovinu duljine horizontalnih i vertikalnih vektora kako ne bi došlo do situacije u kojoj su rubne čestice susjednih dijelova spojene. Jedna od prednosti ovakvog načina određivanja ciljnih pozicija čestica u odnosu na opisan u prošlom algoritmu je i činjenica da dimenzije konačnog oblika dijela nisu konstantne već je moguće da se, ukoliko je zbog djelovanja vanjske sile jedan dio tkanine komprimiran, čestice koje čine taj dio približe jedna drugoj.



Slika 8 - Omjer širine jednog od dijelova tkanine te udaljenost između njemu susjednih centara mase

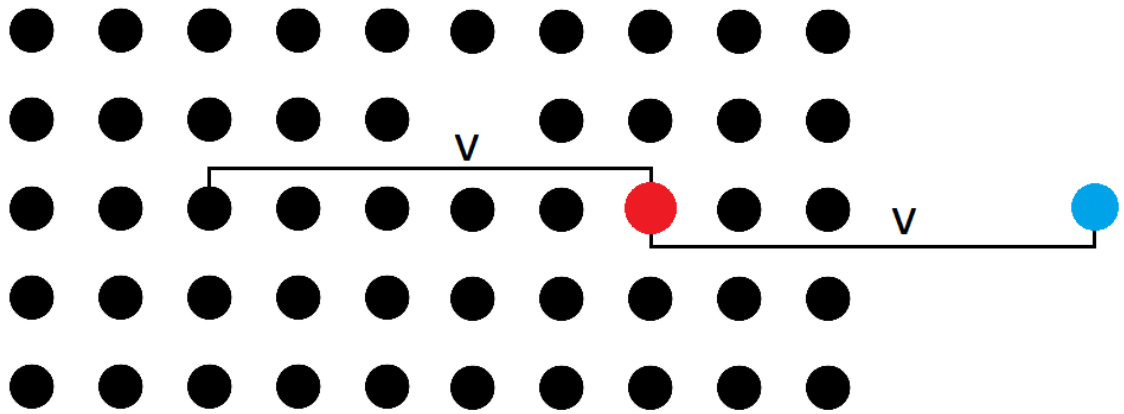
Jedan očiti nedostatak je činjenica da se prilikom računanja ciljnog položaja čestica uopće ne koristi trenutni položaj centra mase tog dijela, već samo njemu susjedni položaji. Moguća korekcija tog problema bi bila da se nakon izračuna ciljnih položaja svih čestica odredi njihov stvarni centar mase te vektor koji vodi od tog „stvarnog“ centra mase prema onom koji je zapisan u memoriji za taj dio. Tako određen vektor bi se primijenio na sve čestice u dijelu, s dodatnim uvjetom da se utjecaj tog vektora smanjuje ovisno o udaljenosti čestica od stvarnog centra mase dijela. Time bi kretanje tkanina dobilo još prirodniji izgled.

Također, očit problem u prethodno opisanom postupku se javlja u slučaju dijelova tkanine uz sam rub, pošto oni nemaju nužno sva četiri susjeda. Kao primjer uzmimo situaciju u kojoj dio ima susjede s lijeve, gornje te donje strane, a nema susjeda s desne strane. Potrebno je pronaći „lažni“ centar mase koji bi simulirao položaj desnog susjeda.

U tom slučaju uzimamo vektor koji vodi od lijevog susjeda prema centru mase dijela kojeg promatramo te pomičemo centar mase za taj isti vektor u desnu stranu. Tako nađenog susjeda možemo primijeniti kako bismo po prethodno opisanom postupu postupku pronašli ciljne pozicije za čestice unutar dijela. Postupak je prikazan na slici 9, a pseudokod koji opisuje kako provodimo taj postupak glasi:

```
ako (lijeviSusjed != null)) lijeviV = lijeviSusjed.centarMase;
inace
{
   privremeniV = centaMase - desniSusjed.centarMase;
   lijeviV = centarMase + privremeniV;
}

ako (desniSusjed != null)) desniV = desniSusjed.centarMase;
inace
{
   privremeniV = centaMase - lijeviSusjed.centarMase;
   desniV = centarMase + privremeniV;
}

ako (gornjiSusjed != null)) gornjiV = gornjiSusjed.centarMase;
inace
{
   privremeniV = centaMase - donjiSusjed.centarMase;
   gornjiV = centarMase + privremeniV;
}

ako (donjiSusjed != null)) donjiV = donjiSusjed.centarMase;
inace
{
```

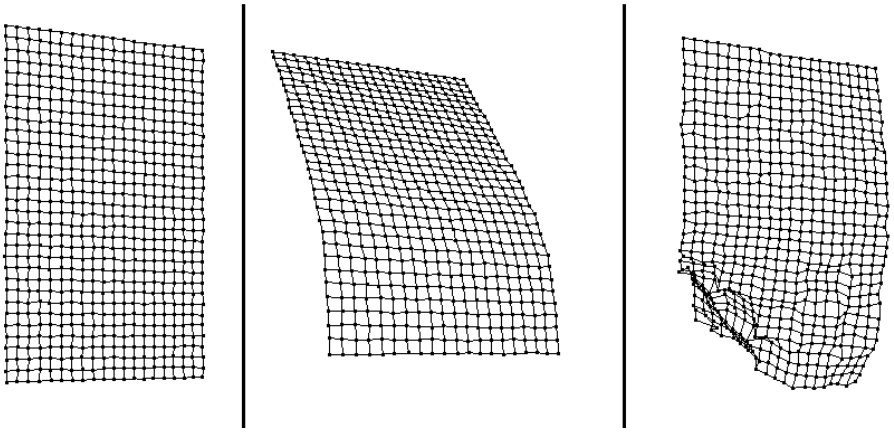
```
   privremeniV = centaMase - gornjiSusjed.centarMase;
   donjiV = centarMase + privremeniV;
}
```



Slika 9 - Prikaz nalaženja "lažnog" centra mase

Očita mana ovog postupka jest nužan preduvjet da se tkanina po svakoj dimenziji sastoji od barem dva dijela. Postoji nekoliko načina na koje bi se riješio problem za dimenzije veličine jedan, ali jednostavnije rješenje bi u tom slučaju bilo jednostavno smanjiti veličinu dijelova, barem po toj jednoj dimenziji.

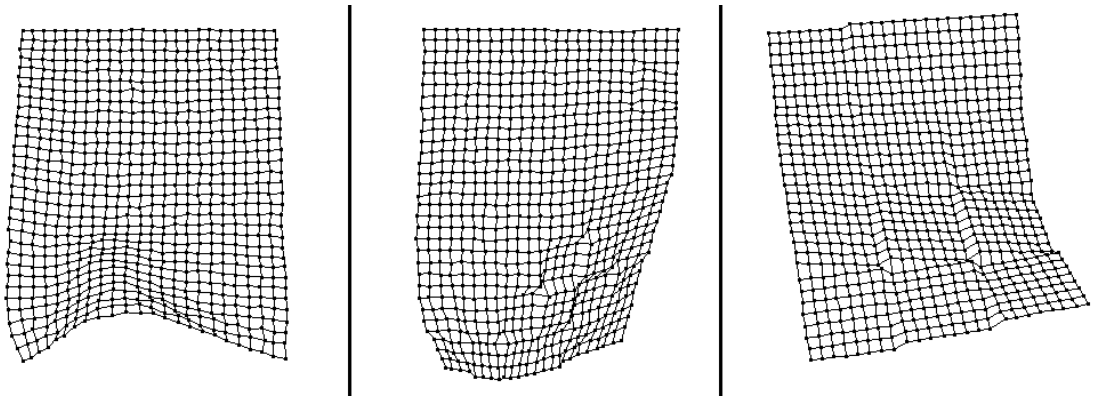
Kao i u prethodnom algoritmu, nakon što je pronađen ciljni položaj pojedinih čestica, na njih djeluje sila proporcionalna udaljenosti od tog ciljnog položaja. Važno je uskladiti sile koje vraćaju čestice prema njihovim ciljnom položajima s onima koje pomiču centre mase dijelova tkanine. Ukoliko to nije napravljeno, može doći do prevelikih udaljenosti između čestica i ciljnih položaja, koje dovode do prevelikih sila te, posljedično, do raspada dijelova tkanine.



