Maciej Matyka

SYMULACJE KOMPUTEROWE W FIZYCE

Wydanie II

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/sykof2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-5496-8

Printed in Poland.

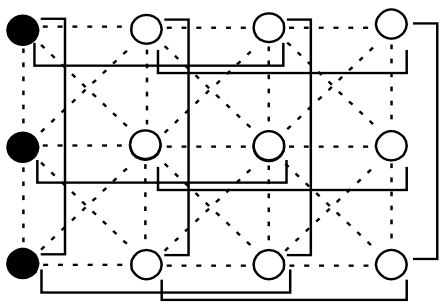
Spis treści

2.9. Ciała miękkie

Czy zastanawiałeś się kiedyś, jak to się dzieje, że piłka rzucona swobodnie odbija się od ziemi? Dlaczego bardziej nadmuchana piłka jest twarda? Co się dzieje na powierzchni piłki przy mocnym kopnięciu? Jednym z zastosowań omówionego w tym rozdziale modelu mas i sprężynek jest symulacja ciał miękkich (ang. *soft bodies*) — swego rodzaju balonów, które odkształcają się pod wpływem przyłożonej siły i po uwolnieniu wracają do swojej pierwotnej postaci [23, 24]. Takiego modelu z powodzeniem można użyć do odpowiedzi na postawione wyżej pytania. I chociaż ta odpowiedź nie będzie całkowicie precyzyjna, to może być bardzo widowiskowa, bo będzie zawierać wizualizację tych zjawisk. Omawianie modelu rozpoczniemy od symulacji flagi, która wyjaśni nam, jak implementuje się siły zewnętrzne (np. wiatr) w tego typu modelach. Następnie wprowadzimy ciśnienie i — budując odpowiednio modele — uzyskamy efekt ciała miękkiego.

2.9.1. Symulacja flagi

Symulację flagi rozpoczniemy od symulacji tkaniny zbudowanej jak na rysunku 2.45. Punkty materialne tworzące tkaninę poddawane są działaniu sił grawitacji i sprężystości połączeń z najbliższymi sąsiadami, tworząc elastyczną powierzchnię. Używamy połączeń poziomych, pionowych i ukośnych. Aby symulować flagę trzepoczącą na wietrze, w tkaninie wprowadzamy kilka modyfikacji. Najpierw unieruchamiamy punkty, które znajdują się na jednej z jej krawędzi, aby flaga miała na czym wisieć (wieszanie flagi na maszcie) — patrz rysunek 2.45. Podobnie jak w poprzednim podrozdziale nieruchomienie jest realizowane przez wyłączenie tych punktów z procedury całkowania równań ruchu.



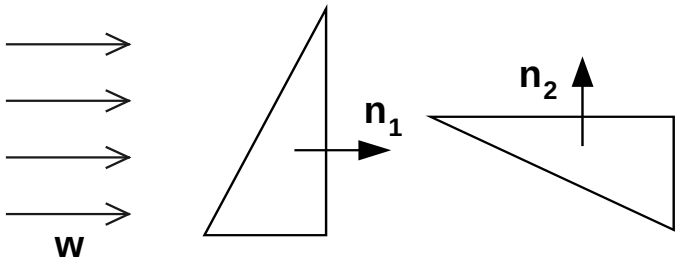
Rysunek 2.45. *Model flagi na wietrze zbudowany ze standardowej tkaniny z unieruchomionymi punktami na lewej krawędzi. Model uproszczony (4×3 punkty) — w praktyce punktów używa się więcej w celu zachowania realizmu symulacji*

W następnym kroku do standardowego algorytmu symulacji tkaniny, w którym wyliczane są siły sprężystości, grawitacji i tłumienia działające na poszczególne punkty materialne, dodajemy kolejną siłę — siłę wiatru, która będzie oddziaływać na powierzchnię flagi. Wiatr uprościmy, przyjmując, że jest jednorodny w całej przestrzeni. To duże uproszczenie, jednak — jak się okaże dalej — jest ono uzasadnione i nie psuje efektu końcowego. Same elastyczne ruchy powierzchni obiektu spowodują, że ruch flagi będzie dość chaotyczny i nieuporządkowany i wcale nie trzeba do tego niejednorodności siły.

Oznaczmy zatem wiatr pojedynczym wektorem *w*, który określa zwrot, kierunek i wartość siły oddziałującej na powierzchnię. Siła wiatru (nazwijmy ją F_w) będzie uzależniona od tego, jak bardzo prostopadle wiatr wieje na daną powierzchnię. Ta intuicja doprowadza nas do prostego wzoru:

$$F_w = (w \cdot n) \cdot n \tag{2.19}$$

gdzie *n* jest wektorem normalnym do powierzchni, a znak mnożenia oznacza mnożenie skalarne wektorów. Łatwo zauważyć, że wiatr wiejący prostopadle do powierzchni będzie działał największą siłą, bo iloczyn skalarny jest proporcjonalny do cosinusa kąta między wektorami, a ten jest wtedy równy 1. Inaczej będzie dla powierzchni równoległej do wiatru — w tym przypadku wektory wiatru i wektor normalny będą do siebie prostopadłe, a iloczyn skalarny znika (bo cos 90 stopni = 0). Ilustruje to rysunek 2.46.



Rysunek 2.46. *Ilustracja modelu pozwalającego wyliczyć siłę wiatru działającą na pojedynczy trójkąt. Jednorodny wiatr, którego kierunek jest oznaczony wektorem w, wiejący prostopadle do powierzchni o wektorze normalnym n_1, wywiera na nią największą siłę. Wiatr wiejący równolegle (na powierzchnię o wektorze normalnym n_2) nie wywiera żadnej dodatkowej siły*

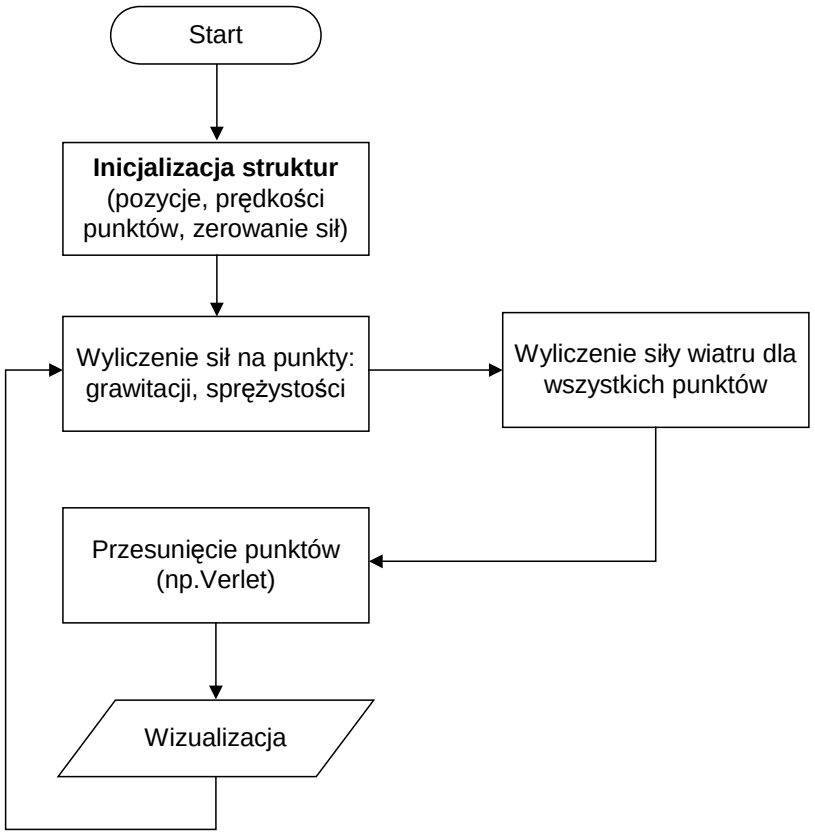
Jednym z możliwych rozszerzeń podstawowego modelu jest uwzględnienie ruchu samego trójkąta, który jednocześnie może modyfikować wpływ wiatru na swoją prędkość. Siła zależy wtedy od względnej prędkości między trójkątem a wiatrem. W takim przypadku dla trójkąta poruszającego się z prędkością *v* w równaniu (2.19) bierzemy *w* -> (*w*–*v*), co skutkuje nowym wzorem na siłę działającą na pojedynczy trójkąt (lub punkt materialny) o normalnej *n*:

$$F_w = ((w - v) \cdot n) \cdot n \tag{2.20}$$

Ze względu na specyficzną budowę tkaniny (na końcu obiekt składa się z powierzchni dużej liczby małych trójkątnych ścian) siła wiatru będzie wyliczana dla każdego punktu materialnego z osobna. Kwestia normalnej do punktu jest oczywiście dyskusyjna, tu jednak posiłkujemy się definicją, zgodnie z którą normalna do punktu jest utworzona przez uśrednienie wkładów od normalnych do ścian, do których ten punkt przynależy.

2.9.1.1. Algorytm

Algorytm symulacji flagi składa się z kilku kroków, a zmiany w stosunku do standardowej symulacji tkaniny polegają na zawieszeniu jej za jedną krawędź i dodaniu siły pochodzącej od wiejącego wiatru. Nadal pracujemy na zbiorze początkowo uporządkowanych punktów materialnych, które posiadają zdefiniowane połączenia sprężyste. Aby wygenerować ruch flagi, należy kolejno wykonać kroki przedstawione na rysunku 2.47.



