# Тема №14
# Физика

проф. П. Бойчев ● КИТ ● ФМИ ● СУ ● 2021

# В тази лекция

## Физика

- Физика в компютърната графика
- Плавност на движението
- Търкаляне
- Вибрация и затихване
- Отблъскване и топане
- Балистика и скачане
- Вълнообразно движение

# Физика

# Физика

## Физика в компютърната графика

- Създава чувство за реалност
- Поддържа естествено поведение на обектите

## Физически точно моделиране

- Създава се точен модел
- Често е по-трудно за реализация

## Приближено моделиране

- Физичните явления се моделират приближено
- Физически неточно, но визуално приемливо
- Прилага се, ако изчисленията се олекотяват значително

# Физични закони

## Често използвани закони и явления

- Запазване на енергията
- Триене и съпротивление
- Привличане и гравитация
- Инерция

## Основни следствия

- Движенията винаги са плавни
- Без рязко тръгване, спиране, завиване

# Плавност

# Плавност

## Биология

- Човек усеща неплавна промяна в движението
- Усетът достига до 2-ра производна (ускорение)

## Плавност на движение

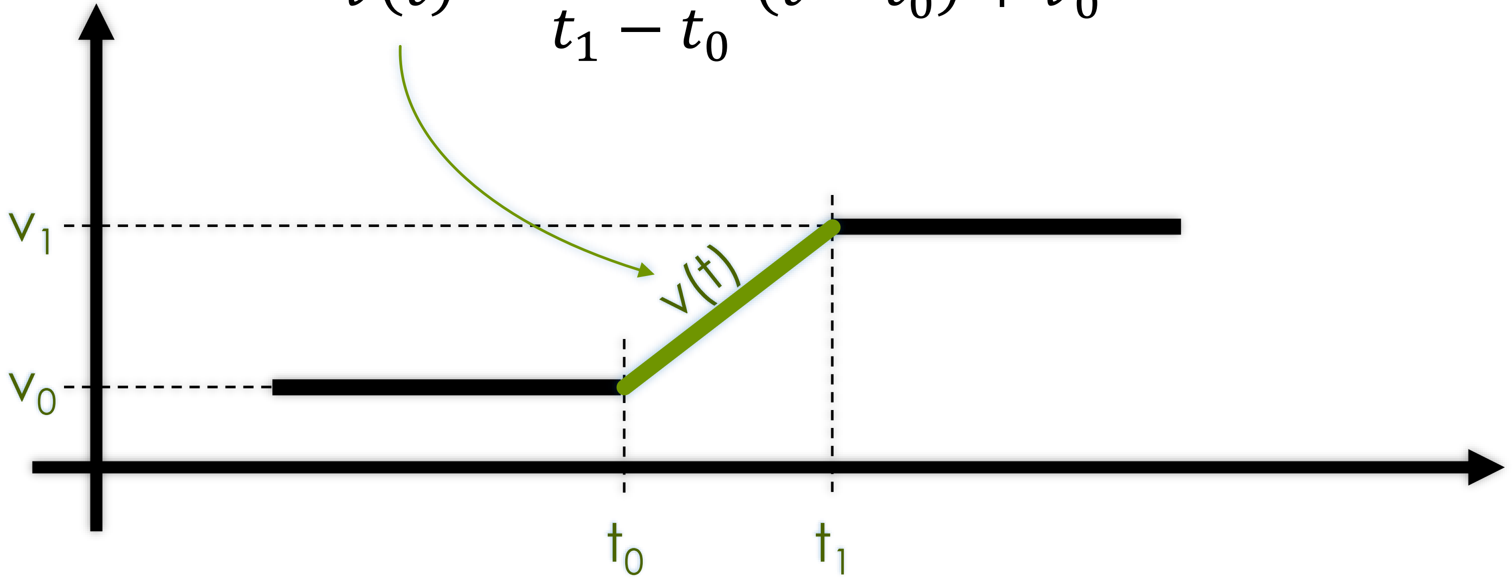
- Линейна плавност
- Плавно навлизане (easy-in)
- Плавно излизане (easy-out)

# Линейна плавност

## Линейна плавност

- Линейна промяна до достигане на желаната стойност
- Има прекъсване на ускорението в точки $\mathbf{t_0}$ и $\mathbf{t_1}$

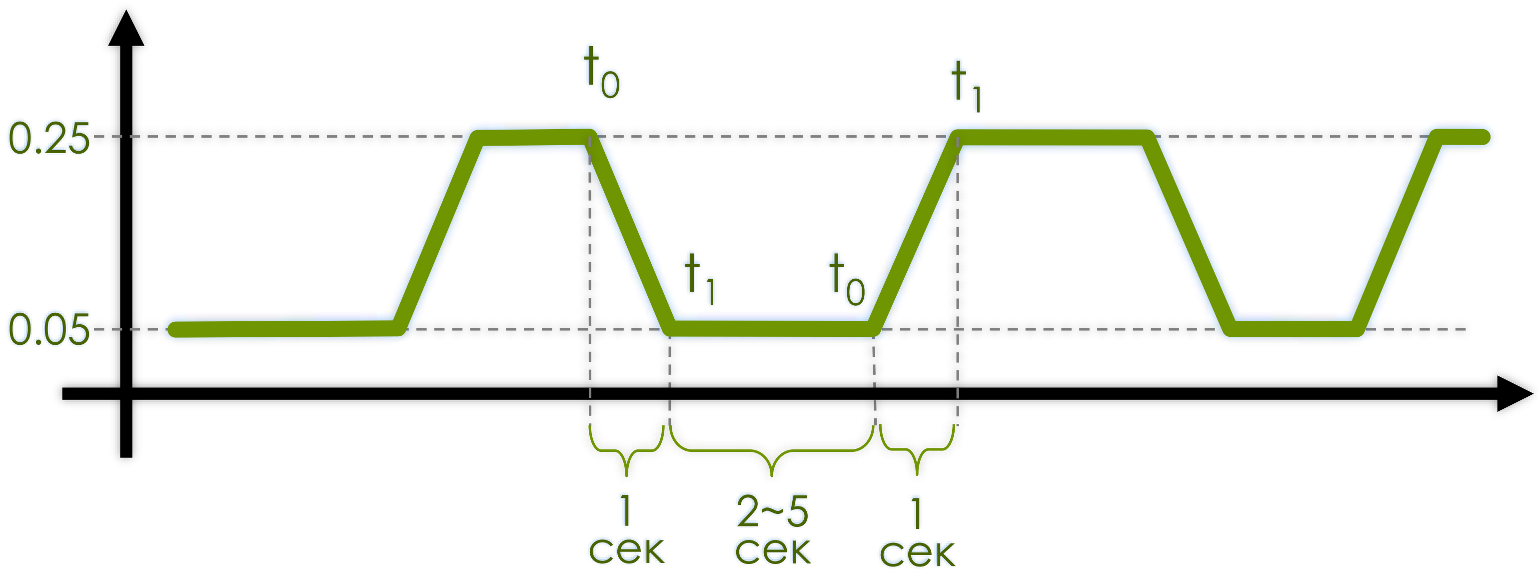
$$v(t) = \frac{v_1 - v_0}{t_1 - t_0}(t - t_0) + v_0$$



# Пример

## Плавен преход между скорости

- Обекти се движат по елипси
- Скоростите са две – една бавна и една 5 пъти по-бърза
- Всеки обект се движи няколко секунди с едната скорост, после няколко секунди с другата
- Преходът между скоростите е 1 секунда



# За всеки обект помним

- Кога е плавната промяна в $t_0$ и $t_1$
- Каква е промяната във $v_0$ и $v_1$
- Преди момент $t_0$ скоростта е $v_0$
- При преминаване на момент $t_1$ скоростта е $v_1$, но веднага определяме следващия интервал на промяна да е след между 2 и 5 секунди и с продължителност 1 секунда

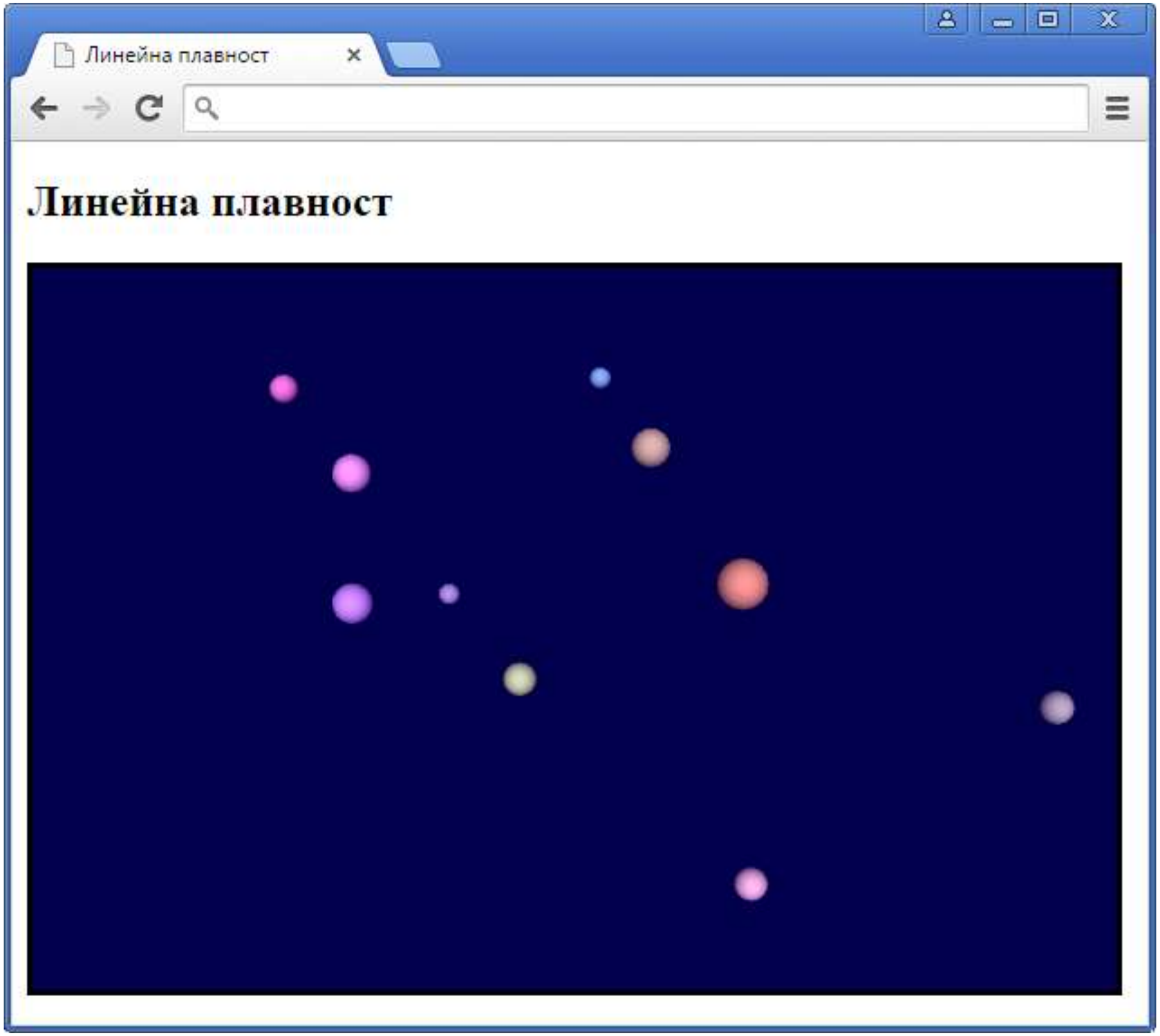
```
if (time<obj[i].t0)
{
   obj[i].v = obj[i].v0;
}
else if (time>obj[i].t1)
{
   obj[i].v = obj[i].v1;
   obj[i].t0 = time+random(2,5);
   obj[i].t1 = obj[i].t0+1;
   obj[i].v0 = obj[i].v1;
   obj[i].v1 = 0.3-obj[i].v1;
}
```