Slika 10 - Prikaz izvođenja hibridnog algoritma

Na slici 10 je prikazano izvođenje hibridnog algoritma. Iz samog primjera je očito kako je riječ o velikom napretku u odnosu na drugi algoritam što se tiče kvalitete prikaza. I dalje se može vidjeti plosnatost na određenim dijelovima, ali uz kvalitetno implementirano sjenčanje taj će se problem znatno manje osjećati.

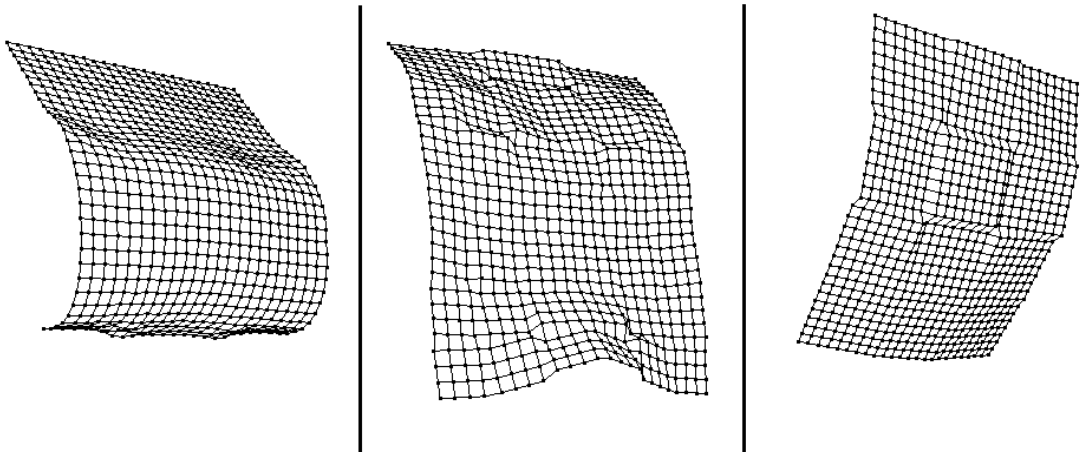
Također je moguće smanjiti dimenzije dijelova kako bi se povećala kvaliteta prikaza, odnosno povećati dimenzije dijelova kako bi se ubrzao rad programa. Naravno, dio s dimenzijama 1x1 bi bio ekvivalent sustavu masa i opruga kakav smo već opisali, dok bi korištenje samo jednog dijela odgovaralo modelu opisanom u prošlom poglavlju. Na slici 11 je prikazan rad algoritma za dimenzije dijelova 2x2 (lijevi), 4x4 (srednji) te 8x8 (desni).



Slika 11 - Prikaz ovisnosti kvalitete prikaza ovisno o dimenziji dijelova unutar tkanine

Dodatna mogućnost za ubrzanje procesa je potpuno zanemariti titranja čestica oko njihovih ciljnih položaja, već se samo fokusirati na te ciljne položaje. Tako dobiven sustav zapravo tkaninu prikazuje kao niz „pločica“ koje su povezane

na rubovima. Kako taj sustav potpuno zanemaruje zadnji korak algoritma, dolazi do određenog ubrzanja rada programa, ali kvaliteta prikaza značajno pada, posebno ukoliko koristimo velike dimenzije dijelova.



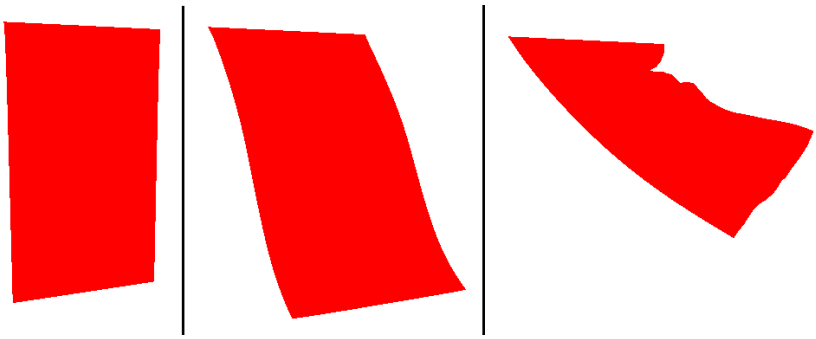
Slika 12 - Pojednostavljena verzija hibridnog algoritma

Za dio veličine 2x2 tkanina izgleda prihvatljivo, no za manje detaljnije uočavaju se nagli prijelazi između dijelova. Na slici 12 je prikazana tkanina koristeći navedeno pojednostavljenje uz veličinu dijelova 2x2 (lijevo), 4x4 (u sredini) i 8x8 (desno).

3.5. Sjenčanje

U prošlim poglavljima je tkanina bila prikazana kao mreža čestica povezanih vezama. Takav prikaz je vrlo prigodan za prikazati kako se ponaša tkanina jer daje točan uvid u čestice te njihovo ponašanje, ali se u stvarnoj primjeni rijetko može vidjeti tako oblikovana tkanina.

Stvarna tkanina redovito sadrži barem jednu boju te vrlo često i neke uzorke na sebi. Bojanje pomoću OpenGLa nije pretjerano zahtjevno, samo je potrebno umjesto točaka iscrtavati poligone, trokute ili kvadrate, a pomoću odgovarajuće naredbe promijeniti boju koja se iscrtava.



Slika 13 - Izgled jednolično obojane tkanine

Problem s takvim postupkom je što će u tom slučaju cijela tkanina biti jednolično obojana, što ne daje realan prikaz, kao što se može vidjeti na slici 13. Kako bi tkanina izgledala realno, potrebno je uvesti neki oblik sjenčanja kako bi se jasno vidjeli naborani dijelovi tkanine. Postupak korišten za sjenčanje će biti opisan u ovom poglavlju.

Dva ključna pojma vezana uz sjenčanje su model osvjetljenja te postupak sjenčanja. Model osvjetljenja označava postupak koji koristimo kako bismo izračunali intenzitet svjetlosti u promatranoj točki. Postupak sjenčanja određuje kako ćemo osjenčati površine uz odabrani model osvjetljenja.

Model osvjetljenja koji je korišten u ovom radu je Phongov model osvjetljenja, koji aproksimira osvjetljenje određenog dijela objekta kao kombinaciju ambijentalne, difuzne te zrcalne komponente svjetlosti. Korišteni postupak sjenčanja poligona je Gouraudov postupak koji se temelji na računanju osvjetljenja u vrhovima na temelju normala u tim vrhovima te dobiven intenzitet svjetlosti interpolira na poligonu.