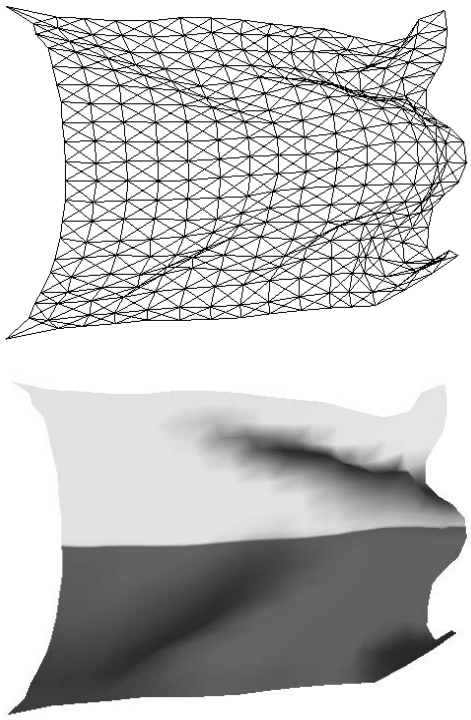
Rysunek 2.47. *Algorytm symulacji flagi z użyciem modelu cząstek i sprężynek oraz dodatkowej siły wiatru*

W algorytmie newralgiczny fragment kodu w języku C++, w którym oblicza się siłę działającą na pojedynczy punkt materialny o normalnej `n`, mieści się w kilku linijkach. Najpierw iterujemy po wszystkich punktach (tu ułożonych w tablicy dwuwymiarowej o indeksach `i`, `j` i wielkości `nx`, `ny`). Następnie sprawdzamy, czy nasz punkt nie został unieruchomiony (jeśli tak, pomijamy dla niego obliczenia siły wiatru). W naszym kodzie flaga punktu jest wtedy ustawiona na `P_ZAW` (stała oznaczająca, że punkt nie bierze udziału w symulacji fizycznej). Na koniec liczymy siłę wprost ze wzoru (2.20), wykorzystując wcześniej wyliczone normalne. W przytoczonym niżej fragmencie kodu wykorzystujemy

obiekt `cloth`, który zawiera wszelkie potrzebne informacje, w szczególności tablicę punktów `cloth.Points` i rozmiar tkaniny `cloth.nx` i `cloth.ny`.

```
for(j=0; j<cloth.ny ; j++)    // wiatr na punkty
for(i=0; i<cloth.nx ; i++)    if(!(cloth.Points[i][j]->flaga & P_ZAW))
    cloth.Points[i][j]->f = cloth.Points[i][j]->f + cloth.Points[i][j]->n * ((Wind -
cloth.Points[i][j]->v)*cloth.Points[i][j]->n);
```

Na rysunku 2.48 przedstawiam jedną klatkę symulacji flagi trzepoczącej na wietrze wykonanej wspomnianym tu algorytmem. Flaga próbuje się oczywiście dostosować kształtem do wiejącego wiatru. Co ciekawe, pomimo że wiatr ma w przestrzeni tę samą wartość, tkanina nie układa się płasko i nie pozostaje w bezruchu. Takiego efektu nie da się uzyskać i nawet przy tak prostej strukturze siły wiatru ruch tkaniny widać cały czas, a zachowanie się tkaniny jest realistyczne — zawija się ona na wietrze, marszczy i tworzy skomplikowane kształty, zachowując dynamikę ruchu.



Rysunek 2.48. *Pojedyncza klatka symulacji flagi trzepoczącej na wietrze [24]. Flaga została wyrenderowana na podstawie modelu fizycznego mas na sprężynkach w OpenGL.*

a)

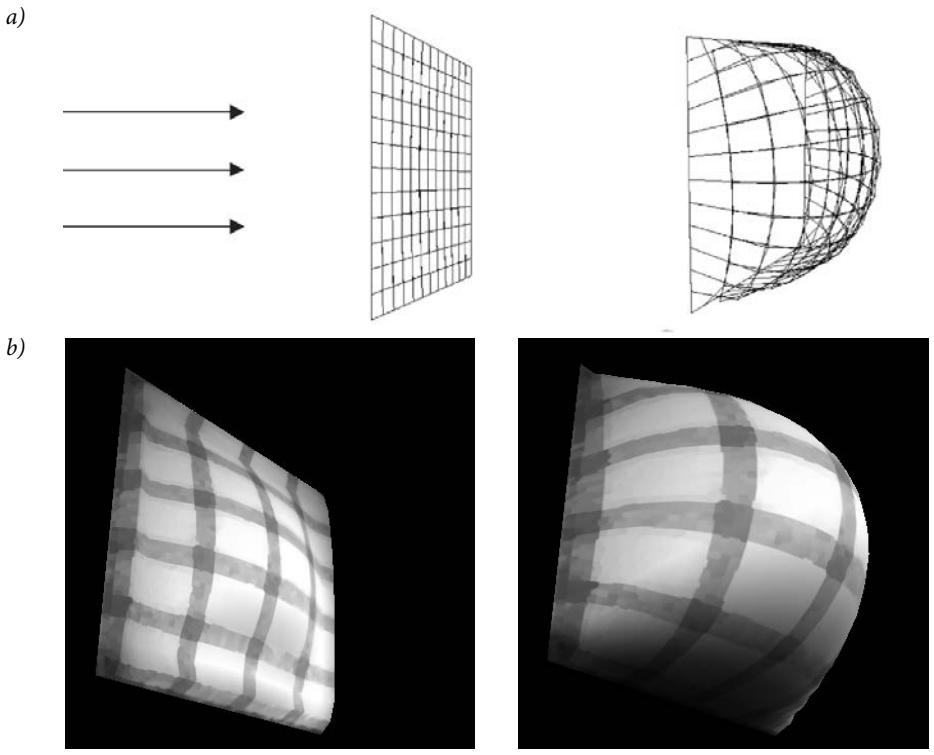
b)

Na rysunku przedstawione zostały w tej samej chwili czasu a) model fizyczny, b) flaga z teksturą. Pełna animacja znajduje się na kanale YouTube: https://youtu.be/J4yKLX3cEb4

2.9.2. Szczypta historii

Kilkanaście lat temu, bawiąc się symulacją z użyciem mas zawieszonych na sprężynach, przygotowywałem symulację trzepoczącej na wietrze flagi zawieszonej za jedną z krawędzi. Wtedy przypadkiem wpadłem na pomysł, aby unieruchomić wszystkie krawędzie zamiast tylko jednej, „puścić" wiatr prostopadle do powierzchni i zobaczyć, co się z nią

stanie. Efekt był bardzo widowiskowy — powierzchnia wypchała się w jednym kierunku, a modyfikując siłę wiatru, można było świetnie bawić się tym efektem. To, co wtedy widziałem, znajduje się na rysunku 2.49b.



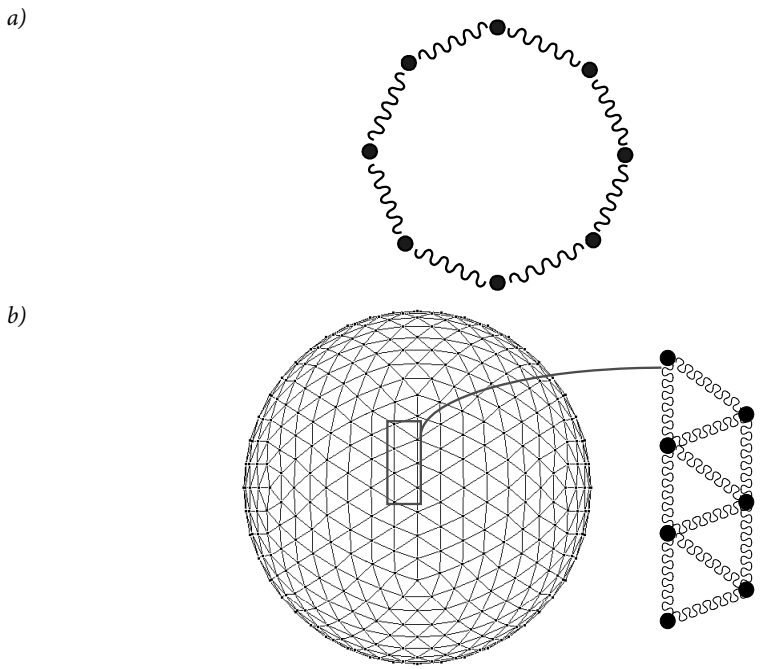
Rysunek 2.49. *Tkanina zawieszona na wszystkich czterech bokach, z wiatrem wiejącym prostopadle do powierzchni. Przedstawione są a) obiekt fizyczny ze sprężyn [23] oraz b) obiekt z symulacji wyrenderowany z teksturą w OpenGL. Na rysunku są dwa przypadki: mocny wiatr (po lewej) i słaby (po prawej). Widać wyraźnie odkształcenie tkaniny pod wpływem siły wiatru (animacja: https://youtu.be/vSqh02PwGng), źródło [23]*

Wtedy nagle mnie olśniło. A gdyby tak zamknąć powierzchnię, a „źródło wiatru umieścić w punkcie centralnym"? Ten zupełnie niefizyczny pomysł, zbudowany oryginalnie na intuicji, że coś może „wiać w każdym kierunku z punktu", doprowadził do modelu ciał miękkich, w których na powierzchnię działa siła skierowana prostopadle do niej. Dziś, po kilku latach pracy, model uwzględnia już ciśnienie i równanie stanu gazu doskonałego, z którego liczymy jego wartość. Ale zacznijmy od początku.

2.9.3. Model fizyczny ciał miękkich

Ciało miękkie reprezentujemy jako zamkniętą powierzchnię uzupełnioną o dodatkową siłę, będącą rezultatem różnicy ciśnień pomiędzy gazem w środku i na zewnątrz ciała. Ciśnie-

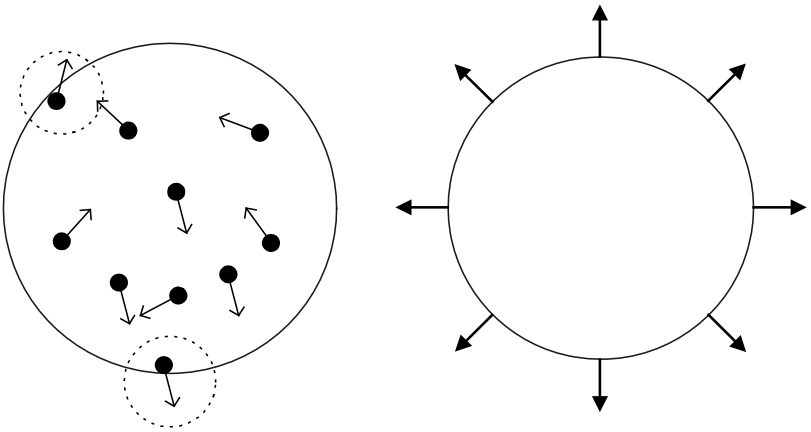
nie jest tutaj kluczowe, bo odpowiada za utrzymywanie przez obiekt stałej objętości. Zaczniemy od modelu dwuwymiarowego, w którym ciałem będzie okrąg zbudowany z punktów rozmieszczonych jak na rysunku 2.50. Spostrzegawczy czytelnik na pewno zauważy, że model nie różni się zbytnio od modeli liny — jedynie początkowy punkt łączy się teraz z końcowym, tworząc zamkniętą pętlę punktów.



