

Vorlesungsreihe „Diskrete Simulation“ (Masterkurs)

## Animation von SLX-Simulationsläufen und andere Visualisierungsaufgaben

Prof. Dr.-Ing. Thomas Wiedemann  
email: [wiedem@informatik.htw-dresden.de](mailto:wiedem@informatik.htw-dresden.de)



HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)  
Fachbereich Informatik/Mathematik

### Animation von Simulationsläufen

- Allgemeine Optionen bei der Animation
- Das Animationsprogramm PROOF
  - Überblick
  - Grundlegende Funktionalität
  - Beispiele
- Die Anbindung des SLX-Simulators an PROOF
  - direkt im Sourcecode
  - mit einem Erweiterungsmodul

## Allgemeine Optionen bei der Animation

### Allgemeine Optionen der Animationsanbindung

- als separates Add-on (meist bei Programmiersprachen (wie SLX))
  - Verfügbarkeit (Markt?), aktuell auch Spiele-Grafikengines nutzbar ?!
- direkt integriert (meist bei großen, fachspezifischen Bausteinsystemen – vgl. VL dazu)
- Bewertung:
  - bessere Flexibilität und Austauschbarkeit bei Add-Ons
  - besseres Look&Feel bei direkter Integration

### Weitere Unterscheidungsmerkmale

- Interaktionsfähigkeit
  - Steuerbarkeit der Zeit - mit Zurück-Option (meist nur bei Add-ons)
  - Steuerbarkeit der Simulation selbst (meist nur integ. Systeme), z.B. durch Druck auf 3D-Button an Maschine -> Maschinen-Stop
- Grafikperformance, max. Anzahl möglicher Grafikobjekte
- Einbindung von externen Grafiken (Bitmaps als Hintergrundbild oder 2D/3D-Grafiken von aus CAD-Daten), auch mit Bitmaps als Objektpräsentation (sogen. Sprites)

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 3

## Animation von SLX-Simulationsläufen mit PROOF

### Visualisierung von SLX-Simulationsläufen

- SLX als Simulationssprache enthält keine direkt integrierte Animation.
- jedoch kann mit beliebigen Visualisierungstools oder mit dem von der gleichen Firma bereitgestellten Animationstool PROOF eine Animation angebunden werden

### Animation mit PROOF

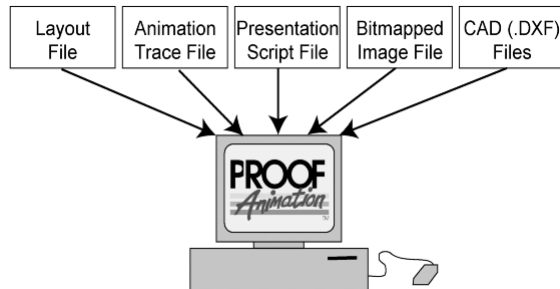
- PROOF ist ein Animations- und Visualisierungsprogramm, welches in mehreren Versionen von Wolverine Software bereitgestellt wird :
  - PROOF4 - 2D-Animation (ältere Version PROOF3)
  - PROOF 3D – neueste Version mit 3D-Unterstützung (die nachfolgenden Beschreibung gelten für diese Version, sind meist aber auch gültig für P4/P3)

### Generelle Eigenschaften aller Versionen:

- Nutzung der DirectX-Schnittstelle von Microsoft (damit nur unter Windows lauffähig und verfügbar) und orientiert auf mod. Hardw.
- PROOF 3D kann auch Multicore-Systeme sinnvoll ausnutzen (z.B. zum verteilten Handling von größeren Mengen dynamischer Objekte)
- GUI und Grundkonzepte (noch) durch historische Entw. geprägt

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 4

## Inputdaten von PROOF 3D



- **Layoutpläne** \*.lay (required) – enthält statische Szenen/Hintergrund,
  - zusätzlich **Bitmap** und **Vektor-Grafikformate (CAD)**
- **Tracefiles** \*.atf (optional) zur eigentlichen Bewegungsbeschreibung aus SLX (oder auch aus anderen Simulationstools heraus generiert) mit
  - Bewegungs- (Delta-X,Y,Z) und Eigenschaftenupdates (Farbe, Form...)
  - Stops und Viewänderungen
- **Skripte** (optional) zur Präsentationsautomatisierung

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 5

## PROOF - Arbeitsmodi

### Entwicklungsmodi (interaktiv) in der Reihenfolge der Nutzung

- Menüpunkt |->Mode-> Draw blendet Zeichenmenü ein
- Menüpunkt |->Mode-> Class erlaubt Klassendefinitionen
- Menüpunkt |->Mode-> Path erlaubt Klassendefinitionen