Ambijentalna komponenta je konstantna u cijeloj sceni te nastaje kao rezultat višestrukog odbijanja zraka svijetlosti. Zbog te komponente nijedan dio scene neće biti potpuno taman. Formula kojom se računa intenzitet ambijentalne komponente glasi:

$$I_g = k_a I_a \tag{24}$$

gdje je k_a koeficijent čija je vrijednost između 0 i 1, specifičan za svaki materijal koji određuje koliko će ambijentalne svjetlosti I_a biti apsorbirano, a koliko reflektirano.

Difuzna komponenta je određena kutom pod kojim svjetlost upada na poligon. Formula za izračun difuzne komponente svjetlost glasi:

$$I_d = I_i k_d \cos\varphi \tag{25}$$

gdje je I_i intenzitet točkastog izvora, k_d koeficijent specifičan za materijal, također vrijednosti između 0 i 1, koji određuje koliko će se difuzne svjetlosti reflektirati, a φ kut između normale promatranog vrha i vektora od izvora do promatrane točke.

Izračun kosinusa kuta φ nije nužno potrebno raditi izravno, preko traženja kuta između vektora, već je dovoljno normalizirati oba vektora te izračunati njihov skalarni produkt, koji će biti jednak tom kosinusu. U tom slučaju formula za izračun difuzne komponente glasi:

$$I_d = I_i k_d (\vec{l} \cdot \vec{n}) \tag{26}$$

gdje *l* označava vektor od izvora svjetlosti do promatranog vrha, a *n* normalu u tom vrhu.

Zrcalna komponenta je određena kutom između reflektirane zrake i zrake prema promatraču. Formula za izračun zrcalne komponente glasi:

$$I_s = I_i k_s (\cos\alpha)^n \tag{27}$$

gdje je I_i, kao i prilikom izračuna difuzne svjetlosti, intenzitet točkastog izvora, k_s koeficijent s vrijednostima između 0 i 1 koji određuje koliki dio zrcalne komponente će se reflektirati, α kut između reflektirane zrake te zrake prema promatraču, a *n* indeks koji opisuje grubоću površine i njezina reflektirajuća svojstva.

Kao i kod difuzne komponente, ukoliko imamo jedinične vektore, moguće je pojednostaviti formulu kao:

$$I_s = I_i k_s (\vec{r} \cdot \vec{n})^n \tag{28}$$

gdje je *r* vektor dobiven reflektiranjem vektora *l*, koji smo koristili prilikom računanja difuzne komponente. Reflektirani vektor se može dobiti kao:

$$\vec{r} = \vec{l} - 2(\vec{l} \cdot \vec{n})\vec{n} \tag{29}$$

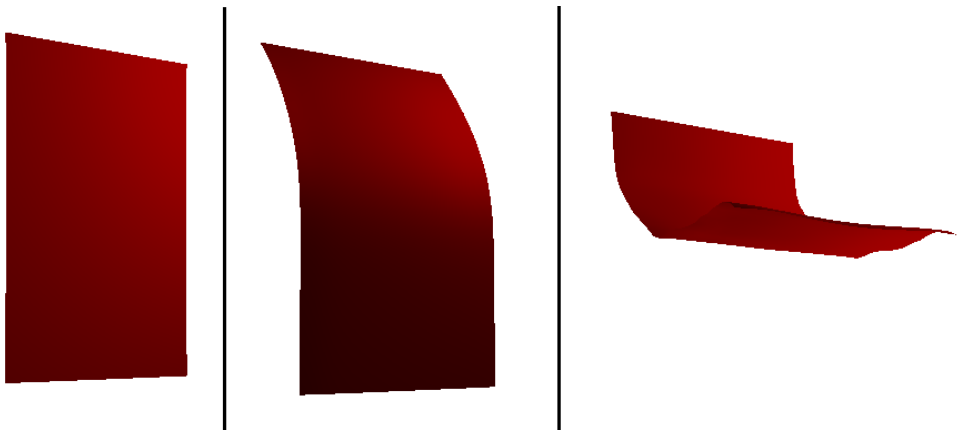
Ukupna formula za računanje osvjetljenja u jednoj točki, koja kombinira ambijentalnu, difuznu i zrcalnu komponentu, glasi:

$$I = k_a I_a + I_i(k_d(\vec{l} \cdot \vec{n}) + k_s(\vec{r} \cdot \vec{n})^n) \tag{30}$$

gdje je važno razlikovati *n* kao vektor koji predstavlja normalu vrha te *n* kao potenciju koja predstavlja reflektirajuća svojstva materijala.

Tijekom opisa računanja osvjetljenja u pojedinim točkama, često se spominjao pojam normale u vrhu poligona. Normala u nekom vrhu u tkanini se računa tako da se gledaju normale u vrhovima susjednih poligona (manjih kvadrata ili trokuta koji čine tkaninu) te se računa prosjek njihovih normala.

Normala jednog poligona se računa kao vektorski produkt dva njegova brida, jer normala je okomita na sve bridove poligona. U našem slučaju, ako imamo poligon koji se sastoji od tri vrha, gledamo vektore koji vode od jednog vrha prema druga dva te tražimo njihov vektorski produkt.



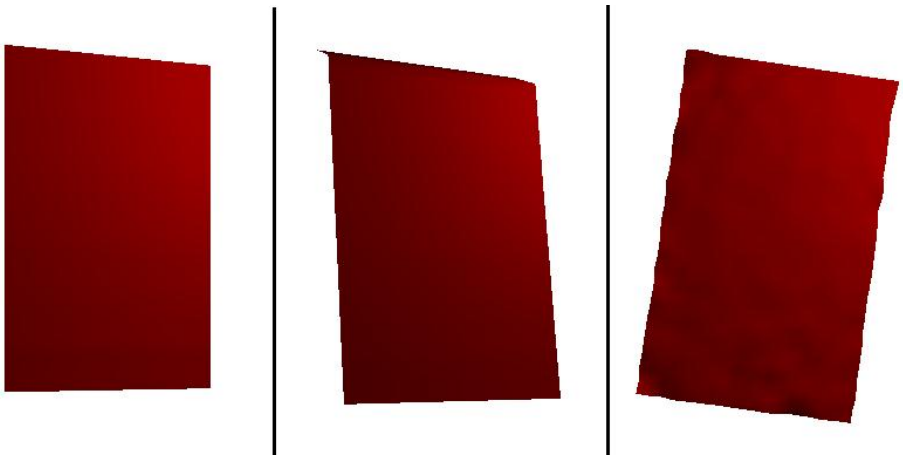
Slika 14 - Primjer rada prvog algoritma uz primjenu Gouraudovog sjenčanja

Kako se tkanina sastoji od niza čestica, a ne poligona, bilo je potrebno odrediti način na koji će se iscrtavati poligoni. Zato se za svaki četverokut koji čini jedan dio tkanine uzela njegova donja desna točka kao referentna za taj poligon. Donja desna točka se činila kao najlogičniji izbor zbog načina stvaranja tkanine (počevši od gornje lijeve točke pa nadolje), jer je ona mogla biti povezana s već postojećim točkama.

Prilikom samog sjenčanja, prvo se za sve čestice, izuzev onih u gornjem redu i lijevom stupcu koje ne posjeduju „svoje“ poligone, pozivala metoda koja je na već opisan način računala normale poligona. Nakon toga se za sve čestice pozivala metoda koja pomoću normala poligona računa normale pojedinih vrhova poligona.

