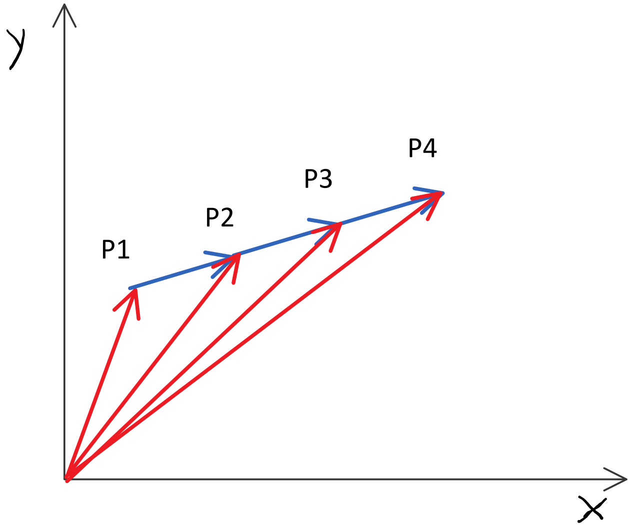
1a Ortsvektor und Geschwindigkeitsvektor fürs Geradeausfahren

Ortsvektor (wo ist das Auto)

Geschwindigkeitsvektor (wie schnell und in welche Richtung bewegt sich das Auto)



$$\vec{P}_{n+1} = \vec{P}_n + \vec{V} \cdot dt$$

Die Position zum Simulationszeitpunkt (n+1) ist
die Position zum Simulationszeitpunkt (n) + der Geschwindigkeitsvektor

ToTo:
Diese Skizze für den Fall, daß der Roboter eine Kurve fährt.
D.h. der Geschwindigkeitsvektor rotiert mit der Winkelgeschwindigkeit **w**

Kinematik
keine Maasen

Dynamik
m, v, s, Gravitation, Reibung

1b Ortsvektor und Geschwindigkeitsvektor Kurvenfahrt

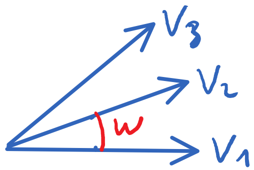
Verglichen mit dem vorhergehenden Fall dreht sich nun der Geschwindigkeitsvektor mit der Winkelgeschwindigkeit **w**

V

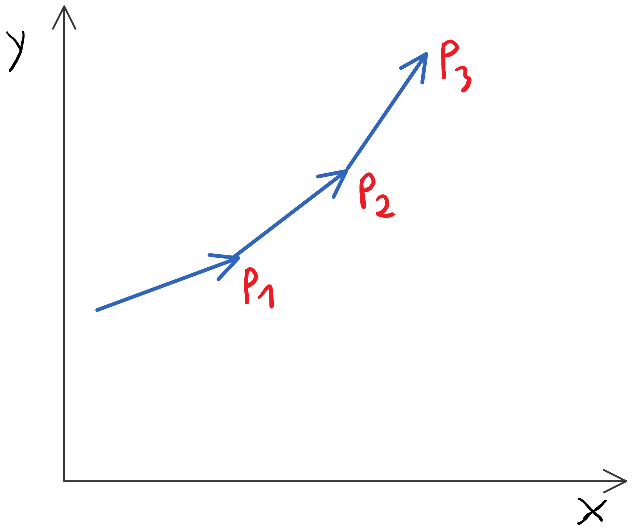
Ortsvektor (wo ist das Auto)

Geschwindigkeitsvektor (wie schnell und in welche Richtung bewegt sich das Auto)

$$\vec{V}_{n+1} = \vec{V}_n \text{ rot } w \cdot dt$$

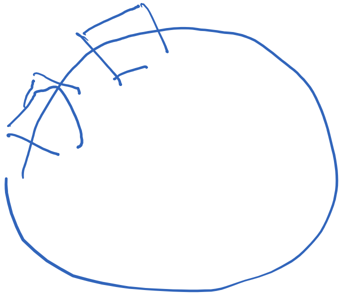


$$\vec{P}_{n+1} = \vec{P}_n + \vec{V} \cdot dt$$



Die Geschwindigkeit zumSimulationszeitpunkt **(n+1)** ist
Die Geschwindigkeit zum Simulationszeitpunkt **(n)** gedreht um **w**

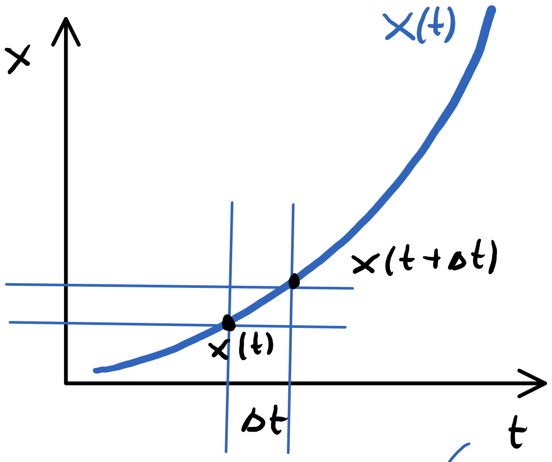
Die Position zum Simulationszeitpunkt **(n+1)** ist
die Position zum Simulationszeitpunkt **(n)** + der Geschwindigkeitsvektor



$$w = d\varphi$$

2 Gleichungen für Beschleunigte (v(t)) und für konstane (v=const) Bewegung

Bewegungsgleichung für beschleunigte Bewegung
beschleunigt heißt v(t) ist nicht konstant



Bei der beschleunigten Bewegung ist x(t) eine Kurve
Bei V=const ist x(t) . . . . .

$$V(t) = \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

Momentangeschw. zum Zeitpunkt t

$$x(t+\Delta t) = x(t) + V(t)\cdot\Delta t$$

Foreward Euler
Backward Euler
Runge Kutta

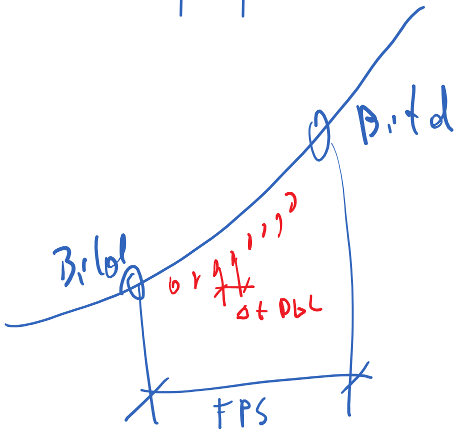
schlecht
↓
best

Simulationsprogramm rechnet in dt-Schritten
Übergang auf Iterationsschritte n, n+1, n+2 . . . .

$$x_{n+1} = x_n + V_{(n)}\cdot\Delta t$$

Allgemeine Gleichung für veränderliches V(t)

n	t
0	0
1	Δt
2	$2\cdot\Delta t$
⋮	⋮



Spezialfall v=const. :

$$x_{n+1} = x_n + V\cdot\Delta t$$

v ist nicht abhängig von n oder t

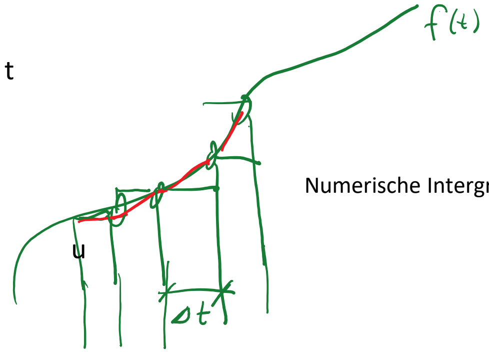
Übergang auf 2D-Vektorrechnung

$$\vec{S}_{n+1} = \vec{S}_n + \vec{V}\cdot\Delta t$$

$$x_{n+1} = x_n + V_x\cdot\Delta t$$

$$y_{n+1} = y_n + V_y\cdot\Delta t$$

2 Komponenten aufgeteilt



Numerische Intergration

2a Die ITER_PER_TICK und dt Geschichte

$$x_{n+1} = x_n + v \cdot \Delta t$$

HL

In allen unseren Bewegungssimulationen gibt es 2 wichtige Zeitgrößen

TIMER_INTERVAL . . . . das sind die Frames per Second

ITER_PER_TICK . . . . So oft werden die Bewegungsgleichungen pro neuem Frame durchgerechnet
Je größer ITER_PER_TICK gewählt wird desto genauer wird die Objektbewegung (Physik) berechnet.