### Run-time und Debug-Modi

- zur Laufzeitsteuerung (siehe auch Animationssteuerung)
- zum Debuggen von Animationen (bzw. Visualisierungen)

### Spezialoptionen:

- Aufruf von SLX oder anderen Datenlieferanten aus dem Code heraus und abschnittsweise Animation

### Hilfe und Dokumentation

- Eine relativ ausführliche Beschreibung aller Befehle ist über |->Help erreichbar ! Programmcode-Befehle sind dort auch komplett dokumentiert – **in diesem Skript nur die wichtigsten !!!**

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 6

## Layoutdefinition – Grundprimitive

- Definition der Layouts über interaktive Menüsteuerung oder per Text
- Groß- und Kleinschreibung ist **signifikant bei Grafikelementen u. Objekten**
- PROOF speichert Definition auch für manuelle Änderungen ab (\*.lay)

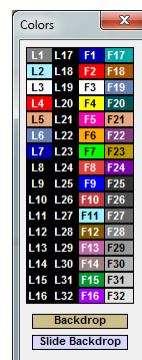
### Geometrische Grundelemente

- **Linie** – Grundform : `line x1 y1 z1 x2 y2 z2`
- Linie – volle Form : `line [ #IDnr ] [ *ExtrusionID Xx Xy Xz ]x1 y1 z1 x2 y2 z2`  
mit Zusatzpar. für PathID bzw. Extrusionsparametern -> bei allen weiteren Primitiven sind diese Zusatzparameter analog vorhanden -> siehe Hilfe !!)
- **Kreis** – Grundform: `arc [filled] radius x y z alpha omega [ normal nx ny nz]`  
wobei alpha und omega die Anfangs- und Endwinkel angeben
- **Polyline** – Grundform (zum Zeichnen von größeren Konturen):  
`polyline`  
`#1 x y z` (einzelne Punkte)  
`...`  
`#N x y z`  
`end | close` (close verbindet Anfang und Ende)

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 7

## Layoutdefinition – weitere Grundelemente

- **Fontdefinition** (mit 2. vordef. Fonts: 0=fixed Courier 1=Arial TT)  
`define font 2 "Times New Roman"`
- **statischer Text** – Grundform :  
`text fontnum height [ billboard ] [ justification ] x y z [ textstring ]`  
Bsp.: `Text 1 2 LJ -60 32 0 Static Text`
- **Dynamischer Text** (veränderbar während Animation) – Grundform :  
`message messagename fontnum height line [textprototype]`  
Bsp.: `Message DynamicText1 1 2 LJ -60 25 0 Text1111`
- **Farbzuordnung mit color colorID**  
Farbcode entsp. rechter Dialogbox mit F\* = Foreground und L\*-Layoutcolor (mit geringerer Priorität), Backcolor geringste Prio.  
Bsp. für Blau als Farbe des Elements : `color F1`  
wobei die Farben den F/L- Codes mit der **Farbdefinition**  
`define color ColorID redvalue greenvalue bluevalue`  
Bsp.: `ColorID F2 1.0 0.0 0.0`  
frei zugeordnet werden können (1.0 = volle RGB-Farbe)



Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 8

## Layoutdefinition – Klassendefinition für die Animation

Für die spätere Animation müssen Klassen definiert werden:

- **Klassendefinition:**

```
define class classname [directional] [clearance fore aft ] [speed s]
...
    geometry (line, arc, text, fill, message commands)
...
end
```

- Bsp. zur Definition der Klasse Kiste4 :

```
Define Class Kiste4 Camera 0 0.0 86.0 86.6 1.5 1.5 0 Clearance 3 3 Speed 1
Line 0 10 0 10 10 0
Arc Filled 3 5 5 0 0 360 normal 0 0 1
End
```

- verwendbar sind die angegebenen Geometrie-Primitiven
- Die Zusatzoptionen definieren bereits das Verhalten auf Pfaden (Abstand zum Vordermann etc.), hier
  - Default-Bewegungsgeschwindigkeit = 1
  - Clearance (Abstand auf Pfaden = 3)

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 9

## Layout – Pfaddefinition für die Animation

Für die spätere Animation können Pfade für komplizierte Bewegungsmuster definiert werden:

- **Klassendefinition:**

```
define path pathname [accumulating| circular] [ label x y z ] [ lag delay ]
    [ leapok ] [ [ speed s ] ] [ [ time t ] ]
    segment #source x1 y1 z1 x2 y2 z2 length [ cw | ccw ]
...
    segment #source x1 y1 z1 x2 y2 z2 length [ cw | ccw ]
end
```