Samo sjenčanje se vrši nakon što su sve potrebne normale izračunate. Za sve čestice koje posjeduju svoje poligone se poziva metoda za bojanje tog poligona. U toj metodi se za dohvaćaju lijevi i gornji susjed te čestice, a zatim, pomoću ta dva i njezin susjed po dijagonali. Za svaki od vrhova se računa vektor od izvora svjetlosti prema tom poligonu te pomoću njega vektor reflektirane zrake prema gore navedenoj formuli.

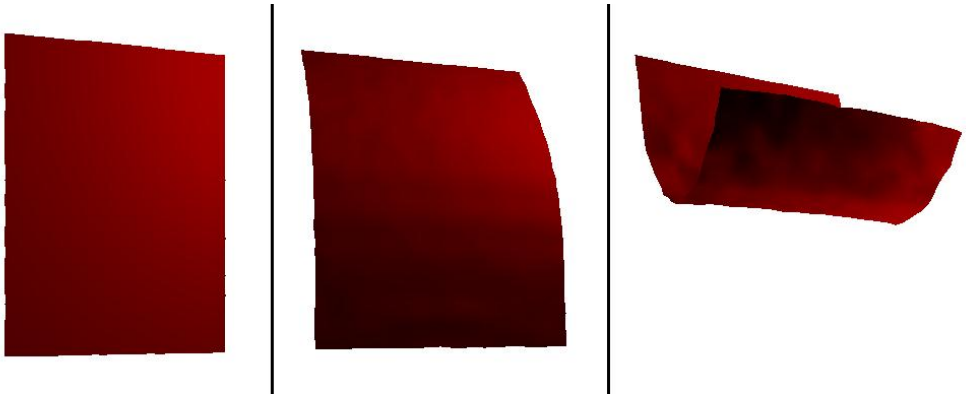
Ambijentalnu komponentu osvjetljenja je dovoljno izračunati na početku procesa, jer će ona biti konstantna za cijelu tkaninu, kako je rađena od jednog materijala. Difuznu komponentu računamo pomoću skalarnog produkta vektora normale i vektora od izvora svjetlosti prema vrhu poligona, a zrcalnu komponentu pomoću skalarnog produkta normale te reflektirane zrake. Moguće je umjesto normale vrha koristiti normalu poligona, ali rezultati dobiveni tim pristupom su znatno slabiji.



Slika 15 - Primjer rada drugog algoritma uz primjenu Gouraudovog sjenčanja

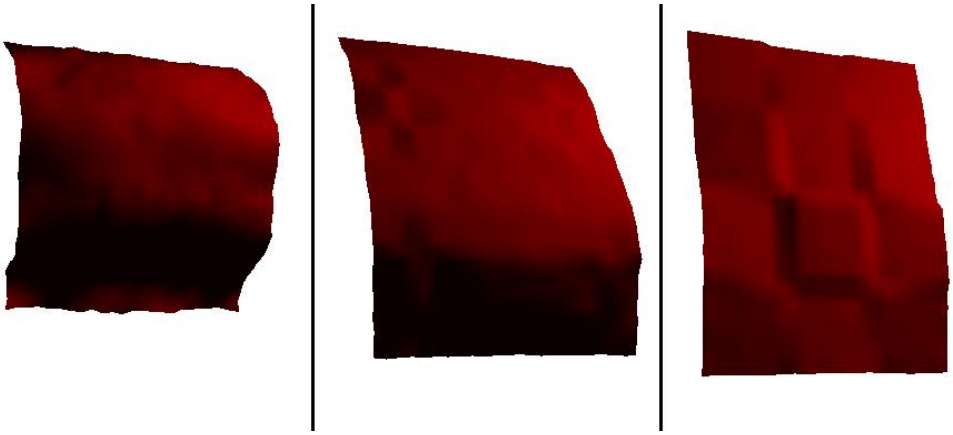
Sumirajući te tri komponente dobivamo osvjetljenje pojedinog materijala. Ukoliko je kosinus nekog od kutova između vektora ulazne svjetlosti i vektora normale manji od 0, potrebno ga je postaviti na nulu kako bi poligon bio osvijetljen samo ambijentalnom komponentom svjetlosti.

Na primjerima prikazanim na slikama 14, 15, 16 i 17 plava i zelena RGB komponenta su postavljene na nulu, a osvjetljenje mijenja crvenu komponentu, tako da su jače osvijetljeni dijelovi crveni, a slabije osvjetljeni dijelovi crni.



Slika 16 - Primjer rada hibridnog algoritma uz primjenu Gouraudovog sjenčanja

Jedan manji problem je što se tkanina prikazuje samo pomoću jednog poligona te su njena prednja i stražnja strana uvijek jednako osvijetljene. Kvalitetnije rješenje bi bilo da se za svaki četverokut stvore dva poligona, jedan koji će se iscrtavati s prednje, a drugi sa stražnje strane četverokuta. Tako bismo uvijek imali jednu osvjetljenu stranu, a jednu potpuno mračnu, odnosno osvijetljenu samo ambijentalnom komponentom.



Slika 17 - Primjer rada hibridnog algoritma uz Gouraudovo sjenčanje za različite veličine dijelova

Na slici 17 je prikazana razlika između rada hibridnog algoritma s dijelovima veličine 2 (lijevo), 4 (u sredini) i 8 (desno). I dalje je vidljiva razlika u kvaliteti,

posebno se ističu „grbe“ kod verzije sa 8 čestica po dijelu, ali su dobiveni rezultati za 2 i 4 čestice vrlo prihvatljivi.

4. Modificiranje parametara

Kao što je već opisano u poglavlju 2.1, kompliciraniji modeli tkanina sastoje se od više desetaka parametara. Veći broj parametara je pokazatelj kvalitetnijeg modela, model s više parametara će u pravilu kvalitetnije simulirati željeni stvaran objekt, u ovom slučaju tkaninu.

Kao primjer uzmimo ponovno model Huamina Wanga, Jamesa O'Briena i Ravija Ramamoorthija, opisan u njihovom radu iz 2011. [2]. Kako bi pronašli model rastezanja tkanine, oni promatraju dva tenzora, ε koji predstavlja bezdimenzijski tenzor naprezanja te σ koji predstavlja tenzor pritiska, a prikazuje silu po dimenzijama površine tkanine.

Ta dva tenzora povezuju simetričnom matricom C, dimenzija 3x3, koja predstavlja prelazak iz bezdimenzijskog tenzora ε u tenzor σ koji služi kako bi se računale sile u tkanini. Iako simetrična matrica dimenzija 3x3 sadrži 6 različitih elemenata, Boisse, Gasser i Hivet su u svom radu iz 2001. [5] pokazali kako je moguće pojednostaviti matricu tako da ima samo 4 različita parametra te konačna formula za model rastezanja glasi:

$$\sigma = \begin{bmatrix} c_{11} & c_{12} & 0 \\ c_{12} & c_{22} & 0 \\ 0 & 0 & c_{33} \end{bmatrix} \varepsilon = C\varepsilon \tag{31}$$

u kojoj c_{11}, c_{12}, c_{22} te c_{33} predstavljaju tražene parametre, a ε i σ prije opisane tenzore. Moguće pojednostavljenje modela bi bilo korištenje konstantne matrice C za tkaninu, ali za potrebe rada je pretpostavljeno kako je matrica funkcija tenzora naprezanja ε. Kako u radu promatraju tkaninu preko 6 ključnih točaka, dolazimo do ukupnog broja od 24 parametra za rastezanje tkanine.