Box2D

```
void OnTimer()
{
  for(i=0; i<ITER_PER_TICK; i++)
    CalcNextPositions();

}
```

Der Geschwindigkeitsmaßstab in unserer simulierten Welt ist *Pixel/FrameUpdate* und wird so gewählt, daß sich die Objekte mit einer beobachtbaren Geschwindigkeit über den Bildschirm bewegen.

Wenn ITER_PER_TICK=1 ist spielt das **dt** für uns eigentlich keine Rolle.
Wenn wir allerdings ITER_PER_TICK verändern wollen (Rechengenaugikeit)
so müssen wir **dt = 1/ITER_PER_TICK** setzen damit die Objekte nicht unbeabsichtigt schneller werden.

Wir werden später noch komplexere Bewegungs-Differentialgleichungen kennen lernen
(z.B. Ball mit Reibung und Gravitation)

In all diesen Differentialgleichungen kommt das **dt** immer wieder vor und ist entsprechend richtig zu setzen.

 Nature of Code

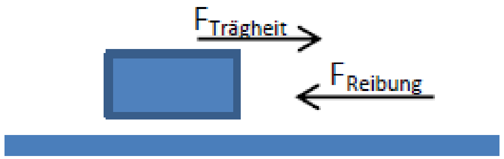
Units of Measurement

Now that we are introducing mass, it's important to make a quick note about units of measurement. In the real world, things are measured in specific units. We say that two objects are 3 meters apart, the baseball is moving at a rate of 90 miles per hour, or this bowling ball has a mass of 6 kilograms. As we'll see later in this book, sometimes we will want to take real-world units into consideration. However, in this chapter, we're going to ignore them for the most part. Our units of measurement are in pixels ("These two circles are 100 pixels apart") and frames of animation ("This circle is moving at a rate of 2 pixels per frame"). In the case of mass, there isn't any unit of measurement for us to use. We're just going to make something up. In this example, we're arbitrarily picking the number 10. There is no unit of measurement, though you might enjoy inventing a unit of your own, like "1 moog" or "1 yurkle." It should also be noted that, for demonstration purposes, we'll tie mass to pixels (drawing, say, a circle with a radius of 10). This will allow us to visualize the mass of an object. In the real world, however, size does not definitely indicate mass. A small metal ball could have a much higher mass than a large balloon due to its higher density.

3 Ball mit Reibung

$$F_{Trägheit} = m \cdot a(t)$$
$$F_{Reibung} = K_R \cdot V(t)$$

Ball mit v abgeschossen wird durch Reibung langsamer



$$\underbrace{m \cdot a(t)}_{\text{Trägheit}} + \underbrace{K_R \cdot V(t)}_{\text{Reibung}} = 0$$

$$m \cdot \frac{\Delta V(t)}{\Delta t} + K_R \cdot V(t) = 0$$

$$m \cdot \frac{V_{n+1} - V_n}{\Delta t} = -K_R \cdot V_n$$

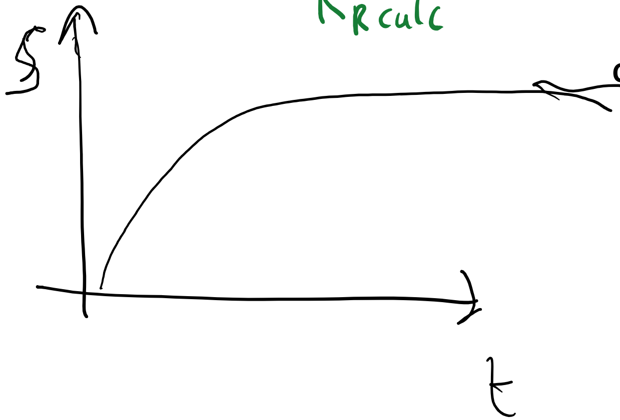
$$V_{n+1} = V_n - \frac{K_R}{m} \cdot V_n \cdot \Delta t$$

$$V_{n+1} = V_n \cdot \underbrace{\left(1 - \frac{K_R}{m} \cdot \Delta t\right)}_{K_{Rcalc}}$$

$$\boxed{\begin{aligned} \vec{V}_{n+1} &= \vec{V}_n \cdot K_{Rcalc} \\ \vec{S}_{n+1} &= \vec{S}_n + \vec{V}_n \cdot \Delta t \end{aligned}}$$

Diese 2 Gleichungen müssen für die Simulation von einem Simulationsschritt zum nächsten immer wieder neu berechnet werden.

Friction Ball

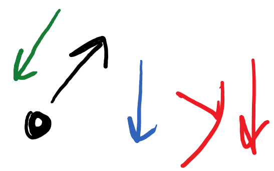


Code in Bulgati.hl/SwDev4te/PhysSim/3Didact

4 Ball mit Gravitation und Reibung

Trägheitskraft, Reibungskraft und Gravit ationskraft müssen im Gleichgewicht sein:

$$\underbrace{m\cdot\frac{\Delta\vec{v}}{\Delta t}}_{\text{Trägheit}} + \underbrace{K_R\cdot\vec{V}(t)}_{\text{Reibung}} + \underbrace{m\cdot g}_{\text{Gravitation}} = 0$$



$$m\cdot\frac{V_{n+1}-V_n}{\Delta t} + K_R\cdot V_n + m\cdot g = 0$$

$$V_{n+1}-V_n + \frac{K_R}{m}\cdot\Delta t\cdot V_n + g\cdot\Delta t = 0$$

$$V_{n+1} = V_n - \frac{K_R}{m}\cdot\Delta t\cdot V_n - g\cdot\Delta t$$

$$\vec{V}_{n+1} = \vec{V}_n\underbrace{\left(1-\frac{K_R}{m}\cdot\Delta t\right)}_{KR_{calc}} - g\cdot\Delta t$$

$$\boxed{\begin{aligned}\vec{V}_{n+1} &= \vec{V}_n\cdot KR_{calc} - g\cdot\Delta t\\ \vec{S}_{n+1} &= \vec{S}_n + \vec{V}_n\cdot\Delta t\end{aligned}}$$

