Symulacja Stałokrokowa

„Bo przez tą zmienną dt to się gry psujom!”

Piotr Martynowicz
Jacek Złydach

Ewolucja aktualizacji stanu gry

Gry niezależne od czasu rzeczywistego.

```
while(true) {
   GrabInput();
   UpdateGame();
   RenderGame();
}

object.velocity += object.acceleration;
object.position += object.velocity;
```

Im szybszy komputer, tym szybciej biegnie czas.

Ewolucja aktualizacji stanu gry

Gry niezależne od czasu rzeczywistego.

Ewolucja aktualizacji stanu gry

Gry niezależne od czasu rzeczywistego.

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   GrabInput();
   UpdateGame(dt);
   RenderGame();
}

object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   GrabInput();
   UpdateGame(dt);
   RenderGame();
}

object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   GrabInput();
   UpdateGame(dt);
   RenderGame();
}

object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   GrabInput();
   UpdateGame(dt);
   RenderGame();
}

object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   GrabInput();
   UpdateGame(dt);
   RenderGame();
}

object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

Ewolucja aktualizacji stanu gry

Gry zależne od czasu rzeczywistego, wersja naiwna.

Nieprzewidywalna fizyka.

Psujące się sprężyny.

Obiekty wypadają przez ściany.

Aktualizacja Stałokrokowa

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Aktualizacja Stałokrokowa

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Aktualizacja Stałokrokowa

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Aktualizacja Stałokrokowa

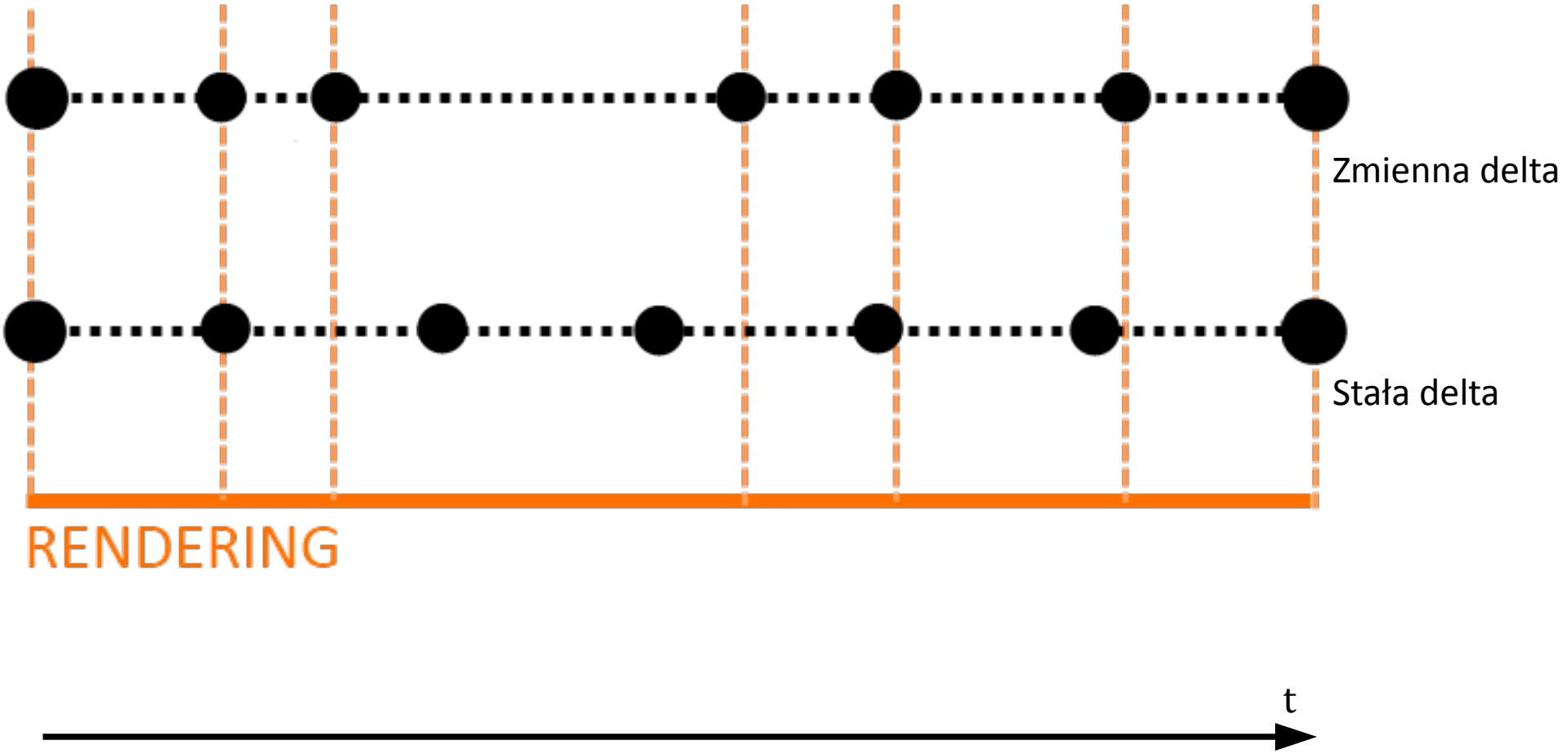
```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Aktualizacja Stałokrokowa

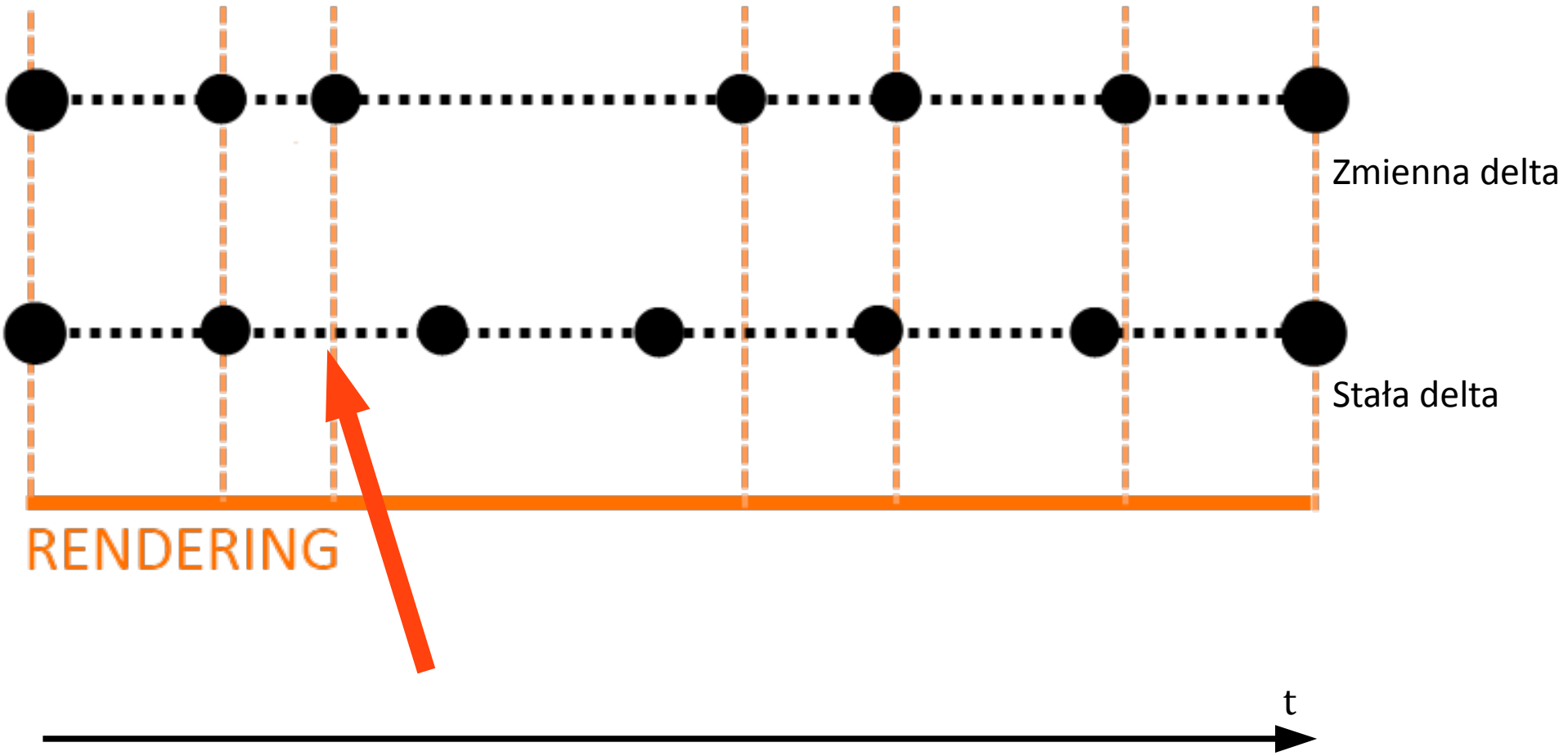
```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Ruch