Tome treba dodati i 15 parametara korištenih za dobivanje modela savijanja tkanine. Kao što je bilo prethodno objašnjeno, za ovaj rad nije bilo izvedivo napraviti tako kompliciran model tkanina. U ovom poglavlju će se prikazati kako

modifikacija malog broja parametara u tri obrađena modela mijenja svojstva tkanina.

Kako u radu prikazujemo tri različita modela, za svaki od njih postoje zasebni parametri. Model elastičnih sustava temeljenih na česticama je najjednostavniji glede broja parametara, pošto ne postoje nikakve veze između čestica, a jedini parametri određuju brzinu kojom se pojedina čestica vraća prema svojem konačnom položaju, potencija koja određuje odnos između udaljenosti čestice od konačnog položaja i sile koja djeluje na istu te prigušenje gibanja čestica.

Model temeljen na sustavu masa i opruga je srednje kompliciran što se broja parametara tiče, u njemu gledamo tri različite vrste veza između čestica kao i sile koje određuju prigušenje gibanja unutar tkanine. Najviše parametara sadrži hibridni model, pošto se u njemu koriste parametri iz oba prethodno opisana modela.

U tom modelu je moguće je razdvojiti prigušenje za pojedinačne čestice od prigušenja za cijelu tkaninu, odnosno dijelove koji čine tkaninu, ali je potrebno paziti da ne dođe do prevelikih odstupanja čestica od njihovih ciljnih položaja što bi moglo dovesti do pretjeranog titranja koje kvari prikaz tkanine.

Prigušenje unutar tkanine ima jednak efekt na sve modele. To je faktor koji određuje kolika sila djeluje na čestice u smjeru suprotnom od smjera gibanja čestice. Drugim riječima, ako imamo faktor prigušenja *k* te brzinu kretanja čestica *v*, sila prigušenja koja djeluje na česticu se može zapisati kao:

$$F = -kv \tag{31}$$

Povećanje faktora prigušenja dovodi do veće krutosti tkanine. Razlika između tkanine u kojoj je faktor prigušenja minimalan, ako uopće postoji i one u kojoj je nekoliko desetaka puta veći se može poistovjetiti s onom između obične zastave ili lakšeg komada odjeće te težih zastora.

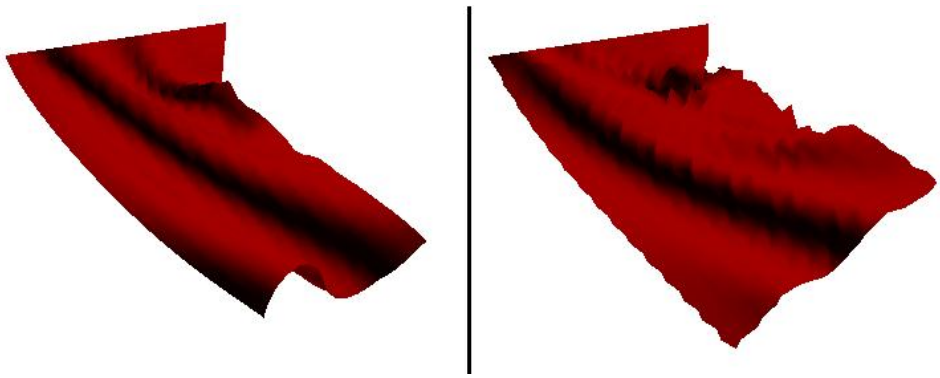
Ostali parametri nemaju tako značajan utjecaj na ponašanje tkanina već samo blago modificiraju njihova ponašanja. Uzmimo kao primjer sustav masa i opruga. Uz naveden faktor prigušenja u njemu se nalaze po tri parmetra:

konstanta strukturnih opruga, konstanta opruga striženja te konstanta opruga savijanja.

Kao što je već opisano, u pravilu su konstante strukturnih opruga veće od preostale dvije konstante. Prihvatljiv omjer između konstanti bi bio 2:1:1 koji daje kvalitetna rješenja te su sve dosad prikazane slike tog modela koristile taj omjer parametara.

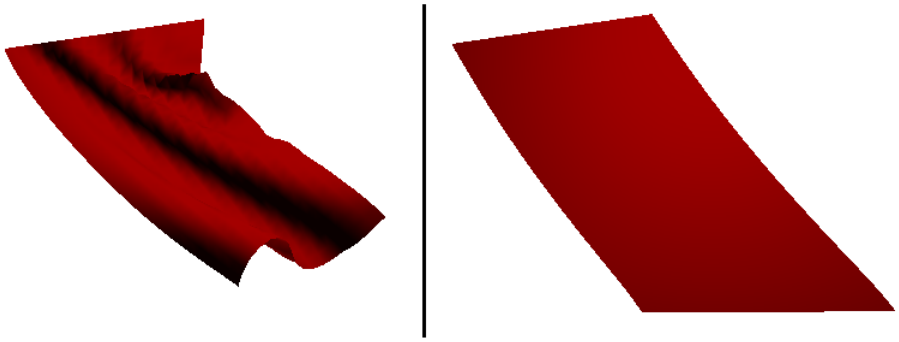
Zanimljivo je promatrati kakav utjecaj na izgled tkanine ima koji od tih parametara. Kao što je ukratko komentirano pri kraju poglavlja 2.2, ovdje će se proučiti kakav efekt ima određena opruga, odnosno izostanak iste, na ponašanje tkanine.

Kao što im ime govori, strukturne opruge pomažu tkaninama zadržati oblik. Izostanak istih se može vidjeti po tome da je tkanina sporija nego inače kad je u pitanju vraćanje u početan položaj nakon deformacije. Razlika se jasno može vidjeti na slici 18 gdje je s lijeve strane prikazana tkanina sa sve tri opruge, a s desne strane tkanina u vrlo sličnoj situaciji bez strukturnih opruga.



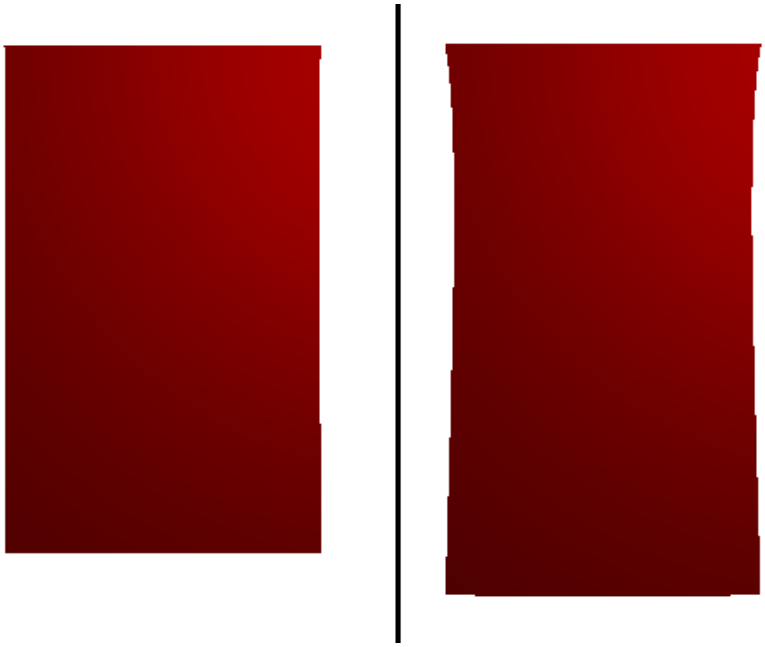
Slika 18 - Izgled tkanine sa i bez strukturnih opruga

Izostanak opruga striženja ima zanimljiv efekt da potpuno onemogući stvaranje nabora na tkaninama. Tkanina se u tom slučaju ponaša vrlo slično onoj iz modela temeljenog na sustavu čestica. Na slici 19 se može vidjeti razlika između tkanine sa sve tri opruge, koja se nalazi lijevo na slici, i tkanine bez opruga striženja, desno na slici.