Rysunek 2.50. *Budowa a) dwuwymiarowego i b) trójwymiarowego modelu ciała miękkiego. Czarne kropki oznaczają punkty materialne, a falista linia — połączenia sprężyste (pomiędzy punktami występuje siła Hooke'a). W celu lepszej wizualizacji w modelu 3D zaznaczone zostały tylko sprężyny ścianek zwróconych przodem do obserwatora*

Tak zbudowany obiekt nie będzie się jeszcze zachowywał jak piłka, dopóki nie uzupełnimy go o siły utrzymujące kształt i objętość. Bez tego obiekt z rysunku 2.50 po umieszczeniu w polu grawitacyjnym opadnie na powierzchnię i zapadnie się. Wskutek utraty energii stworzy płaski obiekt.

W modelu potrzebujemy jeszcze czegoś, co pomoże piłce utrzymać swój kształt. Będzie to gaz (patrz rysunek 2.51). Przy każdym uderzeniu w ściany ciała cząsteczki gazu będą przekazywać im część swojej energii. To będą małe części, ale dodane razem (duża liczba cząsteczek gazu w środku) mogą wygenerować sporą siłę ciśnienia. Jeśli ciśnienie wewnątrz piłki będzie większe niż ciśnienie atmosferyczne na zewnątrz, to siła będzie skierowana na zewnątrz piłki. Można sobie to wyobrazić tak, że liczba cząsteczek gazu w środku piłki,



Rysunek 2.51. *Ciało wypełnione gazem (strona lewa), który oddziałuje ze ściankami i wywiera odpowiednią siłę zwróconą na zewnątrz (strona prawa, wektory siły zaznaczone pełnymi strzałkami) i utrzymującą kształt i objętość ciała. Rysunek jest schematem w dwóch wymiarach, analogiczna sytuacja występuje w 3D*

które uderzają o jej powierzchnię, jest dużo większa od liczby cząsteczek bombardujących piłkę z zewnątrz. Z takich mikroskopijnych uderzeń o powierzchnię ciała wynikają siły, które nazywamy ciśnieniem.

Co ciekawe, dokładnie w ten sam sposób radzimy sobie w życiu codziennym. Na przykład pompujemy piłkę tak, aby ciśnienie wewnątrz było znacznie większe niż na zewnątrz (ciśnienie atmosferyczne). Dzięki różnicy ciśnień powstaje siła działająca na jej elastyczne ściany. Ta siła powoduje, że piłka utrzymuje swój kształt i nie zapada się pod własnym ciężarem. Analogiczna sytuacja zachodzi np. w dętkach w rowerze czy oponach w samochodzie.

2.9.4. Ciśnienie i równanie stanu gazu doskonałego

W naszym modelu ciał miękkich nie będziemy śledzić ruchu pojedynczych cząsteczek gazu ani ich zderzeń ze ścianami obiektów. Do tego celu wykorzystamy fizykę statystyczną i prawa pozwalające wyznaczyć ciśnienie w sposób uśredniony. Aby wyliczyć ciśnienie gazu wewnątrz obiektu, wykorzystamy równanie stanu gazu doskonałego:

$$P = nRT/V \tag{2.21}$$

gdzie ciśnienie wewnątrz jest proporcjonalne do liczby moli gazu (n), stałej gazowej (R) i temperatury (T) oraz odwrotnie proporcjonalne do objętości (V). Na nasze potrzeby uprościmy to równanie i założymy, że ciało jest szczelne (n stałe), a temperatura się nie zmienia (T stałe). Możemy wtedy zapisać prostą zależność:

$$P = 1/V \tag{2.22}$$

oznaczającą odwrotną proporcjonalność ciśnienia od objętości. Przy założeniu, że ciśnienie atmosferyczne wynosi zero (możemy to zrobić, odpowiednio dobierając stałe przy obliczaniu ciśnienia wewnątrz ciała miękkiego), możemy zapisać równanie na siłę działającą na pojedynczą ściankę obiektu:

$$F_p = n \cdot P \tag{2.23}$$

gdzie wartość P jest obliczana ze wzoru (2.22), a n jest wektorem normalnym do ścianki. W tym miejscu trzeba odpowiednio dobrać zwrot wektora n lub P, tak aby siła miała zwrot na zewnątrz obiektu. Podobnie jak w przypadku wiatru działającego na flagę tu siła ciśnienia — pomimo że fizycznie działa na całą ściankę — zostanie przez nas przyłożona tylko do punktów materialnych, które do nich przynależą.

2.9.5. Objętość zamkniętej bryły

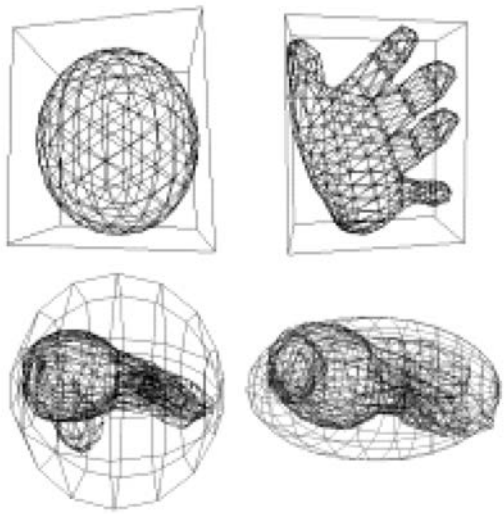
Ciśnienie wewnątrz obiektu obliczamy, wykorzystując wprost wzór (2.22), w którym musimy znać jego objętość. Można tu zastosować różne podejścia: obudowanie bryłą przybliżającą kształt, specjalne sumowanie (całka po powierzchni) wkładu ze wszystkich ścian obiektu lub użycie biblioteki zewnętrznej[25].

W najprostszym podejściu objętość obiektu można przybliżyć opisanymi na nich bryłami, których objętość znamy (prostopadłościan, sfera, elipsoida). W tym celu należy wykonać pętlę po wszystkich punktach obiektu i wyznaczyć minimalne i maksymalne wartości wszystkich składowych położeń punktów w trzech wymiarach. Nazwijmy je *xmin*, *xmax*, *ymin*, *ymax* oraz *zmin*, *zmax*. Znajomość tych wielkości pozwala przybliżyć objętość:

$$V = (xmax - xmin) \cdot (ymax - ymin) \cdot (zmax - zmin)$$

Na rysunku 2.52 znajduje się przykład przybliżenia ciała miękkiego przez prostopadłościan wyznaczony według tego prostego algorytmu. Przybliżenie można też wykonać, używając innych brył (sfer, elipsoid), dla których znamy wzór na objętość. Sposób ten działa dość dobrze dla wszelkich obiektów o zwartej budowie, możliwie dobrze przybliżanej przez wybraną bryłę, których odkształcenia są widoczne na całym obiekcie. Taką procedurę należy oczywiście wykonać zawsze przed wyliczeniem sił dla naszego obiektu. Dla obiektów sferycznych, typu piłka, najlepszym przybliżeniem będzie sfera lub elipsoida, których środek jest wyznaczony jako środek opisanego na nich prostopadłościanu, a promienie z wartości minimum i maksimum dla poszczególnych składowych. W przypadku dwuwymiarowym sytuacja jest analogiczna, tylko zamiast brył trójwymiarowych do opisu używamy figur 2D.

[25] Na przykład GNU Triangulated Surface Library: *http://gts.sourceforge.net/*.



Rysunek 2.52. *Obiekt kuli obudowany sześcianem oraz obiekt ręki, na którym obudowana została bryła (ang. bounding box) w celu przybliżenia objętości. Autorem obiektu jest Mariusz Jarosz. Rysunek pochodzi z pracy [23]*

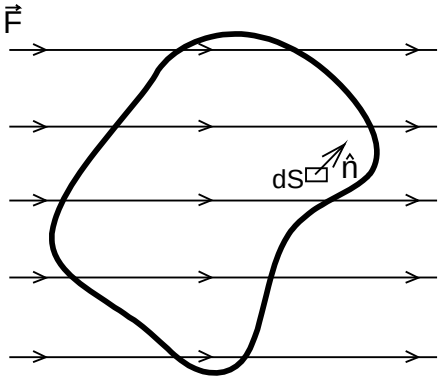
Omówiony wyżej algorytm obliczania objętości jest narażony na niedokładności, szczególnie w przypadku brył o skomplikowanych kształtach, które nie są łatwo przybliżane przez bryły podstawowe. Dlatego omówimy też inny sposób obliczania objętości — wprost z prawa Gaussa-Ostrogradskiego, które mówi, że całka po objętości z dywergencji pola wektorowego może zostać przybliżona przez całkę z pola wektorowego policzoną po powierzchni otaczającej tę objętość[26][27]. Zdaję sobie sprawę, że dla czytelnika nieobeznanego z matematyką to zdanie było chyba najtrudniejsze w całej książce, zachęcam jednak do analizy poniższego wyprowadzenia, bo na koniec czeka nas nagroda w postaci prostego wzoru na objętość (suma po trójkątach lub odcinkach naszego obiektu). Tu będziemy pracować z modelem trójwymiarowym. Rozpocznijmy zatem od wspomnianego prawa Gaussa-Ostrogradskiego, które możemy zapisać w postaci:

$$\iiint_V \vec{\nabla} \cdot \vec{F} dV = \oiint_S \vec{F} d\vec{S}$$

gdzie $\vec{F}$ jest polem wektorowym, V objętością obiektu (stąd całka po lewej stronie przebiega po całym obiekcie), a $d\vec{S}$ jest infinityzemalnym elementem powierzchni przemnożonym przez wektor normalny do niej, czyli $d\vec{S} = \hat{n}dS$, gdzie $\hat{n} = (n_x, n_y, n_z)$ jest wektorem normalnym, a dS jest polem tej powierzchni (patrz rysunek 2.53).

[26] Oczywiście samo twierdzenie istniało dużo wcześniej, ale idea wykorzystania go do liczenia objętości ciał miękkich pochodzi od Josa Stama, który zaproponował takie rozwiązanie z polem $(x,0,0)$ w trakcie dyskusji po moim wystąpieniu z wykładem o ciałach miękkich w ramach konferencji SIGRAD w 2003 roku w Umei (*http://panoramx.ift.uni.wroc.pl/~maq/talks/sigrad03.php*).

