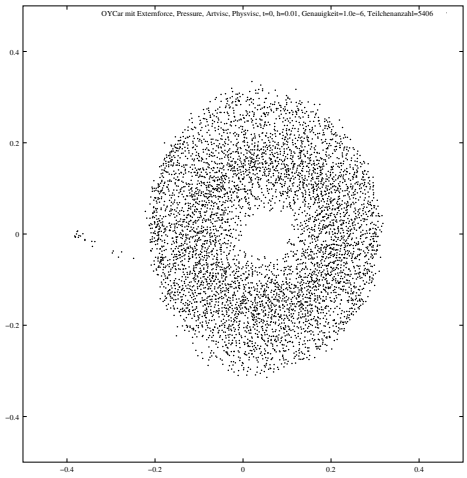
Implementierung von SPH-Methoden in eine objektorientierte Klassenbibliothek mit Namen „SPH++“



Diplomarbeit von Andreas Nagel
Betreuer: Stefan Hüttemann

Vortragsfolien für das Diplomanden- und Doktorandenseminar am 7.7.2000
- Prof. Dr. W. Rosenstiel -

Vortragspunkte:

1. **Die bisherige Entwicklung von SPH++:**

   (a) Die Etablierungsphase (inception): Was soll SPH++ können ?

   (b) Die Entwurfsphase (elaboration): Wie soll die Architektur von SPH++ aussehen ?

   (c) Die Konstruktionsphase (construction): Beschreibung der Implementation von SPH++.

   (d) Die Auslieferungs- und Testphase (transition): Diskussion der Ergebnisse.

2. **Wie sieht die (mögliche) Zukunft von SPH++ aus?**

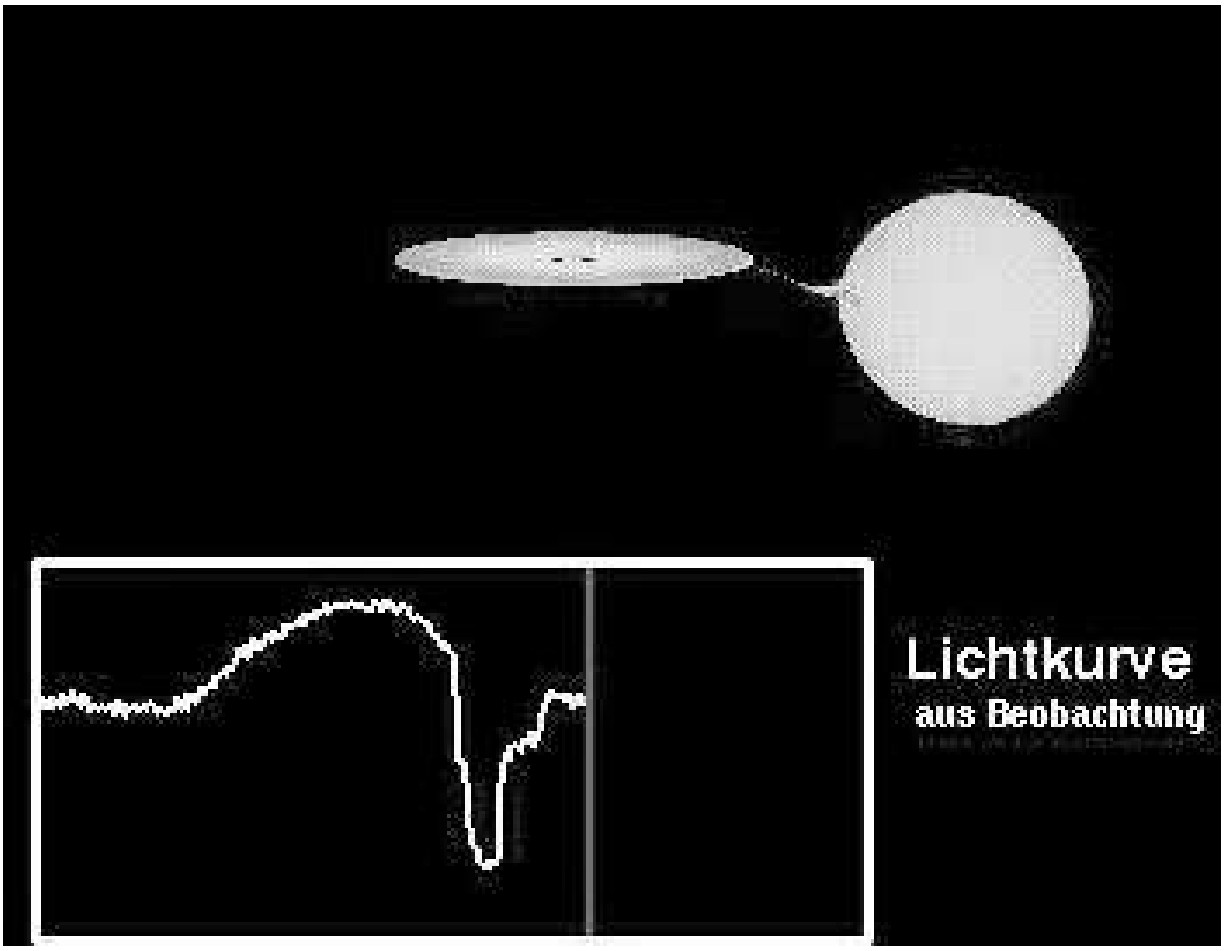
   (a) Verwendung eines Software–Prozeß.

   (b) Durchführung eines neuen Entwicklungszykluses.

1a: Die Etablierungsphase (inception): Was soll SPH++ können ?

