Flash. Akademia matematycznych sztuczek. Wydanie II

Autorzy: Keith Peters, Manny Tan, Jamie MacDonald
Tłumaczenie: Piotr Cieślak
ISBN: 978-83-246-0170-7
Tytuł oryginału: Flash Math Creativity (Second edition)
Format: B5, stron: 272

Tworzenie zaskakujących efektów graficznych

- Jak stworzyć trójwymiarowe wykresy?
- Jak wykorzystać wzory matematyczne w programowaniu?
- W jaki sposób zastosować geometrię fraktalną do generowania grafiki?

Jesteś programistą Flasha i szukasz nowych wyzwań? A może zastanawiasz się, czy matematyka, jaką poznałeś w szkole lub na studiach, może okazać się przydatna w projektach prezentacji flashowych? Jeśli chcesz stworzyć we Flashu coś nowego, świetnie się przy tym bawiąc, możesz wykorzystać w tym celu swoją wiedzę matematyczną. Fraktale, ciąg Fibonacciego, geometria trójwymiarowa, rachunek wektorowy – to wszystko może stać się narzędziem, dzięki któremu uzyskasz zapierające dech w piersiach efekty graficzne.

Książka „Flash. Akademia matematycznych sztuczek. Wydanie II" to zbiór projektów, których twórcy w ciekawy sposób wykorzystują znane zależności matematyczne. Czytając ją, dowiesz się, jak powstają hipnotyczne efekty wirujących kształtów geometrycznych oraz jak generować fraktalne krajobrazy i figury. Poznasz możliwości wykorzystania translacji wektorowych i macierzowych do konstruowania trójwymiarowych wykresów oraz brył. Nauczysz się także uwzględniać dynamikę w animacjach i symulować jej wpływ na animowane obiekty.

- Animowanie kształtów dwuwymiarowych
- Falująca siatka
- Tworzenie fraktalnych roślin i krajobrazów
- Wykorzystanie rekurencji
- Efekty trójwymiarowe

Przekonaj się, że matematyka wcale nie musi być żmudna i męcząca

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

SPIS TREŚCI

Skąd czerpać inspirację? Trudno jest udzielić jednoznacznej odpowiedzi na tak zadane pytanie. Często wpadam na ciekawe pomysły, obserwując rzeczy, które napotykam w moim najbliższym otoczeniu, i wykonując zwykłe, codzienne czynności: oglądając telewizję, filmy czy słuchając muzyki. Pracując we Flashu, często podejmuję próby naśladowania jakiegoś zjawiska lub sceny, którą zaobserwowałem. Choć rozpoczynając swoje eksperymenty, zazwyczaj wyobrażam sobie pewien konkretny rezultat, do którego staram się dążyć, to w trakcie realizowania owego „głównego wątku" niemal zawsze zdarza mi się przypadkiem odkryć nowe, zaskakujące możliwości, które sprawiają, że otrzymuję w efekcie coś, czego zupełnie bym się nie spodziewał.

Jamie Macdonald mieszka i pracuje w Londynie.

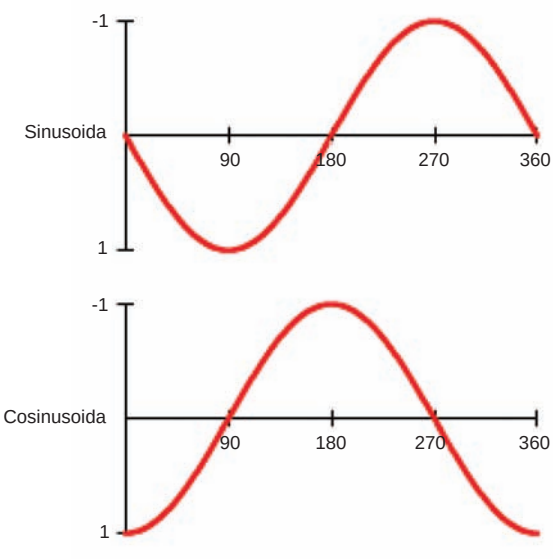
jamie macdonald

www.nooflat.nu

Kilka informacji na początek

Każdy z projektów, które za chwilę omówię, będę rozpoczynał od tego samego pliku wyjściowego. Dla wygody, aby uniknąć czterokrotnego powtarzania tych samych opisów we wstępach do kolejnych eksperymentów, zrobię to tylko raz.

Projekty, które przygotowałem, są stosunkowo nieskomplikowane. Opracowanie każdego z nich nie pochłonęło mi więcej jak kilka godzin, lecz uważam, że wyróżnia je elastyczność i spore możliwości: zwykła zmiana kilku parametrów może prowadzić do niemal całkowitego przeobrażenia uzyskanego efektu. Wspólną cechą wszystkich projektów jest zastosowanie funkcji trygonometrycznych sinus i cosinus do animacji różnych obiektów, zanim więc przejdę do szczegółowego omawiania konkretnych efektów, chciałbym powiedzieć kilka słów o właściwościach tych funkcji. Zarówno sinusoida, jak i cosinusoida są krzywymi okresowymi i w przedziale od 0 do 360 stopni zmieniają swoją wartość w zakresie od 1 do –1 (na przykład sin 90 = 1, zaś sin 270 = –1). Jak łatwo zauważyć, cosinusoidę można potraktować jako szczególny przypadek sinusoidy przesunięty względem niej o pewną wartość.



Biorąc pod uwagę zakres wartości, jaki przyjmują funkcje sinus i cosinus, w celu uzyskania oscylacji z przedziału od –10 do 10 wynik otrzymany dla kolejnych argumentów należy mnożyć przez 10. Jeśli naszym celem byłoby uzyskanie oscylacji z przedziału od 0 do 40, to wartość funkcji trzeba byłoby mnożyć przez 20 i do uzyskanego w ten sposób rezultatu dodawać 20. Warto zauważyć, że ze względu na to, iż funkcje cosinus i sinus są funkcjami okresowymi, to poza granicami podstawowego przedziału argumentów od (0 do 360 stopni) mają one taki sam przebieg, jak w obrębie tego zakresu. Oznacza to, że wartość funkcji sinus dla kąta wynoszącego 10, 370 czy 730 stopni będzie taka sama.

Posługując się funkcjami trygonometrycznymi w programie Flash, należy pamiętać, że ich argumentami nie będą stopnie, lecz radiany — nieco inna jednostka miary.

Przed użyciem funkcji trygonometrycznych we Flashu trzeba zatem zadbać o przekształcenie wartości w stopniach na radiany, co zresztą jest stosunkowo proste:

```
360 stopni = 2*pi radianów
180 stopni = pi radianów
1 stopień = pi/180 radianów
```

W celu przeliczenia stopni na radiany można użyć następującego wzoru:

```
radians = degrees×Math.PI/180
```

To byłoby na tyle, jeśli chodzi o niezbędną wiedzę matematyczną. Przystąpię teraz do omawiania szablonu projektu, który posłuży jako punkt wyjścia do wszystkich dalszych eksperymentów. Przede wszystkim w projekcie tym powinien znaleźć się klip o nazwie ball, który zawierać będzie czerwone kółko o wymiarach 42×42 piksele, umieszczone dokładnie pośrodku sceny. Klip ten powinien zostać wyeksportowany z biblioteki projektu z identyfikatorem ball.