[27] Patrz również: *https://stackoverflow.com/questions/1406029/how-to-calculate-the-volume-of-a-3d-mesh-object-the-surface-of-which-is-made-up*.



Rysunek 2.53. *Obiekt zanurzony w polu wektorowym o składowych (x,0,0), oznaczony element infinityzemalny na powierzchni oraz wektor normalny*

W tym momencie wykonamy założenie o polu $\vec{F}$, które będzie teraz wynosić $\vec{F} = (x,0,0)$, dzięki czemu dywergencja pod całką z lewej strony znika, tzn. $\nabla \cdot \vec{F} = 0$, a po prawej stronie zostanie nam $\vec{F}d\vec{S} = xn_x dS$, stąd dla obiektu o zamkniętej powierzchni, złożonego z NUMT trójkątów numerowanych indeksami *i*, o polach S_i mamy wzór na objętość:

$$V = \iiint_V dV = \oiint_S xn_x dS \cong \sum_{i=1}^{NUMT} x_i n_{x,i} S_i \tag{2.24}$$

gdzie suma po prawej stronie przebiega po wszystkich trójkątach, x_i jest składową *x* danego trójkąta (np. jednego z jego wierzchołków lub średniej), a $n_{x,i}$ jest składową *x* wektora normalnego do tego trójkąta.

Podobny wzór można łatwo wyprowadzić dla przypadku dwuwymiarowego, obniżając stopień całkowania[28]. Otrzymujemy wtedy wzór na pole powierzchni:

$$S = \iint_S dS = \oint_l xn_x dl = \sum_{i=1}^{NUML} x_i n_{x,i} l_i \tag{2.25}$$

gdzie *S* jest powierzchnią ciała dwuwymiarowego, a *l* jest elementem liniowym na konturze obiektu.

Ze względu na specyficzną budowę naszych obiektów, które składają się ze skończonych, dyskretnych elementów na powierzchni (punkty połączone sprężynkami lub małe ścianki

[28] Takie wyprowadzenie można znaleźć np. w *http://panoramx.ift.uni.wroc.pl/~maq/soft2d/node17.html*.

trójkątne, z których budowana jest powierzchnia ciała), implementacja powyższych wzorów nie jest trudna. Sprowadza się ona do wykonania pętli po elementach powierzchni i wykonania sumy ze wzorów (2.24) lub (2.25). Poniżej przedstawiam fragment kodu wprost z programu do symulacji tego modelu, w którym objętość liczona jest wprost ze wzoru (2.24). W poniższym przykładzie objętość (`volume`) obliczamy jako sumę po trójkątnych elementach powierzchni (`Faces`) złożonych z punktów `p1`, `p2` i `p3`. Do objętości wchodzą pozycje punktów na jednej wybranej osi (tu jest to kierunek `x`) przemnożone przez odpowiednią składową wektora `n` (normalnego do powierzchni) oraz przez pole powierzchni `S` trójkąta (w kodzie jest to składowa `Area`).

```
volume = 0;
for(int i=0 ; i< Faces.size() ; i++)
{
  volume += ( Faces[ i ].p1->r.x + Faces[ i ].p2->r.x + Faces[ i ].p3->r.x) *
    Faces[ i ].n.x * Faces[ i ].Area / 3.0;
}
```

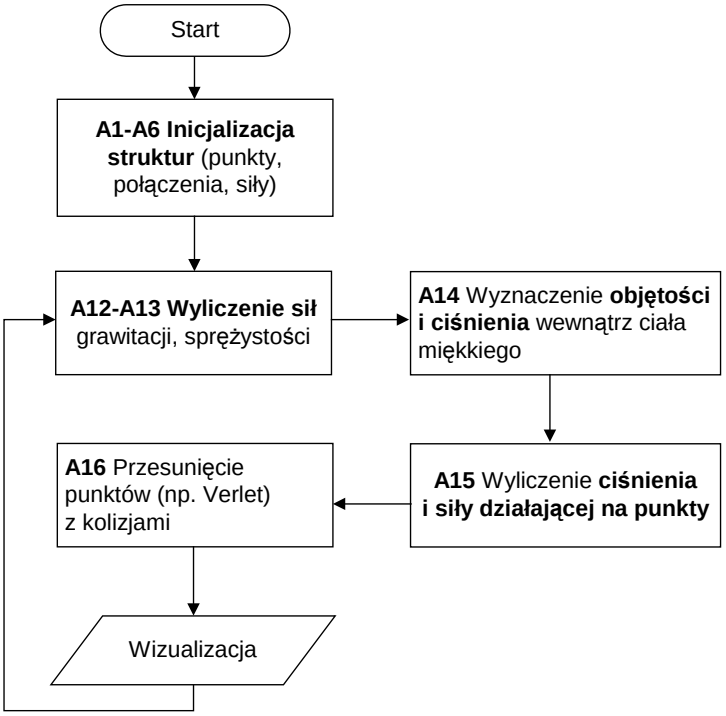
W powyższym kodzie C++ wektor normalny i powierzchnia powinny być dynamicznie obliczane dla każdego z trójkątów w trakcie ruchu obiektu.

2.9.6. Algorytm symulacji ciała miękkiego

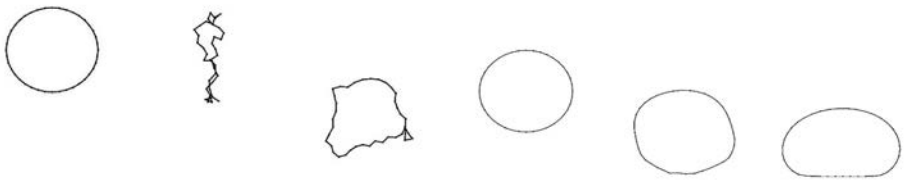
Algorytm działania modelu ciała miękkiego jest podobny do przedstawionego wyżej algorytmu dla tkaniny, z tą różnicą, że uwzględniamy teraz siłę pochodzącą od różnicy ciśnień między wnętrzem ciała a atmosferą (patrz rysunek 2.54).

Kluczowe różnice między algorytmami zwykłej tkaniny i ciała miękkiego zostały pogrubione — w pierwszej kolejności w procedurze tworzenia punktów i połączeń samego obiektu (punkt 1. algorytmu). Budowanie obiektu można zrealizować na wiele sposobów — zarówno generując obiekt ręcznie, jak i wczytując go z pliku w jednym z popularnych formatów (np. bardzo prosty 3d Studio ASCII — patrz podpunkt 2.9.6.1).

Najistotniejszą zmianą w algorytmie dla ciał miękkich (rysunek 2.54) jest dodatkowa siła ciśnienia — wyliczana wprost ze wzoru (2.21) i kolejnych dla każdego punktu materialnego tworzącego powierzchnię ciała. Tu przyda się wiedza z zakresu grafiki komputerowej, ponieważ dla każdego punktu należy wyznaczyć wektor normalny (siłą rzeczy będący konsekwencją budowy całego ciała, tzn. normalna do wektora punktu jest uzależniona od normalnych ścian, które go tworzą — może to być np. prosta średnia po wektorach współtworzonych ścian). Przykład symulacji z uwzględnieniem ciśnienia dla dwuwymiarowej piłki pokazany jest na rysunku 2.55.



Rysunek 2.54. *Algorytm symulacji ciała miękkiego z użyciem modelu cząstek i sprężynek oraz dodatkowej siły pochodzącej od różnicy ciśnień między wnętrzem a ciśnieniem atmosferycznym. Oznaczenia z literą A są znacznikami miejsca w kodzie omawianym w dalszych podrozdziałach*



a) model bez ciśnienia *b)* model z ciśnieniem

Rysunek 2.55. *Zachowanie modelu w polu grawitacyjnym a) bez siły ciśnienia — widać, że łańcuch deformuje się, tracąc początkową objętość i kształt (film: https://youtu.be/pPVE0kqO3xU), b) z uwzględnieniem ciśnienia — łańcuch w przybliżeniu zachowuje zarówno objętość, jak i kształt (film: https://youtu.be/XRpQsw8kzXo)*

W algorytmie uwzględniamy też kolizje, niezbędne przy piłkach i innych obiektach, które w polu grawitacyjnym mogą opadać np. na ziemię i się od niej odbijać. Tu możemy rozpatrywać kolizje ze ścianami, które mogą też być realizowane z uwzględnieniem jednostkowych kolizji punktów materialnych, jak w podpunkcie 2.5.2.3 (metody ogólne

wykrywania kolizji). W dalszej części przedstawimy też metodę uwzględniania przybliżonych kolizji z obiektami o kształcie sferycznym.

