GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

„Nun so wäre denn endlich die Untersuchung in die Geheimnisse der Mathematik gehüllt, damit doch ja niemand so leicht wage, sich diesem Heiligtum zu nähern.“

Johann Wolfgang von Goethe (1749 - 1832)

Adaptive Semi-Lagrange-Advektion mit radialen Basisfunktionen und Moving Least Squares

Diplomarbeit

vorgelegt von
Christian Menz
aus
Kassel

angefertigt
im Institut für Numerische und Angewandte Mathematik
der Georg-August-Universität Göttingen

betreut von
Professor Dr. Holger Wendland

2007

Inhalt

1 Einleitung **1**

1.1 Problembeschreibung 1

1.2 Mathematik 2

1.2.1 Partikelbewegung 2

1.2.2 Kurven 4

1.2.3 Erhaltungssätze 4

1.3 Zielsetzung 6

1.4 Motivation 7

1.5 Anfangsbedingungen 7

1.6 Testreihen 9

2 Methoden **10**

2.1 Upstreampunkt 10

2.2 Bestimmung von Nachbarschaften / KD-Tree 11

2.3 Interpolation mit radialen Basisfunktionen 13

2.4 Moving Least Squares Approximation 16

2.5 Semi-Lagrange-Advektion 19

2.6 Adaption 20

2.6.1 Voronoipunkte / Delaunay-Triangulierung 21

2.6.2 Praktische Anwendung 24

2.7 Clipping 24

3 Numerische Ergebnisse **26**

3.1 Anfangsbedingungen 26

3.1.1 Geschwindigkeitsfeld 26

3.2 Zeitschrittweite 27

3.2.1 Knotenverteilung 27

3.2.2 Konzentrationen 28

3.3 Clipping 29

3.4 Advektion 32

3.4.1 Advektion mit MLS 32
3.4.2 Advektion mit RBF 40
3.5 Adaption 50
3.5.1 Adaption mit MLS 52
3.5.2 Adaption mit RBF - Wendland-Funktion 73
3.5.3 Adaption mit RBF - Thin Plate Spline 78
3.6 Advektion & Adaption 97
3.6.1 Advektion & Adaption mit MLS 100
3.6.2 Advektion & Adaption mit RBF 101

4 Fazit & Ausblick **133**

A Variablen & Bezeichner **138**

B Algorithmen **140**

C Programm **147**
C.1 Klassenstruktur 147
C.2 Klassen 147
C.3 Parameter 166

D Einbindung & Darstellung **173**
D.1 slm_smoothing 173
D.2 ply2vtk 174
D.3 Dateiformat *ply* 174
D.4 Dateiformat *vtk* 175

1 Einleitung

„Man kann beim Studium der Wahrheit drei Hauptziele haben: Einmal, sie zu entdecken, wenn man sie sucht; dann: Sie zu beweisen, wenn man sie besitzt; und zum letzten: Sie vom Falschen zu unterscheiden, wenn man sie prüft.“

Blaise Pascal (1623 - 1662)

1.1 Problembeschreibung

In der Praxis stellt sich oft das Problem, die Dynamik eines Systems mathematisch zu erfassen und zu berechnen.
In einigen Fällen ist es nötig, die Berechnungen besonders effizient durchzuführen, um z.B. in der Meteorologie Vorhersagen über klimatische Veränderungen treffen zu können oder die Strömungen an Flugzeugflügeln zu simulieren. Es existieren zahlreiche Methoden und Theorien, entsprechende Berechnungen durchzuführen.
Eine Verbesserung der Geschwindigkeit geht jedoch meist zu Lasten der Genauigkeit, so dass eine „gute“ Methode stets einen guten Kompromiss zwischen Geschwindigkeit und Genauigkeit finden muss.
Diese Arbeit untersucht die Methode der *adaptiven Semi-Lagrange-Advektion*. Es werden unterschiedliche Varianten der Advektion und Adaption vorgestellt und deren Effizienz diskutiert. Dabei kommen Methoden zur *Interpolation mit radialen Basisfunktionen* sowie zur *Approximation mit Moving Least Squares* zum Einsatz. Wir werden die Vor- und Nachteile beider Methoden betrachten und einige Anwendungsbeispiele geben.
Wir werden die Problemstellung sowohl in der Eulerschen als auch in der Lagrangeschen Weise betrachten und beide Betrachtungsweisen zusammenführen. Danach werden wir einige Begriffe einführen und uns den Verfahren zur Lösung des Problems zuwenden.
Um die Vorgänge in der numerischen Simulation erfassen zu können, müssen wir das Problem zunächst diskretisieren.
Wir gehen bei unseren Betrachtungen stets von einer diskreten Menge von Stützstellen mit je einem zugeordneten Funktionswert, der so genannten *Konzentration* aus. In der Literatur wird hier auch von *Dichte* oder *Masse* gesprochen. Unter dem Einfluss eines einwirkenden Geschwindigkeitsfeldes verlagern sich die Konzentrationen im Laufe der Zeit. Um den Einfluss eines im Zeitablauf veränderten Geschwindigkeitsfeldes auszuschließen, werden wir mit zeitunabhängigen Geschwindigkeitsfeldern arbeiten. Dadurch können wir die