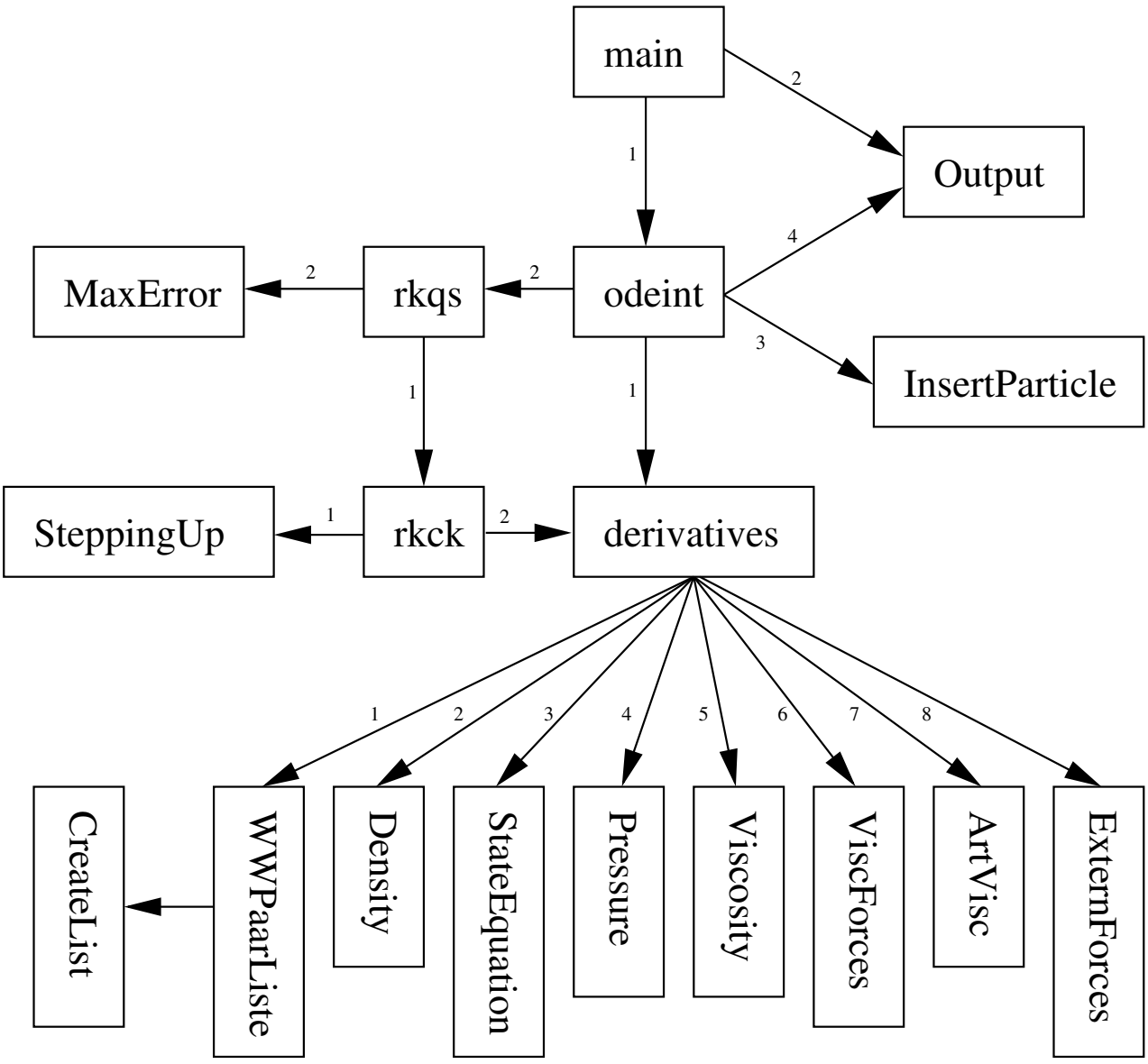
- Die Physik der Akkretionsscheibe ⇒Bild
- Es soll die Funktionalität des **C-Programm** von Stefan Hüttemann und Stefan Kunze (Version 1.1) haben. ⇒Bild
- Eigenschaften der OO-Programmierung mit Hilfe von **Design-Pattern**, wie leichte Erweiterbarkeit und gute Wartbarkeit.
- Es soll auf **parallelen Maschinen** laufen (Cray T3E, Workstation-Cluster, ...) mit Hilfe von dts bzw. TPO++.
- Es soll eine **graphische Oberfläche** erhalten mit Hilfe von Java, Qt oder ähnlichem. ⇒Bild
- **Arbeitsumgebung**: Compiler: gcc.2.95.2; OS:Unix (Solaris); Designtool: Together3.2; UML