2.9.6.1. Format obiektów 3D Studio ASCII

Aby wczytać obiekt 3D z pliku zewnętrznego, można użyć np. formatu programu 3D Studio, który jest prostym formatem tekstowym. Oprócz typowego nagłówka, w którym określamy nazwę obiektu, liczbę ścian i wierzchołków, podajemy listę wierzchołków w przestrzeni 3D oraz indeksy punktów tworzących poszczególne ściany (`A`, `B` i `C` oznaczają kolejne wierzchołki trójkątów). Poniżej zamieszczam definicję sześcianu zapisanego w tym formacie. W pliku jest też kilka dodatkowych informacji (np. indeksy `AB`, `BC` i `CA`) oraz określenie stopnia wygładzenia (`smoothing`), jednak nie są one używane do zdefiniowania podstawowych obiektów.

```
------------------------------------------------------ start cube.asc
Named object: "Cube"
Tri-mesh, Vertices: 8 Faces: 12
Vertex List:
Vertex 0:  X:-1.0    Y:1.0    Z:1.0
Vertex 1:  X:-1.0    Y:-1.0    Z:1.0
Vertex 2:  X:1.0    Y:1.0    Z:1.0
Vertex 3:  X:1.0    Y:-1.0    Z:1.0
Vertex 4:  X:1.0    Y:1.0    Z:-1.0
Vertex 5:  X:1.0    Y:-1.0    Z:-1.0
Vertex 6:  X:-1.0    Y:1.0    Z:-1.0
Vertex 7:  X:-1.0    Y:-1.0    Z:-1.0
Face list:
Face 0: A:0 B:1 C:2 AB:1 BC:1 CA:1
Smoothing: 1
Face 1: A:1 B:3 C:2 AB:1 BC:1 CA:1
Smoothing: 1
Face 2: A:2 B:3 C:4 AB:1 BC:1 CA:1
Smoothing: 1
Face 3: A:3 B:5 C:4 AB:1 BC:1 CA:1
Smoothing: 1
Face 4: A:4 B:5 C:6 AB:1 BC:1 CA:1
Smoothing: 1
Face 5: A:5 B:7 C:6 AB:1 BC:1 CA:1
Smoothing: 1
Face 6: A:6 B:7 C:0 AB:1 BC:1 CA:1
Smoothing: 1
Face 7: A:7 B:1 C:0 AB:1 BC:1 CA:1
Smoothing: 1
Face 8: A:6 B:0 C:4 AB:1 BC:1 CA:1
Smoothing: 1
Face 9: A:0 B:2 C:4 AB:1 BC:1 CA:1
Smoothing: 1
Face 10: A:1 B:7 C:3 AB:1 BC:1 CA:1
Smoothing: 1
Face 11: A:7 B:5 C:3 AB:1 BC:1 CA:1
Smoothing: 1
------------------------------------------------------ koniec cube.asc
```

Proste obiekty (powierzchnie, kule) można wygenerować proceduralnie, ale jeśli naszym celem jest animacja ciał miękkich o kształtach odpowiadających rzeczywistym obiektom, może być niezbędne ich wykonanie w zewnętrznym oprogramowaniu. Wczytując obiekt z zewnątrz, należy pamiętać o dodaniu sprężyn, które zwykle nie występują w obiekcie wyeksportowanym na potrzeby grafiki komputerowej (np. w obiektach graficznych istnieją połączenia między najbliższymi sąsiadami, ale z dalszymi już nie, a mogą być potrzebne). Ze względu na własności modelu ważne jest też, aby obiekt do symulacji był zbudowany z trójkątów o względnie podobnej powierzchni[29].

2.9.7. Kod źródłowy symulacji ciał miękkich — HTML5

Omówimy teraz pełny kod źródłowy symulacji ciała miękkiego oparty na algorytmie przedstawionym w podpunkcie 2.9.6. Przedstawimy tu wersję kodu w dwóch wymiarach. Kod komputerowy znajduje się na serwerze FTP[30] Helionu oraz na stronie domowej autora[31]. Kod napisany został w języku JavaScript z użyciem elementu `Canvas`. Całość zapisana jest w pliku o nazwie *softbody.html*. Aby go uruchomić, wystarczy otworzyć go w dowolnej przeglądarce (Firefox, Opera, Chrome i inne).

Czytelników nieobeznanych z językiem HTML5 może dziwić nazwa pliku (*.html*), nie jest to jednak nic nietypowego. Nasz program został napisany w taki sposób, że jest zwyczajną stroną internetową zawierającą element typu `Canvas` (płótno), na którym rysują procedury z języka JavaScript. Plik *softbody.html* jest zbudowany w następujący sposób:

```
<!DOCTYPE html>
<html>
<head>
  <title>Soft Body JS, Maciej Matyka</title>
```

Po utworzeniu nagłówka i wpisaniu tytułu strony przechodzimy do opisu tzw. metadanych:

```
    <meta name="description" content="Prosty kod symulacji ciała miękkiego przygotowany
na potrzeby ilustracji rozdziału o modelu soft body w książce 'Symulacje komputerowe
w fizyce, wydanie 2 - rozszerzone'. Kod realizuje symulację modelu z ciśnieniem z użyciem
całkowania metodą Verleta.">
</head>
<body>
```

Tu rozpoczyna się właściwa strona internetowa. Najpierw tworzymy płótno, a następnie umieszczamy na stronie skrypt w języku JavaScript:

[29] Ciekawe efekty może dać zabawa z zagęszczaniem i rozrzedzaniem trójkątów na powierzchni, bo taka modyfikacja powoduje, że pojawiają się niejednorodne własności fizyczne. W ten sposób można prawdopodobnie uzyskać efekt wydmuchiwania różnych kształtów itp.

[30] *ftp://ftp.helion.pl/przyklady/sykof2.zip*

[31] *http://panoramix.ift.uni.wroc.pl/~maq/sykofi2/*

```
<!-- płótno do rysowania-->
<canvas id="mycanvas" width="400" height="300"></canvas>
<!-- tekst programu -->
<script type="text/javascript">

    //... tu umieszczony będzie kod JavaScript (punkt 2.9.7.1 i następne)

</script>
</body>
</html>
```

Po przepisaniu powyższego kodu HTML wraz z opisanymi poniżej procedurami należy zapisać całość w pliku z rozszerzeniem *.html*.

Omówienie kodu JavaScript, który powinien być umieszczony między znacznikami `<script ..>` a `</script>`, zaczniemy od bloku danych globalnych.

2.9.7.1. Blok danych globalnych

```
// A1 - stałe w programie
var W = 400;                          // rozmiar ekranu
var H = 300;
var NUMP = 17;                        // liczba punktów
var NUMS = 0;                         // liczba sprężyn
var R = 30;                           // promień ciała
```

W części pierwszej kodu (A1) definiujemy rozmiary płótna do rysowania. Ponieważ w tej wersji programu ciało do symulacji zostanie wygenerowane z kodu, w dalszej części podajemy liczbę punktów na powierzchni (`NUMP`), a liczbę sprężyn (`NUMS`) pozostawiamy wyzerowaną do późniejszego uzupełnienia. Na koniec ustawiamy promień ciała (`R`).

```
// A2 - wielkości fizyczne
var Pressure = 0;                    // początkowe ciśnienie
var PMAX = 6890000;                  // ciśnienie docelowe
var KS = 400, KD = 15;               // stała sprężystości i stała tłumienia dla
sprężynek
var g = -9.8;                        // przyspieszenie grawitacyjne
var m = 4;                           // masa pojedynczego punktu na powierzchni ciała
```

Stałe fizyczne zdefiniowane są w części (A2), gdzie podajemy ciśnienie docelowe (`PMAX`), ciśnienie aktualne (`Pressure`) oraz stałe sprężystości (`KS`) i tłumienia (`KD`) dla połączeń sprężystych na powierzchni. Wszystkie stałe są dobrane bardziej w celu uzyskania poprawnego efektu graficznego i płynnej, stabilnej animacji niż w celu odtworzenia jakiejś konkretnej sytuacji fizycznej. Można oczywiście dobrać stałe tak, aby odzwierciedlały prawdziwy układ fizyczny. Należy wtedy pamiętać o jednolitym układzie jednostek.

```
// A3 - stan obiektu
var x=[], y=[];                          // pozycje
var vx=[], vy=[];                        // prędkość punktów
var fx=[], fy=[];                        // siły
var nx=[], ny=[];                        // wektory normalne
var springs=[];
```

W części (A3) zawarte są tablice z danymi. To tu przechowujemy pozycje punktów (x, y), prędkości (vx, vy), siły działające na punkty (fx, fy) oraz normalne do punktów (nx, ny — liczone one będą z połączeń sprężystych, do których należą punkty). Oprócz tego w tym miejscu trzymamy masę i specjalną tablicę springs, która będzie zawierała definicję wszystkich sprężynek. Wypada wyjaśnić, że ze względu na brak możliwości definiowania obiektów i przeciążania operatorów w podstawowym języku JavaScript struktury, których używamy, są po prostu tablicami jednowymiarowymi. W przypadku bardziej złożonych typów stosujemy prostą metodę trzymania zmiennych jedna za drugą. Przykładowo jeden wpis do tablicy springs zawiera trzy pola: indeks pierwszego punktu, indeks drugiego i odległość w położeniu równowagi. Dlatego w przypadku próby przejścia do odpowiedniego miejsca w tablicy sprężyn należy przemnożyć indeks sprężyny przez 3.

```
// A4 - dane do całkowania
var xm1 = [], ym1 = [];        // kopie punktów dla metody Verlet
var xp1 = [], yp1 = [];
var verletInitialized = 0;    // czy wykonano pierwszy krok metody całkowania?
var dt = 0.08;
```

W dalszej części (A4) mamy dane potrzebne do całkowania równań ruchu. Oprócz tablic, w których przechowujemy pozycje punktów o krok wstecz względem aktualnych (xm1, ym1), oraz tablic dla pozycji dla kroku o jeden w przód (xp1, yp1), mamy tu zmienną verlet ↪Initialized, od której zależy, czy wykonamy tzw. zerowy krok całkowania, niezbędny w używanym schemacie Verleta[32]. Ustawiamy tu też krok czasowy dt, używany w procedurze całkowania. Krok tu równy 0.03 można delikatnie zmieniać. Wartość dobieramy, uwzględniając własności ciała, np. większe stałe sprężystości i tłumienia dla sprężyn wymagają mniejszych kroków całkowania, aby symulacja była stabilna. Podobnie większe ciśnienia narażają symulację na niebezpieczeństwo utraty dokładności. Niestety, zmniejszanie dt do bardzo małych wartości nie jest rozwiązaniem uniwersalnym, bo prowadzi do spowolnienia animacji (mniejsze kroki oznaczają, że trzeba ich wykonać więcej). Cała sztuka polega na dobraniu dobrej i kompromisowej wartości kroku czasowego.

```
// A5 - dane dodatkowe
var ctx;
var logo = new Image();
```

Kolejne dwie linijki to dodatkowe zmienne, które zostaną za chwilę zainicjalizowane; ctx będzie oznaczać kontekst płótna, czyli specjalny obiekt, z użyciem którego będziemy mogli rysować po płótnie (będzie on zawierać wszelkie procedury i dane potrzebne do rysowania). Obiekt logo, który od razu inicjalizujemy przez wywołanie operatora new Image, będzie zawierał niewielkie logo związane z tematyką tej książki. W dalszej części zajmiemy się definicją podstawowych funkcji używanych przez główną pętlę symulacji.