- Bsp. zur Definition des Pfades Kistenpfad4 :

```
Define Path Kistenpfad1 Accumulating Speed 1
Segment #1 -50 0 0 -10 0 0 40
Segment #2 -10 0 0 0 -10 0 14.142136
end
```

- **Die Segmente beziehen sich auf existierende Geometrie! Vorsicht bei Veränderungen dieser → kann PATH zerstören !!!**
- Die Zusatzoptionen definieren wieder das Verhalten , hier
  - Default-Bewegungsgeschwindigkeit = 1
  - Clearance (Abstand auf Pfaden = 3)

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 10

## Layout – mit 3D-Welten

Eine 3D-Animation kann über zusätzliche 3D-Operationen definiert werden:

### Zusatzangaben in den Grafik-Primitiven

- Angaben zur Extrusion im Grafikelement

`arc [ #IDnumber ] [ *ExtrusionID Xx Xy Xz ] [ filled ] radius x y z alpha`

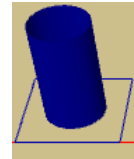
Bsp. zur Extrusion eines Kreises innerhalb der Objektdefinition :

`Arc *100 0 0 1 Filled 3 5 5 0 0 360 normal 0 0 1`

- + globale Definition der Extrusionsparameter

`extrusion ID height cone_height wedge_angle x y z`

`define extrusion *100 10 0 0 0 0 1` (generiert Zylinder)



- Über Zusatzoptionen können
  - geneigte Flächen (-> Kegel oder Pyramiden)
  - Kugeln (Drehung eines Kreises im Raum)
  - und Spiralen (Drehung eines Kreises im Raum mit Offset zur Drehachse) definiert werden.

## Animation - Tracefile – Basisdefinitionen

Der Tracefile enthält die Anweisungen zur Steuerung der Animation.

### 1. Anlegen und Platzieren von Objekten

Anlage und Platzierung von im Layout vorhandenen Klasseninstanzen:

`create Kiste4 1` (mit create classname objectID)

`place 1 at 3.0 0.0 0.0` (place objectID at x y z )

Die weitere Ansprache der Objekte erfolge immer über die objectID.

### 2. Definition einer Bewegungsinformationen

`move 1 30 20.0 0.0 0.0` (move objectID duration xdest ydest zdest [ relative ])

### 3. Definition des Zeitfortschritts

entweder durch absolute Zeitangaben mit

`time 100`

oder relativ (sinnvoll für Tests oder Codebausteine) mit

`dt 10` (= delta t) **Achtung: Alle Bewegungen und**

**Transformationen werden erst bei einem Zeitfortschritt ausgeführt !!**

### 4. Vernichten von Objekten

`destroy Kiste4` (mit destroy objectID) bjectID)

## Animation - Tracefile – weitere Definitionen

### Weitere Transformationen

**Drehen** (auch parallel zu anderen Transformationen):

**yaw 1 speed 2** (dreht um Z-Achse)

**roll 1 speed 2** (dreht um X-Achse)

**pitch 1 speed 2** (dreht um Y-Achse)

**Skalieren** (ebenfalls parallel)

**scale 1 to 2 time 20.0** (scale objectID [ to ] factor [ time duration] | [ speed s ] )

### Ausgabe von Statusinformationen und Statistiken

Schreiben von Textmeldungen (setzt entsprechende Layout Message voraus):

**write Status Starting ...** (write **messagename** textstring)

### Steuerung der Animation

- über die Tasten in entspr. Reihenfolge



- Run (oder Continue Run)
- Stop / Reset
- Einzelevent pro Zeile / Zeitstep einzeln (nur im Debug-Modus |->Mode)
- Jump in Time/Jump back
- Set Speed, Suche Objekt (in größeren Modellen)

Masterkurs „ Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 13

## Tracefile – Path-bezogene Animation

### Anlegen und Plazieren von Objekten auf Pfaden

(Voraussetzung : Define PATH – Layout-File) :

**create classname objectid** (Objektgenerierung wie vorher)

**place objectID on pathname [ squeeze ] [ at offset ] [ at end ]  
[ before objectID2 ] [ after objectID2 ]**

(mit den optionalen Parametern lässt sich die Position genauer bestimmen)

Bsp.: **create Kiste4 1**

**place 1 on Kistenfad0** (Pfad muss im Layout vorhanden sein)

**time 100** (erst)

### Anmerkungen:

- Die [squeeze]-Option veranlasst ein Zusammenschieben mehrerer Objekte.
- Analog zu Objekten können auch Kameras auf einen Path gesetzt werden – dies ermöglicht Kamerafahrten
- Die Groß- und Kleinschreibung ist **NICHT signifikant bei den Tracebefehlen !**

