Parallele Algorithmen mit OpenCL

Universität Osnabrück, Henning Wenke, 2013-05-22

Einschub

OpenCL C Datentypen

Skalare Datentypen

- Integer
  - bool: 0 (false) oder 1 (true)
  - 8Bit: char, uchar / unsigned char
  - 16Bit: (u)short
  - (u)int (32 Bit), (u)long (64Bit)
  - ...
- Floating Point
  - half (16Bit)
  - float (32Bit), double (64Bit)
  - Weitere reserviert: quad(128Bit), complex/imaginary...

Vektorielle Datentypen

- Viele Skalare existieren auch als 2, (3), 4, 8, 16 dimensionale vektorielle Datentypen
- Beispiele:
  - float8: 8-dimensionaler Vektor des Typs float
  - ulong16: 16-dimensionaler Vektor des Typs unsigned long
- Operatoren wirken komponentenweise
- Intern einfach Folge skalarer Datentypen

Beispiele

```
float4 a = (float4) (1.0f, 2.0f, 3.0f, 4.0f);
float2 b = (float2) (1.0f, 2.0f);
float4 c = a + a; // ergibt (2.0, 4.0, 6.0, 8.0)
float4 d = a * a; // ergibt (1.0, 4.0, 9.0, 16.0)
float4 e = 5 * a; // ergibt (5.0, 10.0, 15.0, 20.0)


// Zugriff auf Komponenten
//.s0 liefert Komponente 0, .s1 (Komponente 1) bis .sf (Komponente 15)
a.s2; // Liefert 3.0

// Zugriff auf die ersten 4 Komponenten auch mit x, y, z, w
a.z;  // Liefert ebenfalls 3.0

// Auch möglich: Zugriff auf mehrere (auch vertauschte) Komponenten
a.s31; // Liefert (float2)(4.0, 2.0)

// Built-in functions, z.B. Kreuzprodukt aus a und c:
float4 g = cross(a, c);
```

Anwendung / Übung

- Gegeben: Folge von $2n$ 4-komponentigen Vektoren
- Aufgabe: Berechne von jedem zweiten Vektor Skalarprodukt mit direktem Nachfolger
- Verwende dazu im Kernel einmal float und einmal float4
- Layout der globalen Daten:
  - In beiden Fällen float-Folge
  - Daten der „Objekte" liegen konsekutiv mit aufsteigendem Index im Speicher
  - Vektorielle Daten mit entsprechender Unterordnung ebenfalls
- Ausgangsdaten: Folge von $2 \cdot n \cdot 4$ floats
  - `float* vectors`: $\{ v_{0,x},\ v_{0,y},\ v_{0,z},\ v_{0,w},\ v_{1,x}, \dots ,\ v_{2n-1,z},\ v_{2n-1,w} \}$
- Datenstruktur für Ergebnis: Folge von n floats
  - `float* dotProduct`: $\{ d_0,\ d_1,\ d_2,\ d_3,\ d_4, \dots ,\ d_{n-2},\ d_{n-1} \}$

Kernel 1: Skalare Datentypen

`float* vectors:` $\{v_{0,x}, v_{0,y}, v_{0,z}, v_{0,w}, v_{1,x}, \dots, v_{2n-1,z}, v_{2n-1,w}\}$

```
// Erzeuge  n  Work-Items & verwende  1 - dimensionale Indizierung
kernel void dot1 (
 global float* vectors,
 global float* dotProduct)
{
    int id = get_global_id(0); // Index der Skalarproduktberechnung
    int v1_Id = 2 * id;        // Index des ersten Vektors
    int v2_Id = v1_Id + 1;     // Index des zweiten Vektors

    // Lade ersten Vektor
    float v1_x = vectors[4 * v1_Id    ]; // 4: Komponentenanzahl
    float v1_y = vectors[4 * v1_Id + 1];
    float v1_z = vectors[4 * v1_Id + 2];
    float v1_w = vectors[4 * v1_Id + 3];
    // Lade zweiten Vektor
    float v2_x = vectors[4 * v2_Id    ];
    float v2_y = vectors[4 * v2_Id + 1];
    float v2_z = vectors[4 * v2_Id + 2];
    float v2_w = vectors[4 * v2_Id + 3];
    // Berechne dot(v1, v2):
    float result = v1_x * v2_x + v1_y * v2_y + v1_z * v2_z + v1_w * v2_w;
    dotProduct[    id    ] = result;
}
```

`float* dotProduct:` $\{d_0, d_1, d_2, d_3, d_4, \dots, d_{n-2}, d_{n-1}\}$

Kernel2: Vektorielle Datentypen

`float* vectors:` $\{v_{0,x}, v_{0,y}, v_{0,z}, v_{0,w}, v_{1,x}, \dots, v_{2n-1,z}, v_{2n-1,w}\}$

```
// Erzeuge  n  Work-Items & verwende  1 - dimensionale Indizierung
kernel void dot4(
 global float4* vectors,
 global float* dotProduct)
{
    int id = get_global_id(0); // Index der Skalarproduktberechnung
    int v1_Id = 2 * id;        // Index des ersten Vektors
    int v2_Id = v1_Id + 1;     // Index des zweiten Vektors

    // Lade ersten Vektor
    float4 v1 = vectors[  v1_Id  ]; // Holt 4 konsekutive floats

    // Lade zweiten Vektor
    float4 v2 = vectors[  v2_Id  ];

    // Berechne dot(v1, v2) selbst
    float result = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z + v1.w * v2.w;

    // Oder: Nutze built-in function dot
    result = dot(v1, v2);

    dotProduct[    id    ] = result;
}
```