g nur in y-Richtung

Code in Bulgati.hl/SwDev4te/PhysSim/3Didact

5 Ball mit Reibung CodeSnippets

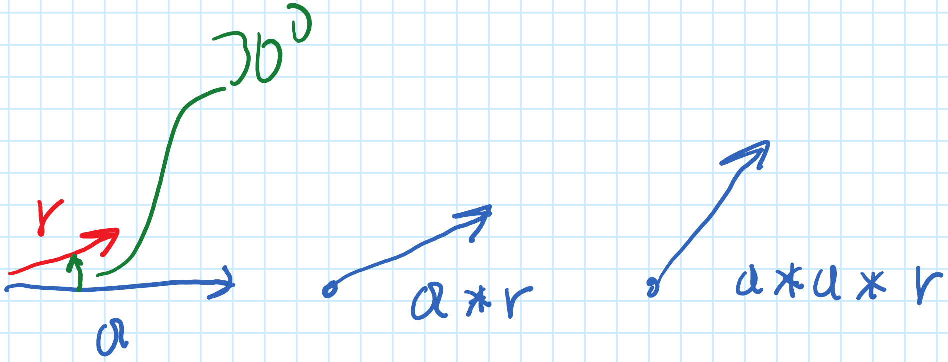
```
// Alle Simulations und Physik-Parameter werden zentral
// an einer Stelle gesetzt
public class Par
{
  // Geschwindigkeitseinheit ist Pixel pro TimerTick
  public const int TIMER_INTERVAL = 40; // 20  in ms

  // wie oft wird die Physik pro FrameUpdate gerechnet
  public const int ITER_PER_TICK = 20; // 5

  public const double DT = 1.0/(double)ITER_PER_TICK;

  public const double KR = 0.005*DT; // Reibungskonstante im Medium

  // Achtung!! hier kein DT
  public const double KRW = 0.1; // Reibungskonstante bei Reflexion

  // Vn+1 = Vn - Vn*KR;
  // Vn+1 = Vn*(1 - KR);
  public const double KR_CALC = 1 - KR;

  public const double KRW_CALC = 1 - KRW;

  public const double EARTH_ACCEL = 0.8 * DT;
```

```
public class FrictionBall : Ball
{
  public override void CalcNextPos()
  {
    // vn+1 = vn*KR_CALC
    m_V = m_V.ScalarMult(Par.KR_CALC);
    //Xn+1 = Vn*dt + Xn;
    m_Pos.AddTo(m_V, Par.DT);
  }
}
```

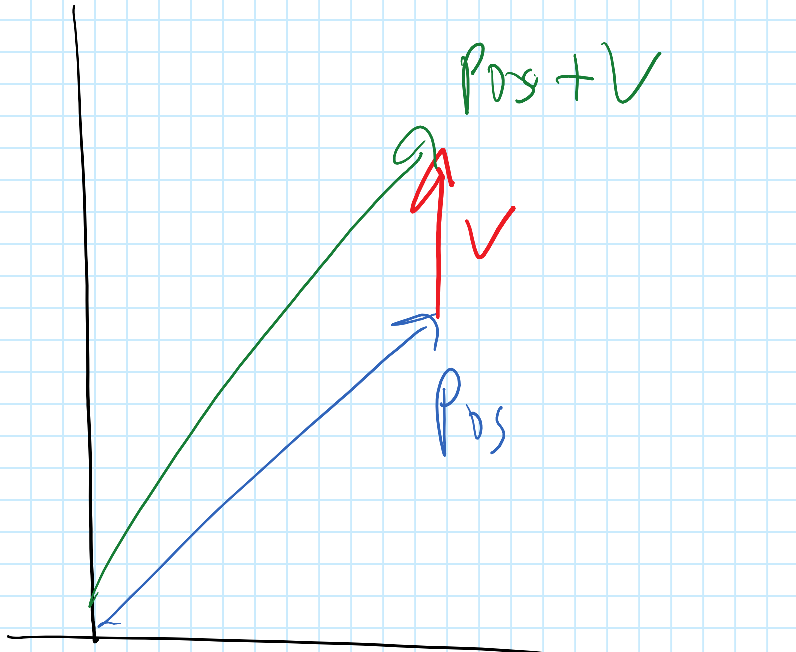
```
// Managen von Lebenszeit, Bewegung, und Reflexionen mehrerer Bälle.
// Die verwalteten Bälle können auch unterschiedliche Flugeigenschaften
// z.B. mit oder ohne Schwerkraft haben
// BasisKlasse für andere erweiterte BallManager wie z.B.
// BillardManager, TwoBallCollider, PingPong-Manager
public class BallManager
{
  protected ArrayList m_BallList = new ArrayList();

  // Basisimplemntierung und Schnittstelle zu
  // abgeleiteten BallManagern
  #region Interface for derived Managers
  public virtual void CalcNextPositions()
  {
    for(int i=0; i<Par.ITER_PER_TICK; i++)
    {
      foreach(Ball bl in m_BallList)
      {
        if( bl.ReflectInWindow(Ball.WndSize) )
          bl.WasReflected();
        bl.CalcNextPos();
      }
    }
  }
```