- Между **$t_0$** и **$t_1$** се изчислява скоростта **v** по линеен начин:

$$v(t) = \frac{v_1 - v_0}{t_1 - t_0}(t - t_0) + v_0$$

- Формулата работи както при забавяне на скоростта, така и при увеличаване на скоростта

```
if (time<obj[i].t0)
{
   :
}
else if (time>obj[i].t1)
{
   :
}
else
{
   obj[i].v = (obj[i].v1-obj[i].v0)/(obj[i].t1-obj[i].t0)*
                                        (time-obj[i].t0)+obj[i].v0;
}
```




Тест

Файлове

# Непрекъснатост

## Проблеми

- При линейната плавност производната е прекъсната
- Ако плавността е в координатите, скоростта е прекъсната
- Ако плавността е в скоростта, ускорението е прекъснато

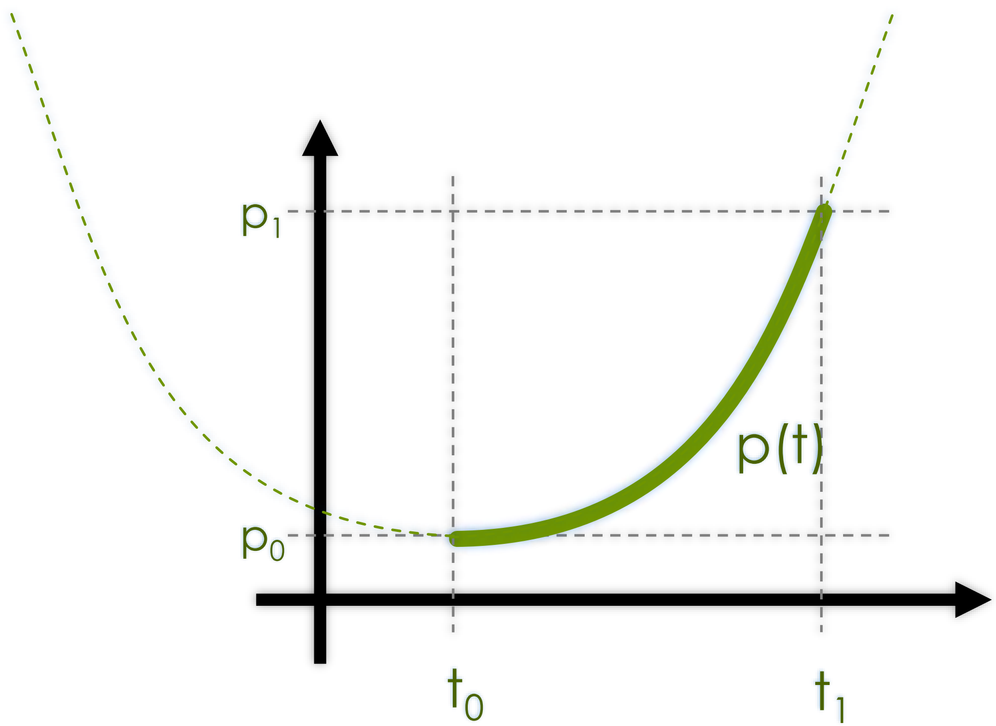
## Решение

- Използване на заглаждане
- Входно заглаждане, в началото на интервала (easy-in)
- Изходно заглаждане, в края на интервала (easy-out)
- Двойно заглаждане (easy-in-out)
- Заглаждаща функция по избор – полиномиална, експоненциална, тригонометрична, …

# Пример

- Входно загладжане с полином
- Загладжаща функция от втора степен:

$$p(t) = \frac{p_1 - p_0}{t_1^2 - t_0^2}(t^2 - t_0^2) + p_0$$

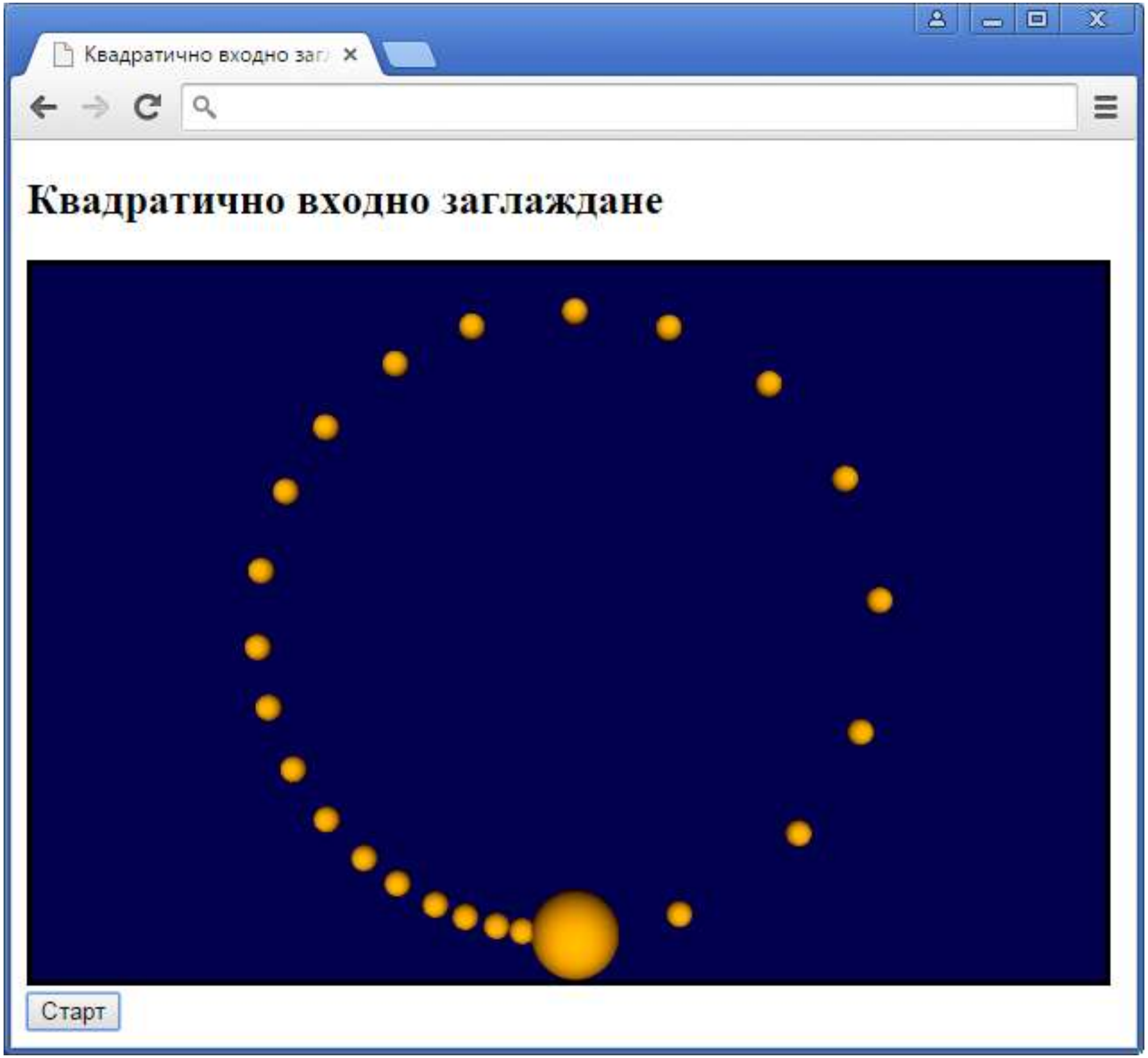


# Реализация

- Топче се движи от **$p_0$** до **$p_1$** във времевия интервал **$t_0$** до **$t_1$**
- Периодично се оставя следа, като се помни последният момент **tObj**, в който е била създадена следа
- Следата е масив от по-малки топчета, които показват къде е било голямото на всеки около 0.1 секунди

```
if (t0<=time && time<=t1)
{
   var a = (p1-p0)*(time*time-t0*t0)/(t1*t1-t0*t0)+p0;
   sph.center = [0,7*cos(a),7*sin(a)];

   if (tObj+0.1<time)
   {
      tObj = time;
      obj.push(new Sphere([0,sph.center[1], sph.center[2]],0.3));
   }
}
```