Masterkurs „ Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 14

## Anbindung von PROOF an SLX

Die An- oder Einbindung von PROOF an SLX ist wie folgt möglich:

### Option A. - direkt im Sourcecode

- Generierung der Trace-Statements mit Standard-Ausgabebefehlen mit
  - Ausgabe der Generierungs- und Bewegungsbefehle
  - Ausgabe der Time-Codes

Bsp.:

```
write file=atf (obid, "Path1", 2) "place _ on _ speed _.___\n";  
write file=atf (time ) "time _.___\n";
```

### • Zusatzaufgaben:

- ggf. unnötige Timestamps unterdrücken (bei Ausgaben zur gleichen Simulationszeit)
- ggf. abweichende Laufzeiten zw. Simulation und Path-Animation beachten und überschreiben (rausnehmen aus PATH...)
- Vernichtung am Ende

## Anbindung von PROOF an SLX II

Die An- oder Einbindung von PROOF an SLX ist wie folgt möglich:

### Option B. - durch Verwendung der Erweiterungsbibliothek PROOF.slx

(ist gleichzeitig auch ein Beispiel zur Erweiterung des SLX-Compilers)

```
PA_Set path "Bob:Base:LeftTrack:TrackPath" speed 0;  
+ {  
+ if (AnimationActive)  
+ {  
+ write_time_stamp();  
+ write file=atf ("Path1", 0) "set path _ speed _.___\n";  
+ }
```

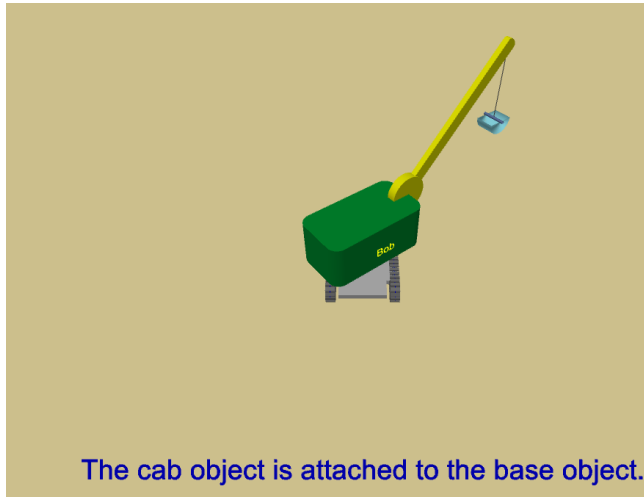
- PA\_Set und weitere Befehle werden definiert durch `import "Proof3d.slx"`
- Dieser Modul erweitert SLX um PROOF-Animationsbefehle, welcher bei der Übersetzung in den SLX-Code (siehe +Zeilen) eingefügt werden
- Komplexes Beispiel - siehe crane.slx

### **Vorteile:**

- Standardtests (wie auch " AnimationActive) leicht versteckbar
- besser lesbarer Code (ggf. aber auch mit Unterprogrammib.)



## Beispiel zur Animationskopplung von SLX



Grafisch relativ aufwändige Animation im Modell „cran.slx“  
mit einzeln ansteuerbaren Modellkomponenten

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 17

## Zusammenfassung zur Animation mit PROOF und SLX

### Allgemein

- Die PROOF-SLX Kopplung kann als ein Beispiel für eine abgesetzte Animationsumgebung gesehen werden, wie diese bei den ersten Generationen von Simulatoren sehr typisch war.

### Vorteile

- PROOF lässt sich sehr gut von der Simulation aus ansteuern
- Installationsaufwand (DirectX) und Laufzeitverhalten sind sehr gut
- Der Aufwand zur Visualisierung kann von einfachen Drahtmodellen bis hin zu schattierten 3D-Modellen beliebig angepasst werden.

**Gewisse Nachteile** liegen in der Natur der Quelltextprogrammierung

- **JEDE** Animationsbewegung **MUSS** im Simulations Quellcode eingebaut werden!
- Bei größeren Modellen schnell unübersichtlich – Ausweg durch automatisierte Codegenerierung analog zu Bausteinsystemen (vgl. VL Bausteinsysteme)

### Fazit:

- bei Abstrichen am Modellierungskomfort (und zeitgemässen 3D-Ansichten) sehr gute Kombination

Masterkurs „Diskrete Simulation“ - Animation mit PROOF - Prof. T.Wiedemann - HTW Dresden - Folie 18