Code in Bulgati.hl/SwDev4te/PhysSim/3Didact

24 Vektorrotation mit complexer Multiplikation

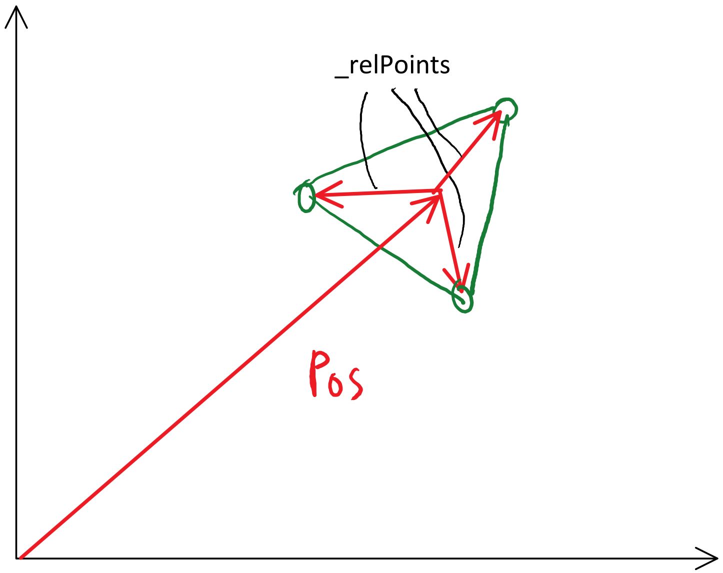


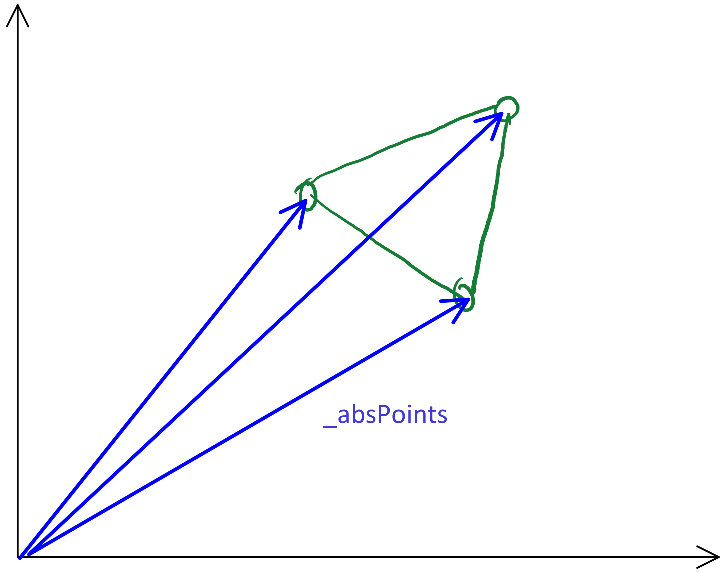
$|\vec{r}| = 1$ muß 1 sein

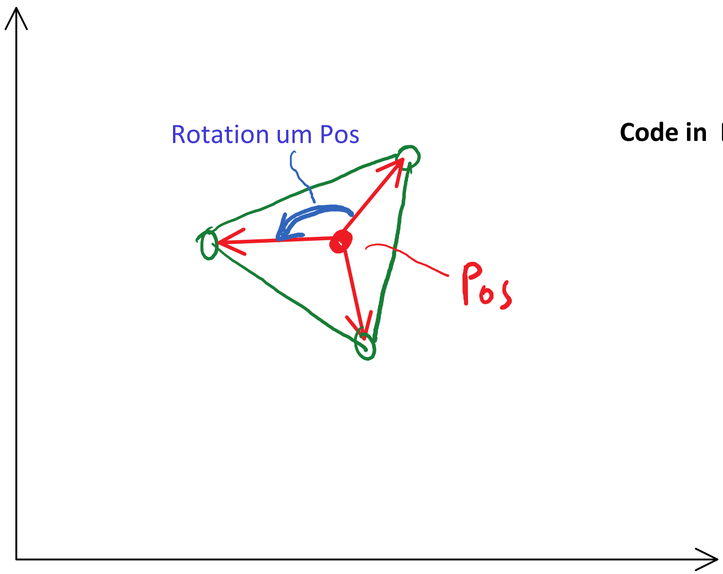


6 Objekte rotieren

_relPoints beschreibt ein Vektorgrafik-Objekt relativ zur momentanen Position Pos des Objekts
durch die Rotation von _relPoints um den relativen Koordinatenursprung Pos kann die Richtung des
RotGraphicObj geändert werden







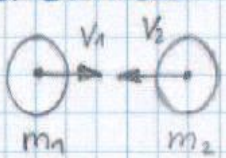
Code in Bulgati.hl/SwDev4te/PhysSim/RotObj

7 Zentraler Stoß

Stoßsatz (Impulssatz):

1) lineare Fall und zentraler Stoß

vor dem Stoß



nach dem Stoß

Gleichungen aus zwei physikalischen Grundgesetzen

1) Energieerhaltungssatz

$$\frac{m_1 \cdot |v_1|^2}{2} + \frac{m_2 \cdot |v_2|^2}{2} = \frac{m_1 \cdot |v_1'|^2}{2} + \frac{m_2 \cdot |v_2'|^2}{2}$$

2) Impulssatz

$$m_1 \cdot \vec{v_1} + m_2 \cdot \vec{v_2} = m_1 \cdot \vec{v_1}' + m_2 \cdot \vec{v_2}'$$

Nach Umformen und Einsetzen:

$$\vec{v_1}' = \frac{(m_1 - m_2) \cdot v_1 + 2 m_2 v_2}{m_1 + m_2}$$

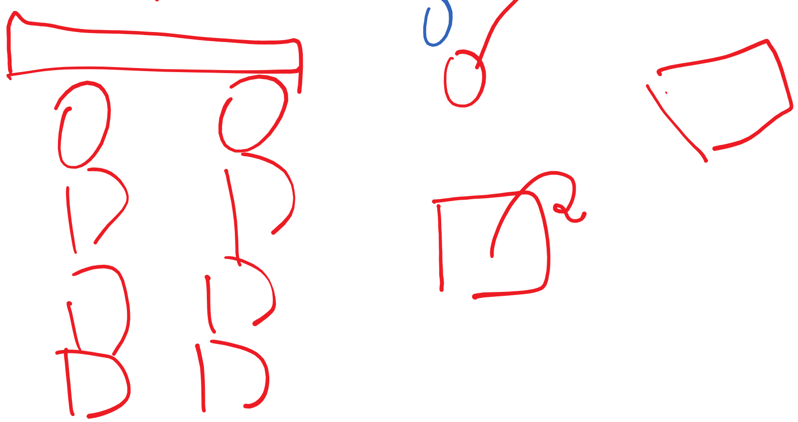
$$\vec{v_2}' = \frac{(m_2 - m_1) \cdot v_2 + 2 m_1 v_1}{m_1 + m_2}$$

bei $m_1 = m_2$: $\vec{v_1}' = \vec{v_2}$

$\vec{v_2}' = \vec{v_1}$

Nature of Code!

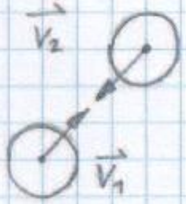
Box2D!



8 Zentraler Stoß

2) zentraler Stoß

liegt nur dann vor, wenn die Geschwindigkeitsvektor[en] beider Stoßpartner in einer Linie liegen.

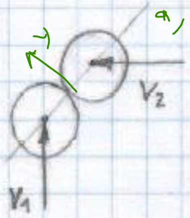


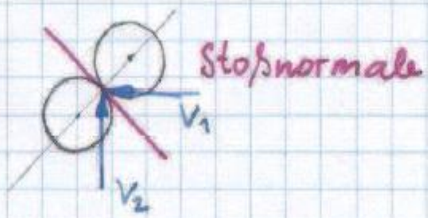
$v_{1x}' = v_{2x}$ $\qquad$ $v_{2x}' = v_{1x}$

$v_{1y}' = v_{2y}$ $\qquad$ $v_{2y}' = v_{1y}$

20. 10.

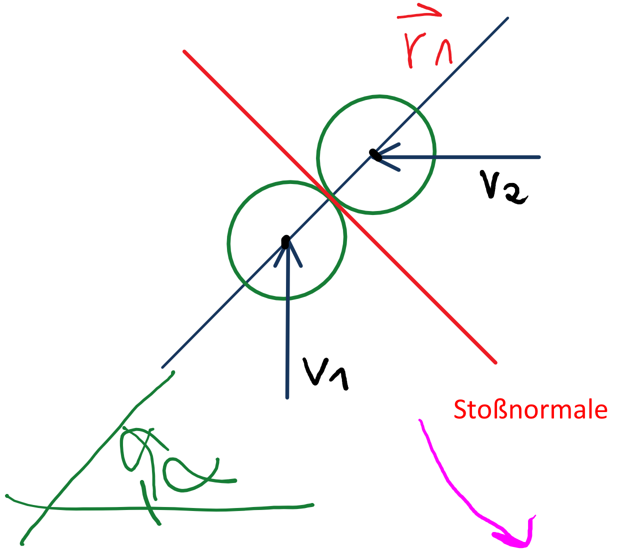
3) nichtzentraler Stoß

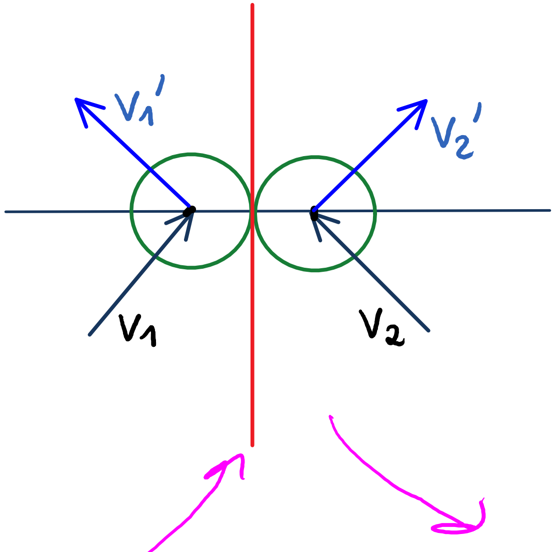


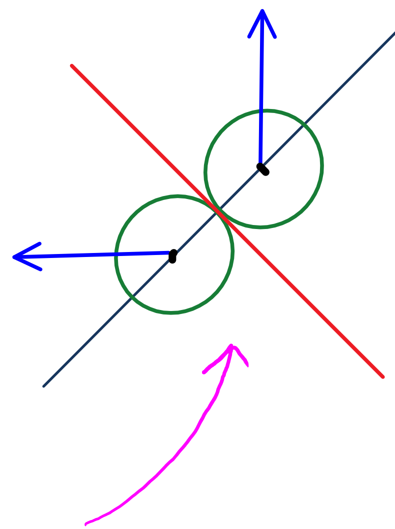


- Transformation der Geschwindigkeitsvektoren in das Koordinatensystem der Stoßnormalen
- Austausch der Geschwindigkeiten (Stoßsatz) (unklar, da beide Komponenten ??)
- Rückdrehen der neuen Geschwindigkeitsvektoren $\vec{v}_1'$ und $\vec{v}_2'$ in das Bildschirmkoordinatensystem