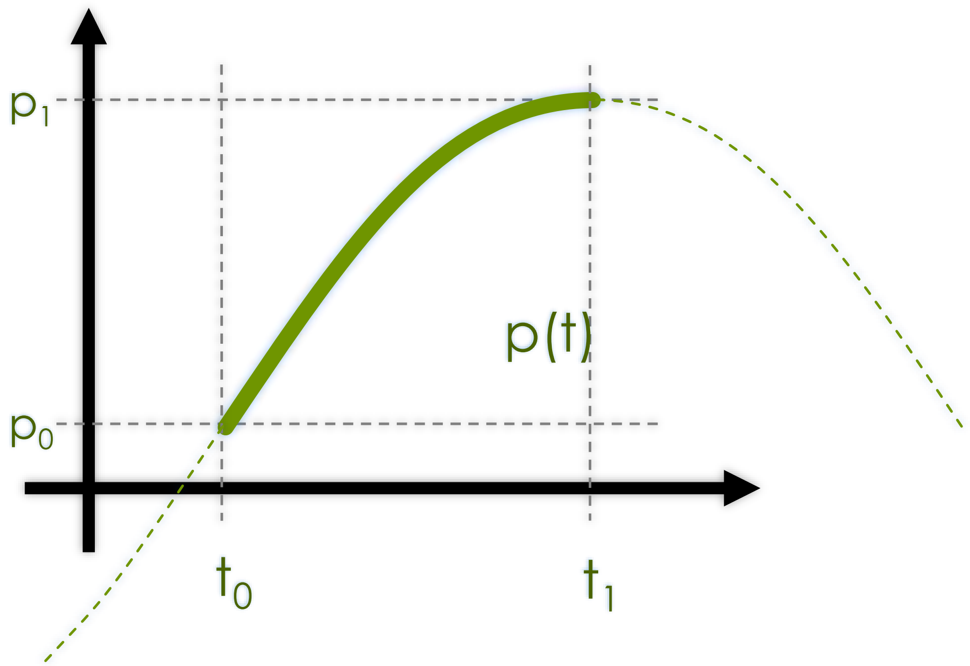


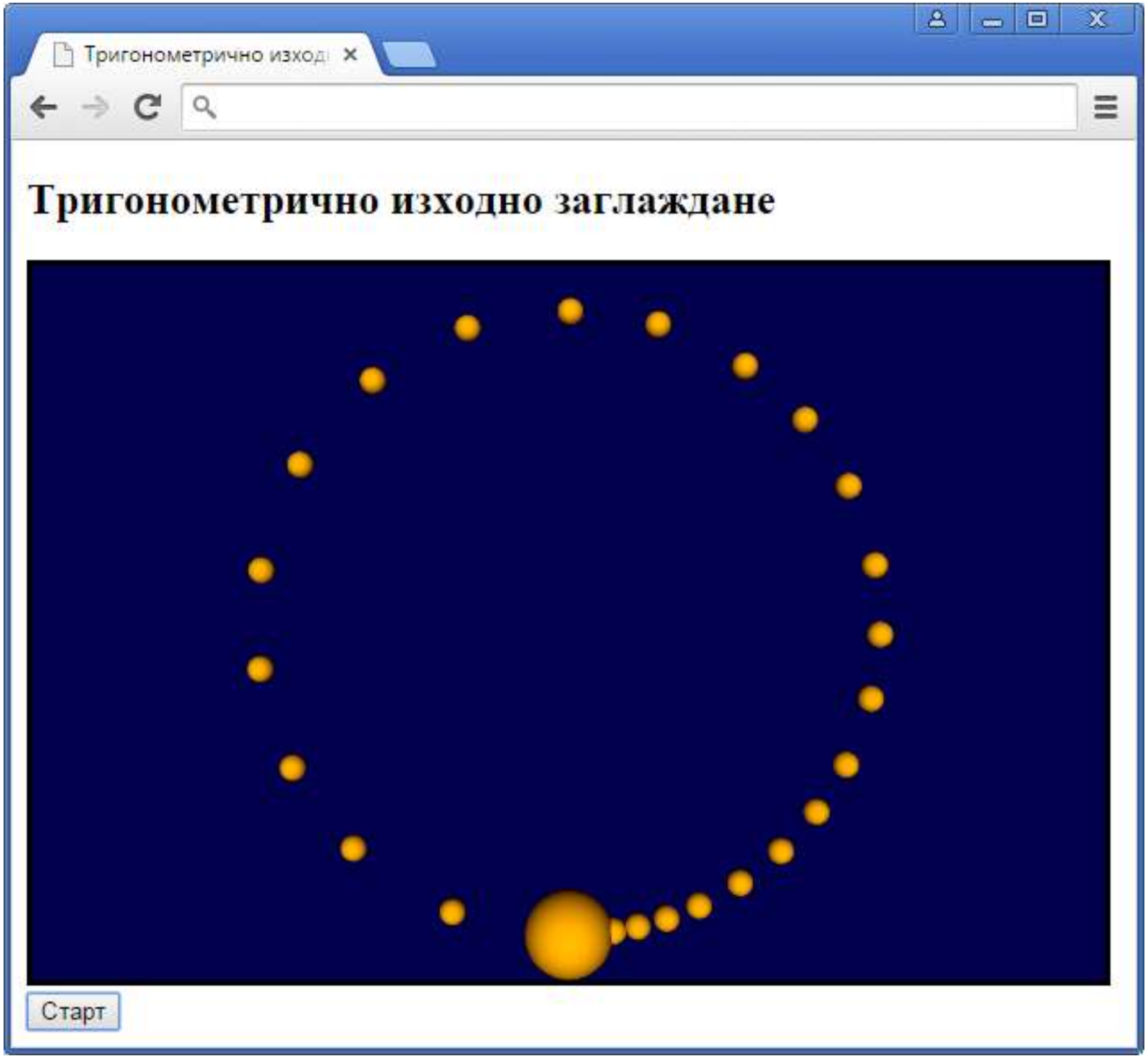

Тест Файлове

# Пример

- Изходно заглаждане с тригонометрична функция
- Заглаждаща функция от втора степен:

$$p(t) = (p_1 - p_0)\sin\left(\frac{\pi}{2}\frac{t - t_0}{t_1 - t_0}\right) + p_0$$





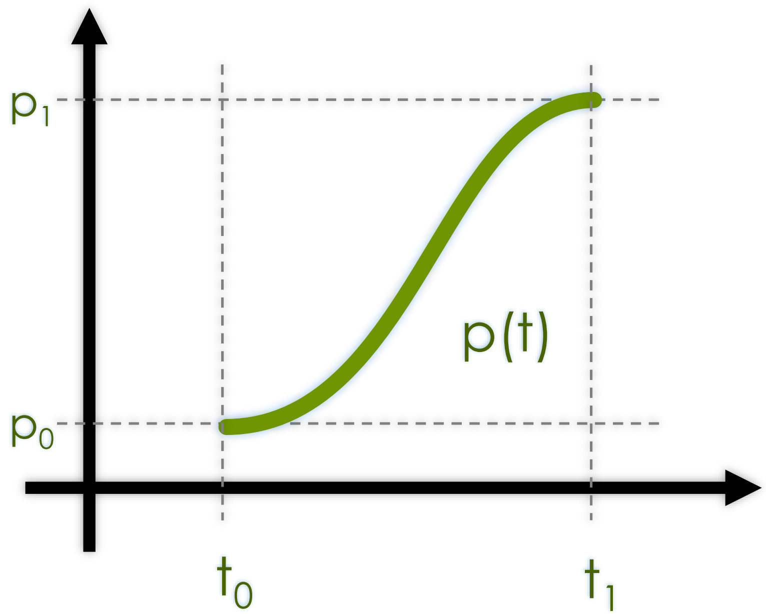

Тест

Файлове

# Пример

- Входно и изходно загладжане с рационална функция
- При k=1 е линейна, при k=2 е s-образна, при k=∞ е стъпало:

$$p(t) = \frac{(p_1 - p_0)}{1 + \left(\frac{t_1 - t_0}{t - t_0} - 1\right)^k} + p_0$$