`float* dotProduct:` $\{d_0, d_1, d_2, d_3, d_4, \dots, d_{n-2}, d_{n-1}\}$

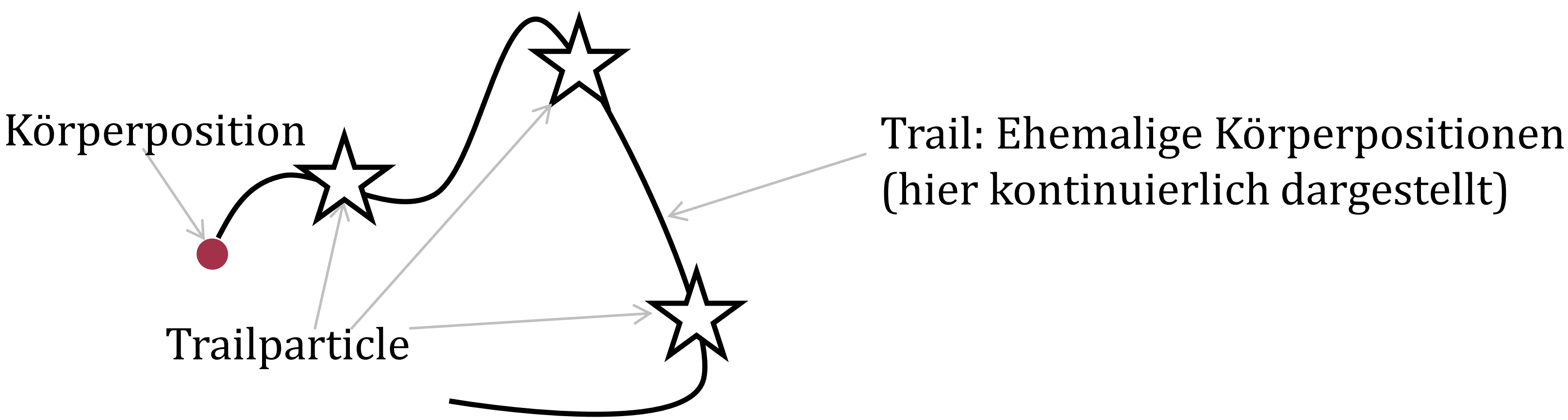
Algorithmus

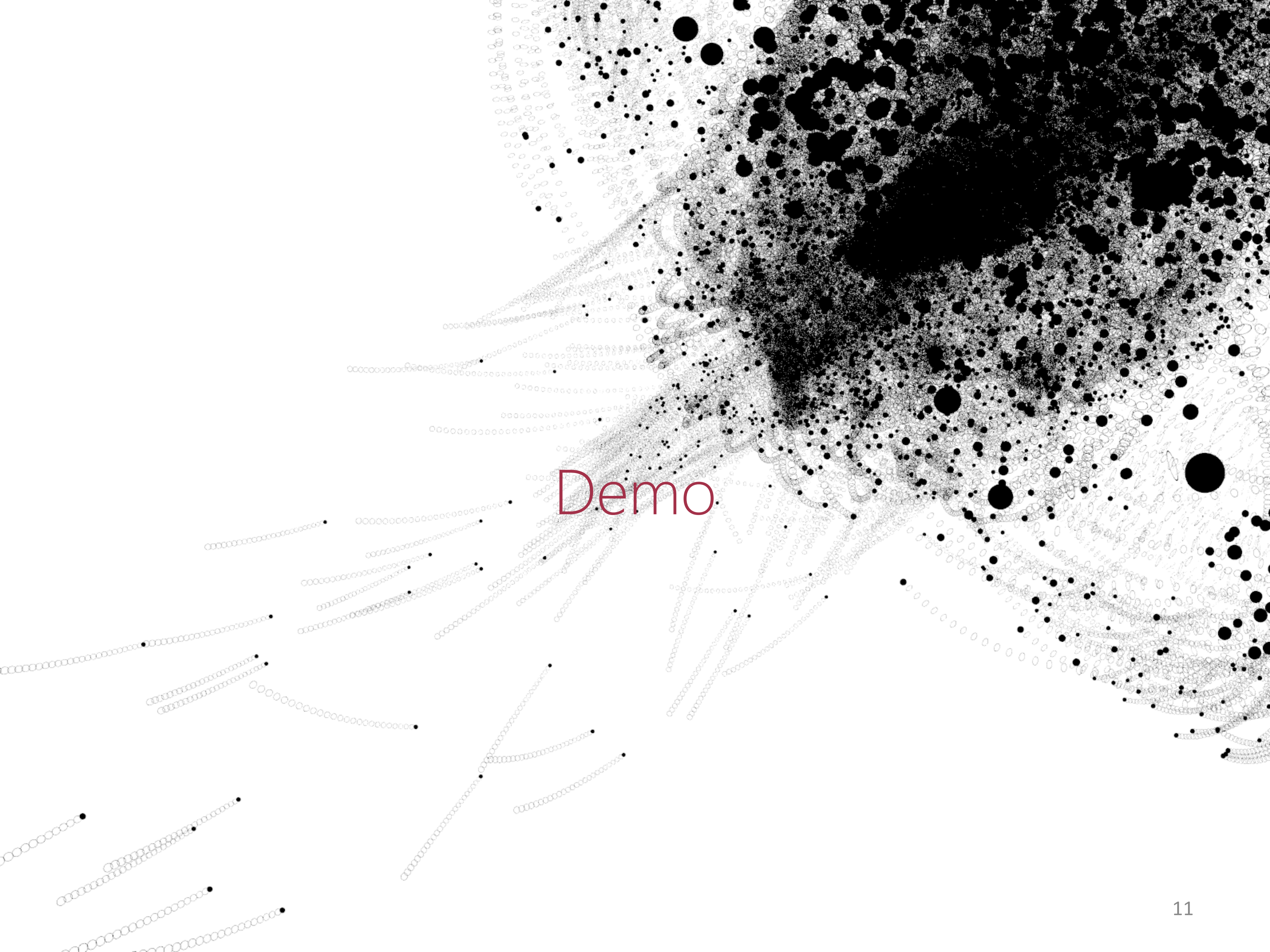
N-Body Simulation + Trailvisualization

Quelle: Selbstgemacht.

Wünsche

- Jedem Körper soll ein Trail folgen
- Soll genau der jüngsten Bahn des Körpers entsprechen
- Visualisierung des Trails durch mehrere masselose „Trail Particles“ (TP)
- Sollen sich auf Trail bewegen
- Geschwindigkeit relativ definierbar zu der des Körpers an der entsprechenden Stelle