I9 Indirekter Stoß 1



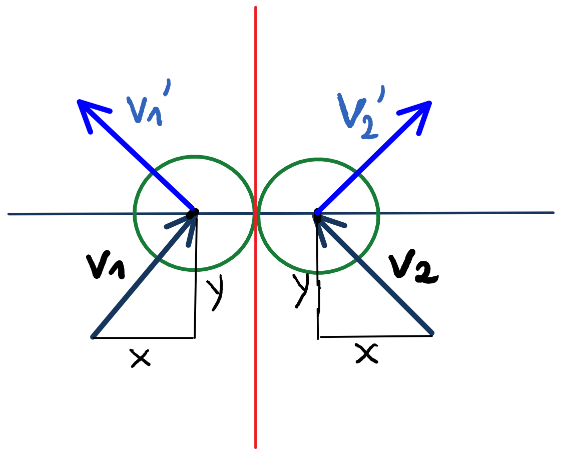




Drehen so daß die Stoßnormale zur Y-Achse wird

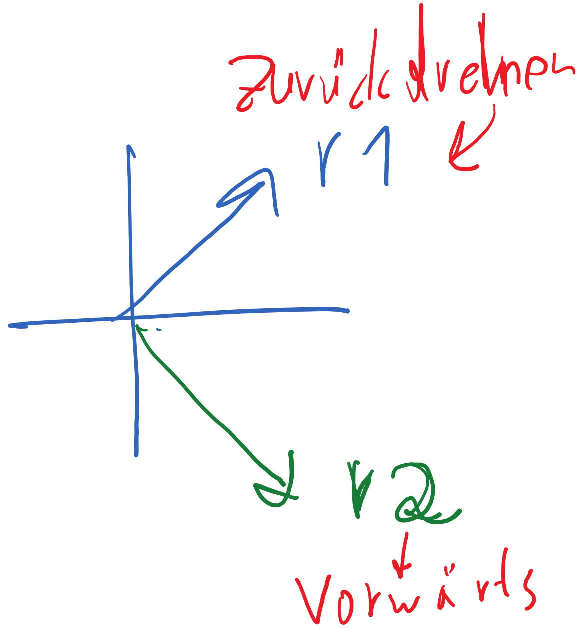
Zurückdrehen in die Ausgangssituation

Y-Komponenten bleiben
X-Komponenten werden ausgetauscht



$$V1'_x = V2_x \qquad V1'_y = V1_y$$

$$V2'_x = V1_x \qquad V2'_y = V2_y$$



10 Indirekter Stoß 2

```
public class Ball
{
  public Vect2D pos, V;
  public int m;
```

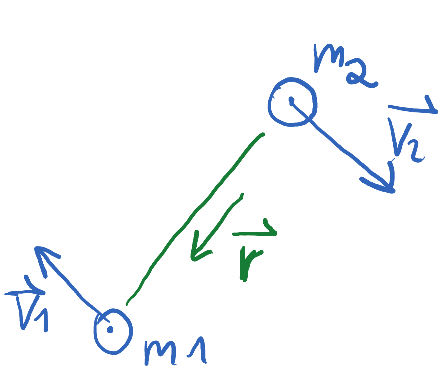
Code in Bulgati.hl/SwDev4te/PhysSim/Collide

```
public static void Collide1(Ball aB1, Ball aB2)
{
  // Vektor von B1 nach B2
  Vect2D r1 = Vect2D.VectBetweenPoints(aB2.pos, aB1.pos);

  // vektor zum Zurückdrehen
  r1 = r1.GetNormalizedVersion();

  // vektor zum Vorwärtsdrehen
  Vect2D r2 = r1.GetComplexConjugate();

  // V-Vektoren so drehen, daß die Stoßnormale mit der Y-Achse zusammenfällt
  aB1.V.CoMultTo(r2); aB2.V.CoMultTo(r2);

  // Vx-Komponenten austauschen Vy-Komponenten bleiben unverändert
  ImpulseRule2(aB1, aB2);

  // V-Vektoren wieder zurückdrehen
  aB1.V.CoMultTo(r1); aB2.V.CoMultTo(r1);
}
```

```
// Mass is !!not!! used
static void ImpulseRule1(Ball aB1, Ball aB2)
{
  double tmp = aB1.V.X;
  aB1.V.X = aB2.V.X;
  aB2.V.X = tmp;
}
```

```
// Mass is used
static void ImpulseRule2(Ball b1, Ball b2)
{
  int nenner = b1.m + b2.m;
  double v1s, v2s;
  v1s = ((b1.m - b2.m) * b1.V.X + 2 * b2.m * b2.V.X) / (double)nenner;
  v2s = ((b2.m - b1.m) * b2.V.X + 2 * b1.m * b1.V.X) / (double)nenner;
  b1.V.X = v1s;
  b2.V.X = v2s;
}
```

11 Sateliten Gleichung



m_2 ... Erde
m_1 ... Sattelit

$$m_1 \cdot \dot{\vec{V_1}} = \frac{-m_1 \cdot m_2 \cdot G}{r^2} \cdot \frac{\vec{r}}{|r|}$$

Beschleunigungskraft und Gravitationskraft
halten sich das Gleichgewicht

$m_1 \cdot \dot{\vec{V_1}}$ .... Beschleunigungskraft

$-\frac{m_1 \cdot m_2 \cdot G}{r^2}$ .... Gravitationskraft

$\frac{\vec{r}}{r}$ ... Einheitsvektor in Radiusrichtung

$\dot{\vec{V_1}}$ ..... 1te Ableitung der Geschw. nach der Zeit => Beschleunigung

$$\dot{\vec{V_1}} = -\frac{m_2 \cdot G}{r^2} \cdot \frac{\vec{r}}{|r|}$$

← $\overrightarrow{accGrav}$

accGrav: wird aus dem Abstand Sat <-> Erde/r² und
dem Einheitsvektor in Radiusrichtung gebildet

$$\frac{\vec{V}_{n+1} - \vec{V_n}}{\Delta t} = -\overrightarrow{accGrav}$$

$$\boxed{\begin{aligned} \vec{V}_{n+1} &= \vec{V_n} - \overrightarrow{accGrav} \cdot \Delta t \\ \vec{S}_{n+1} &= \vec{S_n} + \vec{V_n} \cdot \Delta t \end{aligned}}$$

! Das muß programmiert werden !