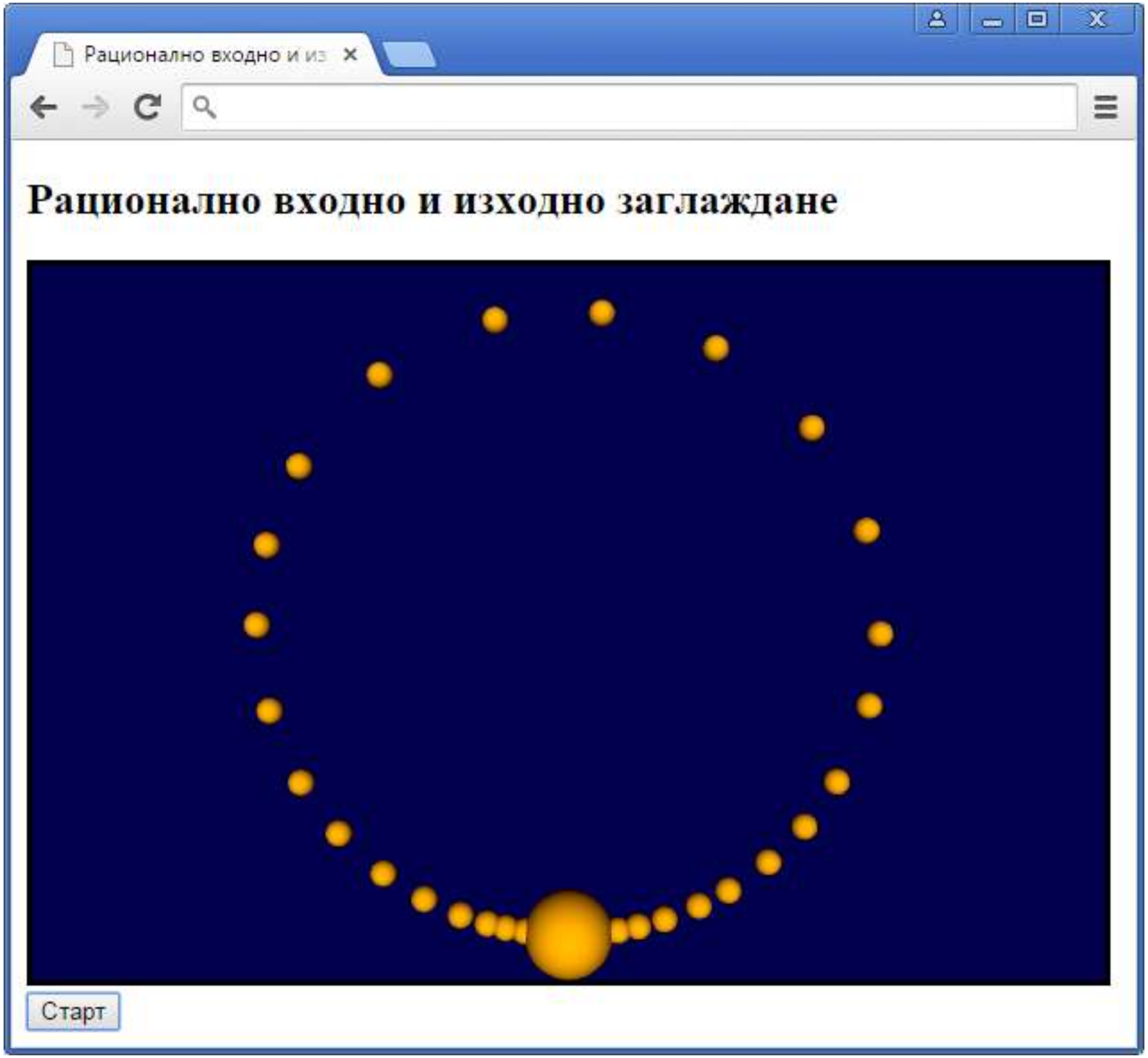



При $p_0 = t_0 = 0$ и $p_1 = t_1 = 1$ получаваме

$$p(t) = \frac{1}{1 + \left(\frac{1}{t} - 1\right)^k}$$

или

$$p(t) = \frac{t^k}{t^k + (1 - t)^k}$$




Тест

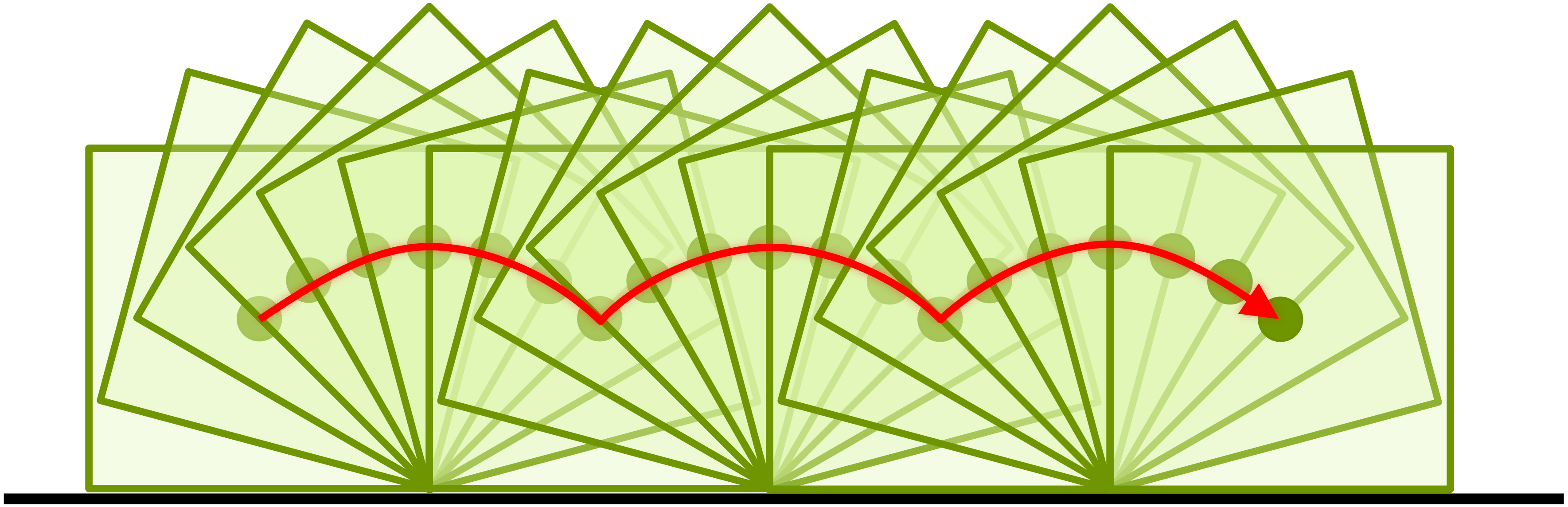
Файлове

# Търкаляне

# Търкаляне

## Моделиране на търкаляне

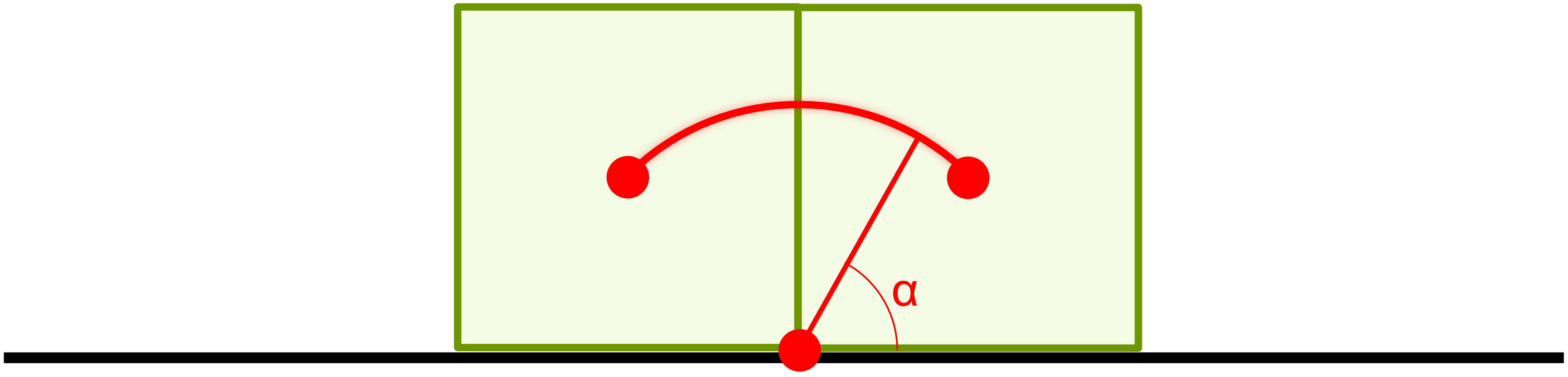
- Тривиално за сфери и цилиндри
- В общия случай се отчита формата на обекта
- Например, търкалянето на куб е около ръбовете му



# Търкалящо се кубче

## Реализация

- Движим центъра по дъги от по 90°
- Гарантираме си фиксирано положение на ръб
- След завъртане, фиксиран става следващия ръб



## Променливи

- Координата **cEdge** на ръб, около който въртим
- Ъгъл **cAngle** на завъртане (от $\pi/4$ до $3\pi/4$)
- Радиус **cRad** на дъгата на завъртане
- Изминало време **dTime** от предишния кадър

## Изчисляване на положение и ориентация

- Зависи от ръба, ъгъла и радиуса
- Използваме преобразуване на полярни координати
- Не променяме центъра на куба, транслираме „ръчно“, понеже въртенето трябва да е преди транслацията

```
pushMatrix();
   translate([0, cEdge-cRad*cos(cAngle), cRad*sin(cAngle)]);
   xRotate(cAngle*180/PI-45);
   c.draw();
popMatrix();
```

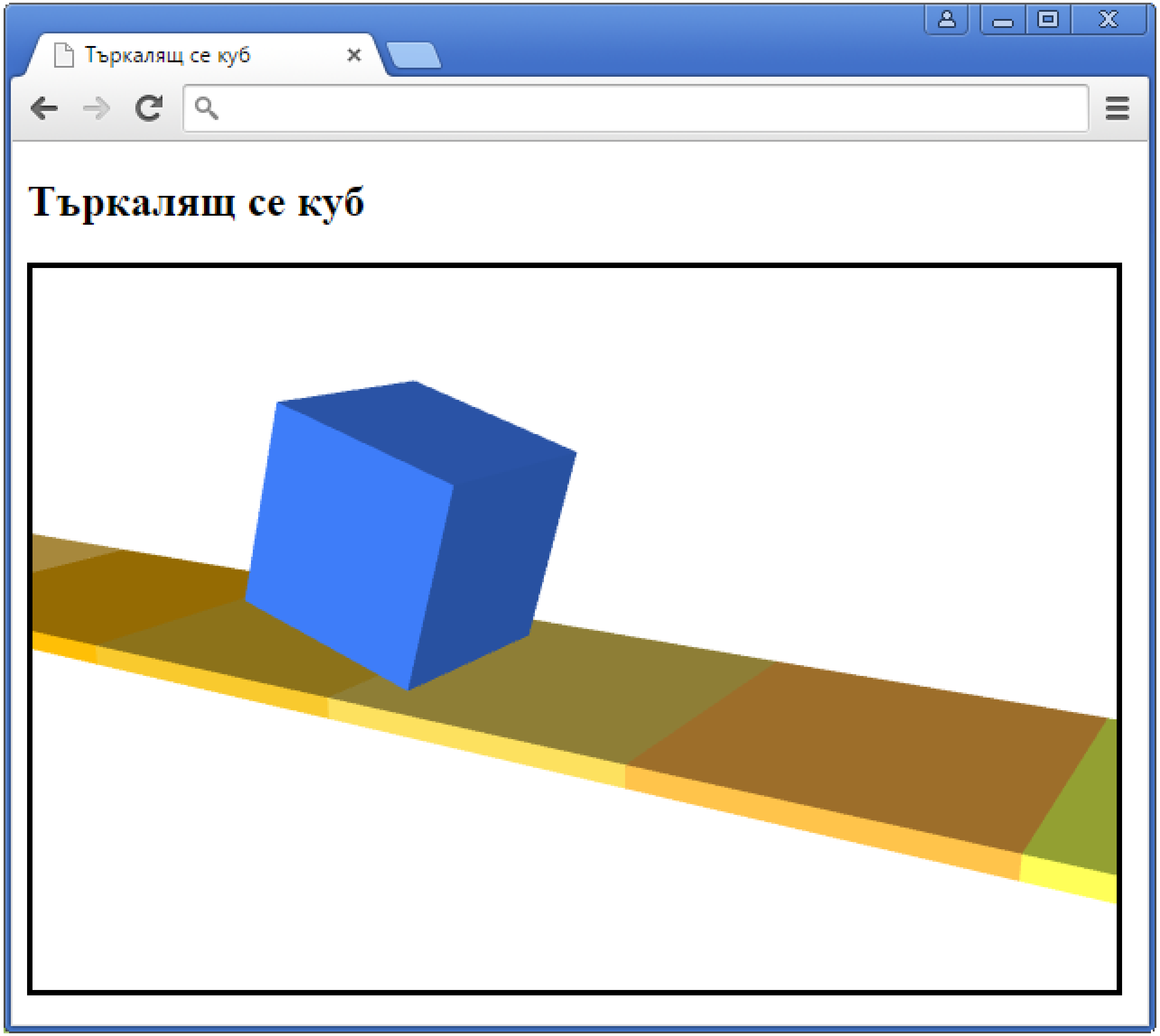
# Промяна на ъгъла и ръба

- Първоначалният ъгъл е $\pi/4$
- На всяка стъпка увеличаваме ъгъла с изминалото време

# Смяна на ръб – ъгълът прехвърля $3\pi/4$

- Превъртаме обратно ъгъла с $\pi/2$
- Изместваме точката на въртене да съвпада със следващия ръб на куба

```
cEdge = -70;
cAngle = PI/4;
:
cAngle += dTime;
if (cAngle>3*PI/4)
{
   cAngle -= PI/2;
   cEdge += c.size;
}
```