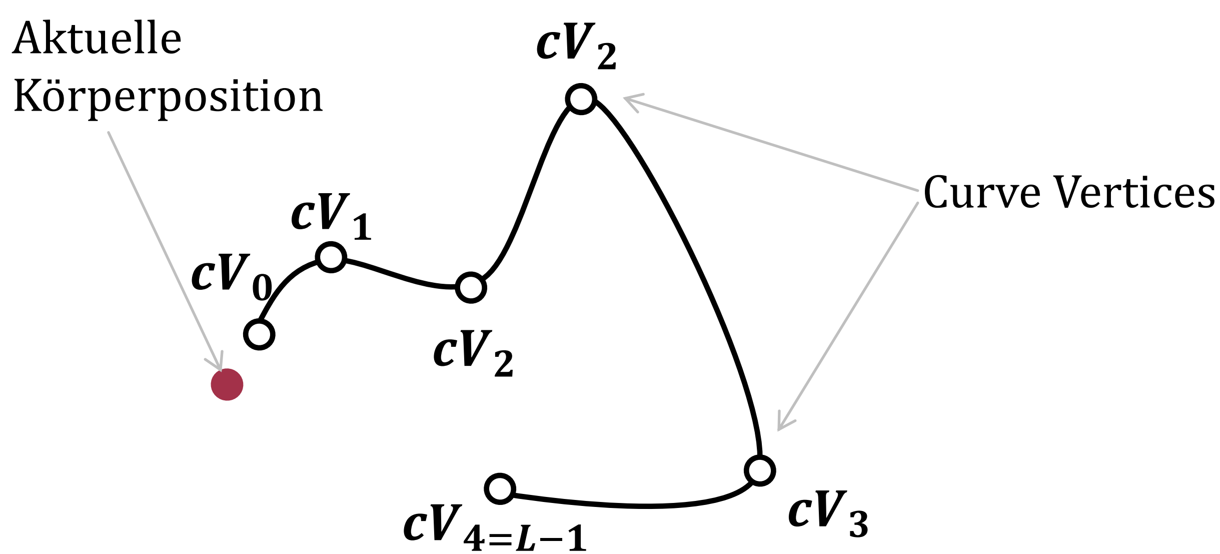




Demo

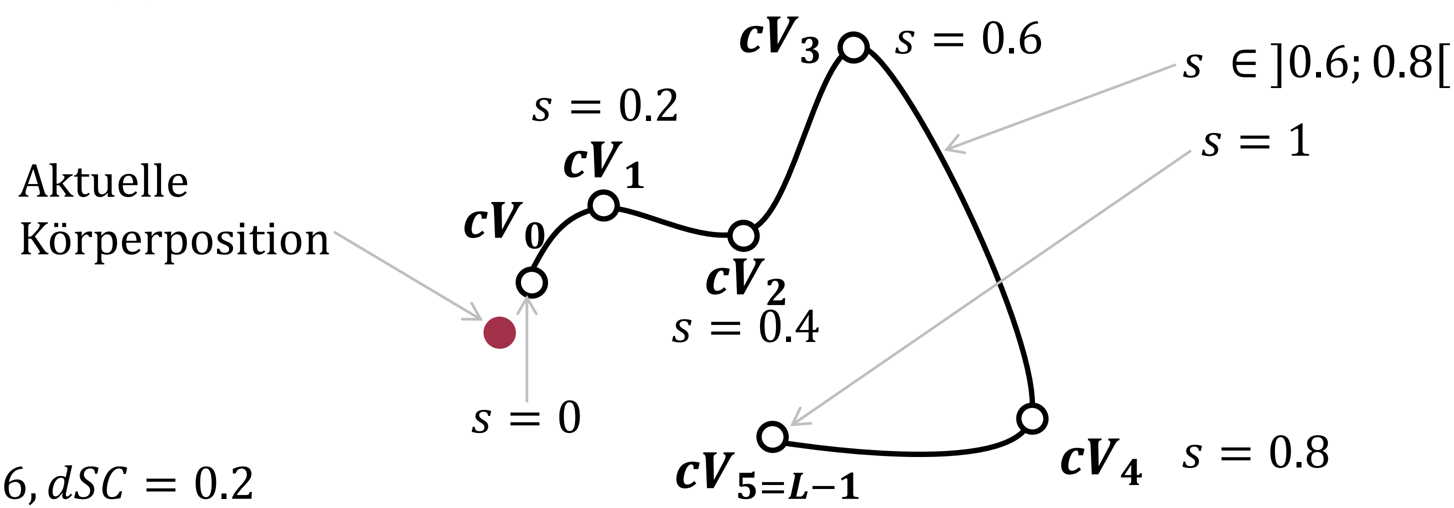
Kurve

- Ziel: Speichere Bahn eines Körpers der letzten L Simulationsschritte
- Kopiere dazu *L* letzte Positionen jedes Körpers
- Nennen wir: *Curve Vertex* (CV)
- Definieren, zusammen mit Interpolationsvorschrift, Kurve
- Hinweis: Bis Folie 8 betrachten wir lediglich <u>eine</u> <u>statische</u> Kurve



Definition des Kurvenparameters

- Definiere Kurvenparameter $s, mit\ s \in [0,1]$ auf der Kurve durch CV
- Es gelte für s an der Stelle eines Curve Vertex mit Index i: $s_{CV,i} = i/(L-1)$
- Definiere CV-Kurvenparameterunterschied: $dSC = 1/(L-1)$