Erste Schritt auf 2 Körper erweiren
2ter Schritt N-Körper Problenm

12 Sateliten Gleichung

```
void CalcNextPositions()
{
  Vect2D rVect, acc;
  double rDist;
  for (int i = 0; i < Par.ITER_PER_TICK; i++)
  {
    // Vector von der Erde zum Satteliten
    rVect = m_Earth.VectBetweenObjects(m_Sat);

    rDist = rVect.GetR(); // Abstand  Erde Sattelit

    // rVect auf 1 normiert ( Länge == 1 )
    rVect = rVect.ScalarMult(1 / rDist);

    // Gravitationsvektor der auf den Sat wirkt g/r^2
    acc = rVect.ScalarMult(Par.EARTH_ACCEL/(rDist*rDist));

    // Vn+1 = Vn - acc*DT
    m_Sat.m_V.SubFrom(acc);

    // Xn+1 = Xn + Vn*DT
    m_Sat.IntegratePosition();
  }
  m_Sat.AddTracePoint();
}
```

```
public const int ITER_PER_TICK = 50; // 50
public const double DT = 1.0 / (double)ITER_PER_TICK;
public const double EPS_V = 1E-3;
public const double COLLIDE_DIST = 25;
// public const double EARTH_ACCEL = 1E3 / ITER_PER_TICK; // 0.2
public const double EARTH_ACCEL = 50E3 / ITER_PER_TICK; // 0.2
public const double ENGINE_ACCEL = 0.005;
```

Code in Bulgati.hl/SwDev4te/PhysSim/Sattelite

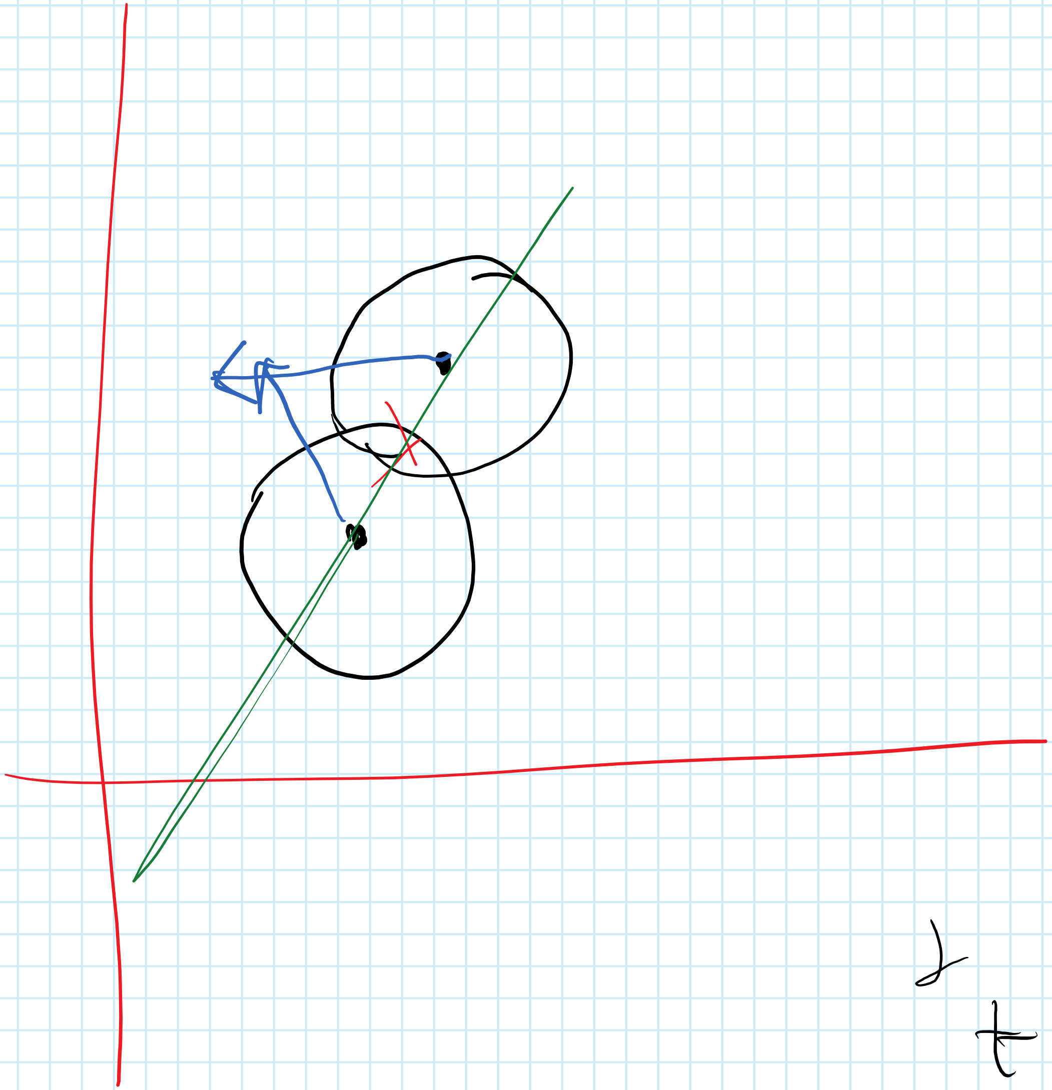
12a N - Körper Problem

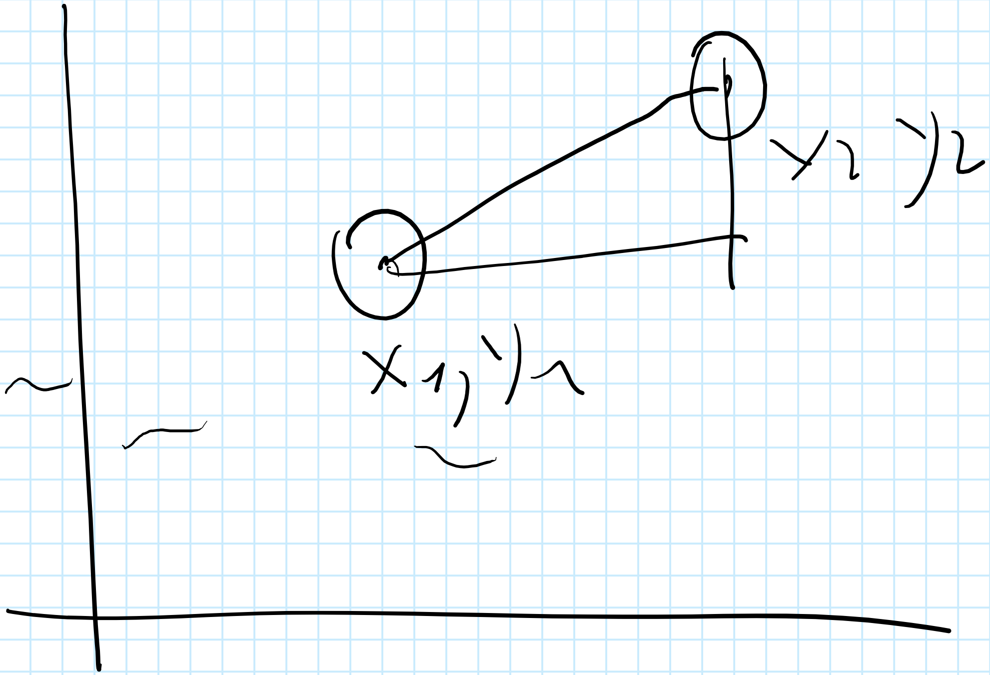
$$\cancel{m_1} \cdot \dot{\vec{v}}_1 = - \sum_i \frac{\cancel{m_1} \cdot m_i \cdot G}{r_i^2} \cdot \vec{r}_i$$

$$\dot{\vec{v}}_1 = - \sum_i \frac{m_i \cdot G}{r_i^2} \cdot \vec{r}_i$$

$$\dot{\vec{v}}_n = - \sum_i m_i \cdot G \cdot \frac{\vec{r}_i}{r_i^2}$$

13 Misc





$$u = \text{const}$$

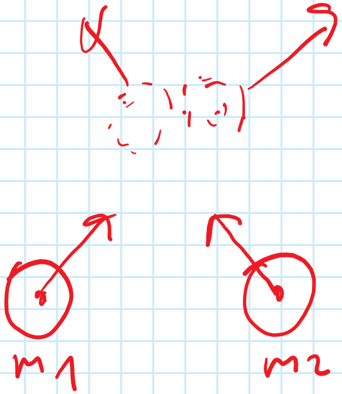
$$l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