angewandten Verfahren störungsfrei betrachten. Anschließend werden wir die erdachten Lösungsverfahren an numerischen Beispielen erproben und die praktische Umsetzung verfeinern.
Abschließend gehen wir auf die für die Implementation der Verfahren verwendeten Klassen und Programmstrukturen ein.

1.2 Mathematik

1.2.1 Partikelbewegung

Um die Bewegung des Systems durch ein einwirkendes Geschwindigkeitsfeld zu beschreiben, nehmen wir eine zweimal stetig differenzierbare Funktion

$$\begin{aligned} F: \quad & \mathcal{I} \times \Omega \to \Omega \\ & (t, x) \;\mapsto F(t, x) \end{aligned}$$

mit $F(0, x) = x$ als gegeben an. Dabei betrachten wir $\mathcal{I} = [0, T] \subseteq \mathbb{R}$, $T > 0$ als Zeitintervall und $\Omega \subseteq \mathbb{R}^d$, $d \geq 1$ als das von uns betrachtete *Gebiet*.
$F(t, x)$ stellt die Position des Partikels zur Zeit t dar, das sich zur Zeit $t_0 = 0$ an der Stelle x befand. Wir nehmen weiterhin an, dass zunächst keine Partikel neu erzeugt werden oder verschwinden, daher ist die Abbildung

$$\begin{aligned} F^t: \quad & \Omega \quad \to \Omega \\ & F^t(x) = F(t, x) \end{aligned}$$

für feste $t \in \mathcal{I}$ eine Bijektion. F^t ist für alle $t \in \mathcal{I}$ eine zweifach stetig differenzierbare Koordinatentransformation, wobei $F^0(x) = x$ die Identität auf Ω darstellt.
Für feste $x \in \Omega$ definieren wir uns

$$\begin{aligned} G^x: \quad & \mathcal{I} \quad \to \Omega \\ & G^x(t) = F(t, x)\,. \end{aligned}$$

Wir wählen die Schreibweise

$$x(t) := G^x(t) \text{ mit } x(0) = x \text{ und } x(t) = (x_1(t), \ldots, x_d(t)) \in \Omega,$$

nehmen also an, dass zum Zeitpunkt $t_0 = 0$ die Positionen aller Partikel $x \in \Omega$ bekannt sei. Dann beschreibt $x(t)$ den Ort, an dem sich das Partikel, das zur

Zeit t_0 an der Stelle x war, zur Zeit t befindet.
Wir bezeichnen die durch $x(t)$ während eines Durchlaufs von t durch $\mathcal{I}$ erzeugte Kurve als die *Bahnkurve* oder *Trajektorie* von x.
Die Funktion F beschreibt die Bewegung der Partikel in so genannten *Eulerschen Koordinaten*. Bei dieser Form der Darstellung wird ein festes Koordinatensystem $(t, x_1, \ldots, x_d) \in \mathcal{I} \times \Omega \subseteq \mathbb{R} \times \mathbb{R}^d$ verwendet.
Wir weisen nun jedem Partikel aus Ω eine so genannte *Konzentration* zu. Dafür definieren wir eine stetig differenzierbare Funktion

$$\begin{aligned} c: \quad & \mathcal{I} \times \Omega \to \mathbb{R} \\ & (t, x) \ \mapsto c(t, x)\,, \end{aligned}$$