- Funktion $curve(s)$ liefert Position auf Kurve für Kurvenparameter s:
  - $curve(s \leftarrow 0 \cdot dSC = 0) = \boldsymbol{cV_0}$
  - $curve(s \leftarrow 1 \cdot dSC) = \boldsymbol{cV_1}$
  - $curve(s \leftarrow (L-1) \cdot dSC = 1) = \boldsymbol{cV_{L-1}}$
  - Sonst: Dazwischen…
  - Größere s weiter „hinten“
- $curve(s)$ nichtlinear, da Abstände der Curve Vertices variieren



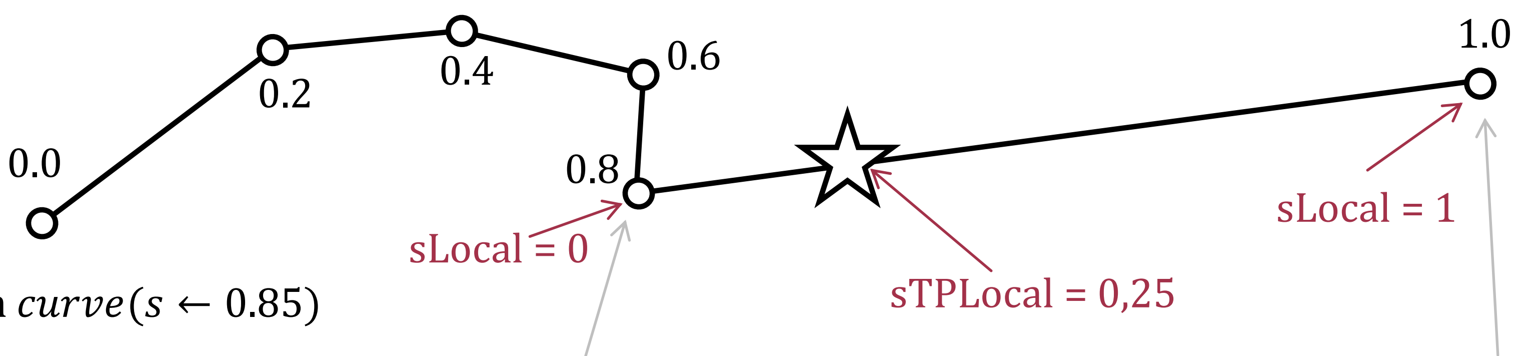
$L = 6, dSC = 0.2$

Interpolation: Abschnittsweise linear

- Gegeben:
  - Kurve mit L Curve Vertices $\Longrightarrow dSC = 1/(L-1)$
  - Kurvenparameter sTP eines TP
- Gesucht: $Position\colon\ curve(sTP)$