Slika 19 - Izgled tkanine sa i bez opruga striženja

Opruge savijanja, unatoč imenu, zapravo služe za sprečavanje situacije u kojoj se opruga jednostavno savija. Najbolji primjer za razliku u izgledu tkanina sa i bez tih opruga je izgled tkanine na početku, kao što se može vidjeti na slici 20. Na toj slici s lijeve strane je tkanina koja sadrži sve tri vrste opruga, a na desnoj strani je tkanina bez opruga savijanja. Vrlo se jasno vidi kako se tkanina na desnoj strani savila prema unutra. Tijekom samog rada programa može se vidjeti kako nastaje puno više nabora nego u prijašnjim slučajevima.



Slika 20 - Izgled tkanina sa i bez opruga savijanja

Važno je napomenuti kako su ovdje prikazani isključivo ekstremni slučajevi, odnosno razlika između tkanina koje sadrže odnosno ne sadrže određene opruge.

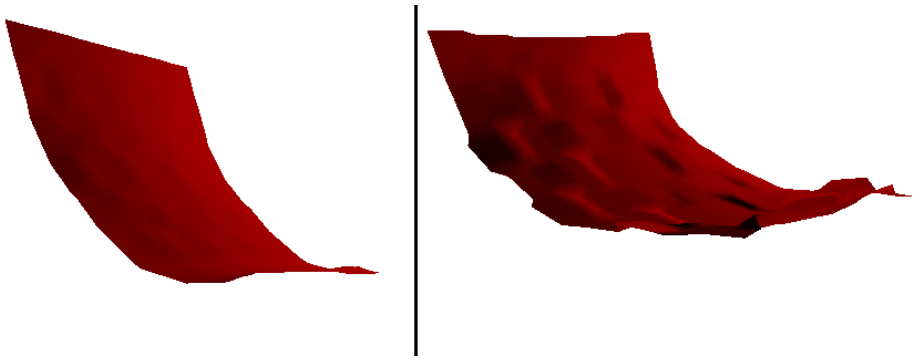
U realnim slučajevima, ovisno o tome kakvu tkaninu pokušavamo prikazati, potrebno je interpolirati između ekstrema kako bi se našao optimalan prikaz tkanine.

Model temeljen na elastičnim modelima temeljenim na sustavima čestica ovisi o svega nekoliko parametara. Kao i u prvom algoritmu, prigušenje razlikuje „teže“ tkanine od „lakših“. Koeficijent svake čestice je vrlo sličan koeficijentima opruga iz prvog algoritma, možemo ga opisati kao koeficijent opruge koja povezuje česticu s njenim ciljnim položajem. Veći koeficijent znači da će se čestica brže vraćati prema svojem ciljnom položaju, odnosno njen period titraja će biti kraći.

Konačno, kao parametar algoritma bi se mogla izdvojiti i potencija koja određuje kakva je veza između sile i udaljenosti čestica od njihovog ciljnog položaja. Ista potencija bi se mogla koristiti i u sustavu masa i opruga kako bi se dodatno reguliralo ponašanje tkanina. Potencija veća od 1 dovodi do više gibanja te nešto fleksibilnije tkanine, dok potencija manja od 1 čini tkaninu čvršćom.

Dodatno, potrebno je paziti da potencija ostane unutar nekih granica, jer u suprotnom zbog prevelikog ili premalog iznosa sila dolazi do prevelikih udaljenosti između čestice i njenog ciljnog položaja, što kvari prikaz tkanina. U ovom algoritmu, kao i u hibridnom za kojeg će parametri biti naknadno komentirani, prihvatljiva vrijednost potencija se u pravilu nalazila između 0.5 te 3.

Kao i u sustavu masa i čestica, u hibridnom algoritmu postoje tri vrste veza koje povezuju dijelove tkanine. Strukturne opruge drže tkaninu kompaktnom, njihova uloga je još značajnija pošto se tkanina bez strukturnih opruga još lakše deformira nego u prvom algoritmu, kao što se može vidjeti na slici 21.



Slika 21 - Izgled hibridnog algoritma sa i bez strukturnih opruga

Uloge koju u hibridnom algoritmu imaju opruge striženja i savijanja su vrlo slične onima koje imaju u modelu masa i čestica. Ukoliko uklonimo opruge striženja, tkanina skoro u potpunosti gubi nabore koji se stvaraju na njoj pod utjecajem vanjskih sila, slično kao na slici 19.

Ukoliko uklonimo opruge savijanja, nestaje sila koja sprječava savijanje tkanina te se one često same od sebe savijaju, što je ponovno uočljivo i pri samom pokretanju programa. Dakako, pošto svaki dio koristi niz individualnih čestica, efekt nije toliko uočljiv kao što je bio u sustavu masa i opruga.

Razlika u izgledu programa kad je u pitanju sustav masa i čestica i hibridni algoritam može se vidjeti na slici 22, gdje je s lijeve strane prikazan sustav masa i opruga, a s desne strane hibridni algoritam. U oba slučaja potpuno su izostavljene opruge savijanja te je uočljivo kako je savijanje tkanine prema unutrašnjosti izraženije na lijevoj slici. Također, na desnoj slici su vidljive granice između pojedinih dijelova tkanine, što kvari izgled, no to je posljedica činjenice da tkaninu gledamo iz neposredne blizine u statičnom položaju, što ne bi bio slučaj u većini animacija.

Slika 22 - Razlika u izgledu sustava masa i čestica te hibridnog algoritma pri izostanku opruga savijanja

Kao i u modelu temeljenom na sustavu čestica, u hibridnom algoritmu je važno obratiti pozornost na to koliki će biti koeficijent „opruge“ kojom se čestica vraća u početni položaj. Kao što je i prije opisano, veći koeficijent će smanjiti period titranja čestice oko svog ciljnog položaja.

U praktičnim primjerima, tkanine s većim koeficijentima titranja u pravilu izgledaju „lakše“, stvara se više nabora, a tkanina djeluje prirodnije. Negativna strana većih koeficijenata titranja je, što u slučaju kada se tkanini dodijeli prevelik koeficijent, dolazi do previše titraja i tkanina u početnom položaju djeluje nestabilno. Takav efekt se može donekle ispraviti mijenjanjem koeficijenta prigušenja.

5. Ocjena brzine izvođenja

Kao i u većini grafičkih aplikacija, i u ovom radu je iznimno važno da se program izvršava uz prihvatljivu brzinu izvođenja. Najčešće se kao mjera brzine izvođenja uzima broj sličica koje se iscrtaju u programu u sekundi, poznatiji kao frames per second (*fps*).

U programu koji je napravljen za potrebu ovog rada, pri svakom pozivanu *OnUpdateFrame* metode, koja vrši novi proračun izgleda scene, se provjerava koliko je vremena proteklo od zadnjeg poziva iste metode. Dobiven period se dijeli s fiksnim brojem milisekundi koji označava jedan „korak“ u izvođenju programa.