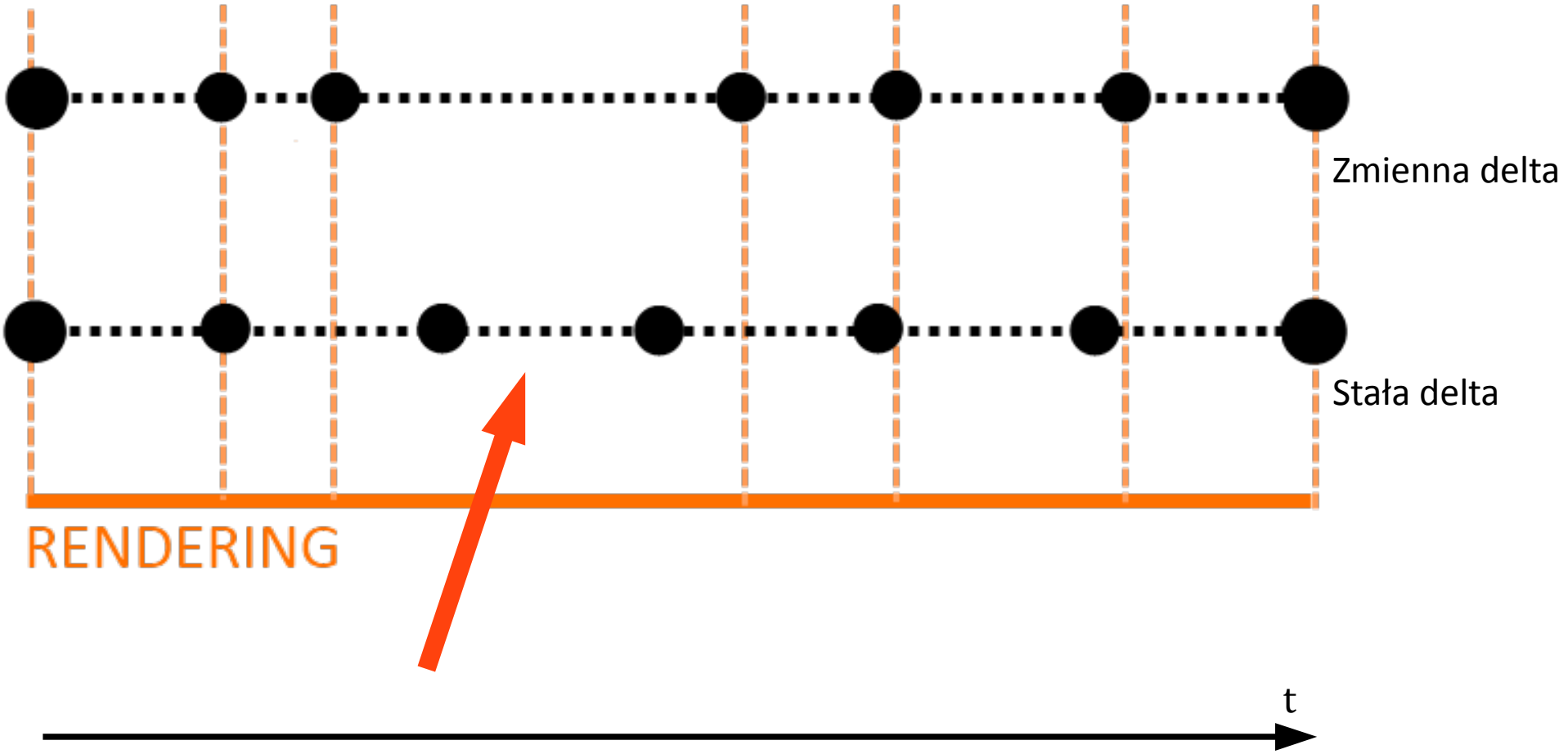
Stała vs zmienna delta - ruch



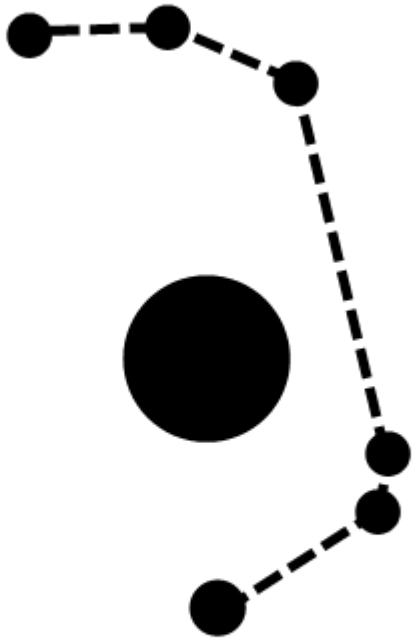
Stała vs zmienna delta - ruch



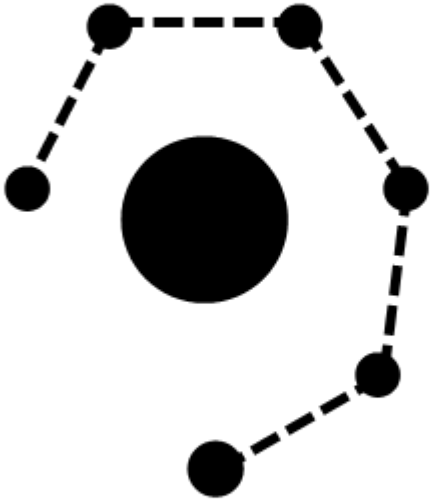
Stała vs zmienna delta - ruch



Stała vs zmienna delta - ruch



Zmienna delta