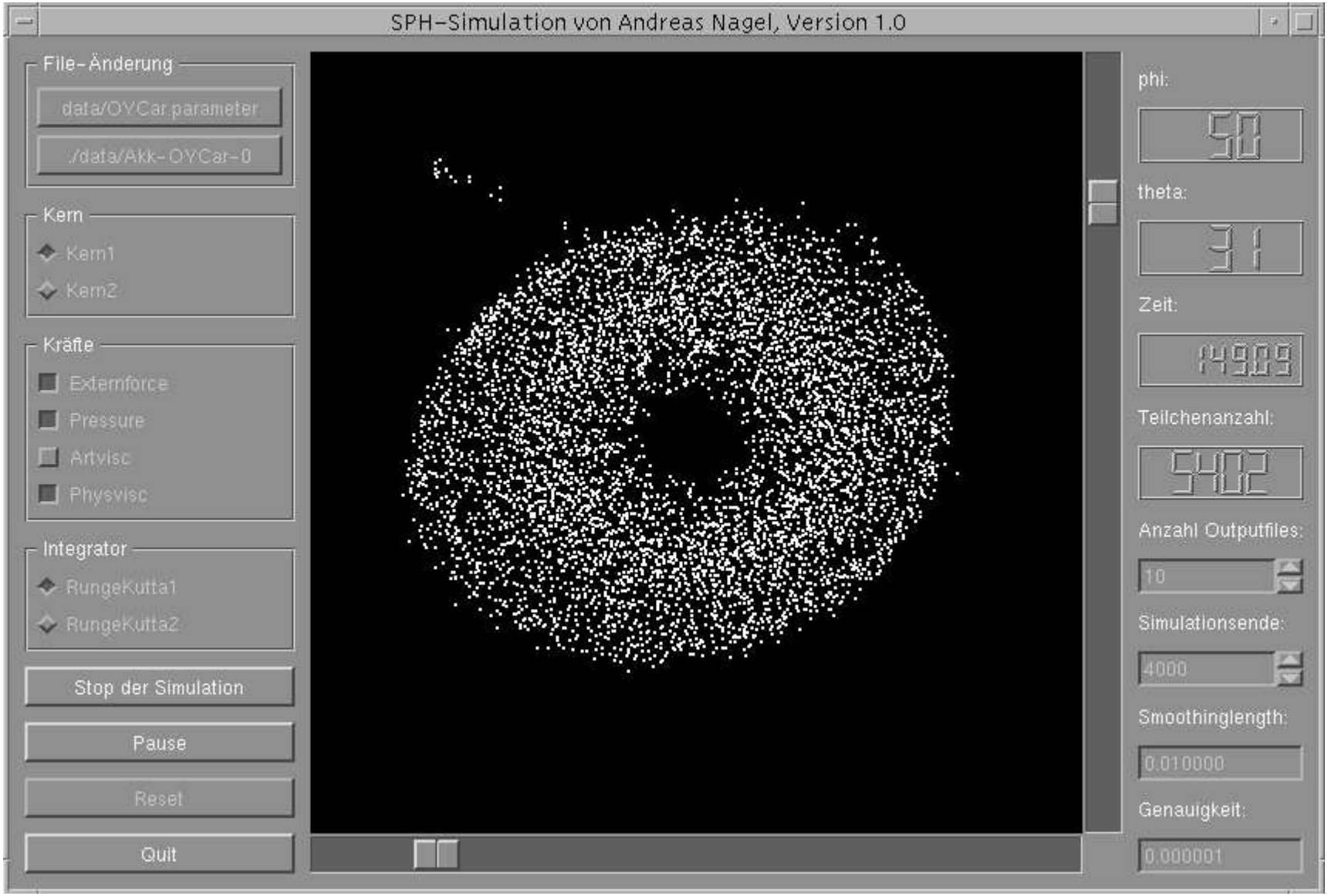
Die Physik der Akkretionsscheibe:



Dieses Bild zeigt die Zwergnova OY Carinae. Dieses System besteht aus einem Weissen Zwerg und einem kühlen, roten Hauptreihenstern. Der Hauptreihenstern verliert Masse an den Weißen Zwerg, es bildet sich eine Akkretionsscheibe mit einem Hot Spot, wo der überfließende Gasstrom auf den Rand der Scheibe trifft.

Struktur des C-Programms:
(Pfeile sind Funktionsaufrufe)




SPH-Simulation von Andreas Nagel, Version 1.0
File-Änderung
data/OYCar.parameter
./data/Akk-OYCar-0
Kern
Kern1
Kern2
Kräfte
Externforce
Pressure
Artvisc
Physvisc
Integrator
RungeKutta1
RungeKutta2
Stop der Simulation
Pause
Reset
Quit
phi:
50
theta:
31
Zeit:
149.09
Teilchenanzahl:
5402
Anzahl Outputfiles:
10
Simulationsende:
4000
Smoothinglength:
0.010000
Genauigkeit:
0.000001

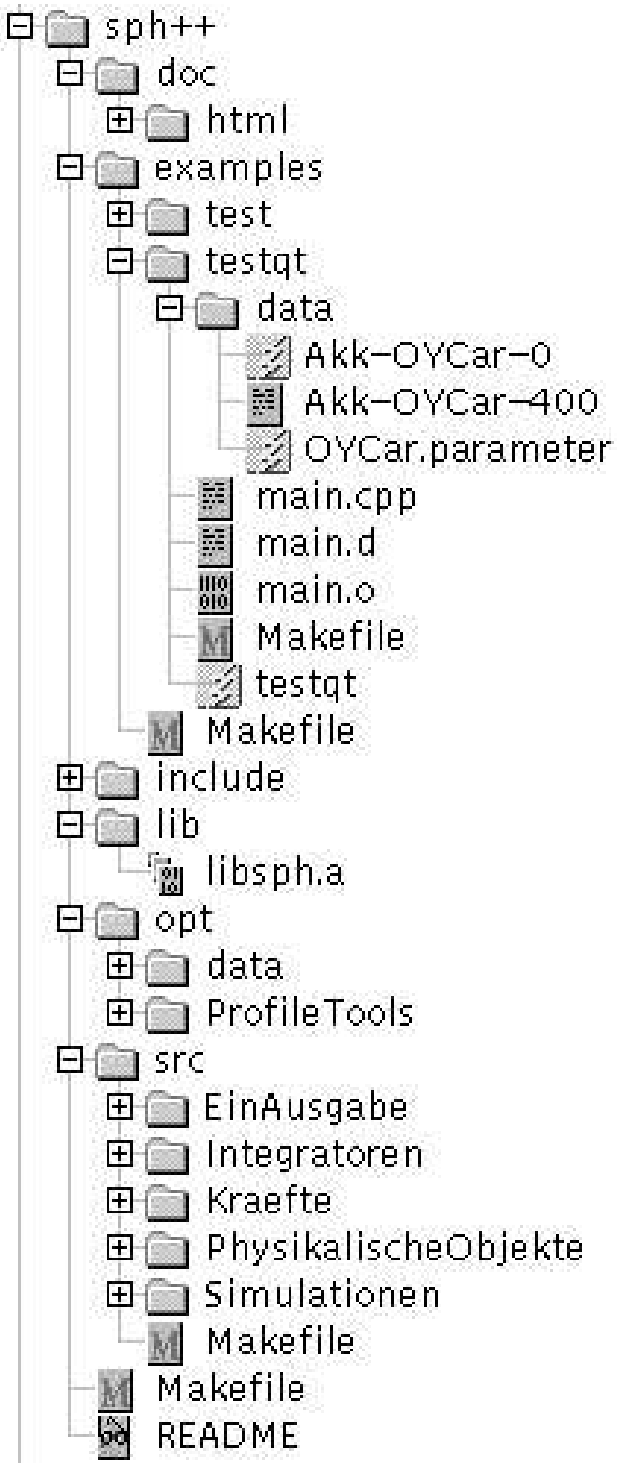
1b: Die Entwurfsphase (elaboration): Wie soll die Architektur von SPH++ aussehen ?