Тест

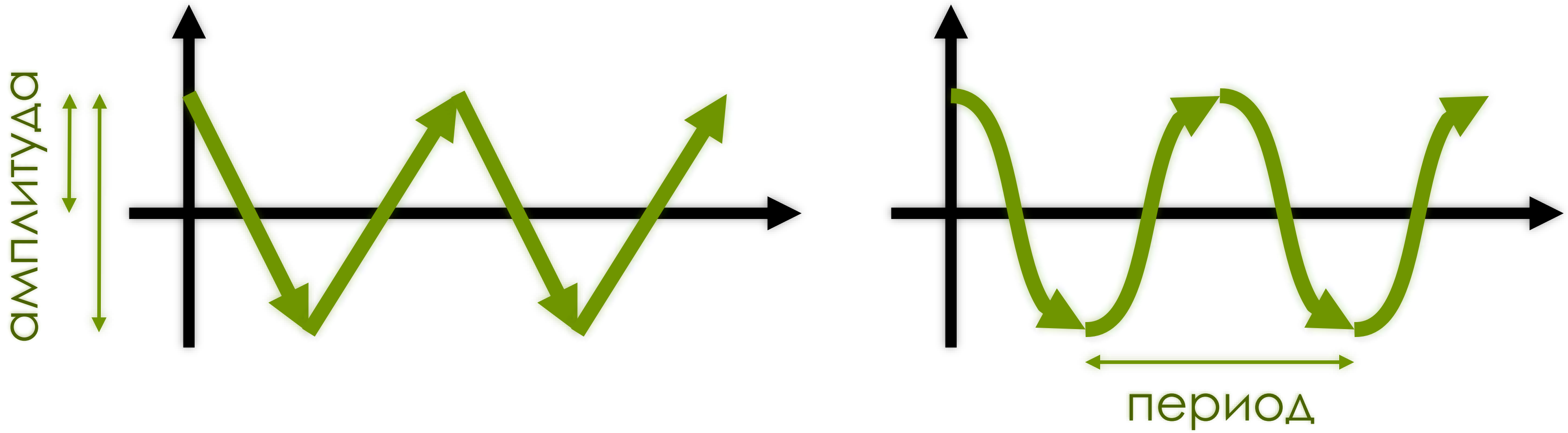
Файлове

# Вибрация

# Вибрация

## Вибрация – периодично трептене

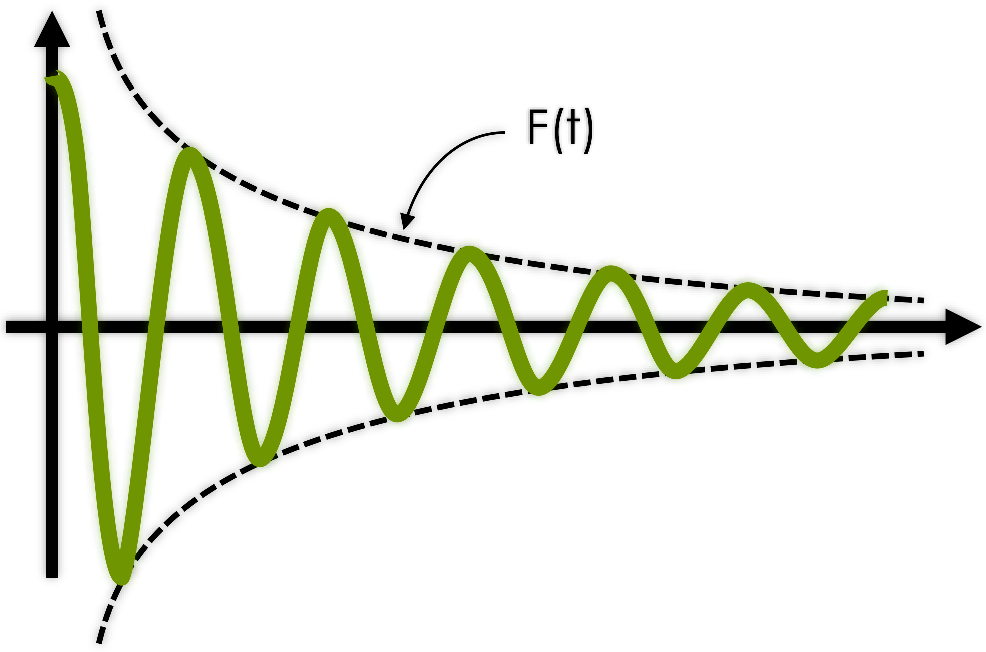
- Амплитуда и период (дължина, честота)
- Профил на вибрацията – по наш избор



# Затихване

- Симулира загуба на енергия
- Ние си избираме как, често е хиперболично или експоненциално

$$F(t) = \frac{a}{bt+c} \quad \text{или} \quad F(t) = ae^{bt+c}$$

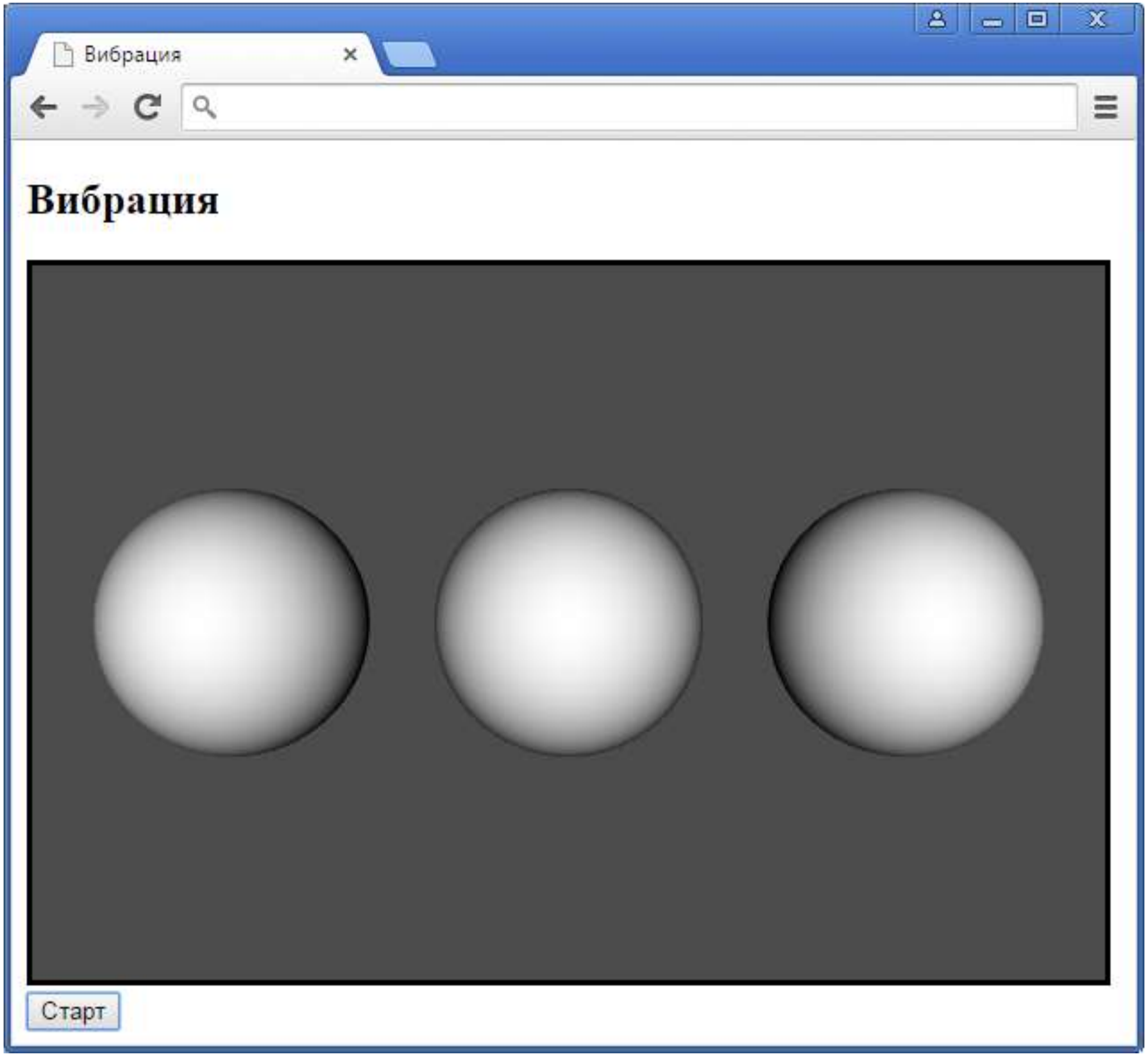


# Реализация

- Три сфери, вибриращи с различно затихване
- Профилни функции $F_1(t) = e^{-t/3}$, $F_2(t) = e^{-t-1}$ и $F_3(t) = e^{-5t-2}$
- Амплитудата се смята според изминалото време от последното активиране на вибрацията **vTime**
- При намаляване на височината се увеличава ширината, като се поддържа еднаква тяхна сума

```
var t = time-vTime;

s1.size[0] = 10+5*sin(15*t)*Math.exp(-t/3);
s1.size[1] = 20-s1.size[0];
s1.center[0] = -s1.size[0];
:
s2.size[0] = 10+10*sin(20*t)*Math.exp(-t-1);
:
s3.size[0] = 10+25*sin(30*t)*Math.exp(-t*5-2);
```




Тест

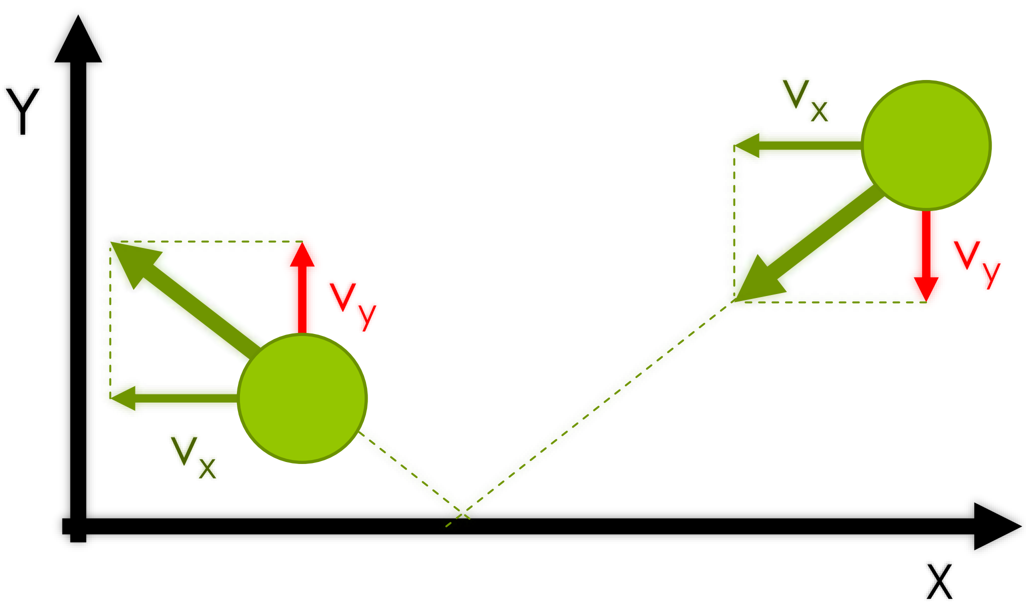
Файлове

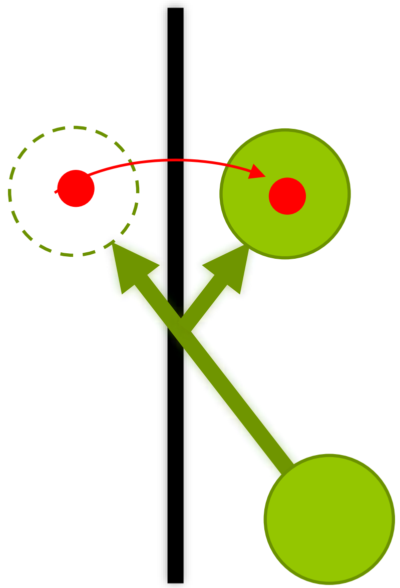
# Отблъскване и топане

# Отблъскване

## Физическа представа

- Движещ се предмет се удря в преграда
- Най-често преградата е успоредна на осите
- Обръща се знакът на някоя от компонентите на скоростта
- Компенсиране при неточен сблъсък