U radu je kao korak korišten period od 16 milisekundi, koji približno odgovara frekvenciji određivanja novih položaja predmeta na sceni od 60 sličica po sekundi. Ta frekvencija je najčešće korištena kao mjerilo pri izvođenju raznih računalnih grafičkih aplikacija te je zato korištena i u ovom radu.

Ovakav način određivanja koliko će predmeti u sceni biti pomaknuti u kojem pozivu *OnUpdateFrame* metode omogućava stabilnost u slučaju da dođe, na primjer, do smrzavanja programa na nekoliko sekundi. Problem u tom slučaju predstavlja činjenica da je donekle otežano mjerenje broja iscrtavanja u sekundi pošto period između dva iscrtavanja ovisi o broju koraka koji su provedeni između ta dva iscrtavanja.

Tako se mjerenjem dobivaju naizmjenični skokovi u brzini izvođenja programa te je preciznije praćenje frekvencije iscrtavanja otežano i neprecizno. Kako u sva tri algoritma proces samog iscrtavanja na ekran uzima približno jednako vrijeme, dovoljno je bilo gledati koliko će trajati računanje novog položaja čestica u jednom koraku *OnUpdateFrame* metode.

Za potrebu mjerenja, sva tri algoritma su bila pokrenuta sa brojem čestica koji predstavlja potencije broja dva. Kako je za 1, 2, 4, 8, 16 te 32 čestice algoritam radio maksimalnom brzinom u sva tri slučaja, oni nisu bili razmatrani. Uz veće potencije su dobiveni rezultati znatno više varirali te su dali jasne dojmove o brzini izvođenja pojedinih algoritama.

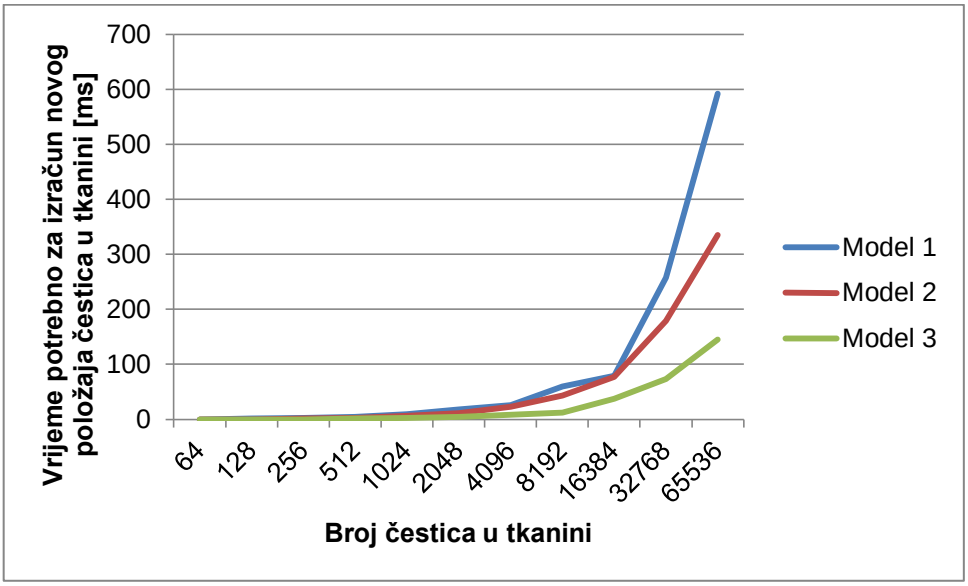
Dodatno, hibridni algoritam je bio testiran s dijelovima veličine 4, 16 te 64 čestica kako bi se dodatno analizirali koja je od te varijante najpraktičnija za korištenje. Svi rezultati su prikazani na slici 23, gdje *Model 3a* predstavlja hibridni model uz dimenzije dijelova od 4 čestice, *Model 3b* predstavlja isti sa 16 čestica te *Model 3c* sa 64 čestice.

	64	128	256	512	1024	2048	4096	8192	16384	32768	65536
Model 1	0	1	2	4	9	18	26	60	79	257	592
Model 2	0	0	1	2	6	11	23	43	77	179	335
Model 3a	0	0	1	1	3	7	14	29	63	120	237
Model 3b	0	0	0	1	2	4	8	12	37	73	145
Model 3c	0	0	0	0	1	3	5	13	31	62	127

Slika 23 - Usporedba vremena izvođenja modela

U slučajevima kad je brzina izvođenja približno jednaka nuli program se izvršava sa, ili vrlo blizu, očekivanih 60 sličica u sekundi. S porastom vremena izvođenja *OnUpdateFrame* metode, ta brzina značajno opada te se, na primjer, model masa i čestica s najvećim isprobanim brojem čestica izvršava s frekvencijom od svega par sličica u sekundi.

Kao što je i očekivano, model temeljen na sustavu masa i čestica je davao najsporije rezultate. Suprotno očekivanjima, model temeljen na metodi elastičnih modela temeljenih na sustavu čestica se pokazao sporijim od sve tri varijante izvođenja hibridnog modela.



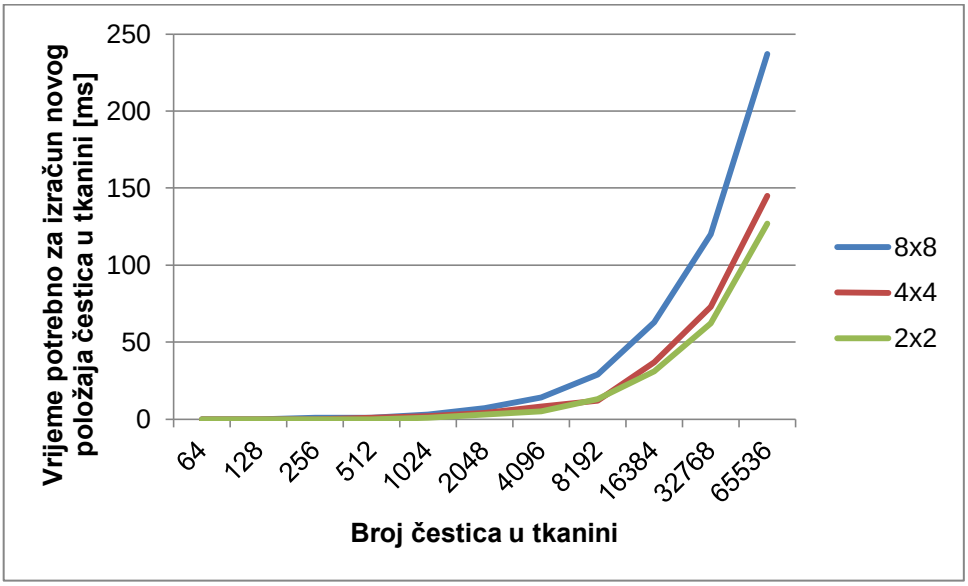
Slika 24 - Grafički prikaz usporedbe vremena izvođenja modela

Objašnjenje za to bi se moglo naći u činjenici da je metoda traženja prosječnog kvaterniona bila matematički daleko najzahtjevniji dio gledajući sva tri algoritma te je posebno usporavala program kad se s većim brojem čestica treba tražiti prosjek sve većeg broja kvaterniona.

Hibridni model ne koristi te komplicirane metode traženja prosječne rotacije već to rješava jednostavnim aproksimacijama što dovodi do znatnog ubrzanja rada u odnosu na model temeljen na sustavu čestica. Usporedba ta tri modela se jasnije može vidjeti na slici 24.