SPH++ ist mit **Together** in **UML** designed. Es wurden nur Klassendiagramme verwendet. (Bem: Die Hardwareanforderungen von Together sind sehr hoch !)

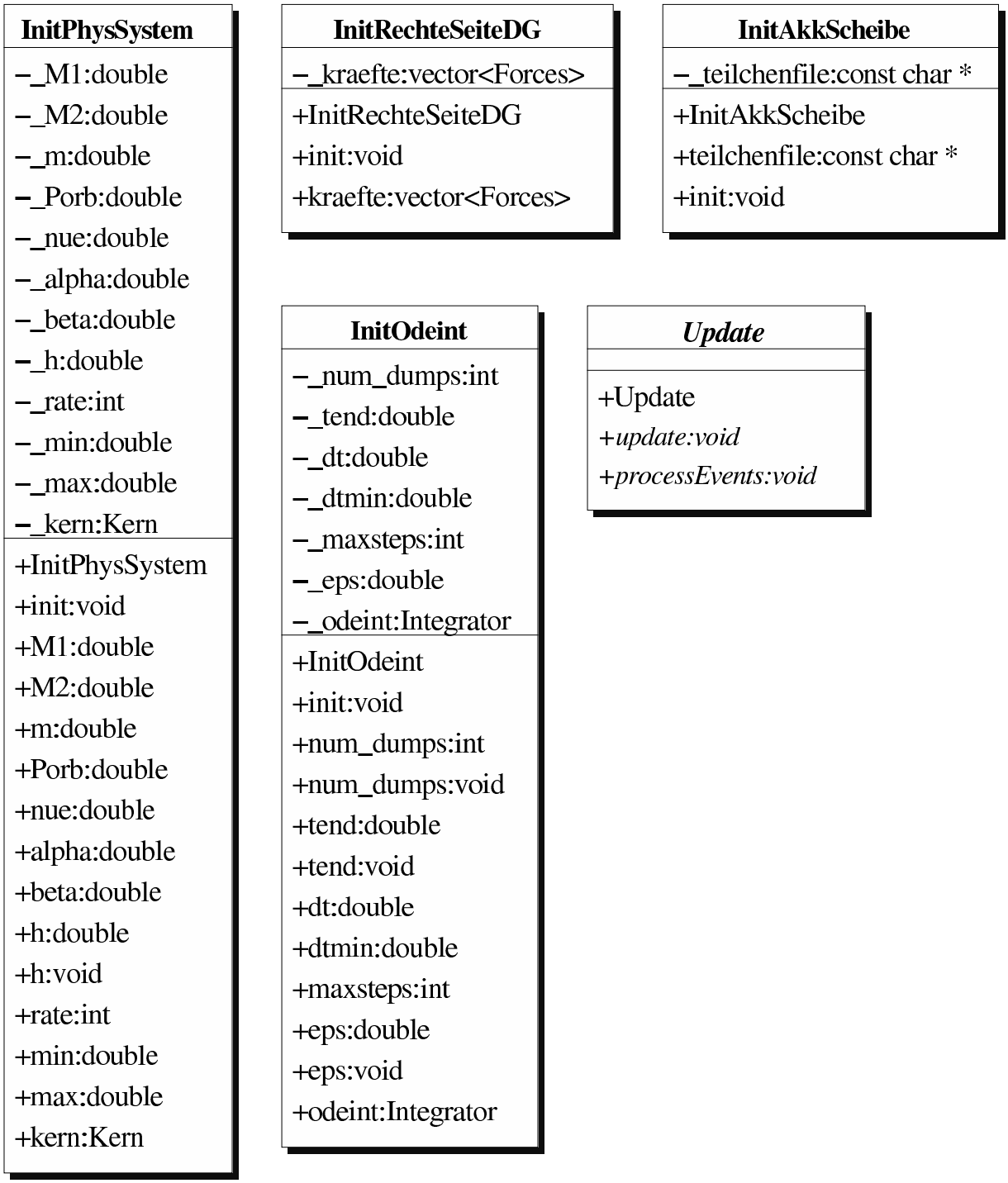
SPH++ wurde in 5 Pakete zerlegt:

- **EinAusgabe** ⇒Bild
- **PhysikalischeObjekte** ⇒Bild
- **Integratoren** (Strategy-Pattern)⇒Bild
- **Das Kraefte** (Composite-Pattern)⇒Bild
- **Simulationen** ⇒Bild