[32] *https://pl.wikipedia.org/wiki/Algorytm_Verleta*

2.9.7.2. Inicjalizacja i funkcje pomocnicze

Kolejnym elementem kodu (A6) jest implementacja funkcji `window.onload`, która zostanie wykonana po pierwszym załadowaniu strony.

```
// A6 - inicjalizacja - uruchom po załadowaniu strony
window.onload = function()
{
  // pobierz identyfikatory dla płótna
  var canvas = document.getElementById('mycanvas');
  canvas.width = W;        // ustaw wielkość płótna
  canvas.height = H;
  ctx = canvas.getContext('2d');

  // stwórz model
  CreateModel();

  // załaduj logo
  logo.src = 'logo.jpg';

  // pierwsze wywołanie pętli
  loop();
}
```

W powyższej funkcji inicjalizujemy strukturę (obiekty) płótna i kontekstu (`ctx`), ustawiając przy okazji żądaną wielkość płótna. W dalszej części wywołujemy funkcję, która ustawi punkty i zdefiniuje połączenia sprężyste na obwodzie (`CreateModel`). Na koniec ładujemy logotyp w formacie *.jpg* (jest on tu w celu identyfikacji i wzbogacenia graficznego) oraz wywołujemy po raz pierwszy pętlę główną symulacji (`loop`).

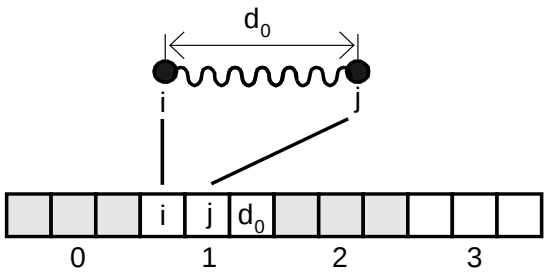
Ważną procedurą pomocniczą, dość często wywoływaną w programie, będzie obliczanie odległości między dwoma wektorami.

```
// A7 - Odległość dwóch wektorów
function dist(_x1, _y1, _x2, _y2)
{
  return Math.hypot(_x1 - _x2, _y1 - _y2);
}
```

Użyta tu funkcja `Math.hypoth` wyznacza pierwiastek sumy kwadratów swoich dwóch argumentów. W dalszej kolejności implementujemy funkcję dodającą sprężynę.

```
// A8 - funkcja dodająca sprężynę
function AddSpring( i, j )
{
  springs[ NUMS*3 + 0 ] = i;      // punkt 1
  springs[ NUMS*3 + 1 ] = j;      // punkt 2
  springs[ NUMS*3 + 2] = dist(x[i], x[j], y[i], y[j]);  // odległość w stanie równowagi
  NUMS++;      // sprężyna została dodana
}
```

Pobiera ona dwa parametry, którymi są indeksy łączonych punktów. Używamy tu jednowymiarowej tablicy do przechowywania sprężyn, które są opisywane przez 3 wielkości: indeks punktu 1, indeks punktu 2 (są to indeksy w tablicy punktów) oraz długość sprężyny w stanie równowagi (`d0`). Długość `d0` liczymy jako odległość od punktu 1 do punktu 2. Ze względu na użycie tablicy jednowymiarowej indeksujemy ją w specyficzny sposób. Najpierw przesuwamy się w niej o `NUMS*3` pozycji, następnie dodajemy 0, 1 lub 2, aby wpisać odpowiednią składową sprężyny o indeksie `NUMS+1`. Mnożenie razy 3 jest tu po to, aby przesuwać się o całe struktury sprężyn do przodu (patrz rysunek 2.56, ilustrujący przesunięcie się w tablicy `springs`).



Rysunek 2.56
Pojedyncza sprężyna, wielkości, które ją opisują, i umiejscowienie w tablicy jednowymiarowej (tu struktura sprężyny o indeksie 1)

```
// A9 - Stwórz model, punkty i sprężyny
function CreateModel()
{
  for(var i=0 ; i < NUMP ; i++)        // punkty na obwodzie
  {
    x[i] = R * Math.sin(i * (2.0 * Math.PI) /  NUMP  )+W/2;
    y[i] = R * Math.cos(i * (2.0 * Math.PI) /  NUMP  )+H/2;
    vx[i] = vy[i] = 0;
  }

  for(var i=0 ; i < NUMP-1 ; i++)    // tworzymy sprężyny
    AddSpring(i,i+1);
  AddSpring(i,0);
}
```

Kolejna procedura najpierw wypełnia wprowadzoną wcześniej tablicę punktów punktami rozłożonymi równomiernie na obwodzie koła o promieniu *R*. Prędkości punktów są zerowane, ustawiamy jednostkową masę. Po utworzeniu punktów należy je połączyć sprężynami. Tu z pomocą przychodzi nam wprowadzona przed chwilą procedura, którą wykonujemy dla każdej pary (`i`, `i+1`) punktów uprzednio rozłożonych obok siebie. Na koniec łączymy punkt pierwszy z ostatnim (`i`,`0`).

```
// A10 - Procedury graficzne
function line(x1,y1, x2,y2)
{
  ctx.beginPath();
  ctx.moveTo(x1, y1);
  ctx.lineTo(x2, y2);
  ctx.stroke();
```

```
}

function resetView()
{
  ctx.clearRect(0,0,W,H);            // czyść ekran
  ctx.strokeStyle = "rgb(0,0,0)";    // ustaw styl linii
}
```

Kolejne dwie procedury pomagają w rysowaniu. Pierwsza z nich (`line`) pobiera cztery parametry: po dwie współrzędne początku i końca linii i rysuje linię na płótnie (kontekst `ctx` z elementu `canvas`) z użyciem procedur wbudowanych w język HTML5. Druga z nich (`resetView`) czyści ekran i ustawia styl linii (kolor czarny).

2.9.7.3. Pętla symulacji

Kluczową częścią programu jest nieskończona pętla symulacji, której implementacja ma następującą postać:

```
// A11 - pętla symulacji
loop=function()
{
  ...    // tu wywołujemy procedury pętli

  requestAnimationFrame(loop);    // wykonaj pętlę jeszcze raz
}
```

W pętli symulacji po wykonaniu wszystkich operacji na końcu wywołujemy specjalną funkcję `requestAnimationFrame`, która spowoduje ponowne wywołanie funkcji z argumentu. Według dokumentacji typową częstotliwością tych wywołań będzie 60 fps (klatek na sekundę), ale to zależy zarówno od odświeżania przeglądarki, jak i od zawartości pętli. Jeśli będziemy mieli dużo operacji do wykonania lub wiele obiektów do narysowania, przeglądarka internetowa nie będzie w stanie tak szybko wywoływać pętli i liczba klatek na sekundę (a zarazem płynność animacji) ulegnie zmniejszeniu. W dalszej części omówimy zawartość pętli symulacji, która składa się z trzech części: liczenia sił, całkowania oraz rysowania aktualnego stanu.

2.9.7.4. Obliczenia sił

Obliczenia sił rozpoczynamy od najprostszej siły ciężkości, która jest związana z przyspieszeniem grawitacyjnym. W naszym układzie siła działa w kierunku y.

```
// A12 - siła ciężkości
for(var i=0; i<NUMP; i++)
{
  fx[i] = 0;
  fy[i] = -m*g;
}
```

Pętlę wykonujemy po wszystkich punktach tworzących ciało miękkie i każdemu przypisujemy wektorowo siłę `Fg = (0, m*g)`, gdzie `m` jest masą punktu, a `g` jest przyspieszeniem grawitacyjnym.

Kolejna część funkcji, bardziej rozbudowana, dotyczy obliczeń sił sprężystości działających pomiędzy punktami na powierzchni ciała. Rozpoczynamy od zdefiniowania potrzebnych zmiennych:

```
// A13 - siła sprężystości
var x1,x2,y1,y2;
var fs;              // siła sprężystości
var dvx, dvy;        // różnica prędkości
var Fx, Fy;          // wartości siły
```

Dalej przechodzimy do pętli po wszystkich sprężynach i odczytujemy indeksy punktów początkowego i końcowego (`p1` i `p2`) oraz odległość w stanie równowagi (`d0`). Następnie pobieramy współrzędne i obliczamy długość sprężyny.

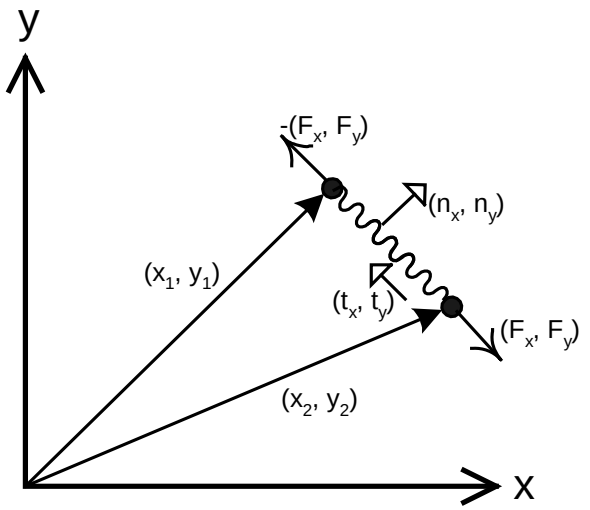
```
for(var i=0 ; i < NUMS ; ++i)              // pętla po sprężynach
{
  var p1 = springs[i*3+0];                // indeks punktu 1
  var p2 = springs[i*3+1];
  var d0 = springs[i*3+2];                // odległość równowagowa

  x1 = x[ p1 ];  y1 = y[ p1 ];            // współrzędne punktów
  x2 = x[ p2 ];  y2 = y[ p2 ];
  var d = dist(x1, y1, x2, y2);           // długość sprężyny
```

W zależności od tego, czy obliczona długość jest większa od 0, obliczamy interesującą nas siłę. Najpierw wyznaczamy wektor jednostkowy skierowany od punktu 2 do punktu 1 (patrz rysunek 2.57). Nasz model zakłada, że na powierzchni ciał oprócz sił sprężystości występuje też tłumienie uzależnione od różnicy prędkości między punktami, dlatego w dalszej kolejności obliczamy różnicę prędkości `dv`, której potrzebujemy, aby móc wyznaczyć tłumienie (patrz wzór 2.18).

```
  if(d > 0)
  {
    var tx = (x1 - x2) / d;          // wektor jednostkowy
    var ty = (y1 - y2) / d;          // od punktu 2 do 1

    dvx = vx[p1] - vx[p2];          // różnica prędkości
    dvy = vy[p1] - vy[p2];

    fs = (d - d0)*KS;               // siła sprężystości Hooke'a
    fs = fs + ( dvx * tx + dvy * ty )*KD;        // tłumienie sprężyny (wzór 2.18)
```