Wszystkie eksperymenty można będzie z powodzeniem zrealizować, używając tylko tego obiektu, jest jednak jeszcze jedna rzecz, którą zdecydowałem się umieścić w szablonie, a mianowicie funkcja umożliwiająca dynamiczną zmianę jasności klipu. Funkcję tę zdefiniowałem na głównej listwie czasowej projektu, nadając jej nazwę setBrightness. Opiera się ona na metodzie setTransform i można użyć jej w odniesieniu do dowolnego kolorowego obiektu, który narysujesz. Warto zauważyć, ze funkcję tę napisałem w taki sposób, by przyjmowała ona wyłącznie wartości dodatnie. Umieść następujący kod w pierwszej klatce głównej listwy czasowej projektu:

```
function setBrightness(col:Color, brightness
➥ Number):Void {
   var anum:Number = 100 - brightness;
   var bnum:Number = 255/100 * brightness;
   col.setTransform( {ra:anum, ga:anum, ba:anum,
   ➥ rb:bnum, gb:bnum, bb:bnum, aa:100, ab:0} );
};
```

Szablon pliku jest gotowy. Podczas realizacji kolejnych projektów wystarczy uzupełniać go o odpowiedni kod źródłowy, który należy umieścić w pierwszej klatce głównej listwy czasowej.

towardsUs (atak piłek)

Pierwszy efekt polega na generowaniu serii klipów, które stopniowo zwiększają swoje rozmiary, oscylując przy tym względem środka ekranu, by w pewnym momencie sprawić wrażenie przeniknięcia przez ekran monitora — wówczas znikają na dobre. Sercem całego projektu są dwie funkcje. Pierwsza z nich obsługuje zdarzenie onEnterFrame i służy do generowania klipów filmowych, zaś druga, o nazwie expand, powoduje ich powiększanie. Przypominam, że kod źródłowy projektu należy umieścić w pierwszej klatce głównej listwy czasowej. Oto on:

```
var scaleMax:Number = 800;
var fadeOut:Number = 0.93;
var frequency:Number = 10;
var colMin:Number = 0;
var colMax:Number = 40;
var colVariance:Number = colMax - colMin;
function hRad(Void):Number {
   return 4 + Math.random();
}
```

```
function vRad(Void):Number {
   return 4 + Math.random();
}
function vRadInc(Void):Number {
   return 0.1;
}
function hRadInc(Void):Number {
   return 0.1;
}
function lrSpeed(Void):Number {
   return 5 + Math.random() * 40;
}
function scaleUpSpeed(Void):Number {
   return 1.02;
}
function nooCol(Void):Number {
   return colMin + Math.random() * colVariance;
}
var depth:Number = 0;
onEnterFrame = function () {
   if (Math.floor(Math.random() * frequency) == 0) {
      depth++;
      var noo:MovieClip = _root.attachMovie("ball", "ball"+depth, depth);
      var col:Color = new Color(noo);
      setBrightness(col, nooCol());
      noo._x = -50;
      noo._y = -50;
      noo._xscale = noo._yscale = 10;
      noo.scaleSpeed = scaleUpSpeed();
      noo.lrSpeed = lrSpeed();
      noo.hRad = hRad();
      noo.vRad = vRad();
      noo.hRadInc = hRadInc();
      noo.vRadInc = vRadInc();
      noo.lr = 0;
      noo.onEnterFrame = _root.expand;
   }
}
function expand() {
   this.lr += this.lrSpeed;
   this.hRad += this.hRadInc;
   this.vRad += this.vRadInc;
   this._x = Stage.width / 2 + this.hRad * Math.sin(this.lr * Math.PI/180);
   this._y = Stage.height / 2 + this.vRad * Math.cos(this.lr * Math.PI/180);
   this._yscale = this._xscale *= this.scaleSpeed;
   this.swapDepths(Math.floor(this._xscale));
   if (this._xscale > _root.scaleMax) {
      this._alpha *= _root.fadeOut;
      if (this._alpha < 3) {
         this.removeMovieClip();
      }
   }
}
```

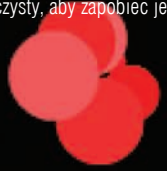
Zauważ, że większą część kodu napisałem w postaci funkcji, gdyż takie rozwiązanie jest moim zdaniem bardziej czytelne: jeśli cały kod podzielony zostanie na niewielkie, logiczne części, łatwiej będzie znaleźć w nim potrzebne fragmenty, zrozumieć jego działanie i zmodyfikować go. Spora liczba funkcji pozwala jednocześnie odwołać się do rozmaitych wartości i skorzystać z nich w razie potrzeby.

Najważniejsze zmienne i funkcje

`scaleMax` — maksymalny rozmiar klipu przed zaniknięciem,
`fadeout` — szybkość zanikania,
`frequency` — częstotliwość tworzenia nowych klipów,
`colMin` — minimalna jasność,
`colMax` — maksymalna jasność,
`colVariances` — zróżnicowanie jasności,
`lr` — wartość definiująca położenie piłki na sinusoidzie lub cosinusoidzie; im szybciej rośnie ta wartość, tym gwałtowniejsze będą oscylacje piłek,
`hRad` — wartość oscylacji względem punktu środkowego w kierunku poziomym,
`vRad` — wartość oscylacji względem punktu środkowego w kierunku pionowym,
`hRadInc` — przyrost oscylacji w kierunku poziomym,
`vRadInc` — przyrost oscylacji w kierunku pionowym,
`lrSpeed` — szybkość oscylacji,
`scaleUpSpeed` — szybkość, z jaką zwiększa się wielkość obiektu,
`nooCol` — jasność poszczególnych klipów,
`noo` — nazwa bieżącego obiektu.

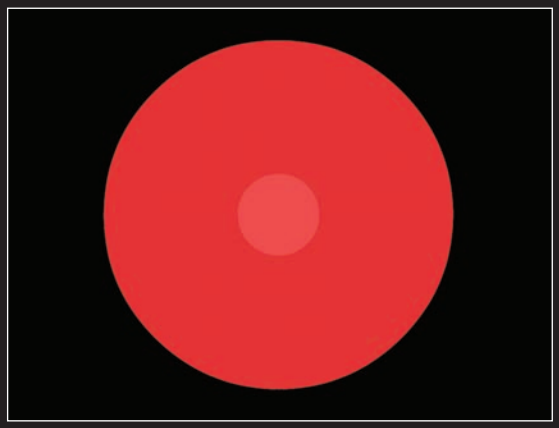
Omówię teraz działanie funkcji onEnterFrame oraz expand. Funkcja powiązana ze zdarzeniem onEnterFrame wywoływana jest w każdej klatce klipu, a jej działanie rozpoczyna się od wygenerowania losowej liczby całkowitej z przedziału od zera do wartości zawartej w zmiennej frequency (zadeklarowanej wcześniej). Jeśli otrzymana liczba będzie miała wartość zero, to przy użyciu metody attachMovie utworzona zostanie nowa kopia klipu ball. W następnym kroku funkcja onEnterFrame powoduje umieszczenie tego klipu w odpowiednim miejscu na scenie i nadaje mu odpowiednią jasność i rozmiar. Ponadto w obrębie tej funkcji wywoływane są kolejne funkcje decydujące o wyglądzie i parametrach utworzonej kopii klipu ball. Na koniec funkcja przekazuje uchwyt zdarzenia onEnterFrame funkcji expand, którą omówię za chwilę. Dzięki takiemu rozwiązaniu za każdym razem, gdy w obrębie funkcji expand nastąpi odwołanie do parametru this._x, będzie ono dotyczyło współrzędnej *x* tej kopii klipu z piłką, w której to odwołanie nastąpiło.