1. Suche diejenigen zwei Curve Vertices, zwischen denen s liegt
   - $localIndexPrevCV \leftarrow \lfloor sTP \cdot (L - 1) \rfloor$ (Achtung: Lokaler Index in Kurve)
   - $localIndexNextCV \leftarrow localIndexPrevCV + 1$
2. Bestimme Parameter *sTPLocal* für lineare Interpolation zwischen diesen
   - $sPrevCV \leftarrow localIndexPrevCV \cdot dSC$
   - $sTPLocal \leftarrow (sTP - sPrevCV)/dSC$ (Offset abschneiden & Intervall normieren)
3. Gesuchter Punkt: $\boldsymbol{posPrevCV} \cdot (1 - sTPLocal) + sTPLocal \cdot \boldsymbol{posNextCV}$

Beispiel: L = 6,
dSC ← 0,2.
Gesucht: Position $curve(s \leftarrow 0.85)$



localIndexPrevCV ← $\lfloor 0.85 \cdot (6 - 1) \rfloor = \lfloor 4{,}25 \rfloor = 4$

sPrevCV ← 4 · 0,2 = 0,8

localIndexNextCV ← 5

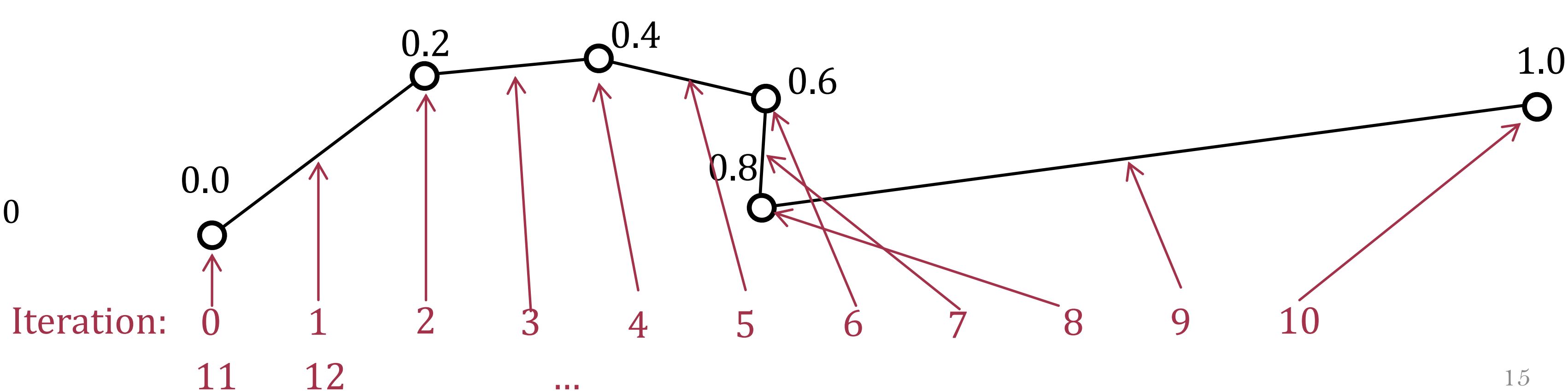
sTPLocal ← (0,85 – 0,8) / 0,2 = 0,25

Bewegung eines TP auf statischer Kurve

- Eine Lösung: Jedes TP bekommt Kurvenparameter sTP…
- …für jeden TP einer Kurve initial anders
- Globale Konstante für Änderung von sTP in jedem Schritt: dSTP
- Berechne Position eines TP gemäß:

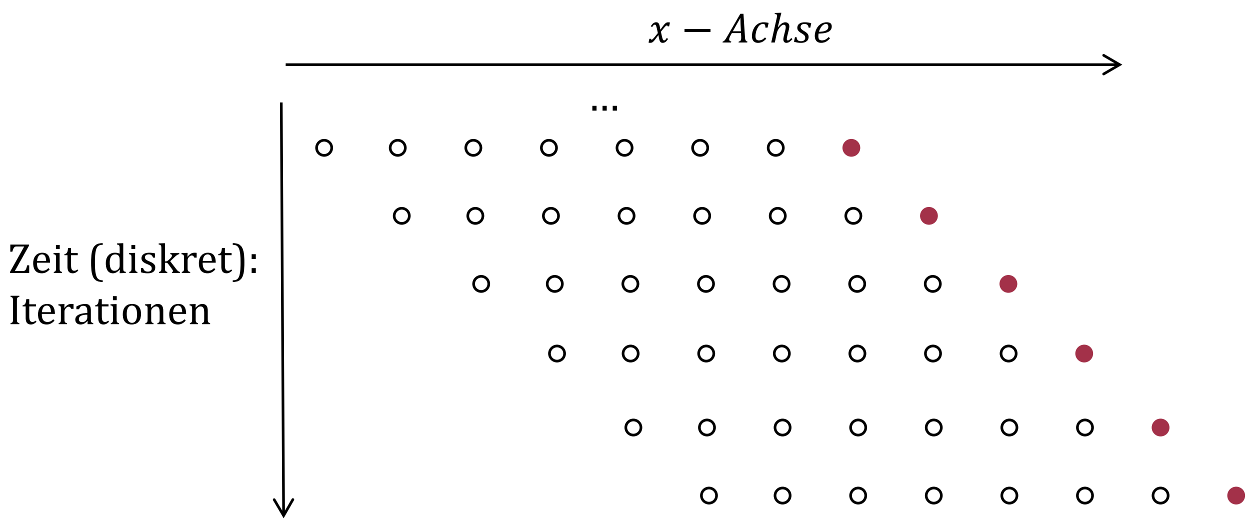
```
sTP ← initialVal // Unterschiedlich, damit Partikel auf Kurve verteilt
While (Iterating)
   pos ← call curve(sTP)
   sTP ← sTP + dSTP
   if (sTP > 1)      // Zyklisch: Falls Kurve hinten verlassen...
      sTP ← sTP – 1 //  setze TP nach vorne
   End
End
```