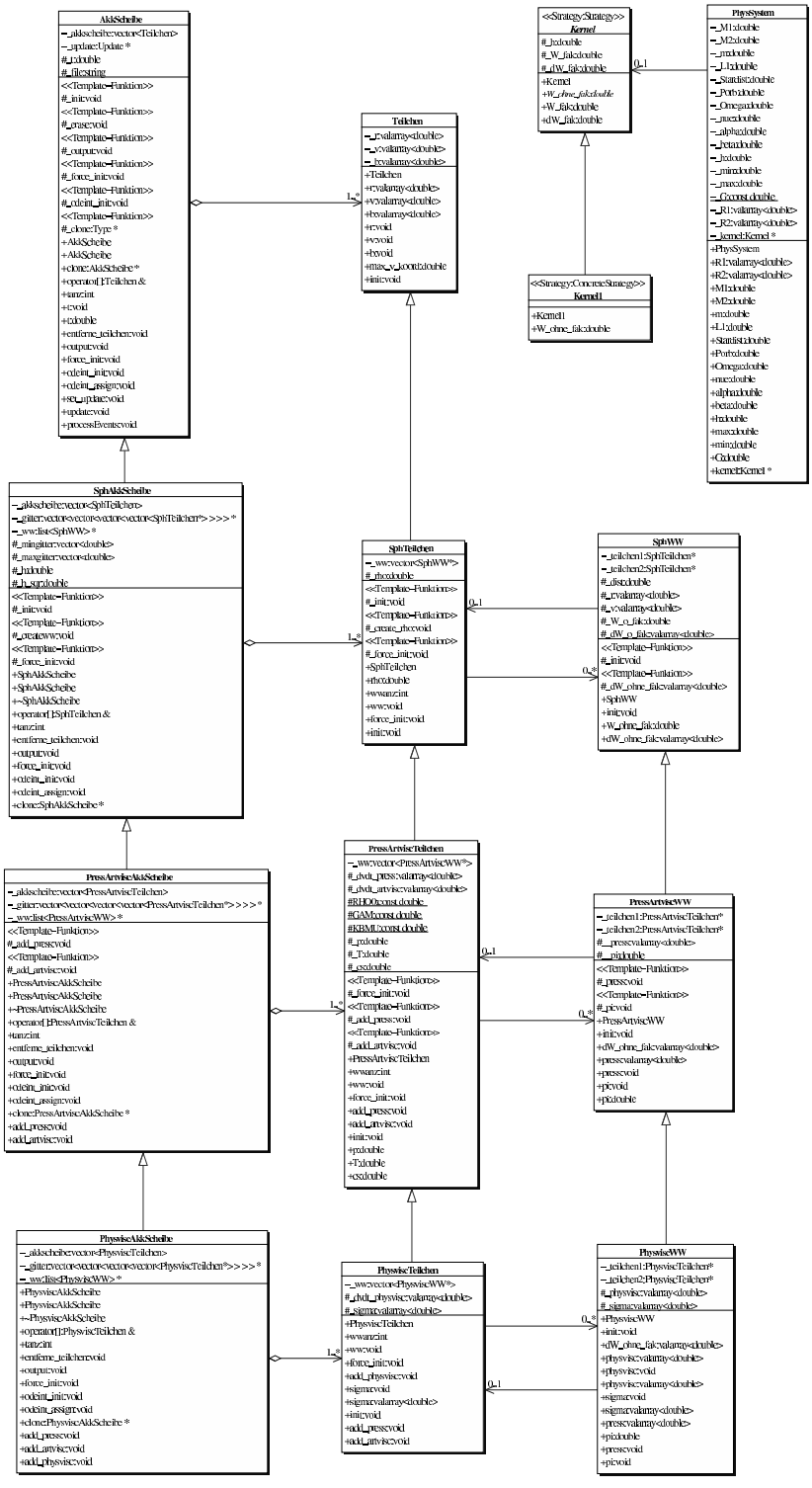
Die Verzeichnisstruktur des Programms:



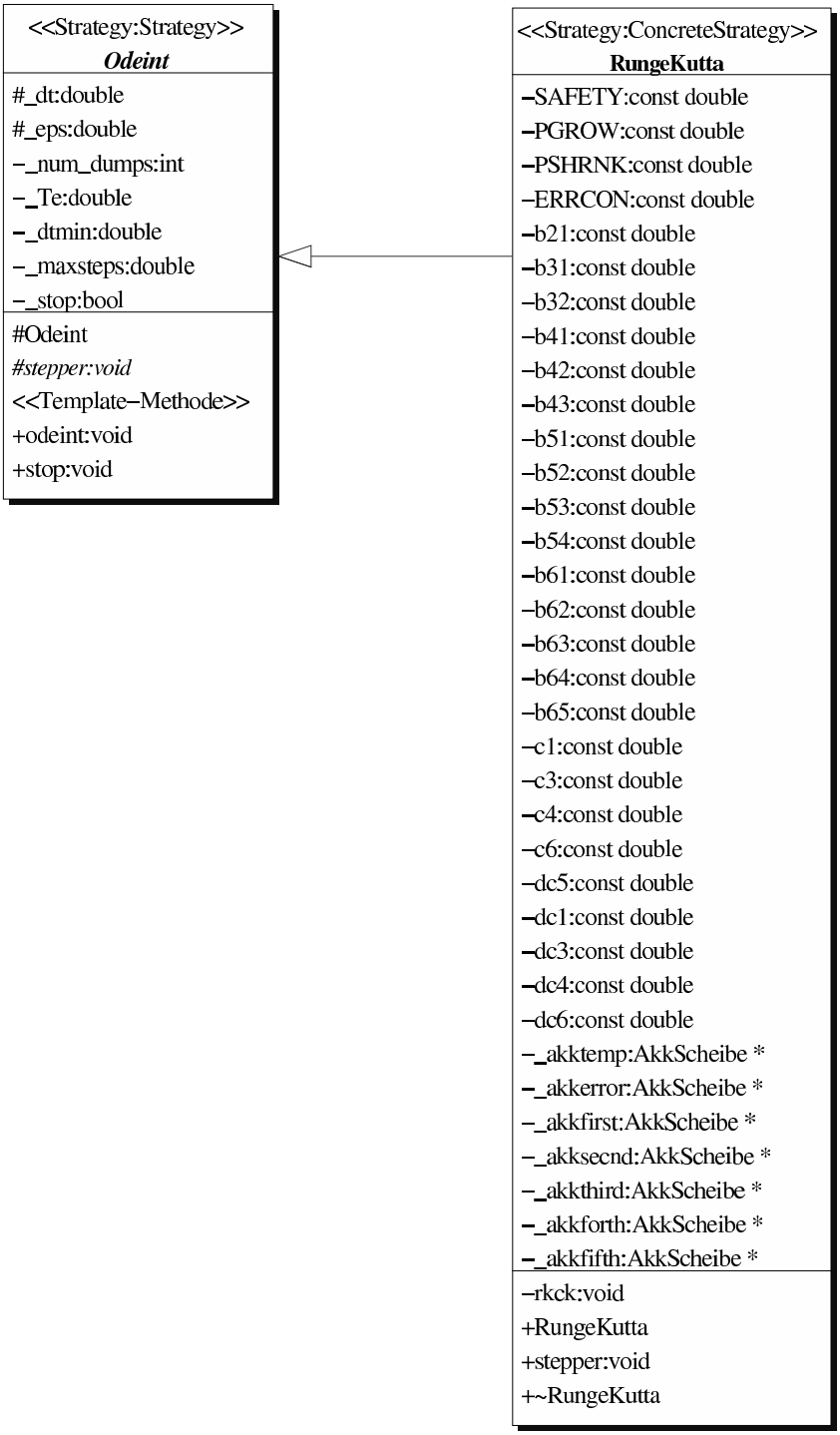
Das EinAusgabe-Paket:



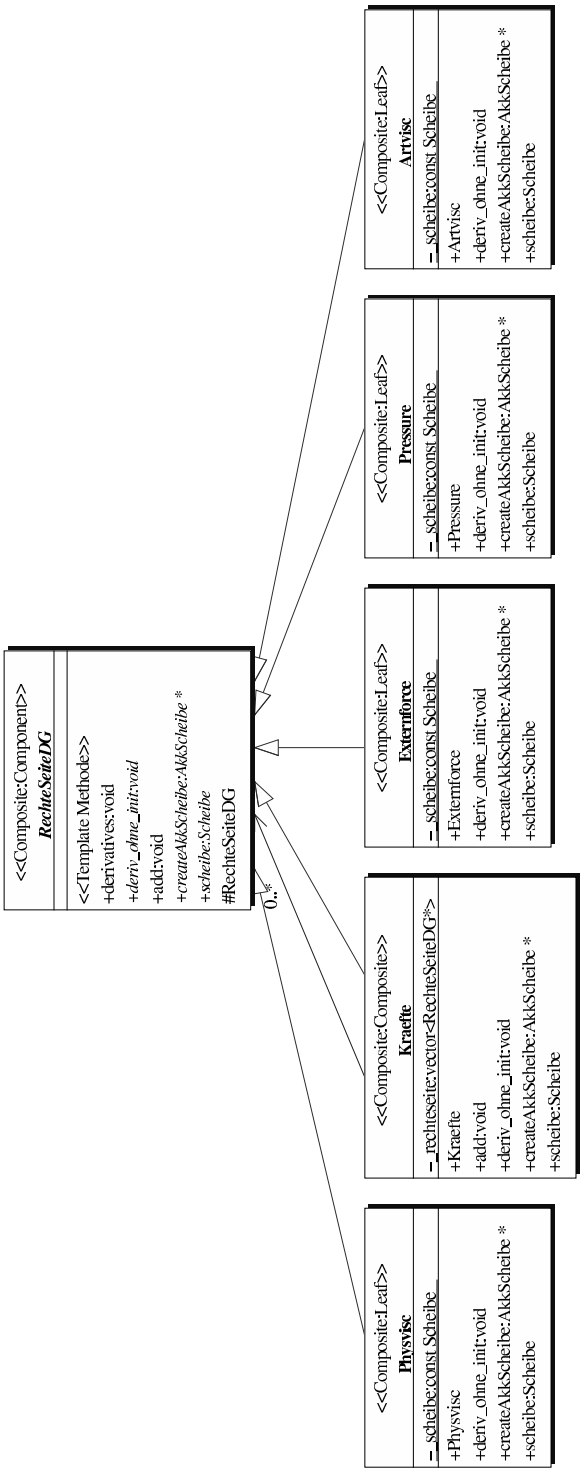
Das PhysikalischeObjekte-Paket:



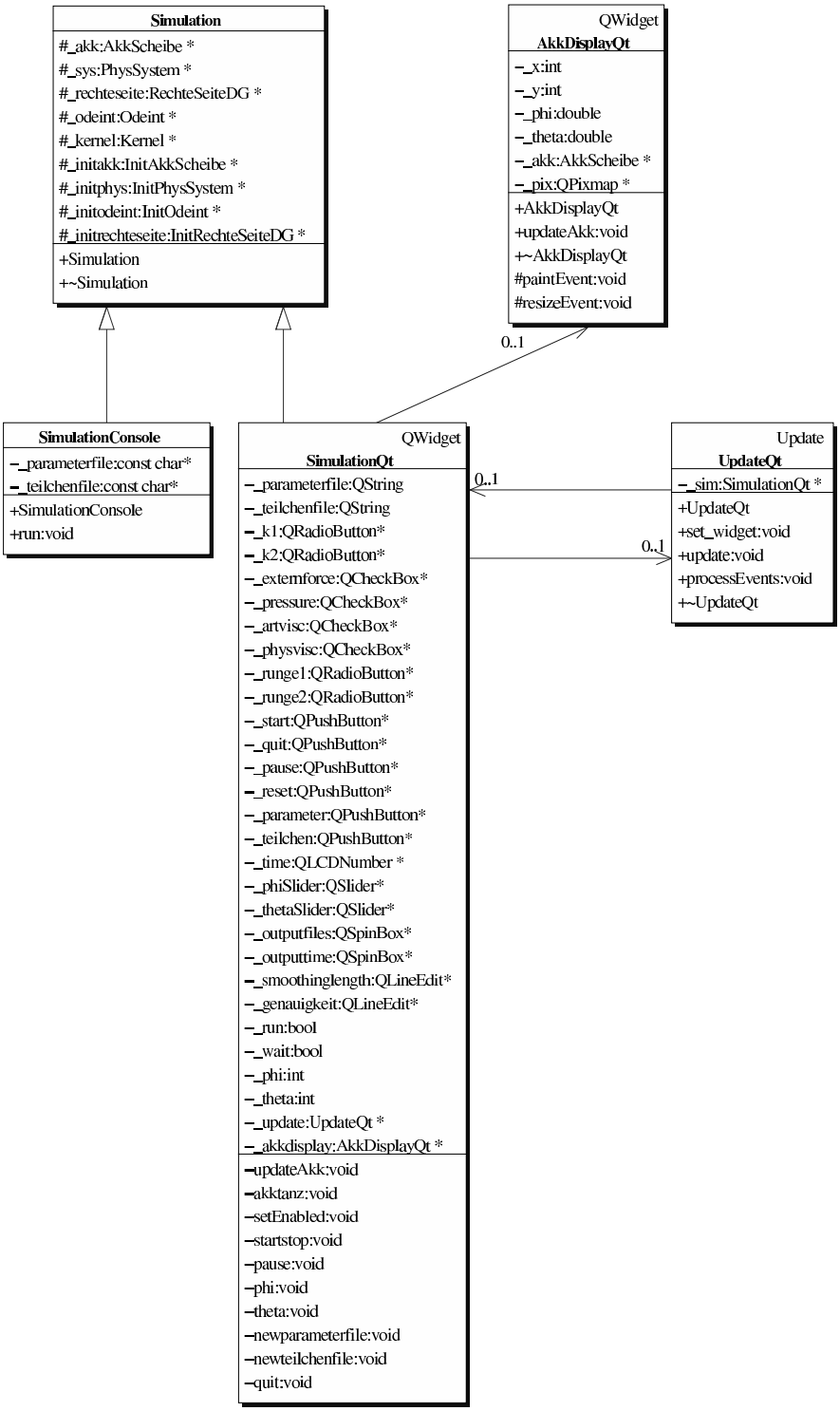
Das Integratoren-Paket:



Das Kräfte-Paket:



Das Simulationspaket:



1c: Die Konstruktionsphase (construction): Beschreibung der Implementation von SPH++.

Die Implementierung lief parallel zur Designphase. Dabei gab es folgende größere Probleme:

- valarray-Problem (gcc-2.95.2, Intel).
- Template-Funktionen (zu viele, CompilerOption:-ftemplate-depth-30).
- Nichtbeachtung der Art des Wachstums des STL-Vektors.
- Probleme mit dem Debugger gdb 4.18 .

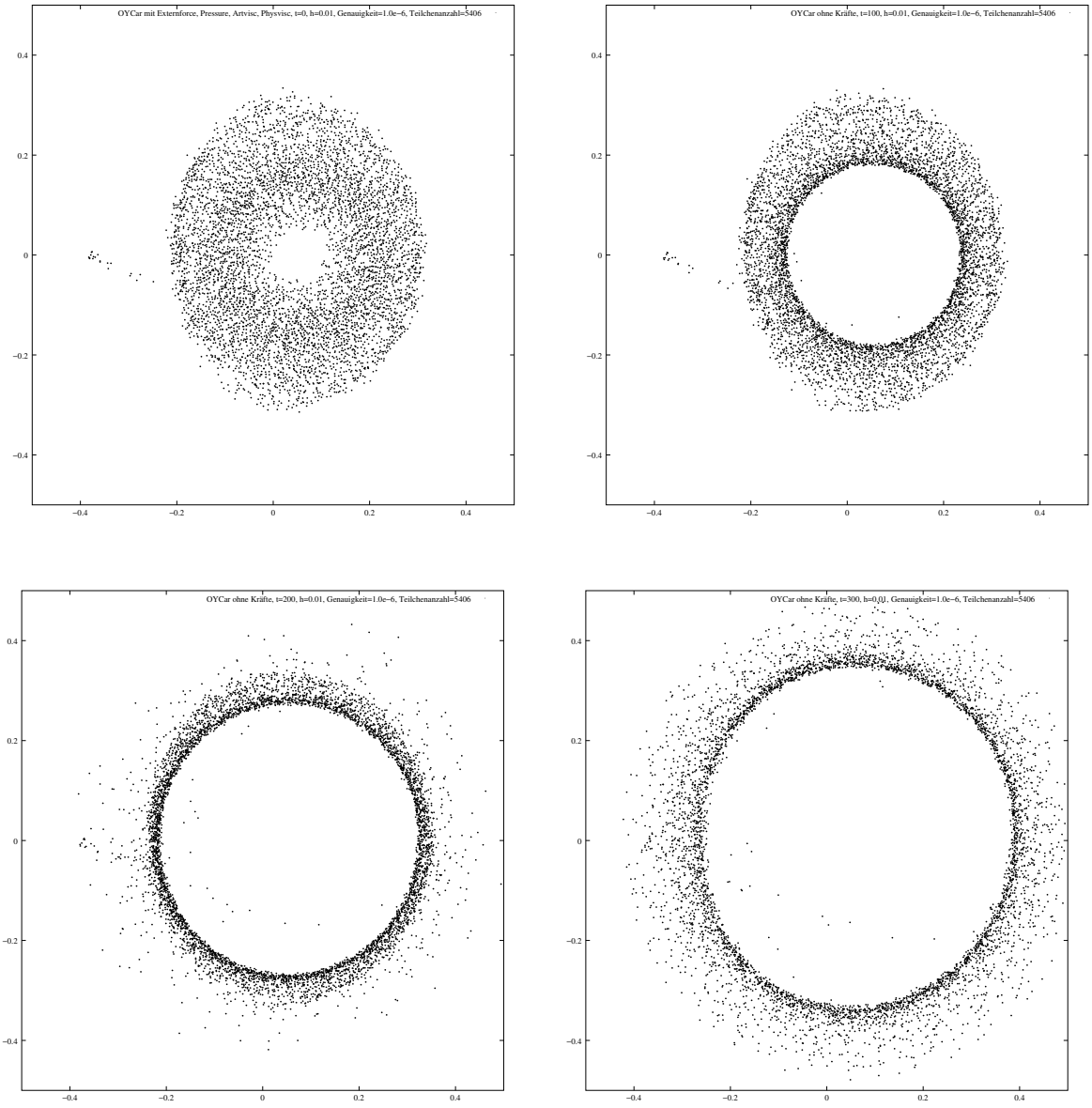
1d: Die Auslieferungs- und Testphase (transition): Diskussion der Ergebnisse.