W funkcji expand mają miejsce wszystkie ważniejsze obliczenia, jakie wykonywane są podczas odtwarzania filmu. Przede wszystkim następuje w niej zwiększenie wartości decydujących o charakterze oscylacji — lr, hRad oraz vRad. Jak widać, w wyrażeniach odwołujących się do parametrów _x oraz _y klipu wykorzystane są funkcje trygonometryczne. Wartość tych parametrów obliczana jest na podstawie bieżącej wartości zmiennych hRad i vRad, pomnożonej przez sinus lub cosinus wyrażenia opierającego się na zmiennej lr. Prócz tego w omawianych wyrażeniach uwzględnione są też współrzędne środka sceny. Jak wspomniałem we wstępie, sinus dowolnego kąta osiąga wartości z przedziału od –1 do 1, a zatem wartość obliczonej w ten sposób współrzędnej *x* klipu mieści się w granicach wyznaczonych z jednej strony przez wartość parametru stage.width pomniejszoną o zmienną hRad, z drugiej zaś przez wartość parametru stage.width powiększoną o tę samą zmienną. Dzięki właściwościom funkcji sinus położenie klipu w osi poziomej będzie płynnie zmieniać się pomiędzy tymi dwoma granicami. Klip jest następnie proporcjonalnie skalowany na podstawie wartości parametru scaleSpeed, a na koniec — jeśli w wyniku skalowania osiągnięty został zakładany rozmiar maksymalny — stopniowo zmniejszana jest jasność obiektu. Obiekt jest usuwany ze sceny dopiero w chwili, gdy jest niemal przezroczysty, aby zapobiec jego nagłemu zniknięciu z ekranu.



towardsUs2

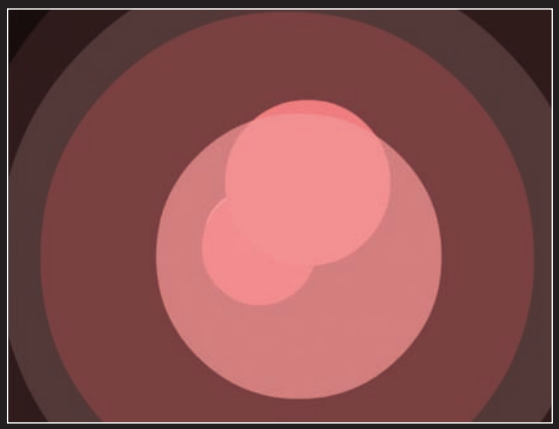
W tej wersji efektu postanowiłem zmienić sposób animacji obiektu tak, by zamiast oscylować względem wspólnego środka, piłki po prostu gwałtownie przybliżały się do oglądającego. Przy okazji przyspieszyłem ich animację, by całość sprawiała lepsze wrażenie i była bardziej dynamiczna. Pozostałe modyfikacje to już czysto kosmetyczne poprawki. Oto lista zmian, które wprowadziłem w znanym Ci już kodzie:



```
var scaleMax:Number = 600;
var frequency:Number = 5;
var colMax:Number = 90;
function hRad(Void):Number {
   return 0;
}
function vRad(Void):Number {
   return 0;
}
function vRadInc(Void):Number {
   return 0;
}
function hRadInc(Void):Number {
   return 0;
}
function scaleUpSpeed(Void):Number {
   return 1.2;
}
```

towardsUs3

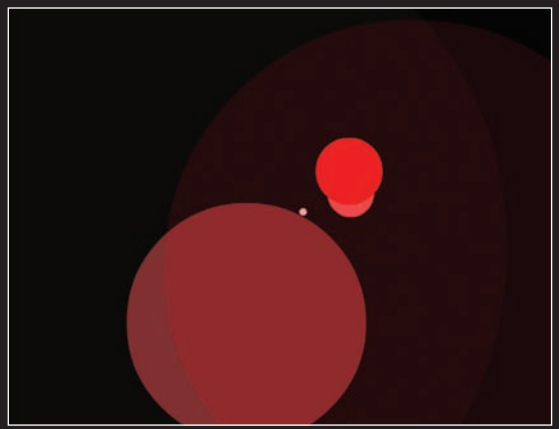
W tym wariancie oscylacje piłek są stałe, a nie — jak w pierwszej wersji projektu — narastające. Skróciłem też czas, po którym piłki znikają z ekranu, otrzymując w ten sposób dynamiczny efekt polegający na wirowaniu gwałtownie przybliżających się obiektów.



```
var fadeOut:Number = 0.7;
function hRad(Void):Number {
   return 40;
}
function vRad(Void):Number {
   return 40;
}
```

towardsUs4

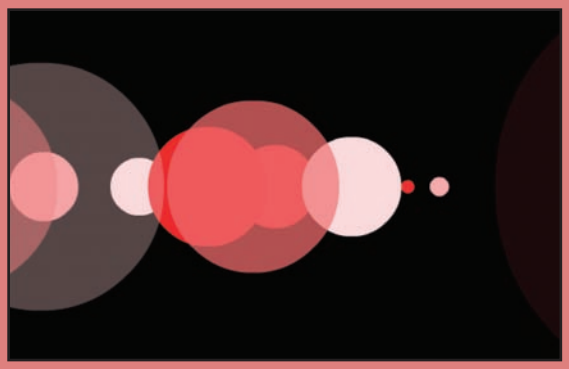
W tej wersji piłki pojawiają się na ekranie „znikąd", by zatoczając coraz szersze, spiralne kręgi, zniknąć za jego krawędziami.



```
var scaleMax:Number = 400;
function hRad(Void):Number {
   return 0;
}
function vRad(Void):Number {
   return 0;
}
function vRadInc(Void):Number {
   return 5;
}
function hRadInc(Void):Number {
   return 5;
}
```

towardsUs5

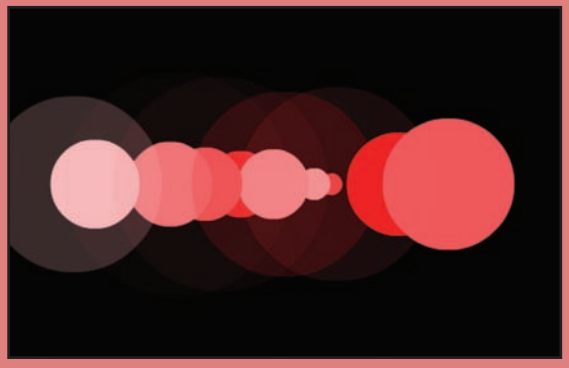
W tym wariancie postanowiłem przemieszczać klipy jedynie w kierunku poziomym; nieprzerwany strumień piłek już po chwili zajmuje całą szerokość ekranu.



```
var frequency:Number = 2;
function vRadInc(Void):Number {
   return 0;
}
function hRadInc(Void):Number {
   return 20;
}
```

towardsUs6

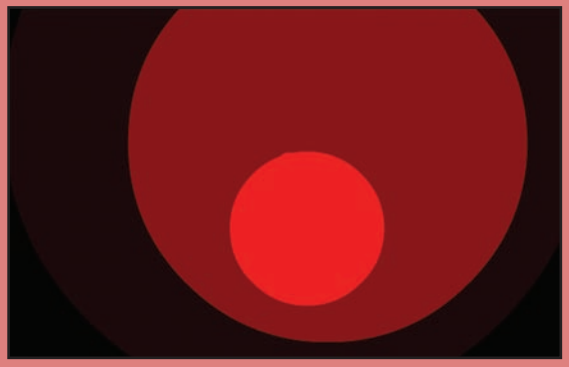
Tym razem postanowiłem nieznacznie zmniejszyć dynamikę animacji. Piłki poruszają się bardziej leniwie, przemieszczając się pomiędzy bocznymi krawędziami sceny, by wreszcie zniknąć w chwili, gdy nabiorą odpowiednich rozmiarów.



```
function hRadInc(Void):Number {
   return 5 * Math.random();
}
function lrSpeed(Void):Number {
   return 3 + Math.random() * 3;
}
function scaleUpSpeed(Void):Number {
   return 1.05;
}
```

towardsUs7

W kolejnej odmianie omawianego efektu postanowiłem ponownie poeksperymentować z ruchem spiralnym, tym razem jednak — dzięki temu, że zrezygnowałem z wartości losowych — wszystkie kółka podążają w przybliżeniu tym samym torem.



```
var scaleMax:Number = 900;
var frequency:Number = 10;
var colMax:Number = 20;
function vRadInc(Void):Number {
   return 1.5;
}
function hRadInc(Void):Number {
   return 1.3;
}
function lrSpeed(Void):Number {
   return 10;
}
function scaleUpSpeed(Void):Number {
   return 1.1;
}
```

towardsUs8

W tym wariancie kółka bardzo szybko zbliżają się do oglądającego.

```
var colMin:Number = 70;
var colMax:Number = 90;
function hRad(Void):Number {
   return 5;
}
function vRad(Void):Number {
   return 25;
}
function vRadInc(Void):Number {
   return 0;
}
function hRadInc(Void):Number {
   return 0;
}
function lrSpeed(Void):Number {
   return 40;
}
```

towardsUs9

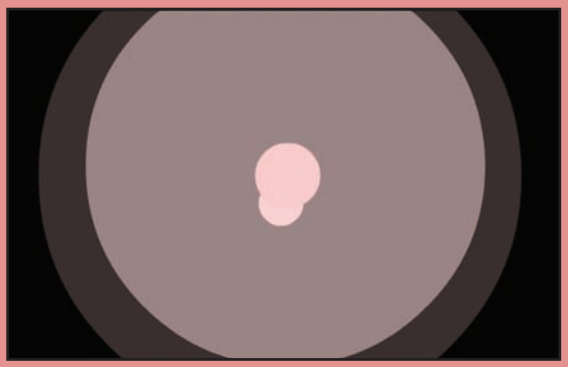
W tym eksperymencie zwiększyłem szybkość poziomych oscylacji klipu w taki sposób, że kółka wydają się pojawiać w trzech wyraźnie określonych punktach na ekranie.