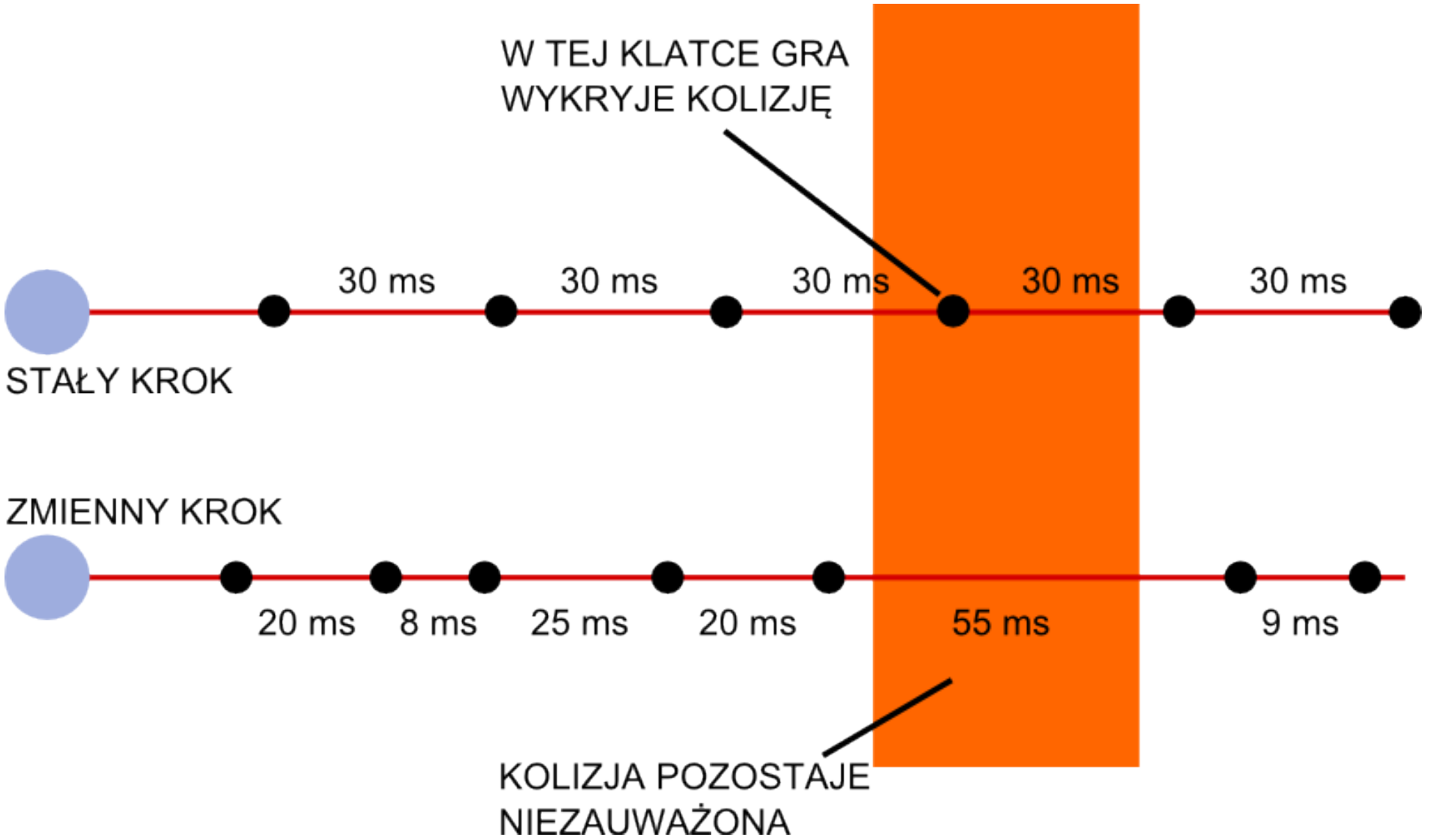


Stała delta


SCORE
0
HIGHSCORE
35730

Kolizje

Stała vs zmienna delta - kolizje



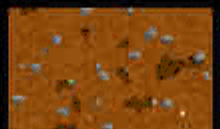
Kills: 1

Flags captured: 0

Kills: 4

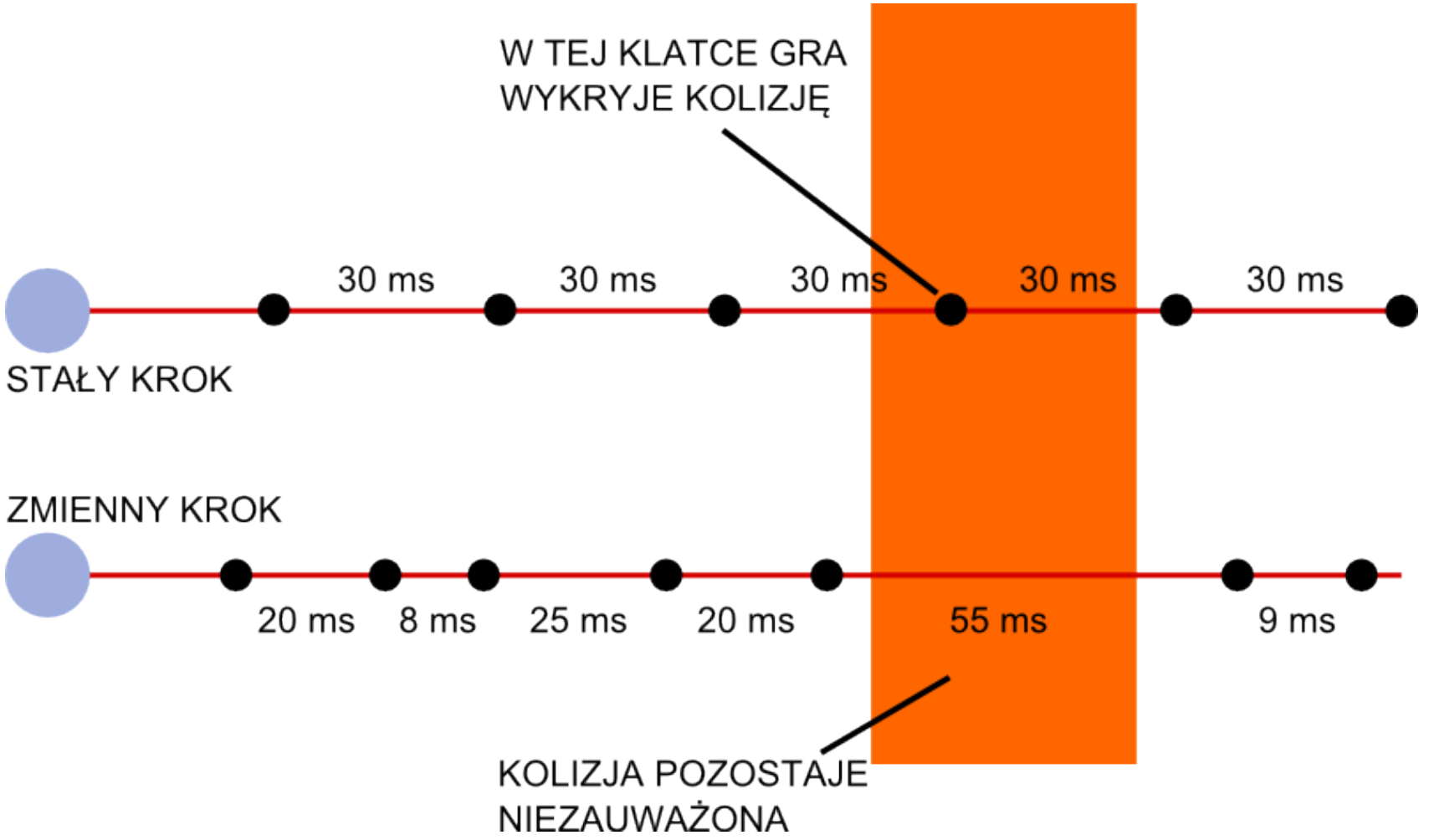