Rezultati dobiveni mjerenjem različitih varijanti hibridnog algoritma su bili u skladu s očekivanjima. Iz tablice na slici 23 je vidljivo da je program osjetno ubrzan kao kompenzacija činjenici da takav model ipak ne izgleda kvalitetno kao sustav masa i čestica.

Kao najbolji balans između kvalitete prikaza i brzine izvođenja nameće se varijanta algoritma koja koristi dijelove veličine 16 čestica, odnosno dimenzija 4x4. Varijanta sa 4 čestice je po brzini izvođenja ipak nešto bliže modelu masa i opruga nego što bismo željeli, dok varijanta sa 64 čestice jednostavno ne izgleda dovoljno kvalitetno da bismo ju poželjeli koristiti, a ušteda na brzini izvođenja nije toliko značajna. Na slici 25 se mogu vidjeti razlike u brzini izvođenja te tri varijante algoritma.



Slika 25 - Grafička usporedba vremena izvođenja različitih varijanti hibridnog algoritma

Sva mjerenja su bila izvršena na istom laptopu na kojem je program bio izrađen. Na laptopu je korišten 32-bitni Windows 7 operacijski sustav, uz 1.5GHz procesor, 4GB procesorske memorije te Radeonovu HD 6470M grafičku karticu s 1GB memorije. Također, u svim mjerenjima je korišten model sa implementiranim sjenčanjem. Ukoliko bismo isključili sjenčanje, brzina izvođenja bi bila nešto veća.

6. Zaključak

U svom radu iz 1996. [6], Ng i Grimsdale su prikazali mnogo načina na koje je moguće prikazati tkanine u računalnoj grafici. Kako je prošlo skoro 20 godina otkad je rad napisan, vrlo je vjerojatno da se broj mogućih načina prikaza uvišestručio. Kao što je opisano u prvom poglavlju, riječ je o iznimno atraktivnom području unutar računalne grafike te je u ovom radu prikazano nekoliko načina simulacije tkanina.

Modeli koji su obrađivani u ovom radu su ostavili dosta različite dojmove. Model temeljen na sustavu masa i opruga je djelovao vrlo kvalitetno te su mu i performanse bile prihvatljive, iako i dalje najsporije od tri obrađivana modela. Model temeljen na elastičnim sustavima je potpuno podbacio gledajući očekivanja, ideja koja je bila njegova osnova se jednostavno pokazala prejednostavnom za simulaciju kompliciranijih tkanina.

Hibridni model je izgledao prihvatljivo, a brzina izvođenja mu je bila bolja od prijašnjih modela. Moguće je da bi taj model bio primjenjivan u raznim grafičkim aplikacijama u budućnosti, ali sigurno je da su mu potrebne određene modifikacije kako bi se umanjio dojam da je sastavljen od niza dijelova te dodatne optimizacije kako bi se većim dobitkom u brzini izvođenja kompenzirao potencijalni nedostatak u prikazu.

7. Literatura

[1] Müller , M., Heidelberger , B., Teschner , M., Gross , M. 2005. Meshless Deformations Based on Shape Matching, ACM. Trans. Graph.

[2] Wang, H., O'Brien, J., Mamamoorthi, R. 2011. Data-Driven Elastic Models for Cloth: Modeling and Measurement, University Of California, Berkely

[3] Mihajlović, Ž., Definiranje objekata, putanja i pokreta, http://www.zemris.fer.hr/predmeti/ra/predavanja/1_defin.pdf

[4] Čupić, M., Mihajlović, Ž. 2011. Interaktivna računalna grafika kroz primjere u OpenGLu, http://www.zemris.fer.hr/predmeti/irg/irg-knjiga.pdf

[5] Boisse, P., Gasser, A., Hivet, G. 2001. Analyses of fabric tensile behavior: determination of the biaxial tension-strain surfaces and their use in forming simulations.

[6] Ng, H. N., Grimdsdale, R. L. 1996. Computer Graphics Techniques for Modelling Cloth

[7] Hauth, M., Etzmuss, O., Eberhardt, B., Klein, R., Sarlette, R., Sattler, M., Daubert, K., Kautz, J., 2002. Tutorial T3: Cloth Animation and Rendering, The Eurographics Association

[8] Terzepoulos, D., Platt, J., Barr, A., Fleischer, K., 1987, Elastically Deformable Mdels, Computer Graphics, Volume 21

[9] Sladović, J., 2012. Elastični modeli temeljeni na sustavu čestica, http://www.zemris.fer.hr/predmeti/irg/Zavrsni/12_Sladovic/Sladovic_Janko.pdf

[10] Sladović, J., 2013, Elastični modeli temeljeni na sustavu čestica, Rad izrađen za potrebe Diplomskog seminara na Fakultetu Elektrotehnike i Računarstva

8. Sažetak

Naslov: Model elastične tkanine

Prikaz tkanina u računalnoj grafici je vrlo opsežno i zahtjevno područje s potencijalno velikom primjenom. U ovom radu je objašnjeno na koje je sve načine moguće simulirati tkanine te su dana tri potencijalno iskoristiva modela. Ti modeli su implementirani i detaljno analizirani. Svakom modelu su istaknute prednosti i nedostaci u odnosu na ostale. Dodatno je analizirano kakav utjecaj na pojedini model imaju njegovi parametri te je analizirana razlika u brzini izvođenja pojedinih modela. Sav programski kod napisan u radu je pisan u programskom jeziku C# te OpenGLu.

Ključne riječi: tkanine, čestice, elastični modeli, sustav masa i opruga, sjenčanje, C#, OpenGL

9. Abstract

Title: Elastic cloth models

Cloth simulation in computer graphics is an extensive and complicated area with a potentially wide array of uses. This thesis analyses various methods of simulating cloth and presents three potentially usebale models. Those models have been implemented and analysed in detail. Every model had it's advantages and disadvantages presented in comparison to the others. In addition, the effect of various parameters for each model has been analysed and their performances have been tested. All the programming code has been written in the programming language C# and in OpenGL.

Key words: cloth particles, elastic models, mass-springs system, shading, C#, OpenGL

10. Skraćenice

API	*Application Programming Interface*	Aplikacijsko programsko sučelje
FPS	*Frames-Per-Second*	Sličice u sekundi
OpenGL	*Open Graphics Library*	Otvorena grafička biblioteka
OpenTK	*Open ToolKit*	Otvoren alat

11. Privitak

Svi opisani programi su priloženi uz rad. Tri modela su podijeljeni u tri zasebna C# projekta, imena *DiplomskiRadV1*, *DiplomskiRadV2* te *DiplomskiRadV3* za prvi, drugi te treći model. U sva tri modela se koriste tipke W, A, S, D, Q i E za rotaciju kamere oko promatranog predmeta, O i P za smanjenje te povećanje utjecaja vjetra, G za aktivaciju odnosno deaktivaciju utjecaja gravitacije te numeričke tipke 2, 4, 6 i 8 za promjenu smjera vjetra. Program se gasi pritiskom na tipku *Escape*.

Za mijenjati veličinu tkanine koriste se za prva dva algoritma varijable *CurtainWidth* te *CurtainHeight* u klasi *Game*, a za treći i varijable *PartWidth* te *PartHeight*. Dodatno je moguće mijenjati vrijednosti konstanta u klasi Game kako bi se isprobale razni načini prikaza tkanina te vrijednost gravitacije kako bi se prikazali razni načini ponašanja tkanina.