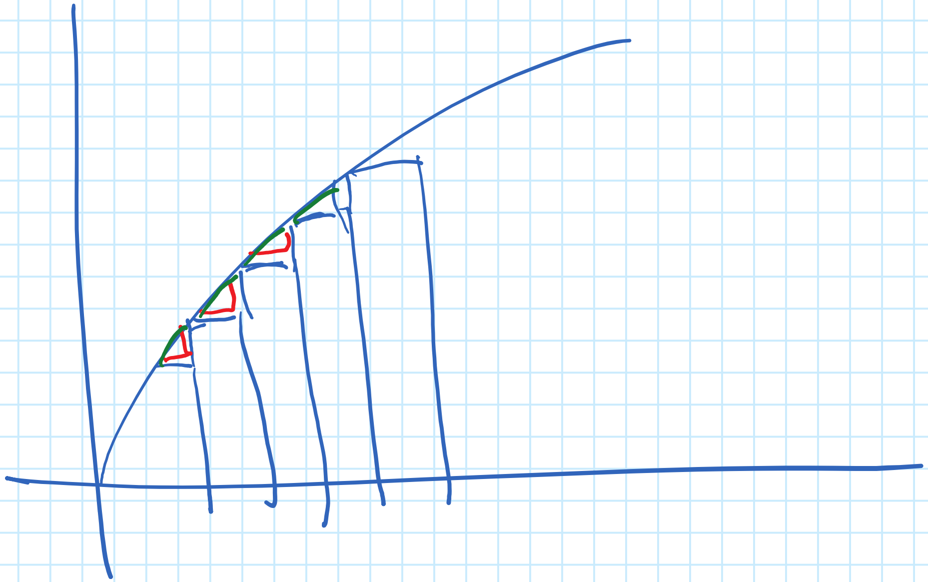
13 Misc

b = 1 /-45°

c = ?

a = 1 / 45°



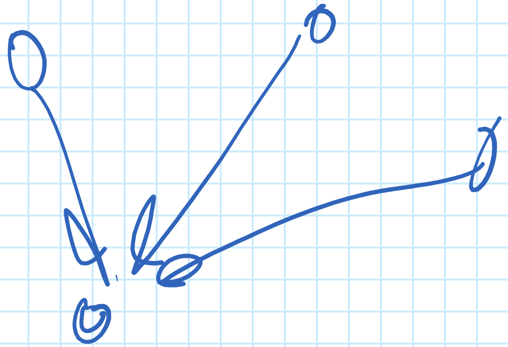


13 Misc

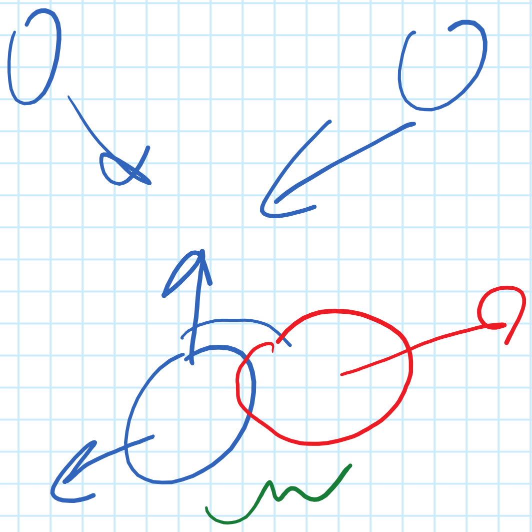
b = 1 /-45°

c = ?