Flags captured: 0

Kills: 0
Lives: 15



Kills: 0
Lives: 15

Stała vs zmienna delta - kolizje




```
minWallSize = maxObjectSpeed * TIMESTEP;
```

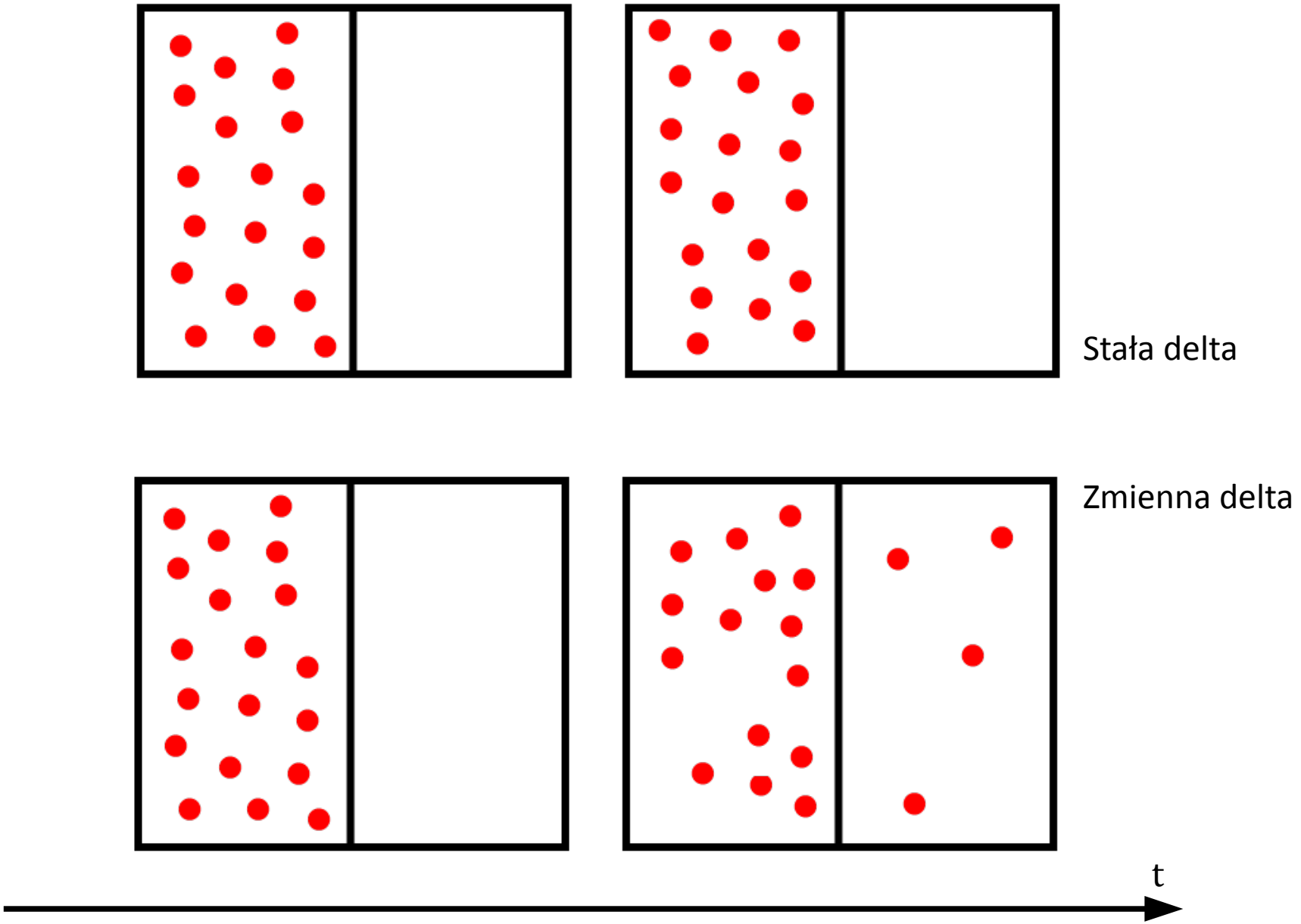
Stała vs zmienna delta - kolizje

```
minWallSize = maxObjectSpeed * TIMESTEP;
```