Beispiel:
L = 6,
dSTP = 0,1
initialVal = 0



Dynamische Kurve

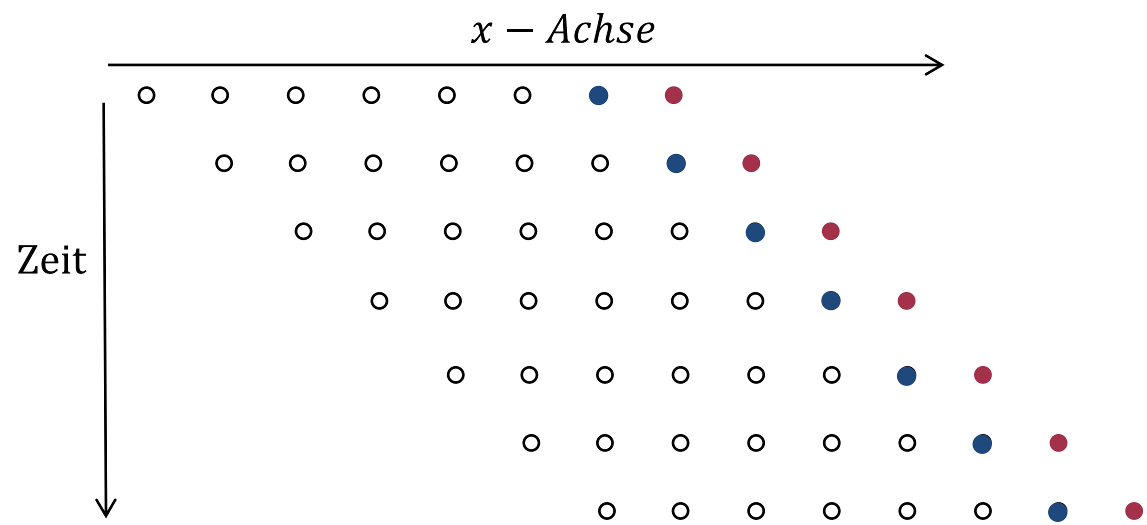
- Kurve bekommt in jedem Zeitschritt vorne einen Curve Vertex hinzu und verliert hinten einen
- Veranschaulichung für Bewegung mit konstanter Geschwindigkeit in x-Richtung:



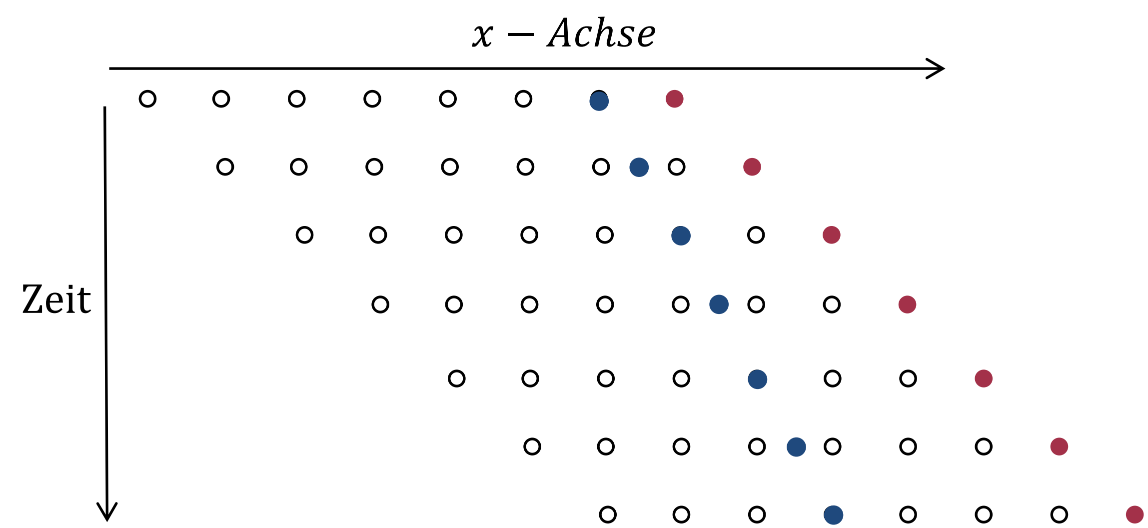
- ● Aktuelle Position des Partikels
- ○ Alte Position des Partikels