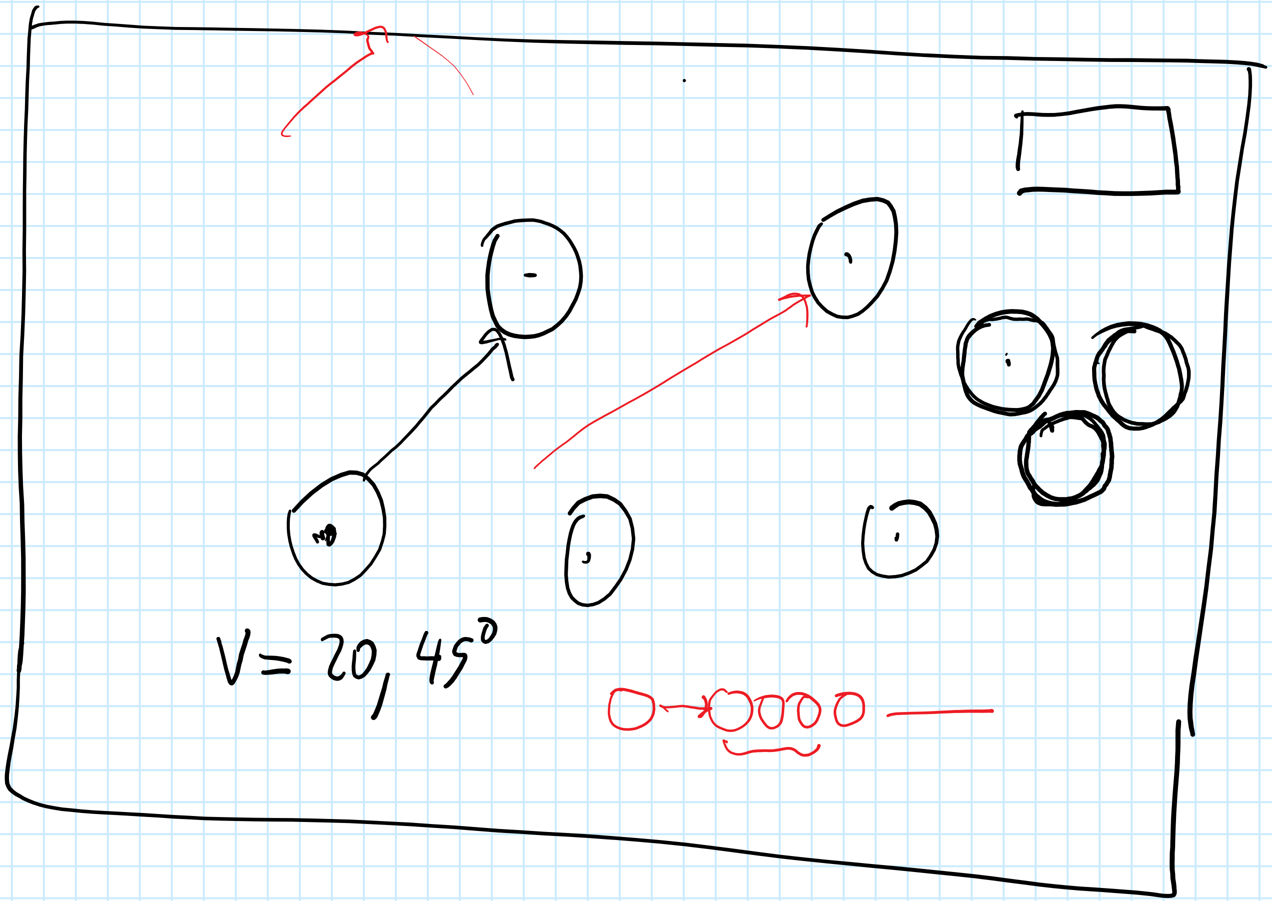
a = 1 / 45°



N- Body

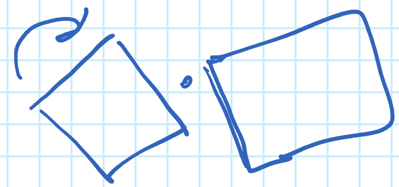


13 Misc

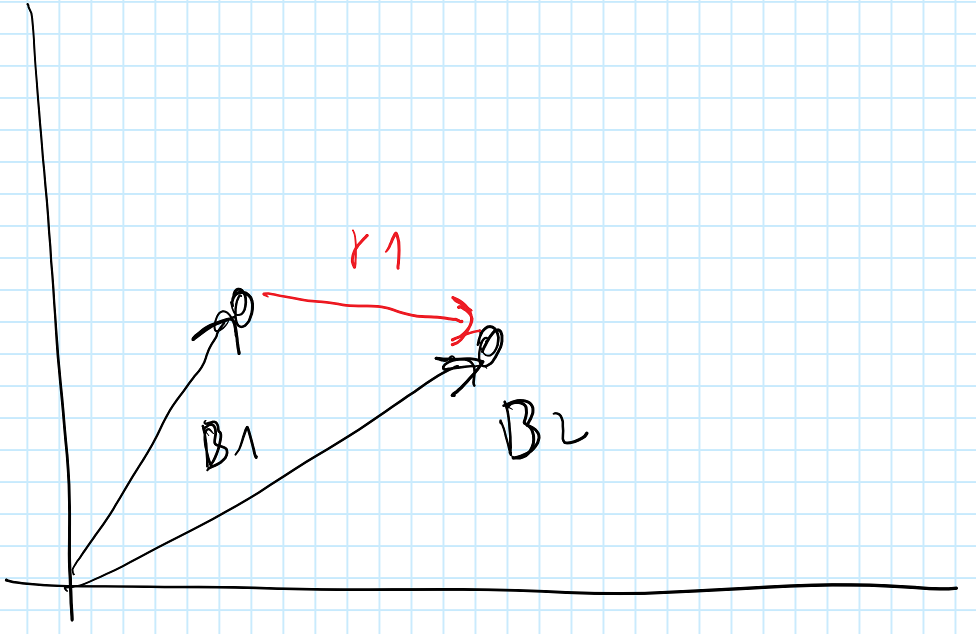


$n \to n+1$

Singhly Sten

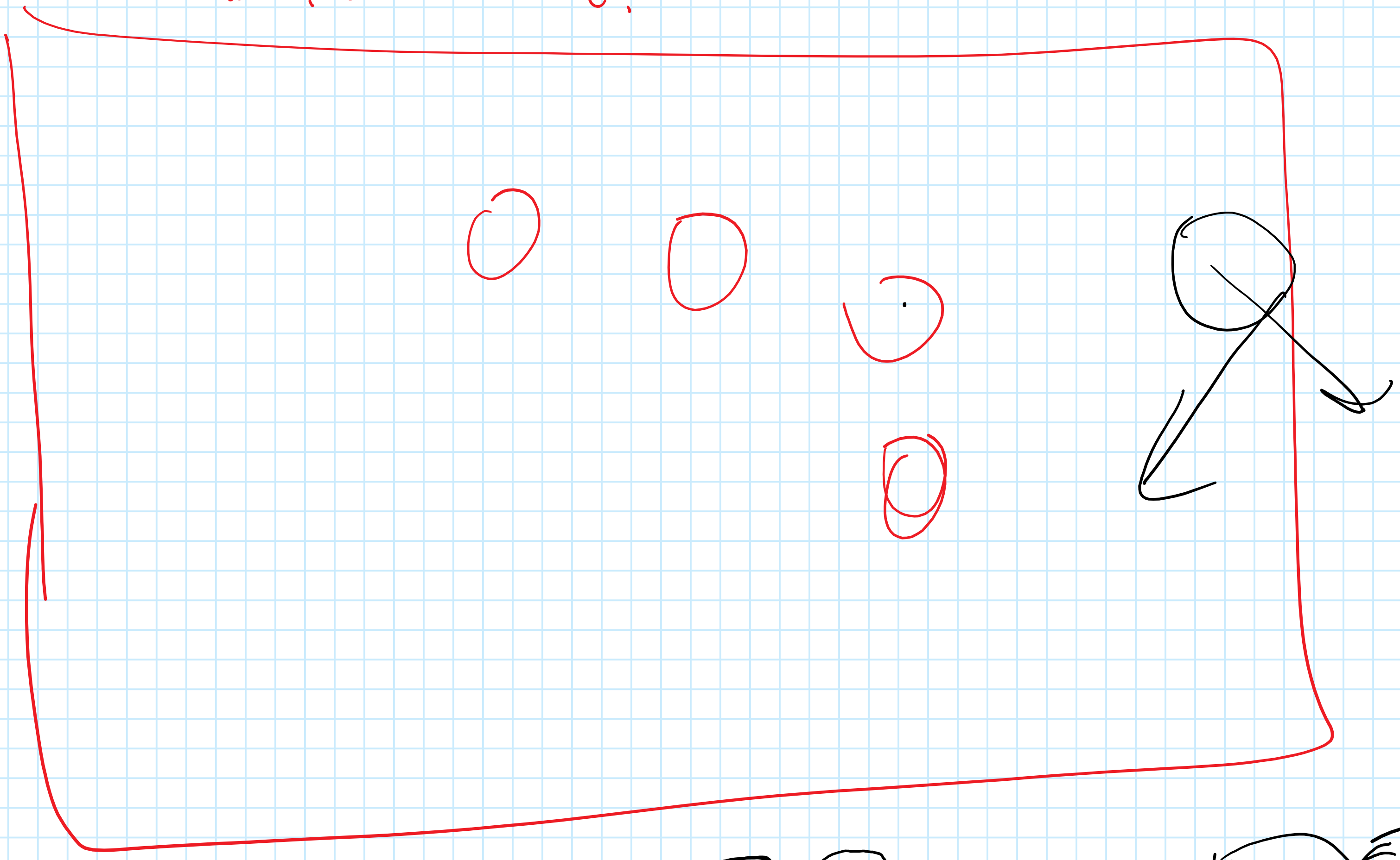


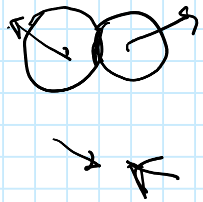
13 Misc

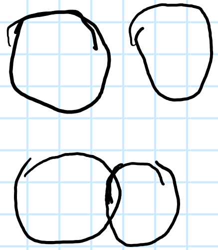


13 Misc

m : v   2 : 1

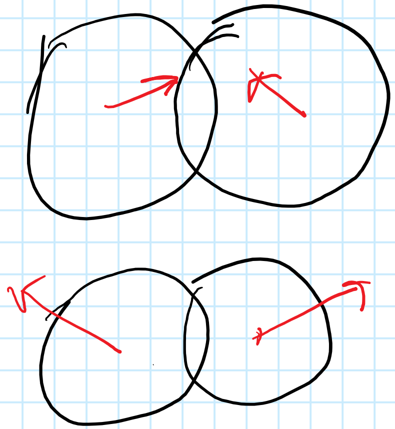






n

n + 1



n + 1

2

13 Misc