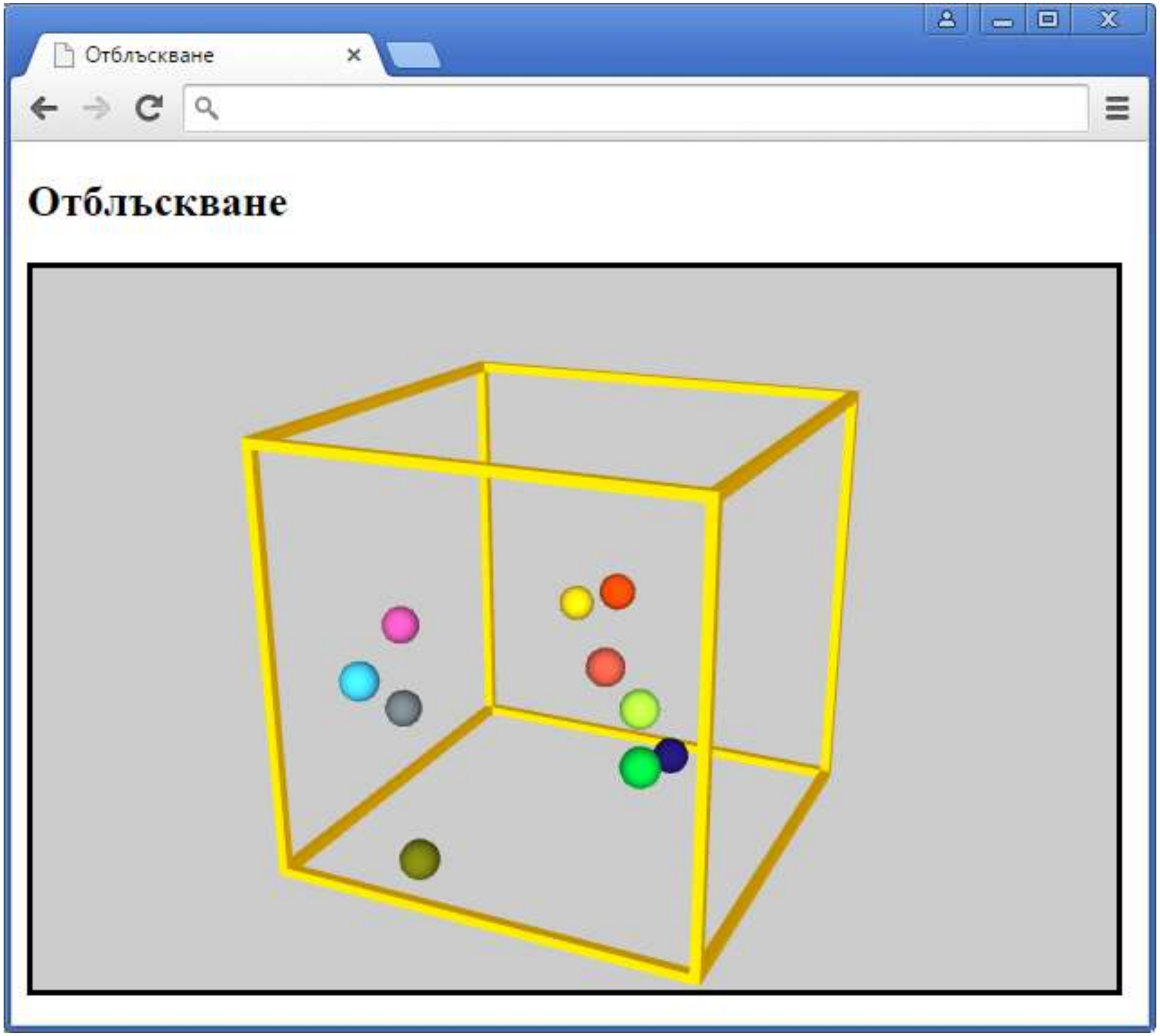


# Реализация

- Координатите **j** обработваме по един и същ начин
- При преминаване на горната или долната граница обръщаме знака на съответната компонента на скоростта
- Промяната в центъра е само компенсация, ако стъпката не стига точно до границата

```
for (var j=0; j<3; j++)
{
   o.center[j] += 30*o.v[j]*dTime;
   if (o.center[j]>20)
   {
      o.center[j] = 40-o.center[j];
      o.v[j] *= -1;
   }
   if (o.center[j]<-20)
   {
      o.center[j] = -(40+o.center[j]);
      o.v[j] *= -1;
   }
}
```




Тест

Файлове

# Топане

## Физическа представа

- Отблъскване на твърда повърхност
- Движение породено от гравитация – има ускорение

## Идеи за реализация

- За физическа точност – уравнения от балистиката, отчитат се маса, скорост и земно привличане
- При симулация – заменя се топането със **sin(x)** или **cos(x)** – физически грешно, но визуално приемливо

# Реализация

- Формула на симулирано топане

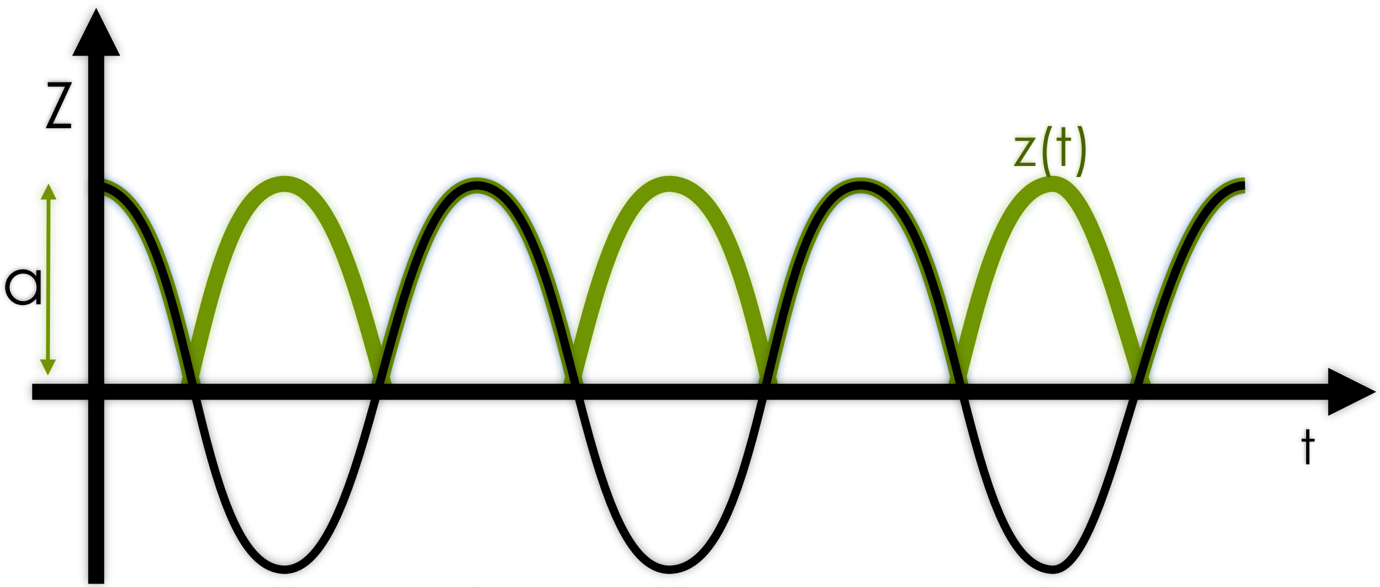
Отместване във времето

Абсолютна стойност

$$z(t) = a|\cos(bt + c)|$$

Амплитуда

Скорост на топане



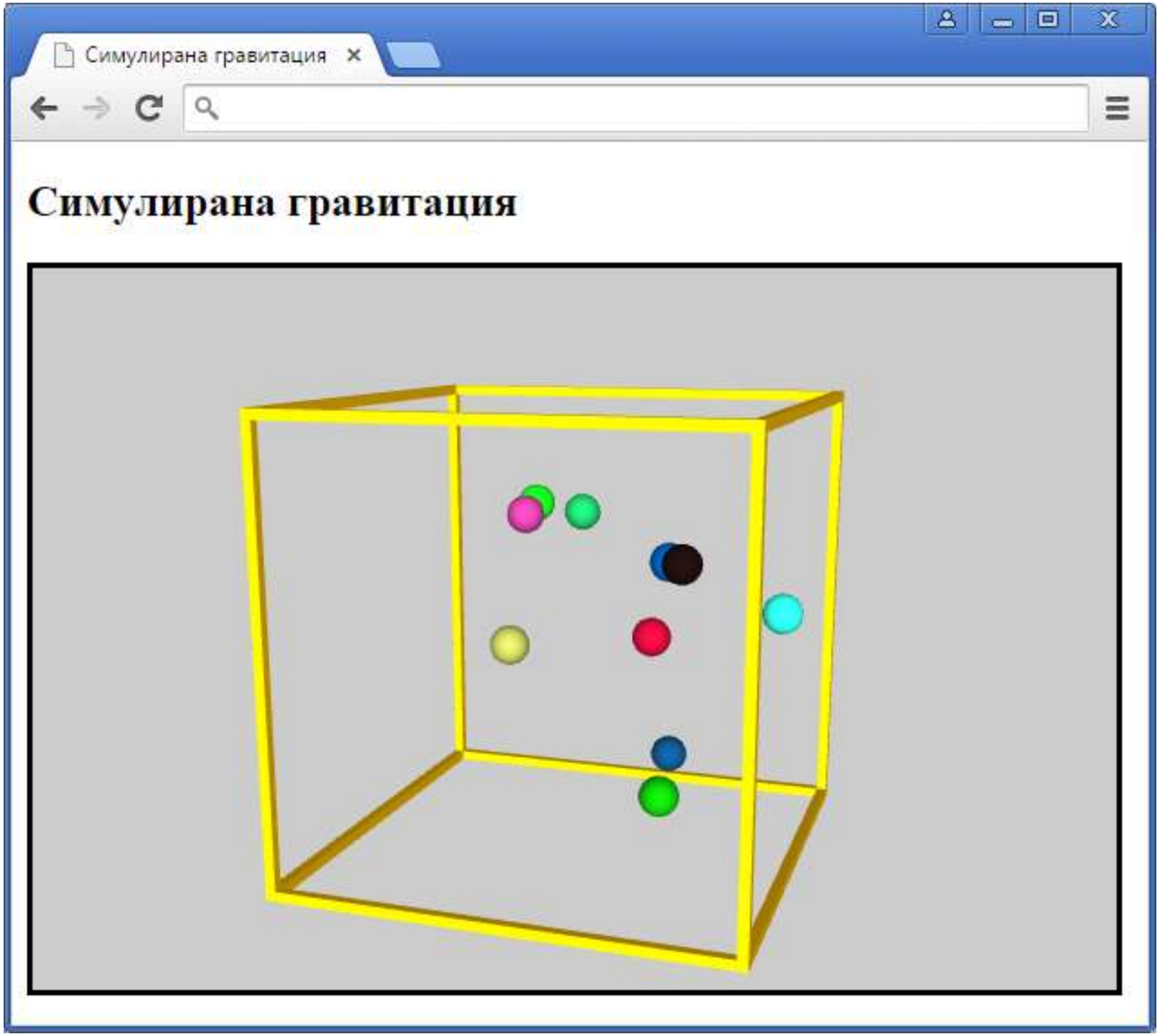
# За всеки топащ се обект определяме:

- Вектор на хоризонталната скорост **v**
- Скорост на самото топане **s**
- Отместване във времето **a**
- Амплитуда на топането **h**

# Движение

- Хоризонталното е подобно на отблъсване
- Вертикалното е по синусоида – симулира гравитация

```
o.v = [cos(a),sin(a)];    // вектор на скоростта
o.s = random(1.5,2.5);    // скорост на топане
o.a = random(0,2*PI);     // отместване във времето
o.h = random(20,40);      // амплитуда на топане
:
o.center[2] = -20+o.h*Math.abs(sin(o.s*time+o.a));
```




Тест

Файлове

# Балистика

# Балистика

## Основи

- Движение по парабола под влияние само на гравитацията
- Необходими данни са началното положение и скоростта

## Изчисляване на параболата

- Чрез уравнение

$$p(t) = \frac{1}{2}gt^2 + v_0 t + p_0$$

- Чрез постъпково изчисление

$$g = const$$
$$v_i = v_{i-1} + g\Delta t$$
$$p_i = p_{i-1} + v_i \Delta t$$

# Пример

## Гейзер от тухли

- Тухли извират от една точка
- Случайна посока, но преобладаващо нагоре
- Движат се по парабола
- След падане на тухла, тя извира отново

## Още изисквания

- Цветовете да са кафеникави
- При летенето тухлите да се въртят
- Да има клас **Brick**, в който е кодирано движението