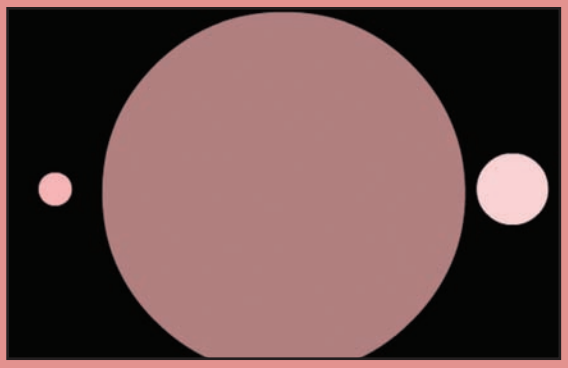
```
function hRad(Void):Number {
   return 300;
}
function vRad(Void):Number {
   return 9;
}
function lrSpeed(Void):Number {
   return 60;
}
```

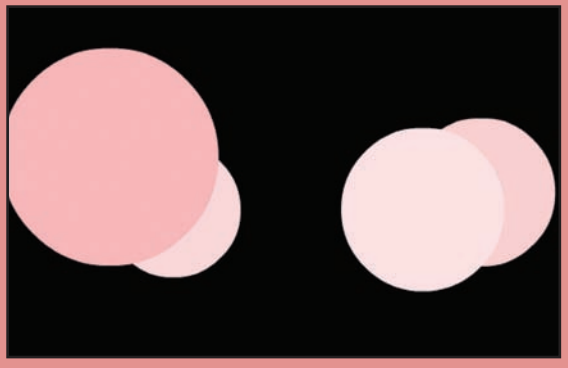
towardsUs10

Ostatnia odmiana tego projektu różni się od poprzednich zmniejszoną dynamiką poziomych oscylacji i wytłumieniem szybkości animacji. Dzięki temu piłki poruszają się po łagodniejszej, szerszej spirali.

```
function hRad(Void):Number {
   return 200;
}
function vRadInc(Void):Number {
   return 1;
}
// szybkość, z jaką rośnie pozioma składowa
// oscylacji
function hRadInc(Void):Number {
   return 1;
}
// szybkość oscylacji
function lrSpeed(Void):Number {
   return 10 + Math.random() * 10;
}
```







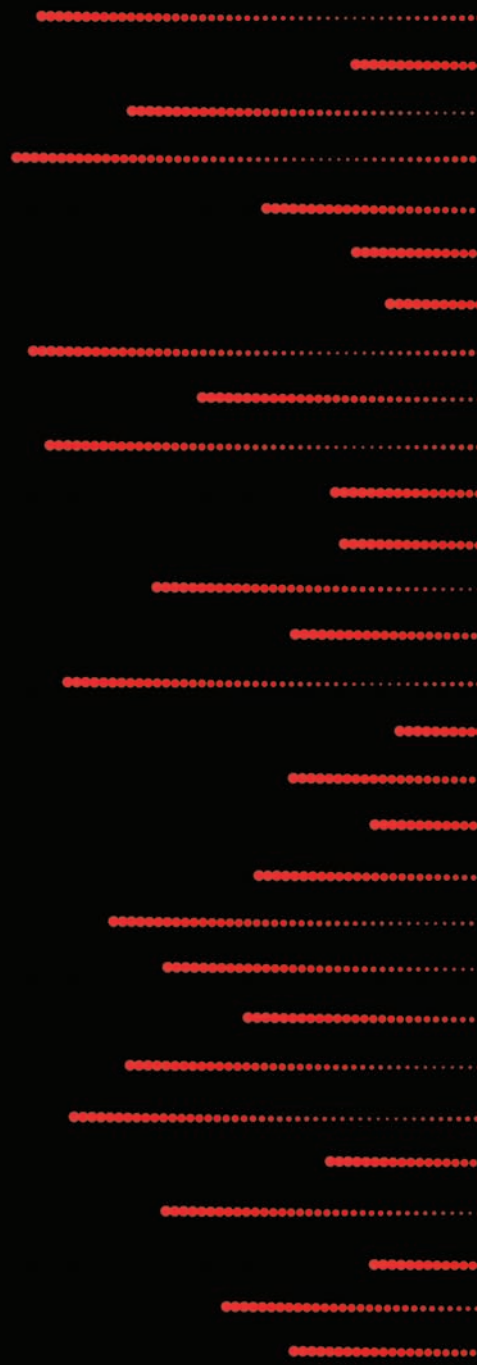
Omówiony projekt można modyfikować na wiele różnych sposobów. Interesujące efekty da się uzyskać choćby poprzez zmianę niektórych parametrów: warto wypróbować warianty polegające na oscylacji wyłącznie względem osi *x*, podjąć próbę uzyskania dynamicznej zmiany szybkości oscylacji w trakcie „życia" pojedynczego klipu, zaprogramować zmianę przezroczystości tak, by odbywała się ona wcześniej, czy spowolnić tempo skalowania klipów. Prócz tego warto zastąpić funkcje sinus i cosinus innymi funkcjami trygonometrycznymi — na przykład funkcją tangens — by przekonać się, jaki wpływ będzie miała taka zmiana na trajektorie ruchu piłek.

inLine (w jednej linii)

Efekt ten polega na cyklicznych zmianach wielkości obiektów ułożonych w jednej linii. Zmiany te obliczane są przy użyciu funkcji sinus, a początkowa wielkość każdego obiektu (wynikająca z wartości kąta będącego argumentem funkcji) uzależniona jest od jego odległości od wybranego punktu sceny. Cykliczne zmiany wszystkich obiektów w linii tworzą efekt fali — „reakcji łańcuchowej" — przesuwającej się od środka na zewnątrz linii.

Podobnie jak poprzednio, cały kod źródłowy powinien zostać umieszczony w pierwszej klatce głównej listwy czasowej (wraz z funkcją setBrightness, o której mówiłem na wstępie):

```
var hsp:Number = 4;
var total:Number = 70;
var twidth:Number = (total-1)*(hsp);
var brmin:Number = 0;
var brmax:Number = 40;

function inc(val:Number):Number {
   return 3;
}
function colinc(val:Number):Number {
   return 4;
}
function yMag(val:Number):Number {
   return 1;
}
function minScale(val:Number):Number {
   return 1;
}
function maxScale(val:Number):Number {
   return 12;
}
function startDegree(val:Number) {
   return 3 * val;
}
for(var i:Number = 0; i<total; i++) {
   var noo:MovieClip = _root.attachMovie("ball", "ball"
   ➥ + i, i);
   var offset:Number = Math.abs((total / 2) - i);
   noo._y = Stage.height / 2;
   noo._x = (Stage.width - twidth) / 2 + hsp * i;
   noo.baseY = Stage.height / 2;
   noo._xscale = noo._yscale = minScale();
   noo.inc = inc(offset);
   noo.colinc = colinc(offset);
   noo.col = new Color(noo);
   noo.brmin = brmin;
   noo.brmax = brmax;
   noo.degree = noo.coldegree = startDegree(offset);
   noo.brvariation = noo.brmax - noo.brmin;
   noo.yMag = yMag(offset);
   noo.minScale = minScale(offset);
   noo.maxScale = maxScale(offset);
   noo.variation = noo.maxScale - noo.minScale;
   noo.onEnterFrame = oscillate;
}
```



```
function oscillate(Void):Void {
   this.degree += this.inc;
   var value:Number = Math.sin(this.degree * Math.PI/180);
   this._xscale = this._yscale = this.minScale + (this.variation/2)
   ➥ + (this.variation/2) * value;
   this.coldegree += this.colinc;
   var value2:Number = Math.sin(this.coldegree * Math.PI/180);
   var brightness:Number = this.brmin + (this.brvariation/2) + (this.brvariation/2) * value2;
   setBrightness(this.col, brightness);
   this._y = this.baseY + value2 * this.yMag;
}
```