W kolejnym kroku obliczamy obie składowe siły sprężystości (`Fx`, `Fy`) i przypisujemy te wartości do sił działających na oba punkty (oczywiście ze zmienionym znakiem). Na koniec wyznaczamy wektor normalny (`nx` i `ny`), obracając wektory `tx` i `ty` o 90 stopni.

```
    Fx = tx * fs;                          // wartość siły "x"
    Fy = ty * fs;

    fx[ p1 ] -= Fx;  y[ p1 ] -= Fy;     // dodaj siłę dla punktu 1
    fx[ p2 ] += Fx;  fy[ p2 ] += Fy;     // jw. dla punktu 2
```



Rysunek 2.57. *Pojedyncze połączenie sprężyste — oznaczenia wektorów i sił. Strzałki z pełnymi grotami oznaczają wektory pozycji punktów, groty otwarte to wektory jednostkowe. Siły oznaczone są grotami zakrzywionymi, osie zwykłymi strzałkami. Na ilustracji mamy sytuację ściśniętej sprężyny (stąd siły działające „na zewnątrz")*

```
    nx[i] =  ty;                        // wyznacz wektor normalny z obrotu
    ny[i] =  -tx;                       // tx,ty
  }  // warunek d>0
}  // pętla po sprężynach (for)
```

Po zakończeniu powyższej pętli po sprężynach w siłach działających na punkty uwzględnione są już oddziaływania sprężyste. Ponieważ symulujemy ciało miękkie, musimy dodać jeszcze siłę pochodzącą od różnicy ciśnień między gazem wewnątrz a ciśnieniem atmosferycznym (patrz algorytm na rysunku 2.54). Pierwszym krokiem będzie obliczenie objętości (ang. *volume*) ciała.

```
// A14 - objętość ciała
var volume = 0;
for(var i=0 ; i<NUMS ; i++)
{
  var p1 = springs[i*3+0];          // pobierz punkty sprężyny
  var p2 = springs[i*3+1];
  x1 = x[ p1 ];      y1 = y[ p1 ];  // współrzędne punktów
  x2 = x[ p2 ];      y2 = y[ p2 ];
  d = dist(x1, y1, x2, y2);                 // długość sprężyny
  volume += 0.5 * Math.abs(x1 - x2) * Math.abs(nx[i]) * (d);    // wzór 2.25 z 2.9.5
}
```

W celu obliczenia objętości sumujemy wkłady od wszystkich ścianek ciała (tu ścianką jest każda sprężynka na obwodzie). W pętli najpierw pobieramy współrzędne punktu 1 i 2, a następnie obliczamy aktualną długość sprężyny i, korzystając z wzoru (2.25) z punktu 2.9.5, obliczamy kolejne przyczynki do objętości. Objętość wykorzystamy dalej do obliczenia

sił ciśnienia. Najpierw jednak wykonamy „pompowanie" ciała, ustalając ciśnienie gazu na wartość od 0 do PMAX z krokiem 0.01·PMAX. Takie dodawanie jest potrzebne, bo gwałtowne ustalenie ciśnienia na PMAX na początku symulacji spowoduje, że staje się ona niestabilna. Dodając je powoli, uzyskamy płynne przejście do obiektu nadmuchanego.

```
// pompowanie
if(pressure < PMAX)
  pressure = pressure + 0.01 * PMAX;
```

W dalszej części obliczamy ciśnienie, korzystając z równania stanu gazu doskonałego.

```
// A15 - ciśnienie
var pressurev;
for(i=0 ; i<NUMS ; i++)
{
  var p1 = springs[i*3+0];              // pobierz punkty sprężyny
  var p2 = springs[i*3+1];
  x1 = x[ p1 ];      y1 = y[ p1 ];     // współrzędne punktów
  x2 = x[ p2 ];      y2 = y[ p2 ];
  d = dist(x1, y1, x2, y2);                              // długość sprężyny
  pressurev = 0.5 * d * pressure * (1.0/volume);         // wartość siły ciśnienia
  fx[ p1 ] += nx[i] * pressurev;      fy[ p1 ] += ny[i] * pressurev;    // siła
  fx[ p2 ] += nx[i] * pressurev;      fy[ p2 ] += ny[i] * pressurev;
}
```

Po standardowym pobraniu współrzędnych wykonujemy sumowanie wkładu ciśnienia od wszystkich ścianek. Wartość ciśnienia (pressurev), zgodnie z wzorami, których używamy (patrz podpunkt 2.9.4), będzie odwrotnie proporcjonalna do objętości ciała (im mniejsza objętość, tym większe ciśnienie na ścianki). Dodatkowo mnożymy siłę przez długość sprężyny, aby uwzględnić, że siła działa na jednostkę powierzchni. Tak obliczona wartość siły jest dzielona przez 2, mnożona przez wektor normalny do powierzchni oraz dodawana (wektorowo) do obu punktów tworzących daną sprężynkę.

2.9.7.5. Metoda całkowania Verleta i kolizje

Przechodzimy teraz do kroku całkowania, w którym wykorzystamy algorytm Verleta. Jest to metoda numeryczna rozwiązywania równań różniczkowych zwyczajnych, która cechuje się większą stabilnością niż metody Eulera oraz zachowuje energię układu [27]. Jej cechą szczególną jest to, że wymaga zapamiętania jednego kroku wstecz, czyli oprócz kroku n musimy pamiętać stan układu w kroku $n–1$. Wzorem, jaki wykorzystamy do wyznaczenia nowego położenia wszystkich punktów, jest metoda Verleta bez prędkości:

$$x_{n+1} = 2 \cdot x_n - x_{n-1} + dt^2 \cdot f_x / m \qquad (2.26)$$
$$y_{n+1} = 2 \cdot y_n - y_{n-1} + dt^2 \cdot f_y / m$$

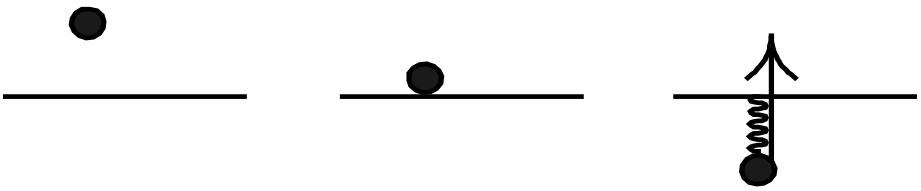
Jak w takim razie wykonać pierwszy krok całkowania, skoro startujemy z warunków początkowych, które są określone tylko dla czasu $t = 0$? W tym celu wykorzystamy pojedynczy krok całkowania metodą Eulera, który opisuje wzór:

$$x_{n+1} = x_n + dt^2 \cdot f_x / m \qquad (2.27)$$
$$y_{n+1} = y_n + dt^2 \cdot f_y / m$$

Zaczniemy zatem od kodu kroku inicjalizującego, który wykonany zostanie tylko raz, przy pierwszym wywołaniu procedury całkującej.

```
// A16 całkowanie
if(verletInitialized == 0)
{
  verletInitialized = 1;
  for(var i=0; i<NUMP; i++)
  {
    xp1[i] = x[i]+dt*dt*fx[i]/m;       // wzór (2.27)
    yp1[i] = y[i]+dt*dt*fy[i]/m;
  }
}  else
```

W dalszej części całkujemy równania z użyciem wzoru (2.27), który będzie używany w kolejnych iteracjach pętli (stąd `else`, które wykorzystuje zmienną `verletInitialized` i po pierwszym wywołaniu będzie już nas zawsze przenosić do właściwego całkowania). Oprócz całkowania metodą Verleta w tym miejscu zostanie też dołożona dodatkowa siła, wynikająca ze zderzenia z powierzchnią (`YBOTTOM`). Potrzebne jest tu kilka słów wyjaśnienia, bo w celu obsługi tego zderzenia wykorzystaliśmy inną technikę niż do tej pory stosowana w tej książce. Ponieważ mamy do czynienia z ciałem miękkim, zamiast odbijać punkty od twardej powierzchni, założyliśmy tu, że jest ona również miękka.



Rysunek 2.58. *Wirtualna sprężynka pojawiająca się przy zderzeniu ze ścianą. Ilustracja prezentuje trzy momenty: przed kolizją (po lewej), przy zetknięciu ze ścianą (na środku) i w trakcie kolizji (po prawej). Siła pochodząca od wirtualnej sprężynki ma tutaj niezerową tylko składową y. Model jest jednak bardziej ogólny i takie wirtualne sprężynki można stosować dla dowolnej ściany (również zakrzywionej)*

Dlatego odbicie jest realizowane przez dodanie dodatkowej, wirtualnej sprężynki (patrz rysunek), która „pojawia się" wtedy, kiedy punkt wykroczy poza dozwolony obszar (czyli tu poniżej wartości y `YBOTTOM`). Będzie to taka sama sprężynka jak na powierzchni ciała, ale o innym współczynniku sprężystości (tu nazywa się on `KSCOLLISION`). Wartość siły pochodzącej od tej wirtualnej sprężyny obliczamy wprost z prawa Hooke'a:

$$f_{virtual} = -\texttt{KSCOLLISION} \cdot (y - \texttt{YBOTTOM})$$

W samym kodzie nałożenie tej dodatkowej siły wykonujemy tuż przed całkowaniem — oczywiście można zrobić to wcześniej, jednak ze względu na organizację kodu oraz historyczne użycie w nim odbić, a nie sił ta dodatkowa siła znajduje się w pętli całkującej.

```
{ // wykonuj tylko dla iteracji > 0
  var YBOTTOM = 50;                          // pozycja podłogi
  var KSCOLLISION = 220;                     // sprężystość przy odbiciu
  for(var i=0; i<NUMP; i++)
  {
    // wirtualna sprężyna (kolizje ze ścianą)
    if(y[i]<YBOTTOM)  { fy[i] = fy[i] - KSCOLLISION*(y[i]-YBOTTOM);}

    // całkuj równania ruchu metodą Verlet
    xp1[i] = 2*x[i] - xm1[i] + dt*dt*fx[i]/m;    // wzór (2.26)
    yp1[i] = 2*y[i] - ym1[i] + dt*dt*fy[i]/m;
  }
}
```

W kroku całkowania do wyznaczenia nowych wartości pozycji punktów (`xp1`, `yp1`) wykorzystujemy zarówno aktualny stan układu (`x`, `y`), jak i stan poprzedni (`xm1`, `ym1`). Dlatego po całkowaniu trzeba wykonać przeniesienie danych, bo w kolejnym kroku nowe pozycje staną się aktualnymi, a aktualne tymi z poprzedniego kroku, czyli:

```
x -> xm1,
y -> ym1,
```

oraz

```
xp1 -> x,
yp1 -> y.
```

Procedurę tę realizuje kod poniżej:

```
// A17 przepisanie stanów dla metody Verleta i obliczenie prędkości
for(var i=0; i<NUMP; i++)
{
  xm1[i] = x[i];                             // x -> xm1
  ym1[i] = y[i];
  x[i] = xp1[i];                             // xp1->x
  y[i] = yp1[i];

  vx[i] = (x[i] - xm1[i]) / (2*dt);          // oblicz prędkość
  vy[i] = (y[i] - ym1[i]) / (2*dt);
}
```

Ponieważ w metodzie, której używamy, nie istnieje jawnie całkowanie dla prędkości, wykorzystujemy tę pętlę do wyznaczenia prędkości, biorąc średnią z prędkości bieżącej i poprzedniej.