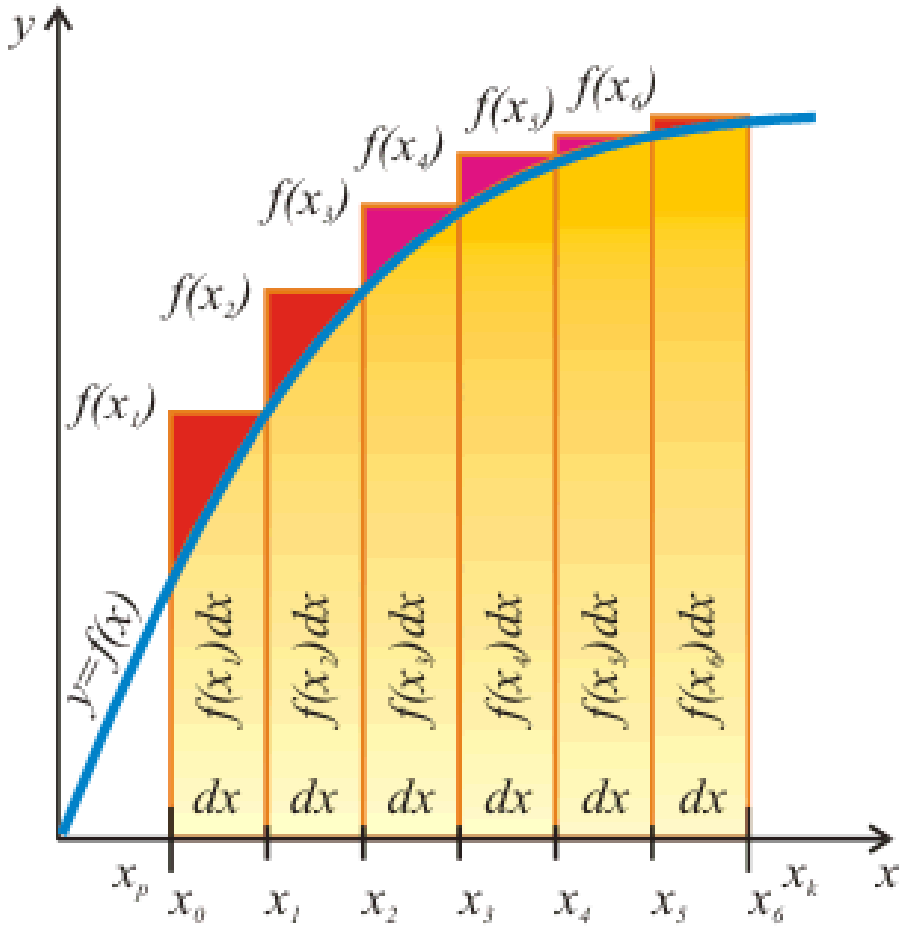
Dla pocisków i bardzo szybkich obiektów używajmy **continous collision detection**.