Najważniejsze zmienne i funkcje

`hsp` — odstęp pomiędzy obiektami na osi *x*,
`total` — całkowita liczba obiektów,
`twidth` — całkowita szerokość łańcucha obiektów,
`brmin` — minimalna jasność,
`brmax` — maksymalna jasność,
`inc` — szybkość zmian wielkości,
`colinc` — szybkość zmian jasności,
`yMag` — dynamika oscylacji w pionie,
`minScale` — minimalna wielkość obiektu,
`maxScale` — maksymalna wielkość obiektu,
`variation` — różnica pomiędzy wartościami parametrów minScale i maxScale,
`brvariation` — różnica pomiędzy wartościami parametrów brmin i brmax,
`startDegree` — kąt początkowy dla każdego obiektu,
`offset` — odległość od punktu odniesienia (środka sceny),
`noo` — nazwa bieżącego obiektu,
`degree` i `coldegree` — liczniki,
`value` — sinus kąta wyrażonego parametrem degree,
`value2` — sinus kąta wyrażonego parametrem coldegree.

Zwróć uwagę, że niektóre z wymienionych funkcji przyjmują argument o nazwie val, który określa odległość każdego obiektu od środka sceny. Ponieważ w przypadku tego projektu argumentem tym jest zawsze zmienna offset, wartości te możesz traktować jako tożsame.

W tej wersji projektu niepotrzebna okazała się funkcja onEnterFrame, gdyż parametry poszczególnych klipów definiowane są na samym początku działania programu, a potem animacja niejako „dzieje się sama". Wystarczy rozmieścić poszczególne klipy przy użyciu pętli i powiązać uchwyt zdarzenia onEnterFrame każdego z nich z funkcją oscillate. W funkcji tej następuje cykliczne zwiększenie wartości dwóch zmiennych — liczników — degree i coldegree. Liczniki te stanowią argumenty funkcji sinus, która wykorzystana została do obliczenia jasności i wielkości poszczególnych obiektów.

inLine2

Po wprowadzeniu następujących zmian odstępy pomiędzy obiektami ulegną zmniejszeniu, dzięki czemu łańcuch obiektów będzie sprawiał bardziej jednolite, spójne wrażenie.

```
var hsp:Number = 3;
function inc(val:Number):Number {
   return 10;
}
function colinc(val:Number):Number {
   return 40;
}
function minScale(val:Number):Number {
   return 10;
}
function maxScale(val:Number):Number {
   return 24;
}
```

inLine3

W tym wariancie łańcuch obiektów jest nieco grubszy i bardziej sprężysty; nieco silniej został też zaakcentowany efekt fali.

```
function yMag(val:Number):Number {
   return 3;
}
function minScale(val:Number):Number {
   return 20;
}
function maxScale(val:Number):Number {
   return 30;
}
function startDegree(val:Number) {
   return 9 * val;
}
```

inLine4

W tej wersji skorygowałem sposób obliczania wartości funkcji `inc`, aby uzyskać bardziej dynamiczny, pulsujący efekt, a jednocześnie zmniejszyłem „sprężystość” łańcucha.

```
function inc(val:Number):Number {
   return 3 * val;
}
function yMag(val:Number):Number {
   return 0;
}
function startDegree(val:Number) {
   return 5 * val;
}
```

inLine5

Tym razem zmodyfikowałem sposób obliczania wielu różnych wartości, w tym również definicję funkcji `startDegree`, dzięki czemu poszczególne obiekty są bardziej rozproszone i przestają sprawiać wrażenie jednolitej konstrukcji.

```
function inc(val:Number):Number {
   return 10 * val;
}
function yMag(val:Number):Number {
   return 3 * val;
}
function minScale(val:Number):Number {
   return 10;
}
function maxScale(val:Number):Number {
   return 20;
}
function startDegree(val:Number) {
   return 35 * val;
}
```

inLine6

W tej wersji, dzięki całkowitej zmianie sposobu obliczania wartości zwracanej przez funkcje `startDegree` i `yMag` (zastosowałem dzielenie zamiast mnożenia), uzyskałem efekt o zupełnie odmiennym charakterze.

```
var hsp:Number = 4;
var total:Number = 50;
function yMag(val:Number):Number {
   return 30 / (val / 3);
}
function startDegree(val:Number) {
   return 35 / val;
}
```

inLine7

I ponownie prosty łańcuch obiektów, tym razem jednak w postaci szybko pulsującej fali.

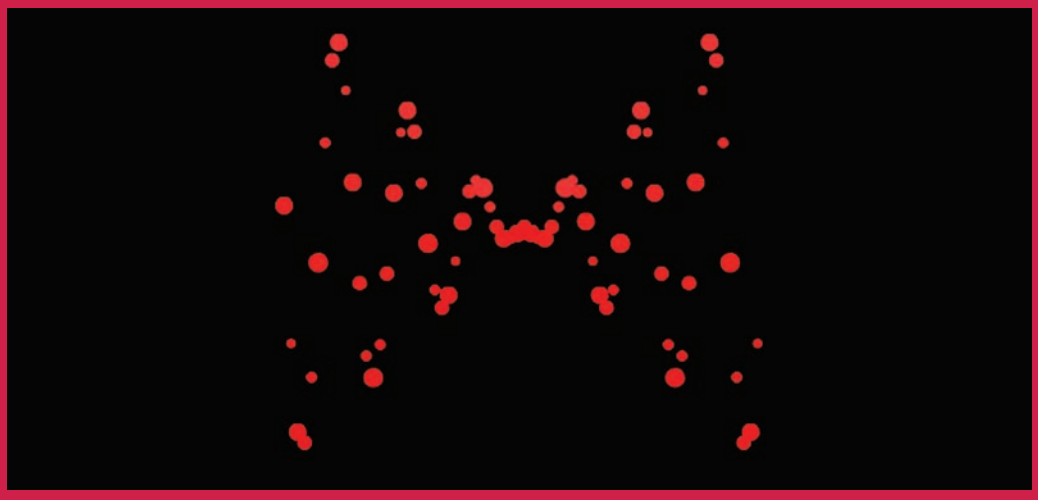
```
function yMag(val:Number):Number {
   return 0;
}
function maxScale(val:Number):Number {
   return 30 + 10 / (val / 3);
}
function startDegree(val:Number) {
   return val;
}
```

Podobnie jak w przypadku poprzedniego projektu, wystarczy zmiana wybranych parametrów lub sposobu obliczania wartości zwracanych przez niektóre funkcje, by całkowicie zmienić charakter otrzymanego efektu. Pomijając opisane warianty, projekt ten można przekształcić na szereg rozmaitych sposobów: na przykład przenieść punkt odniesienia służący do obliczania położenia każdego z obiektów ze środka na lewą stronę sceny i uzależnić od niego dynamikę oscylacji obiektów.