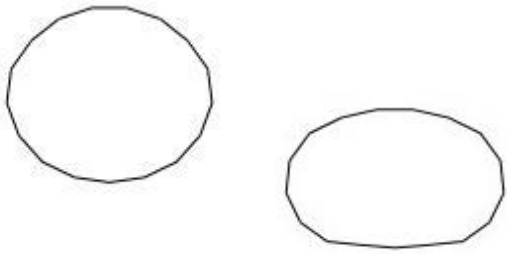
2.9.7.6. Wizualizacja

W naszym przykładzie wizualizacja sprowadza się do narysowania linii łączących po kolei punkty ciała:

```
// A18 - rysowanie ciała na płótnie
resetView();

// rysuj ciało miękkie
for(i=0 ; i<NUMS ; i++)
{
  var p1 = springs[i*3+0];                  // pobierz punkty sprężyny
  var p2 = springs[i*3+1];
  x1 = x[ p1 ];        y1 = y[ p1 ];        // współrzędne punktów
  x2 = x[ p2 ];        y2 = y[ p2 ];
  line(x1, y1, x2, y2);                     // rysuj sprężynę
}

// narysuj logo
ctx.drawImage(logo,0,0);
```

W powyższej procedurze najpierw czyścimy płótno i ustawiamy kolor dla linii, a następnie rysujemy wszystkie połączenia sprężyste, iterując po tablicy `springs`. Na koniec, już poza pętlą, rysujemy logotyp (to oczywiście tylko ozdobnik).

Wyniki działania symulacji wykonanej opisanym kodem znajdują się na rysunku 2.59.

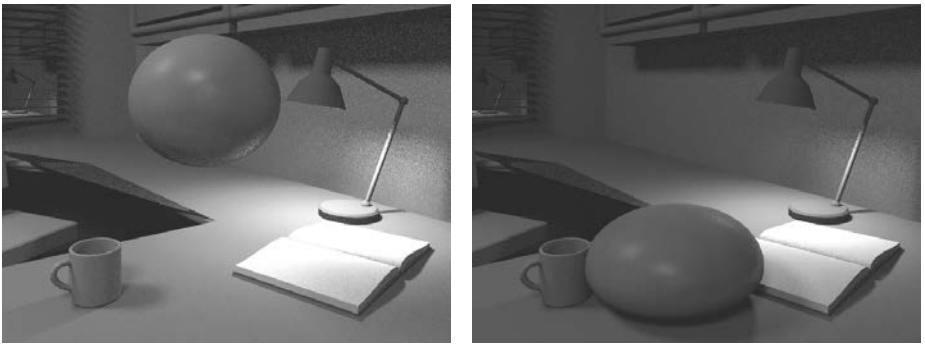


Rysunek 2.59. *Symulacja ciała miękkiego z użyciem kodu omówionego w tym podrozdziale*

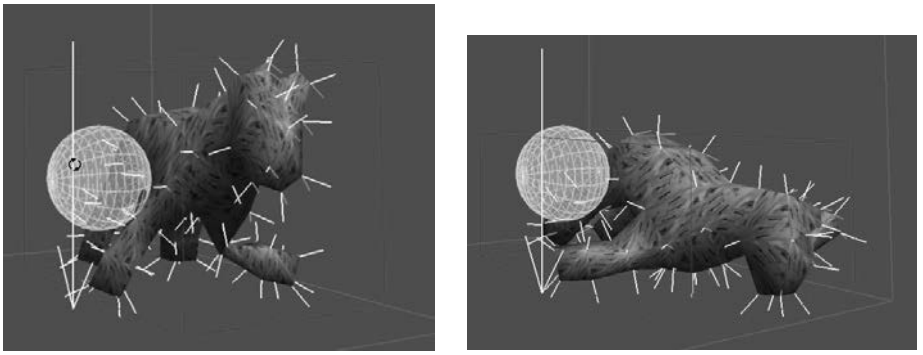
2.9.8. Przykłady symulacji

W tym podrozdziale prezentujemy symulacje trójwymiarowe, które realizują przedstawiony wcześniej algorytm. Symulacje zostały napisane z użyciem języka C++ i od strony rachunkowej i fizycznej nie różnią się wcale od implementacji przedstawionej w poprzednich podrozdziałach. Pierwszym przykładem jest nadmuchana piłka (2.60), której wizualizacja została dodatkowo wykonana metodą uwzględniającą oświetlenie i cienie.

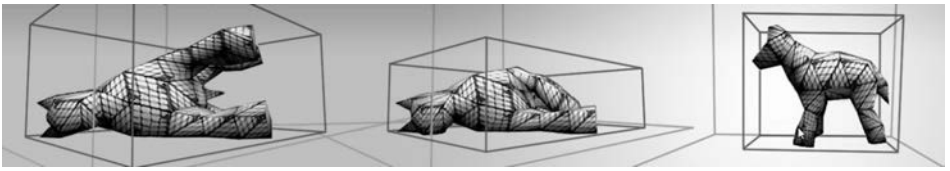
Model ciał miękkich pozwala też na symulację brył o kształtach bardziej złożonych niż piłki. Obiekty złożone symulujemy, wczytując punkty i ściany z zewnętrznego pliku (można je też oczywiście wygenerować; możliwe są różne podejścia). Takie punkty mogą być jednak przygotowane w dowolnym programie graficznym 3D (np. Blender, 3D Studio, LightWave i inne) przez grafika komputerowego. Przykład takiej animacji — ruch nadmuchanego konia i interakcję z użytkownikiem — przedstawia rysunek 2.61.



Rysunek 2.60. *Dmuchana piłka w polu grawitacyjnym, rendering metodą raytracingu monte carlo (autorem kodu programu do renderowania jest Kamil Trojan). Film: https://youtu.be/hMYNpS1_YOA*



a) interakcja z użytkownikiem



b) ciało miękkie przyjmuje różne pozy

Rysunek 2.61. *Model trójwymiarowego konia w symulacji ciała miękkiego a) w interakcji z użytkownikiem (sztywna biała sfera zderza się z punktami ciała miękkiego, białe linie to wektory normalne do punktów), b) rendering modelu w kilku pozach. Symulacja została wykonana programem Soft Body 2, który można znaleźć m.in. na stronie domowej autora: http://panoramx.ift.uni.wroc.pl/~maq/eng/softbody.php. Animacja: https://youtu.be/bd5Z-t7mfcg. Model trójwymiarowy stworzył Barnaba Mikułowski*

Skorowidz

G

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

Z


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA Helion

Symulacje komputerowe — od podstaw!

- Poznaj metody numeryczne
- Naucz się stosować je w praktyce
- Odkryj świat symulacji komputerowych

Ostatnie kilkadziesiąt lat to okres burzliwego rozwoju technologii informatycznych i ciągłego zwiększania możliwości komputerów dostępnych dla coraz to szerszych rzesz użytkowników. Z zaawansowanych narzędzi graficznych i symulacyjnych mogą obecnie korzystać już nie tylko profesjonaliści zatrudnieni w dużych korporacjach dysponujących ogromnymi zasobami finansowymi, ale również pasjonaci, programiści i graficy pracujący dla niewielkich firm, które tworzą animacje komputerowe na użytek przemysłu czy branży rozrywkowej.

Realistyczne, uniwersalne i interaktywne efekty można tu uzyskać tylko w jeden sposób: wykorzystując fizyczne metody symulacji ruchu, do tego zaś niezbędna jest odpowiednia wiedza. Drugie wydanie książki *Symulacje komputerowe w fizyce* wprowadzi Cię w podstawy metod numerycznych oraz przedstawi ich zastosowanie w praktyce. Dowiesz się, jak przeprowadzać symulacje prostych i bardziej złożonych zjawisk fizycznych, rozwiązywać równanie falowe oraz symulować zachowanie cieczy nieściśliwych i gazów, a nawet wkroczysz w tajemniczy świat fizyki kwantowej. Symulacja flagi trzepoczącej na wietrze, miękkiej piłki odbijającej się od powierzchni czy oporu, który powietrze stawia jadącemu samochodowi? Przekonasz się, że to nic trudnego!

Książka jest przeznaczona zarówno dla studentów, jak i uczniów starszych klas szkół średnich, dla których może stanowić wstęp do praktycznego programowania symulacji i modelowania fizycznego w animacji komputerowej. Pozycja szczególnie zainteresuje studentów oraz kadrę dydaktyczną kierunków ścisłych, na przykład wykładowców modelowania komputerowego czy fizyki komputerowej, dla których może być punktem wyjścia do dalszego zgłębiania przedstawionych modeli. Do publikacji dołączony jest zbiór napisanych w językach C++, OpenGL oraz HTML5 programów, które ilustrują sposób przeprowadzania opisanych symulacji.

- Symulacje zjawisk mechaniki klasycznej
- Dynamika ciał miękkich w czasie rzeczywistym
- Rozwiązanie numeryczne równania falowego
- Symulacje dynamiki płynów (CFD)
- Rozwiązanie numeryczne równania Schrödingera

Poznaj metody numeryczne od podszewki!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-5496-8

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 79,00 zł