Fizyka

Stała vs zmienna delta - kolizje



Całkowanie Eulera

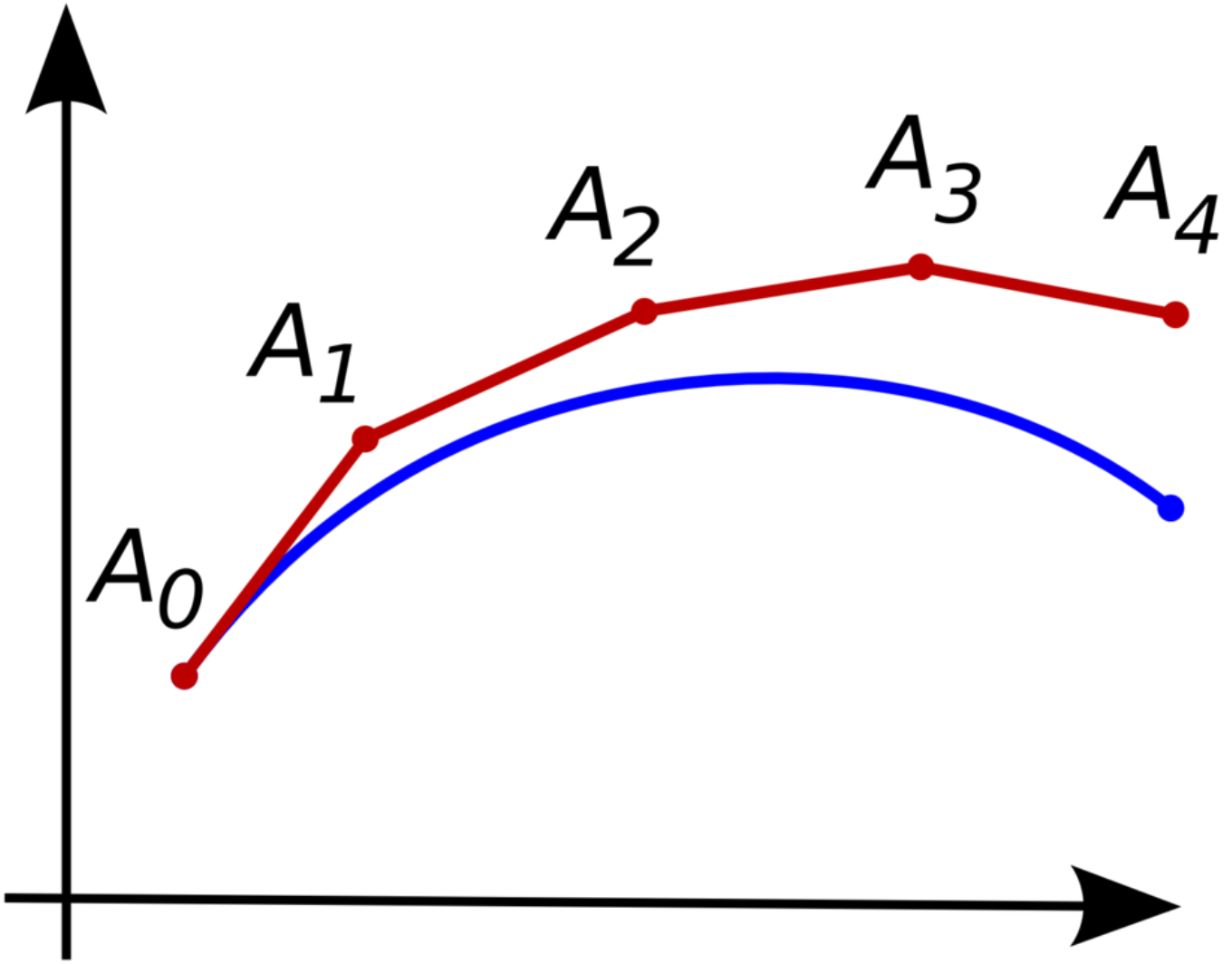


$$x(t) = \frac{a(t) \cdot t^2}{2} + v(t) \cdot t + C$$

$$x(t) = \frac{x''(t) \cdot t^2}{2} + x'(t) \cdot t + C$$

```
object.velocity += object.acceleration * dt;
object.position += object.velocity * dt;
```

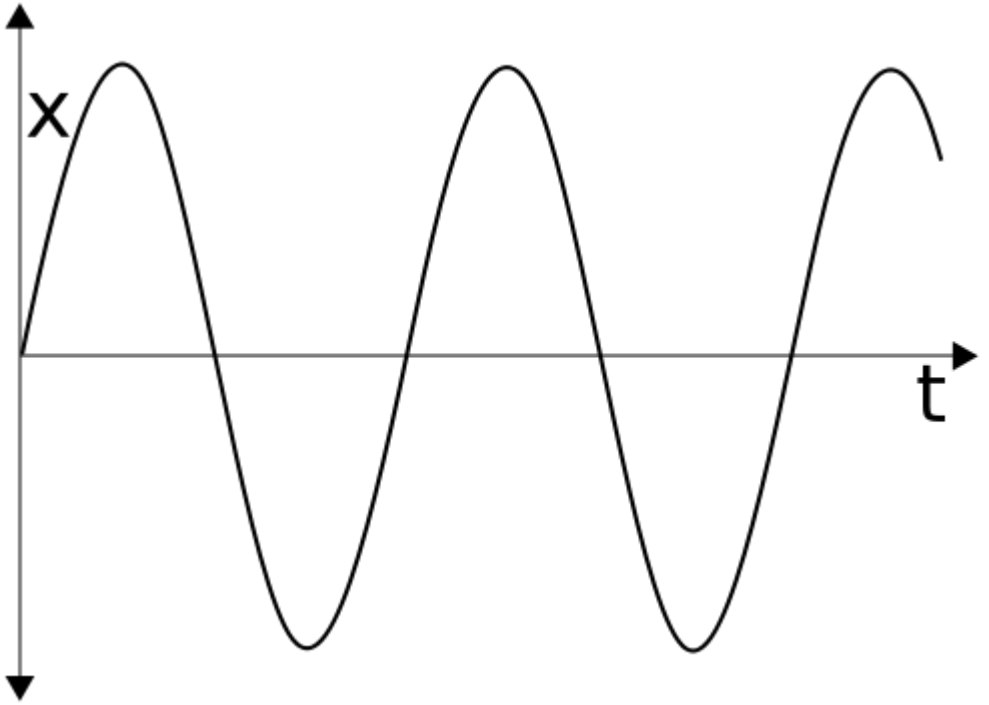
Całkowanie Eulera



Tym dokładniejsze, im mniejsza delta.