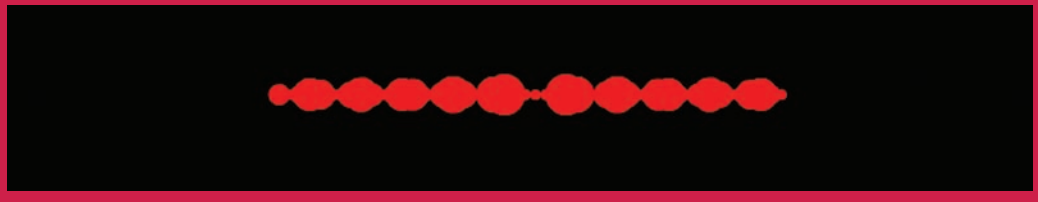

[2]


[3]


[4]


[5]


[6]

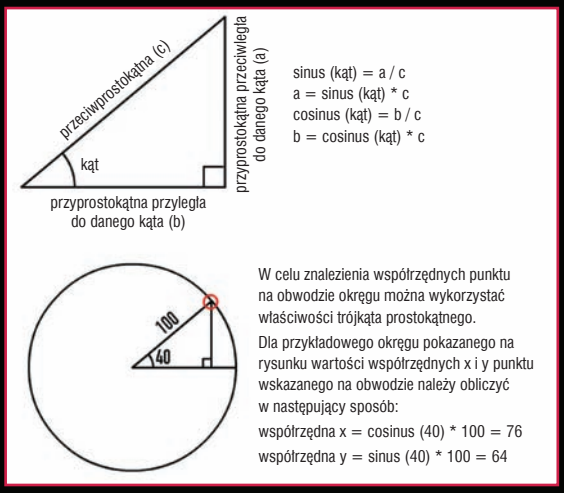

[7]

rightToLeft (prawy do lewego)

Efekt ten polega na utworzeniu pewnej liczby kółek wirujących wokół punktu, który przemieszcza się od prawej do lewej strony ekranu, nieznacznie oscylując przy tym w górę i w dół. Pionowa składowa ruchu tego punktu obliczona zostanie w sposób analogiczny jak w przypadku poprzedniego efektu: na podstawie sinusa parametru, którego wartość będzie cyklicznie zwiększana. W celu uzyskania wrażenia ruchu wirowego trzeba będzie użyć funkcji sinus i cosinus w nieco inny sposób. Otóż dla dowolnego okręgu można wyliczyć współrzędne *x* oraz *y* dowolnego punktu leżącego na jego obwodzie, jeśli tylko znany jest promień tego okręgu i kąt, pod jakim punkt ten jest położony względem jego środka. Współrzędne te można obliczyć na podstawie właściwości trójkąta prostokątnego, którego wierzchołki wyznaczone są przez środek okręgu oraz szukany punkt, znajdujący się na jego obwodzie (patrz rysunek).

Przełożone na język Flasha wzory umożliwiające obliczenie współrzędnych punktu wyglądają następująco:

```
xposition = radius * Math.cos(degrees*Math.PI/180)
yposition = radius * Math.sin(degrees*Math.PI/180)
```



Teraz można już przystąpić do programowania. Oto kod, który należy umieścić w pierwszej klatce głównej listwy czasowej projektu:

```
var frequency:Number = 30;
var colMin:Number = 0;
var colMax:Number = 50;
var colVariance:Number = colMax - colMin;

function leftRightSpeed(Void):Number {
   return -2;
}
function maxScale(Void):Number {
   return 120;
}
function minScale(Void):Number {
   return 60;
}
function leftRightRadius(Void):Number {
   return 150;
}
function circlingSpeed(Void):Number {
   return 5;
}
function circleStartPoint(Void):Number {
   return 0;
}
function upDownRange(Void):Number {
   return 10;
}
function yFreqInc(Void):Number {
   return 12;
}
```

```
function nooCol(val):Number {
   val *= 30;
   return colMin + colVariance * 0.5 + (0.5 *
   ➥ colVariance) * Math.sin(val * Math.PI / 180);
}

var g:Number = 0;
var depth:Number = 0;
onEnterFrame = function(Void):Void {
   g++;
   if (g > frequency) {
      g = 0;
      depth++;
      var noo = attachMovie("ball", "ball"
      ➥ + depth, depth);
      noo._y = Stage.height / 2;
      noo.fulcrumX = noo._x = Stage.width + 30;
      noo.maxScale = maxScale();
      noo.minScale = minScale();
      var col:Color = new Color(noo);
      setBrightness(col, nooCol(depth));
      noo.variance = noo.maxScale - noo.minScale;
      noo.acrossRadius = leftRightRadius();
      noo.upDownRange = upDownRange();
      noo.degree = circleStartPoint();
      noo.degreeInc = circlingSpeed();
      noo.yFreq = 0;
      noo.yFreqInc = yFreqInc();
      noo.leftRightSpeed = leftRightSpeed();
      noo.onEnterFrame = shootMeAcross;
   }
};
```

```
function shootMeAcross(Void):Void {
   this.fulcrumX += this.leftRightSpeed;
   this.degree += this.degreeInc;
   this._x = this.fulcrumX+Math.cos(this.degree * Math.PI / 180) * this.acrossRadius;
   this._xscale = this._yscale = this.minScale+(this.variance * 0.5) + (this.variance * 0.5)
   ➥ * Math.sin(this.degree * Math.PI / 180);
   this.yFreq += this.yFreqInc;
   this._y = Stage.height / 2 + this.upDownRange * Math.sin(this.yFreq * Math.PI / 180);
   this.swapDepths(Math.floor(this._xscale));
   if (this._x < -40) {
      this.removeMovieClip();
   }
}
```

Najważniejsze zmienne i funkcje

`frequency` — częstotliwość tworzenia nowych obiektów,
`colMin` — minimalna jasność obiektu,
`colMax` — maksymalna jasność obiektu,
`colVariance` — zróżnicowanie jasności,
`leftRightSpeed` — szybkość, z jaką przemieszczają się obiekty w poprzek ekranu,
`maxScale` — maksymalna wielkość obiektu,
`minScale` — minimalna wielkość obiektu,
`leftRightRadius` — promień okręgu, po którym poruszają się obiekty,
`circlingSpeed` — szybkość, z jaką obiekt porusza się po okręgu,
`circleStartPoint` — punkt początkowy, w którym obiekt rozpoczyna ruch po okręgu,
`upDownRange` — zasięg oscylacji w kierunku pionowym,
`yFreqInc` — szybkość oscylacji w kierunku pionowym,
`nooCol` —jasność poszczególnych obiektów; zwiększenie wartości parametru val spowoduje przyspieszenie oscylacji kolorów,
`noo` — nazwa bieżącego obiektu,
`fulcrumX` — punkt, wokół którego obracają się obiekty; zmiana położenia tego punktu powoduje przesunięcie całego wirującego „układu".

Większość obliczeń odbywa się w funkcji shootMeAcross, która powiązana jest z uchwytem zdarzenia onEnterFrame każdego klipu. Działanie tej funkcji rozpoczyna się od przesunięcia punktu, wokół którego orbituje klip, po czym następuje zwiększenie wartości parametru degree, który odpowiada za uzyskanie ruchu wirowego. Zwiększana jest również wartość parametru yfreq, od którego uzależnione jest położenie obiektu względem osi *y*. Współrzędna *x* obiektu obliczana jest na podstawie cosinusa wartości parametru degree, zaś wielkość tego obiektu — na podstawie sinusa tego samego parametru (zmiana wielkości obiektu zastępuje użycie parametru *y* i ma imitować efekt jego przybliżania się i oddalania od ekranu, zaś samo kółko — będąc obiektem dwuwymiarowym — zwrócone jest cały czas tą samą stroną do oglądającego). Po wykonaniu tych obliczeń na podstawie parametru yfreq ustalana jest wartość współrzędnej *y*, która decyduje o sposobie oscylacji obiektu w górę i w dół. Ostatni fragment funkcji shootMeAcross powoduje usunięcie klipu, który znalazł się poza lewą krawędzią ekranu.