Trailparticle in dynamischer Kurve

$dSTP = 0$



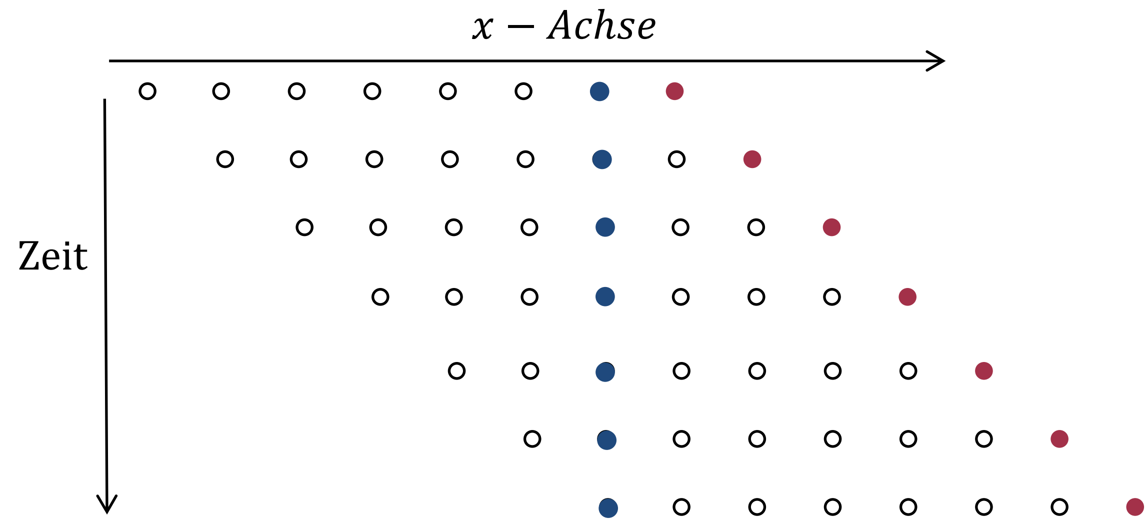
$dSTP = 0{,}5 \cdot dSC$



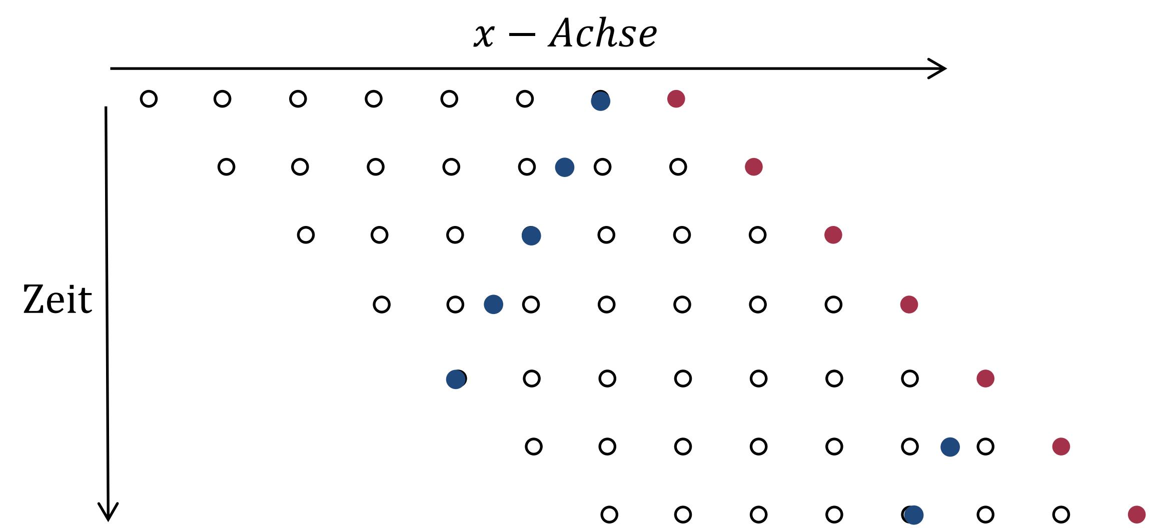
● Aktuelle Körper-Position ○ Alte Körper-position ● Trace-Particle Position initialVal = 0