SPH++ ist bis jetzt noch **kein fertiges** und stabiles Produkt. Da das Design einige Schwachstellen besitzt, ist es besser, erst diese Schwachstellen im nächsten Zyklus zu beheben und erst dann dieser Phase mehr Zeit zu widmen.
Trotzdem wurden einige Test mit folgenden Ergebnissen gemacht:

- Die Externe Kraft scheint richtig implementiert. Die anderen Kräfte wurden noch nicht getestet. ⇒Bild

- Die Geschwindigkeit von SPH++ ist ca. 50% langsamer als das C-Programm.

Die mit dem C++-Programm berechnete Akkretionsscheiben ohne Krafteinwirkung:



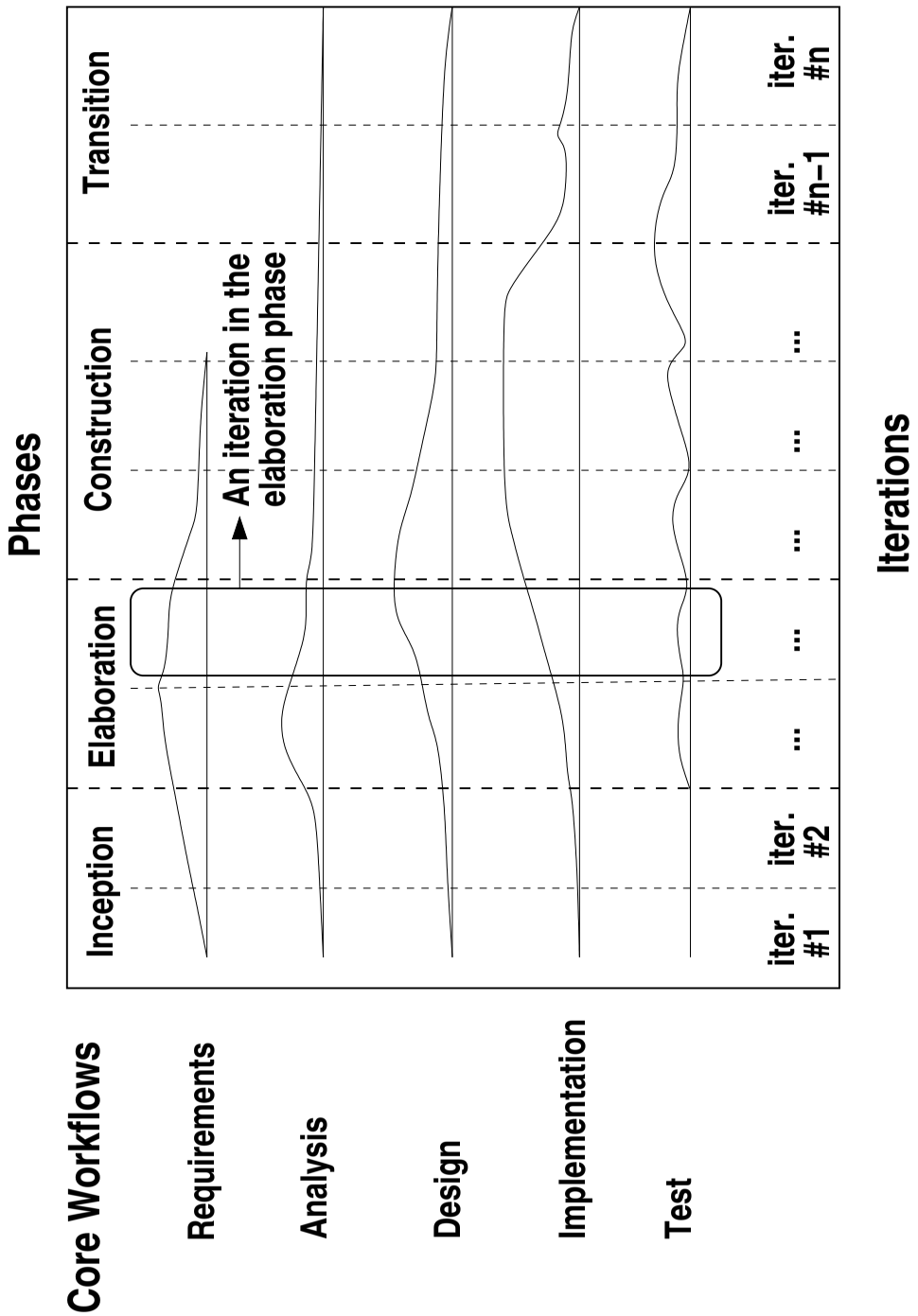
2a: Verwendung eines Software-Prozeß:

Je größer und komplexer eine Software ist und je mehr Personen bei deren Herstellung beteiligt sind, umso wichtiger wird die Verwendung eines **Softwareprozesses** (vgl. Hausbau).

Die wichtigsten sind:

- **Feature-Driven Development Prozeß** der Firma **Together**.
- **Unified Process** der Firma **Rational**.
- **Catalysis**
- **OPEN-Process**, siehe www.open.org.au
- **Object-Oriented Software Process (OOSP)**
- **Extreme Programming (XP)**

Die 5 Arbeitsflüsse des **Unified Process** werden über die 4 Phasen aufgetragen:



2b: Durchführung eines neuen Entwicklungszykluses:

Folgendes gibt es noch zu tun:

- **Etablierungsphase:** Zusammen mit den Physikern müssen die Anforderungen an das Programm bestimmt werden
  (Parallelisierung, ...)
  (Benutzung von Use Case-, Interaction, ... -Diagrammen)

- **Entwurfsphase:** Neues Design: Statt Akkretionsscheibe $\Rightarrow$ Simulationssgebiet (keine Templatefunktionen).

- **Implementierungsphase:** Verwendung eines neueren Compilers (gcc-3.x, libstdc++-v3 mit gdb-5.0).
  Verwendung von externen Paketen
  (Blitz++, LAPACK++,...)

Verweis auf Diplomarbeit von Swen Ganzenmüller und Frank Heuser.