rightToLeft2

Dzięki dużej zmianie wartości parametru `upDownRange` w tej wersji efektu zasięg pionowych oscylacji obiektów został znacznie zwiększony. Zmiana ta miała na celu uzyskanie jeszcze lepszego złudzenia trójwymiarowej przestrzeni, w której poruszają się obiekty.

```
function leftRightRadius(Void):Number {
   return 50;
}
function circlingSpeed(Void):Number {
   return 10;
}
function upDownRange(Void):Number {
   return 70;
}
function yFreqInc(Void):Number {
   return 4;

}
```

rightToLeft3

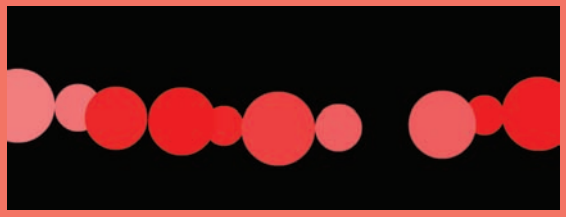
W tej wersji zwiększyłem liczbę obiektów, które wydają się teraz płynąć nieprzerwanym strumieniem. Nieznacznie zmodyfikowałem też ścieżkę, po której się poruszają.

```
var frequency:Number = 3;
function leftRightSpeed(Void):Number {
   return -3;
}
function maxScale(Void):Number {
return 180;
}
function leftRightRadius(Void):Number {
   return 60;
}
function circlingSpeed(Void):Number {
   return 8;
}
```

rightToLeft4

W tym wariancie zmodyfikowałem sposób skalowania oraz wartość zwracaną przez funkcję `upDownRange`, by obiekty ułożyły się w bardziej jednolity, niemal płaski łańcuch, poprzerywany tu i ówdzie niewielkimi szczelinami.

```
var frequency:Number = 4;
function minScale(Void):Number {
   return 0;
}
function leftRightRadius(Void):Number {
   return 30;
}
function upDownRange(Void):Number {
   return 20;
}
function yFreqInc(Void):Number {
   return 1;
}
```



rightToLeft5

W tym przypadku zmieniłem wartości zwracane przez funkcje `upDownRange` oraz `yFreqInc`, aby wpłynąć na trajektorię ruchu obiektów: tym razem poruszają się one po pętli przypominającej ósemkę.

```
var colMax:Number = 100;
function maxScale(Void):Number {
   return 60;
}
function upDownRange(Void):Number {
   return 60;
}
function yFreqInc(Void):Number {
   return 10;
}
```

rightToLeft6

I jeszcze inny wariant pętli.

```
var colMax:Number = 20;
function leftRightRadius(Void):Number {
   return 50;
}
function upDownRange(Void):Number {
   return 120;
}
function yFreqInc(Void):Number {
   return 5;
}
```

rightToLeft7

W tej wersji znacznie zmniejszyłem wartość zwracaną przez funkcję upDownRange i poszerzyłem pętlę, po której poruszają się obiekty, by poprawić głębię i plastyczność animacji.

```
var frequency:Number = 1;
function leftRightSpeed(Void):Number {
   return -4;
}
function maxScale(Void):Number {
   return 40;
}
function leftRightRadius(Void):Number {
   return 80;
}
function circlingSpeed(Void):Number {
   return 5;
}
function upDownRange(Void):Number {
   return 19;
}
```

rightToLeft8

Nieznaczna modyfikacja kodu źródłowego pozwoliła na ujednolicenie trajektorii ruchu wszystkich obiektów; teraz układają się one w wijącego się węża.

```
var frequency:Number = 2;
function leftRightSpeed(Void):Number {
   return -3;
}
function maxScale(Void):Number {
   return 90;
}
function circlingSpeed(Void):Number {
   return 3;
}
function yFreqInc(Void):Number {
   return 10;
}
```

rightToLeft9

W ostatniej wersji efektu postanowiłem dodać trochę elementów losowych, które nadają całości nieco chaotyczny wygląd.

```
var frequency:Number = 3;
function leftRightSpeed(Void):Number {
   return -2 - Math.random();
}
function maxScale(Void):Number {
   return 120;
}
function upDownRange(Void):Number {
   return Math.random() * 50;
}
```

Opisany efekt można modyfikować na wiele różnych sposobów — wystarczy jedynie poeksperymentować z wartościami poszczególnych parametrów. Można na przykład zmieniać szybkość „wirowania" i szybkość przemieszczania się obiektów w poprzek ekranu, uzyskując w ten sposób rozmaite spirale i pętle. Warto też wypróbować możliwości, jakie daje zmiana kształtu poruszających się obiektów.

sinGrid (falująca siatka)

Ostatni efekt polega na cyklicznej zmianie koloru i jasności obiektów-kółek, ułożonych w postaci siatki. Stosując zmiany oparte na wartościach funkcji sinus i przypisując różne kąty początkowe sąsiadującym obiektom, w siatce takiej można wywołać efekt przypominający falę. Różnice pomiędzy wartościami kątów dla sąsiadujących obiektów można obliczać w różny sposób, lecz niezależnie od tego, który z nich zostanie użyty, warto poczynić jedno istotne założenie: podzielić cały użyteczny zakres argumentów funkcji sinus — czyli 360 stopni — przez liczbę obiektów w siatce. Przypuśćmy, że siatka składa się z dziesięciu kółek; wówczas należałoby przypisać im następujące kąty początkowe: 36, 72, 108, 144, 180, 216, 252, 288 oraz 324 stopnie. Przy takim założeniu początkowa wartość funkcji sinus dla każdego obiektu będzie nieco inna, lecz różnice pomiędzy tymi wartościami podczas animacji całego układu (gdy sinus dla każdego obiektu będzie wahał się pomiędzy 1 a –1) będą stałe. Takie rozwiązanie stanowi klucz do uzyskania efektu fali, która rozpoczynać się będzie w prawym dolnym rogu siatki i stopniowo, rząd po rzędzie przemieszczać się będzie w górę. Jeśli zmodyfikujesz sposób obliczania kątów początkowych tak, by obejmowały połowę użytecznego zakresu argumentów funkcji sinus — czyli od 0 do 180 stopni — to zgodnie z intuicją uzyskany efekt będzie odzwierciedlał jedynie połowę okresu funkcji sinus.

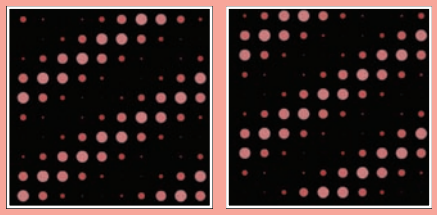
Podobnie jak w przypadku poprzednich projektów, cały podany kod należy umieścić w pierwszej klatce głównej listwy czasowej przygotowanego wcześniej „szablonu".

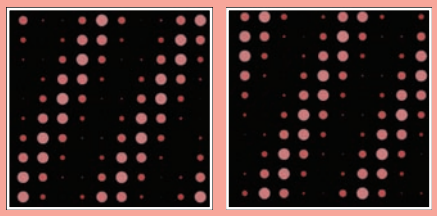
```
var across:Number = 10;
var down:Number = 10;
var total:Number = across*down;
var hsp:Number = 20;
var vsp:Number = 20;
var degInc:Number = 360 / total;
var numberOfOscillations:Number = 1;
var bx:Number = (Stage.width - hsp * across) / 2;
var by:Number = (Stage.height - vsp * down) / 2;

function increment(offset):Number {
   return 30;
}
function minScale(Void):Number {
   return 3;
}
function maxScale(Void):Number {
   return 54;
}
function minBrt(Void):Number {
   return 0;
}
function maxBrt(Void):Number {
   return 50;
}
var row:Number = 0;
var column:Number = 0;

for (i=0; i<total; i++) {
   var noo:MovieClip = attachMovie("ball", "circ" +
   ➥ i, i);
   noo._x = bx + column * hsp;
   noo._y = by + row * vsp;
   noo.col = new Color(noo);
   var offset:Number = Math.abs(total / 2 - i);
   noo.myInc = increment(offset);
   noo.minScale = minScale();
   noo.maxScale = maxScale();
   noo.variance = noo.maxScale - noo.minScale;
   noo.minBrt = minBrt();
   noo.maxBrt = maxBrt();
   noo.colVariance = noo.maxBrt - noo.minBrt;
   noo.onEnterFrame = undulate;
   noo.degree = i * degInc * numberOfOscillations;
   column++;
   if (column == across) {
      column = 0;
      row++;
   }
}

function undulate(Void):Void {
   this.degree += this.myInc;
   var sinVal:Number = Math.sin(this.degree *
   ➥ Math.PI/180);
   this._xscale = this._yscale = this.minScale +
   ➥ (this.variance * 0.5) + (this.variance * 0.5)
   ➥ * sinVal;
   var brightness:Number = this.minBrt +
   ➥ (0.5 * this.colVariance) + (0.5 * this.
   ➥ colVariance) * sinVal;
   setBrightness(this.col, brightness);
```