# Реализация

- Обектът **Brick** ще е подобен на **Cuboid**
- Конструкторът е без параметри
- Завъртането е в **rot**, а скоростта на завъртане е във **vRot**
- Скоростта на движение на тухлата е във **v**, като по Z е изкуствено засилено двукратно
- За графичен примитив се ползва каноничния куб

```
constructor()
{
   this.center = [0,0,-10];
   this.color = [random(0.4,0.6),random(0.1,0.3),random(0,0.25)];
   this.rot = [0,0,0];
   this.vRot = [random(50,100),random(50,100),random(50,100)];

   var a = random(0,2*PI);
   var b = radians(random(50,90));
   this.v = [cos(a)*cos(b),sin(a)*cos(b),2*sin(b)];

   if (!canonicalCube) canonicalCube = new CanonicalCube();
}
```

# Рисуване на тухлата

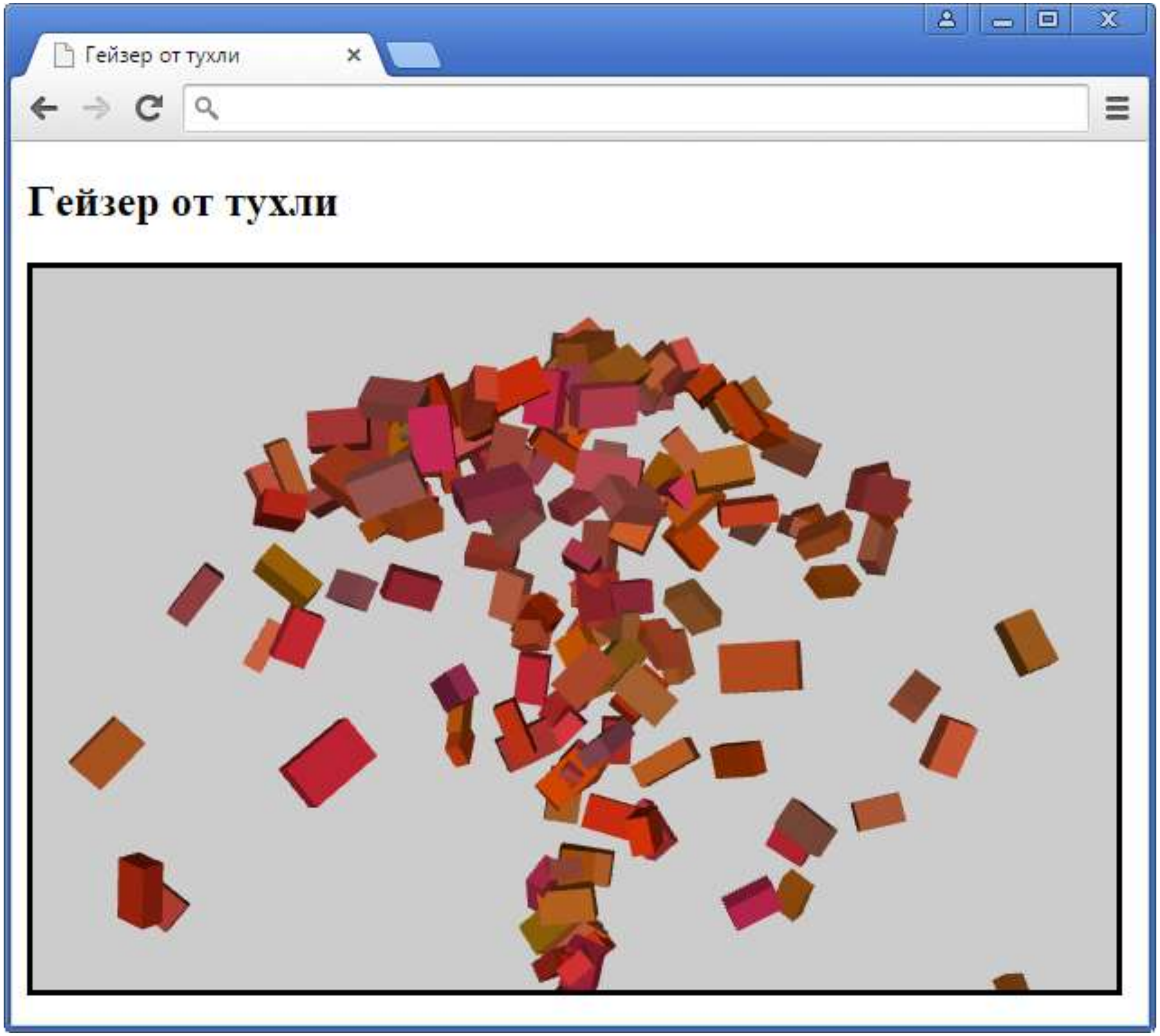
- Аналогично на рисуването на **Cuboid**
- Размерът на тухлата е фиксиран на 1.6x1x0.6
- Между транслацията и мащабирането се извършва ротация по три оси

```
draw()
{
   pushMatrix();
      gl.vertexAttrib3fv(aColor,this.color);
      translate(this.center);
      xRotate(this.rot[0]);
      yRotate(this.rot[1]);
      zRotate(this.rot[2]);
      scale([1.6,1.0,0.6]);
      useMatrix();
      canonicalCube.draw();
   popMatrix();
}
```

# Движение на тухлата

- Метод **move** с параметър изминало време **dT**
- Премества центъра според скоростта **v**
- Актуализира вертикалната скорост с ускорение -0.8
- Променя завъртането **rot** със скоростта на завъртане **vRot**
- Ако вертикално тухлата е под -20 и пада надолу, тогава се премества в началото и се дава нова скорост

```
move(dT)
{
   for (var j=0; j<3; j++) this.center[j] += 8*this.v[j]*dT;
   this.v[2] -= 0.8*dT;
   for (var j=0; j<3; j++) this.rot[j] += this.vRot[j]*dT;
   if (this.center[2]<-20 && this.v[2]<0)
   {
      this.center = [0,0,-10];
      var a = random(0,2*PI);
      var b = radians(random(60,90));
      this.v = [cos(a)*cos(b),sin(a)*cos(b),2*sin(b)];
   }
}
```




Тест

Файлове

# Скачане

## Моделиране на скачане

- По същество това е балистична крива
- Точен модел – с парабола
- Приближени модели – с елипса, с тригонометрична функция или с постъпково изчисляване

## При (пре-)скачане

- Обикновено се извършва в по-ранен момент спрямо евентуалния удар

# Пример

## Прескачане на препятствия

- Четири прегради се движат в кръг
- Топчета се движат срещу тях и ги прескачат

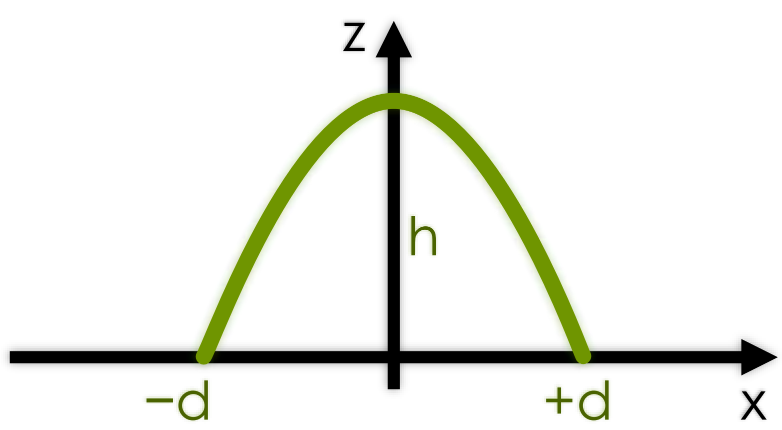
## Реализация

- Преградите са 2 перпендикулярни сплескани сфероида
- Ако на ъгъл **α** има преграда, то другите са на **α+90°**, **α+180°** и **α+270°**
- Всяко топче знае положението си като ъгъл и то се сверява с 4-те ъгъла с прегради

# Самото прескачане

- Симулация с **cos(x)** в интервала $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
- Параметри на скока: ширина **d** и височина **h**
- Ъглово разстояние **x** до препятствието

$$h(x) = \begin{cases} h\cos\dfrac{\pi x}{2d} : x \in [-d, d] \\ 0 : x \notin [-d, d] \end{cases}$$

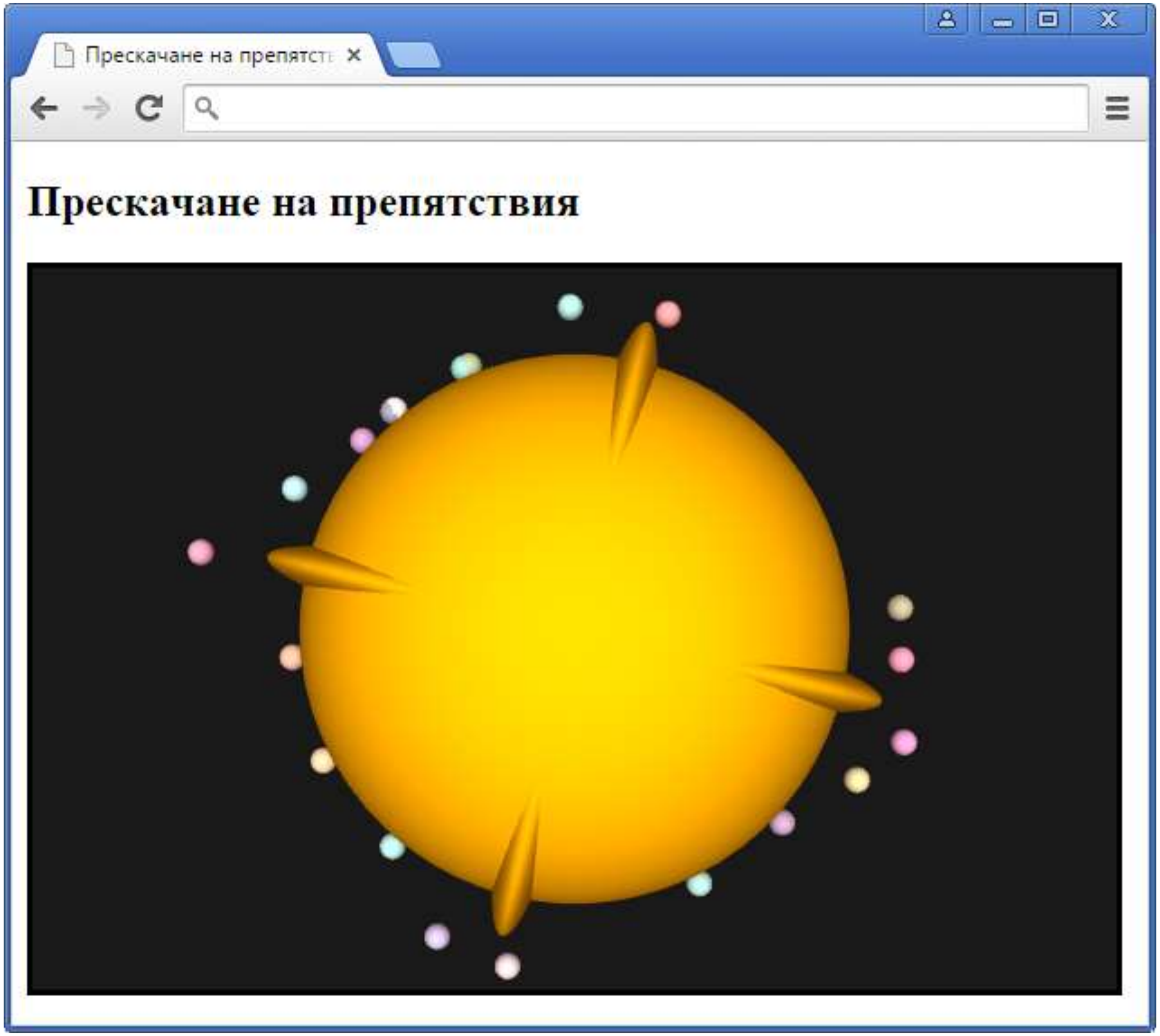


# Функция за скачане

- При ъгли на топче и преграда намира височината на скока
- Ъглите се нормализират до $[-2\pi, 2\pi]$
- Ако разстоянието между тях не е **d** или по-малко, стеснява диапазона, като премества долната граница с $2\pi$ нагоре

```
function jump(a,b,ball)
{
   var d = ball.d;
   var h = ball.h;

   a = a % (2*PI);
   b = b % (2*PI);
   for (var i=0; i<3; i++)
   {
      if (Math.abs(a-b)<d) return h*cos((a-b)/d*PI/2);
      b+=2*PI;
   }
   return 0;
}
```