Trailparticle in dynamischer Kurve II

$dSTP = dSC$



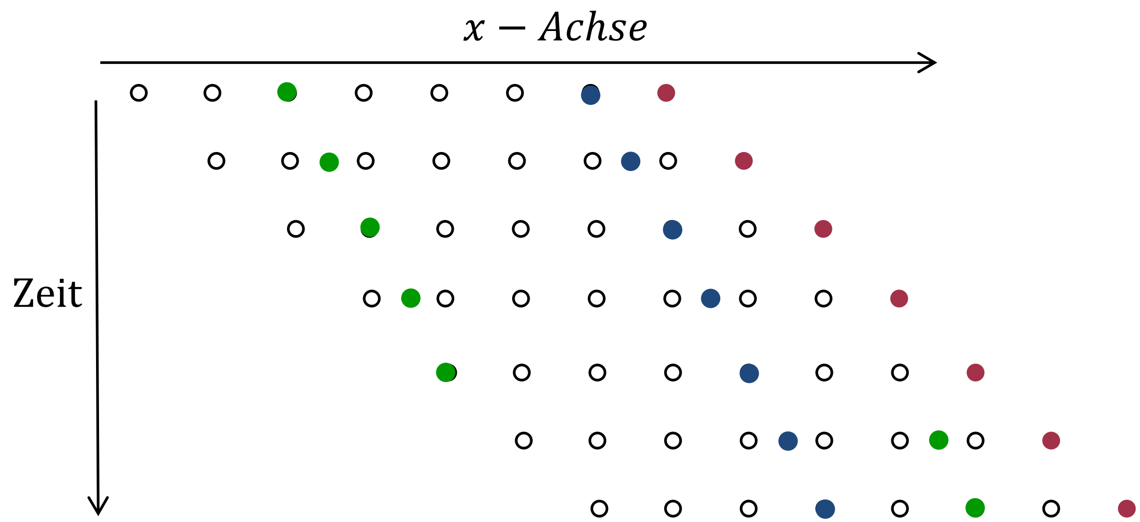
$dTPS = 1{,}5 \cdot dSC$



- ● Aktuelle Körper-Position
- ○ Alte Körper-position
- ● Trace-Particle Position

Trailparticle in dynamischer Kurve III

$$dSTP = 0{,}5 \cdot dSC$$



dSTP < 0?
Partikel schneller als Kurve. Eher nicht…

- Trace-Particle1 Position, initialVal: 0
- Aktuelle Körper-Position
- Alte Körper-position
- Trace-Particle2 Position, initialVal: 4 dSC

Daten für Algorithmus

- Globale Konstanten
  - `DELTA_T, EPSILON_SQUARED`
  - `N    // Body Count`
  - `L    // VERTICES_PER_CURVE`
  - `TPpC // TRAIL_PARTICLES_PER_CURVE`
  - `dSC  // Kurvenparameterunterschied zw. 2 Curve Vertices`
  - `dSTP //  " zw. zwei Zeitschritten eines Trail Particle`
- Body: N Stück, jeweils aus:
  - `float4: body_Pos`
  - `float4: body_V`
- Curve: N Stück, 1 zu 1 Beziehung mit Body. Besteht aus L Curve Vertices. Keine eigenen Daten
- CurveVertex: Einer Curve bzw. Body sind je L Stück zugeordnet
  - `float4: curveVertex_Pos    // Ehemalige body_Pos`
- Trail Particle: Einer Curve bzw. Body sind je TPpC Stück zugeordnet
  - `float4: trailParticle_Pos`
  - `float:  trailParticle_S    // Kurvenparameter`
  - `float4: trailParticle_Dir  // Optional, für Visualis.`

Beispiel: Datenlayout

- Für jede Variable jedes Objekttyps eigener Pointer
- Daten der Objekte liegen konsekutiv mit aufsteigendem Index im Speicher:
  - `float4* body_Pos:` $\{p_0, \quad p_1, \quad \dots, \quad \boxed{p_{N-2},} \quad p_{N-1}\}$
  - `float4* body_V:` $\{v_0, \quad v_1, \quad \dots, \quad \boxed{v_{N-2},} \quad v_{N-1}\}$
  - `float4* curveVertex_Pos:` $\{p_0, \quad p_1, \quad \dots, \quad p_{L \cdot N-2}, \quad p_{L \cdot N-1}\}$
  - `float4* trailParticle_Pos:` $\{p_0, \quad p_1, \quad \dots, \quad p_{TPpC \cdot N-1}\}$
  - `float* trailParticle_S:` $\{s_0, \quad s_1, \quad \dots, \quad s_{TPpC \cdot N-1}\}$
  - `float4* trailParticle_Dir:` $\{dir_0, \quad dir_1, \quad \dots, \quad dir_{TPpC \cdot N-1}\}$
- Beispiel: Ein „Objekt" des Typs `body` besteht aus:
  - Index: Arrayposition
  - Daten: Pos(ition) & V(elocity)
  - Markiert: Daten des "Body-Objekts" mit Index $N - 2$
- Beziehungen statisch, daher durch festes Indexschema berechenbar
- Beispiele: Finde Indices der mit…
  - …Body-Index b assoziierten Curve Vertices: $\{ \quad L \cdot b, \dots, \quad L \cdot (b + 1) - 1\}$
  - …Body-Index b assoziierten Trail Particles: $\{TPpC \cdot b, \dots, TPpC \cdot (b + 1) - 1\}$

Algorithmus

```
Initialize body_Pos, body_V, trailParticle_S
While (Iterating)
   For each (Body b, b in {0, ..., N - 1}) in parallel do
      call nBody_CalcNewV  (body_Pos, body_V, N, DELTA_T, EPSILON, b)
   End
   For each (Body b, b in {0, ..., N - 1}) in parallel do
      call nBody_CalcNewPos(body_Pos, body_V, DELTA_T, b)
   End
   For each (Body b, b in {0, ..., N - 1}) in parallel do
      call passPositionOn(body_Pos, curveVertex_Pos, L, b)
   End
   For each (TrailParticle tp, tp in {0, ..., TPpC * N - 1}) in parallel do
      call setTrailParticle(
            trailPerticle_Pos, trailParticle_S, trailParticle_Dir,
            curveVertex_Pos,
            L,
            TPpC,
            dSC
            dSTP,
            tp
      )
   End
   call visualizeBody(body_Pos)
   call visualizeTP(trailPerticle_Pos, trailParticle_Dir)
End
```