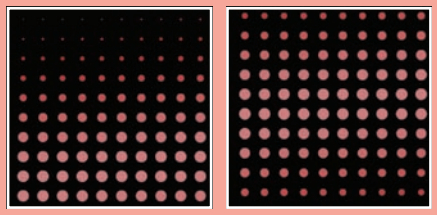
Najważniejsze zmienne i funkcje

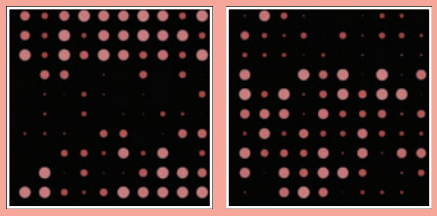
`across` — liczba kolumn,
`down` — liczba rzędów,
`total` — całkowita liczba obiektów,
`hsp` — odstępy w poziomie,
`vsp` — odstępy w pionie,
`degInc` — liczba stopni przypadająca na każdy obiekt, obliczona na podstawie całkowitej liczby obiektów potrzebnej do wyświetlenia pełnej fali,
`numberOfOscillations` — jaka część fali ma być jednocześnie wyświetlona na ekranie,
`bx` — początkowe położenie względem osi *x*,
`by` — początkowe położenie względem osi *y*,
`increment` — szybkość przemieszczania się fali; do funkcji tej można przekazać argument `offset`, odzwierciedlający odległość danego obiektu od wybranego punktu odniesienia siatki (początkowo możliwość ta nie jest wykorzystana, lecz zostanie zaimplementowana w kolejnych wariantach projektu),
`minScale` — minimalna wielkość obiektów,
`maxScale` — maksymalna wielkość obiektów,
`minBrt` — minimalna jasność obiektów,
`maxBrt` — maksymalna jasność obiektów,
`column` i `row` — bieżąca kolumna i wiersz siatki.

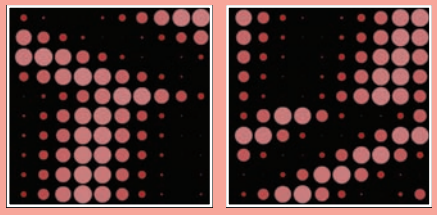
Siatka generowana jest przy użyciu zwykłej pętli for, a następnie cyklicznie aktualizowana za pomocą funkcji undulate, w której zawarłem większą część wszystkich niezbędnych obliczeń. Funkcja ta jest stosunkowo prosta: jej działanie, podobnie jak w poprzednich projektach, polega na zwiększaniu wartości parametru degree i zmianie wielkości oraz jasności poszczególnych klipów w oparciu o tę wartość.

[2]


[3]


[4]


[5]


[6]


sinGrid02

Ta prosta zmiana umożliwia uzyskanie pulsujących, dynamicznych fal.

```
var numberOfOscillations:Number = 12;
```

sinGrid03

Dalsze zwiększanie wartości parametru `numberOfOscillations` pozwala zmienić sposób pulsowania: fale rozchodzą się pionowo.

```
var numberOfOscillations:Number = 21;
```

sinGrid04

Zmniejszenie wartości wspomnianego parametru poniżej 1 sprawia, że w siatce pojawia się pojedynczy, poziomy impuls, biegnący od dołu do góry.

```
var numberOfOscillations:Number = 0.5;
```

sinGrid05

Włączenie elementu losowego do sposobu obliczania wartości zwracanej przez funkcję `increment` sprawia, że efekt rozpoczyna się wprawdzie od pojedynczego impulsu, lecz zaraz potem przeradza się w niekontrolowany chaos.

```
function increment(offset:Number):Number {
   return 10 * Math.floor(5 * Math.random());
}
```

sinGrid06

Ponownie zmieniłem treść funkcji `increment`. Tym razem chciałem uzyskać płynną zmiany kierunku rozchodzenia się impulsów, używając zmiennej `offset`.

```
function increment(offset:Number):Number {
   return 4 + offset;
}
```

sinGrid07

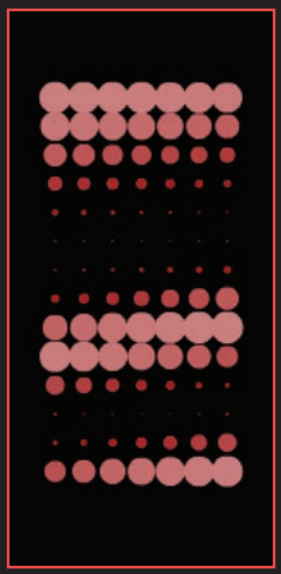
W tym przypadku nadałem siatce kształt pionowego, wydłużonego prostokąta.

```
var across:Number = 7;
var down:Number = 14;
var numberOfOscillations:Number = 100;
function increment(offset:Number):Number {
   return 14 + offset / 20;
}
```

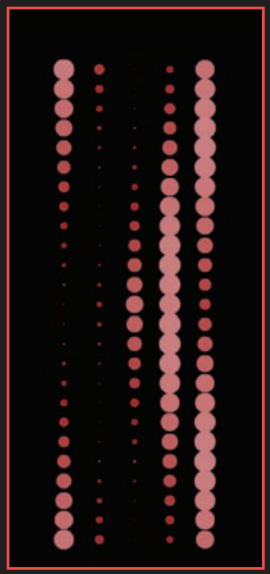
sinGrid08

W tej wersji postanowiłem po raz kolejny zmienić kształt siatki — tym razem jest ona bardzo wysoka i wąska. Parametry rozchodzenia się fali pozostały niezmienione.

```
var across:Number = 5;
var down:Number = 25;
```



[7]



[8]

sinGrid09

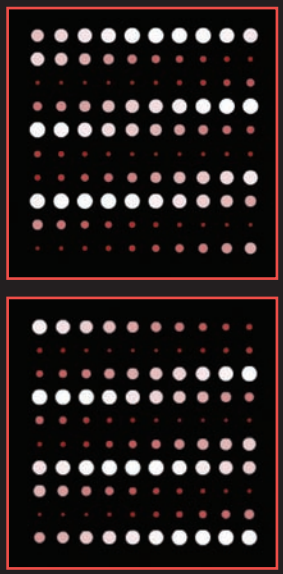
Postanowiłem przywrócić kwadratowy kształt siatki, tym razem jednak zmieniłem sposób zmiany jasności obiektów.

```
var across:Number = 10;
var down:Number = 10;
var numberOfOscillations:Number = 3;
function minScale(Void):Number {
   return 15;
}
function maxScale(Void):Number {
   return 60;
}
function maxBrt(Void):Number {
   return 100;
}
```

sinGrid10

Zmniejszenie odstępów pomiędzy obiektami i jednoczesne powiększenie rozmiarów samych obiektów sprawiło, że przestały one wyglądać jak oddzielne elementy siatki, lecz raczej jak bardziej złożone struktury, które rytmicznie pulsując, zmieniają zarówno swój kształt, jak i kolor.

```
var hsp:Number = 7;
var vsp:Number = 7;
var numberOfOscillations:Number = .33;
function minScale(Void):Number {
   return 45;
}
function maxScale(Void):Number {
   return 130;
}
```



[9]

[10]

Eksperymentując z tym efektem, możesz spróbować na przykład wykorzystać wartość parametru `offset` w taki sposób, by szybkość oscylacji wynikała z odległości poszczególnych obiektów od wybranego punktu odniesienia: od środka siatki, od któregoś z jej narożników itp. Zmiana wartości parametru `numberOfOscillations` umożliwia uzyskanie rozmaitych odmian falowania i pulsowania. Warto zauważyć, że kierunek rozchodzenia się fali w dowolnym z uzyskanych efektów można odwrócić, odejmując przyrost kąta dla każdego obiektu od 360 (gdyż wartość otrzymana dla kąta 320 stopni będzie przeciwna niż dla kąta wynoszącego 40 stopni).