Тест

Файлове

# Вълнообразно движение

# Вълнообразно движение

## Модел на водна повърхност

- Има вълни (като в басейн)
- Физически модел – прекалено сложен

## Опростен модел

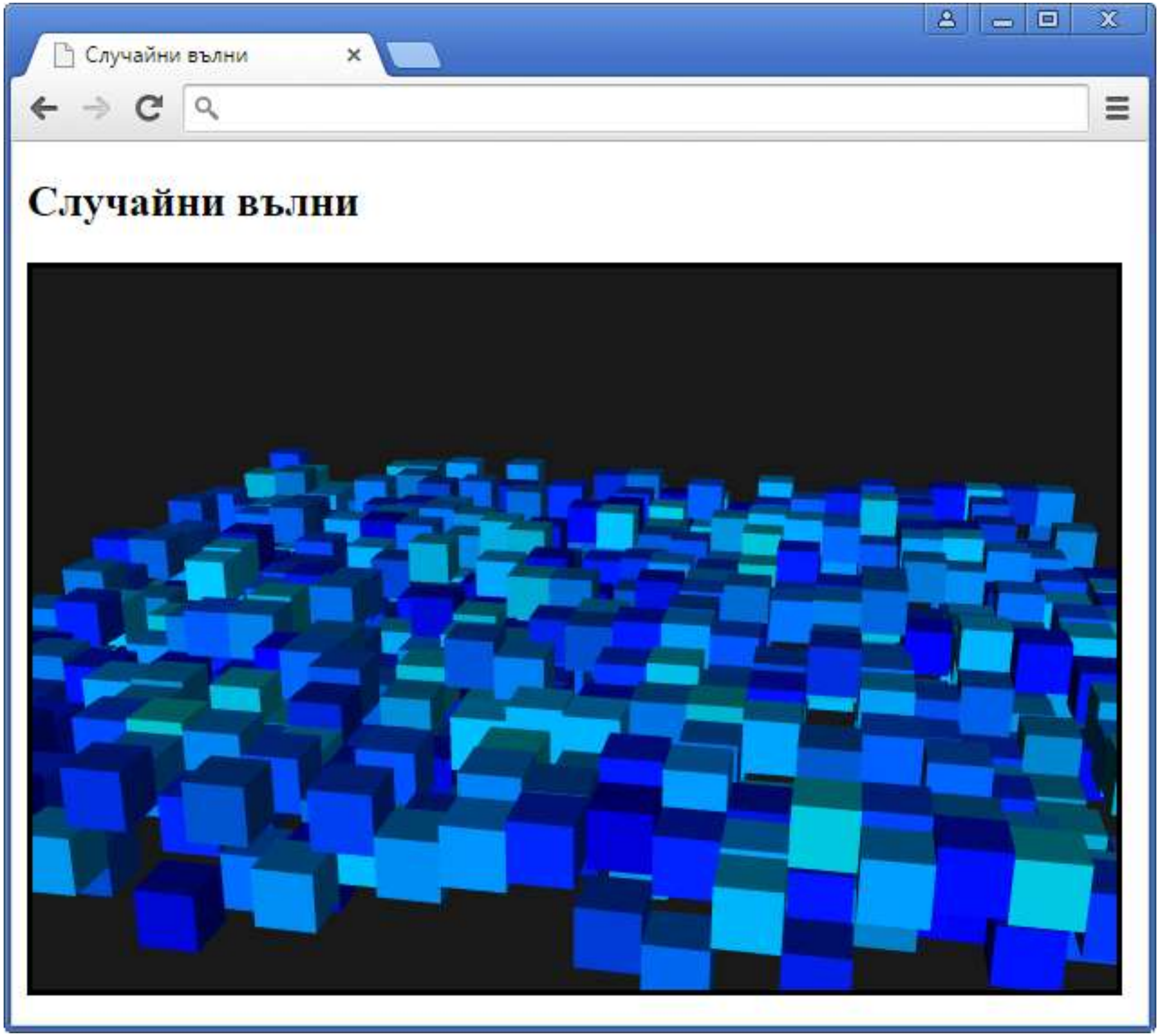
- Мрежа от обекти (точки, кубове, …)
- Обектите се движат нагоре-надолу
- Движат се случайно, но не изцяло случайно

# Пример

## Модел на вълни

- Повърхността е от кубове
- Всеки се движи вълнообразно нагоре-надолу
- За да не са едновременно, всеки куб има случайно отместване във времето

```
for (var x=0; x<n; x++)
{
   for (var y=0; y<n; y++)
   {
      water[x][y] = new Cube([x-(n-1)/2,y-(n-1)/2,0],1);
      water[x][y].a = random(0,2*PI);
   }
}
:
for (var x=0; x<n; x++)
   for (var y=0; y<n; y++)
      water[x][y].center[2] = sin(2*time+water[x][y].a);
```




Тест

Файлове

# Недостатък

- Всяко кубче се движи само за себе си

# Решение

- Съседни кубчета да се движат съседно
- Измисляме си функция, която се гърчи в 2D
- Ако индексите на куб са **x** и **y**, а времето е **t**, отместването на движението във времето може да се зададе с:

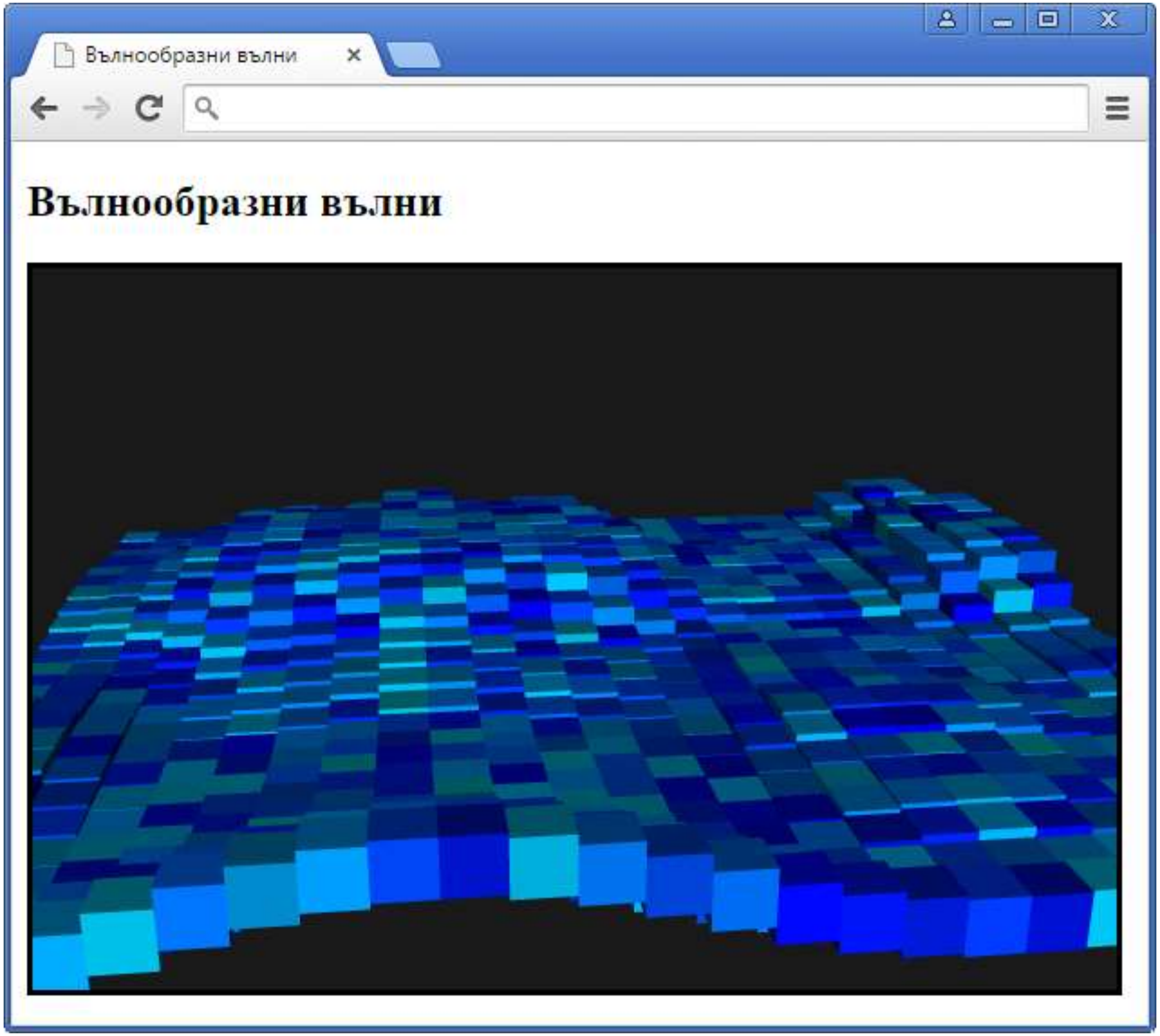
$$\mathrm{a}(x,y) = \sin\left(\frac{x}{3.0} + t + \frac{y}{2n}\cos(2t - x)\right) + \cos\left(\frac{y}{3.5} - t + \frac{x}{2n}\sin(3t + y)\right)$$

Забележка: функцията е измислена с опити и грешки, няма нужда да се ползва точно този шаблон

# Нова реализация

- Кубовете не помнят своето отместване
- Отместването им се изчислява в реално време чрез функцията **a(x,y)**
- В крайна сметка движението на всеки куб си остава синусоидално движение

```
function a(x,y)
{
   var k1 = sin(x/3.0+time+y*cos(2*time-x)/n/2);
   var k2 = cos(y/3.5-time+x*sin(3*time+y)/n/2);
   return k1+k2;
}
:
for (var x=0; x<n; x++)
   for (var y=0; y<n; y++)
      water[x][y].center[2] = sin(2*time+a(x,y));
```




Тест

Файлове

# Въпроси и коментари

# Край