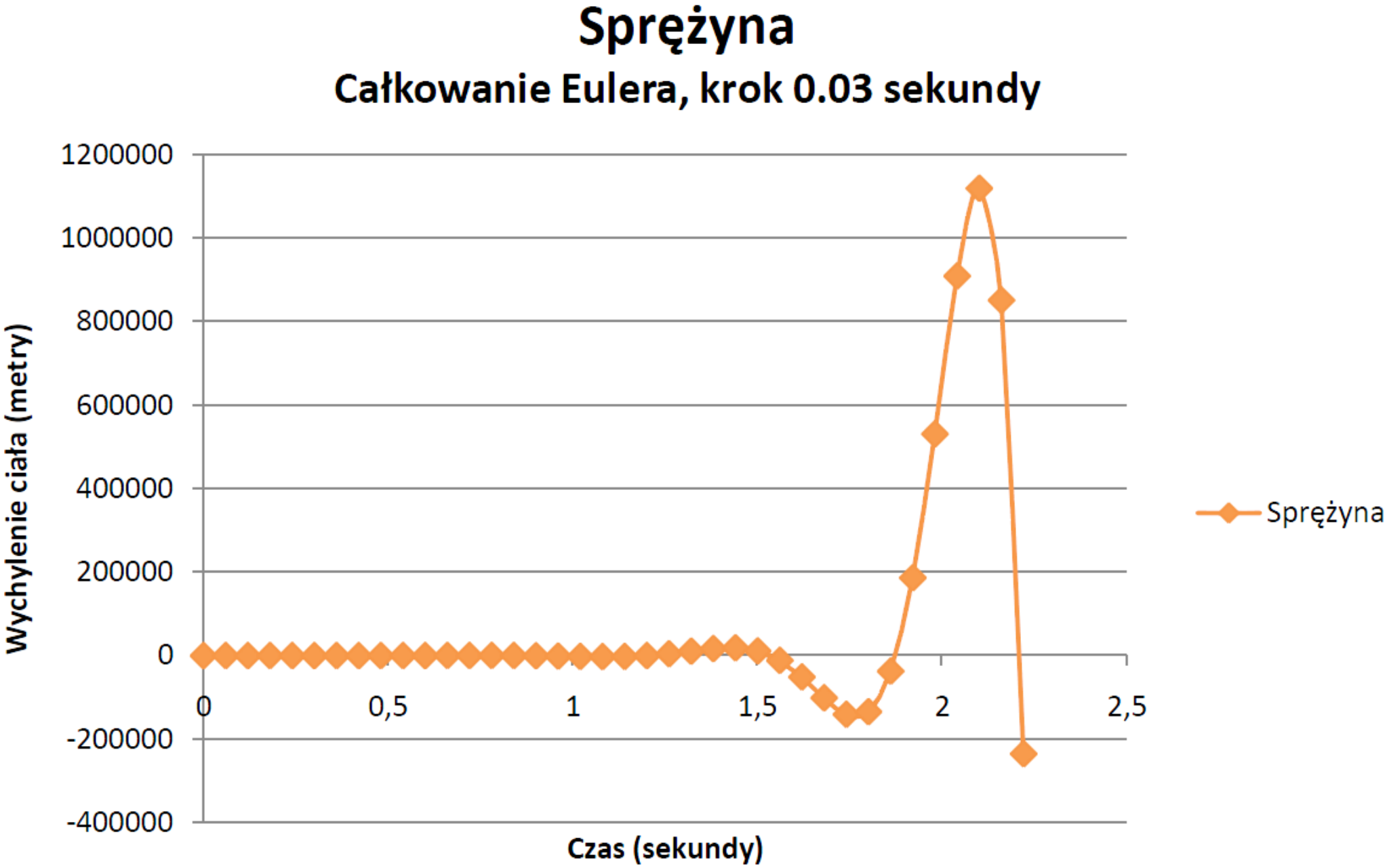
Niestabilne numerycznie.

Oscylator harmoniczny



$$F = -kx$$

Eksplodujące sprężyny



Eksplodujące sprężyny

$$F_t = -kx_t$$

$$x_{t+1} = x_t + F(t) \cdot \frac{t^2}{2} + C$$

Alternatywy do całkowania Eulera

Odwrotna metoda Eulera

Verlet

Velocity Verlet

RK4

Inne zalety

Replaye

Zapisujemy tylko wejścia i seedy generatorów liczb losowych

Sieć

Wystarczy zsynchronizować jedynie stan świata i zegar.

Debugging

Fizyka zawsze zachowuje się tak samo.

Ostatnie poprawki

Ostatnie poprawki

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();

float accumulator = 0.0f;
const float TIME_STEP = 0.03;

while(true)
{
   dt = GetCurrentTime() - lastUpdateTime;
   lastUpdate += dt;
   accumulator += dt;
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Ostatnie poprawki

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();
float currentTime = lastUpdateTime;
float accumulator = 0.0f;
const float TIME_STEP = 0.03;
float MAX_ACCUM_TIME = 1.0;
while(true)
{
   currentTime = GetCurrentTime();
   dt = std::max(0, currentTime - lastUpdateTime);
   lastUpdate = currentTime;
   accumulator = clamp(accumulator + dt, 0, MAX_ACCUM_TIME);
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Ostatnie poprawki

```
float dt = 0.0f;
float lastUpdateTime = GetCurrentTime();
float currentTime = lastUpdateTime;
float accumulator = 0.0f;
const float TIME_STEP = 0.03;
float MAX_ACCUM_TIME = 1.0;
while(true)
{
   currentTime = GetCurrentTime();
   dt = std::max(0, currentTime - lastUpdateTime);
   lastUpdate = currentTime;
   accumulator = clamp(accumulator + dt, 0, MAX_ACCUM_TIME);
   GrabInput();
   while(accumulator > TIME_STEP)
   {
      UpdateGame(TIME_STEP);
      accumulator -= TIME_STEP;
   }
   RenderGame();
}
```

Symulacja Stałokrokowa

FOR GREAT JUSTICE

Piotr Martynowicz
Jacek Złydach