welche die Veränderung der Konzentrationen in Raum und Zeit in Eulerschen Koordinaten beschreibt.
Durch die Existenz von F können wir das System auch in *Lagrange-Koordinaten* betrachten. Das Lagrangesche Koordinatensystem
$(t, x_1(t), \ldots, x_d(t)) \in \mathcal{I} \times \Omega \subseteq \mathbb{R} \times \mathbb{R}^d$ bewegt sich dabei mit den Partikeln des Systems mit.
Durch die Wahl von F können wir von den Eulerschen auf die Lagrangeschen Koordinaten und umgekehrt zweimal stetig differenzierbar transformieren.
Bei gegebener Funktion c in Eulerschen Koordinaten beschreibt die Abbildung $t \mapsto c(t, x(t))$ die zeitliche Veränderung der Konzentration, die das Partikel mit $x(0) = x$ während seiner Bewegung auf der Bahnkurve erfährt.
Um die Bewegung eines Partikels zu einer Zeit t zu erfassen, verwenden wir das so genannte *Geschwindigkeitsfeld* $\vec{a}$, dass zu jedem Partikel $x \in \Omega$ den Vektor mit der Richtung seiner momentanen Bewegung beschreibt. Wir setzen

$$\begin{aligned} \vec{a}: \quad & \mathcal{I} \times \Omega \to \Omega \\ & (t, x) \ \mapsto \vec{a}(t, x) := \frac{\partial}{\partial t} F(t, x) = \frac{dx}{dt}(t) =: \dot{x}(t)\,. \end{aligned}$$

Nun können wir die Eulersche und die Lagrangesche Betrachtungsweise in Relation setzen durch die nach Anwendung der Kettenregel erhaltene Beziehung

$$\frac{d}{dt} c(t, x(t)) = \frac{\partial c}{\partial t} + \sum_{j=1}^{d} \frac{\partial c}{\partial x_j} \frac{dx_j}{dt}(t) = \frac{\partial c}{\partial t} + \vec{a} \cdot \nabla_x c\,, \tag{1.1}$$

wobei der *Gradient* $\nabla_x = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d})$ ist.
Dabei beschreibt die *substantielle* oder *materielle Ableitung* $\frac{d}{dt}$ die zeitliche Veränderung, die ein mitbewegter Beobachter wahrnimmt. Die partiellen Ableitungen auf der rechten Seite von (1.1) sind Ableitungen der Konzentration c in Eulerschen Koordinaten und beschreiben die raum-zeitliche Veränderung,

die ein ruhender Beobachter wahrnimmt. $\frac{\partial}{\partial t}$ wird dabei als *lokale zeitliche Ableitung*, die Richtungsableitung $\vec{a} \cdot \nabla_x$ als *konvektive Ableitung* bezeichnet.

1.2.2 Kurven

Wir wollen die oben beschriebenen Bewegungen der Partikel anschaulich darstellen. Dafür werden wir die Bahnkurven verwenden. Die von Warnecke in [9] beschriebenen ebenfalls zur Darstellung verwendeten *Stromlinien* und *Einspritzkurven* sind für unsere Betrachtungen identisch mit den Bahnkurven, da wir mit stationären Geschwindigkeitsfeldern arbeiten, um die durchgeführten Verfahren zur Simulation der Partikelbewegung zu testen. Diese Geschwindigkeitsfelder sind zeitunabhängig, so dass $\vec{a}(t,x) \equiv \vec{a}(x)$ gilt für alle $x \in \Omega$.
Wegen der Differenzierbarkeitsvoraussetzung sind die Bahnkurven Lösungen des Anfangswertproblems

$$\frac{d}{dt} z_{t_0}(t) = \frac{\partial}{\partial t} F(t, z_{t_0}(t)) = \vec{a}(t, z_{t_0}(t)), \text{ mit } z_{t_0}(t_0) = z_0 \,. \tag{1.2}$$

Wir verwenden nun den

Satz 1.1. (Satz von Picard-Lindelöf) Sei $\vec{g} : [m,n] \times \mathbb{R}^d \to \mathbb{R}^d$ eine stetige Funktion, welche eine Lipschitzbedingung erfüllt, d.h. es existiert eine *Lipschitzkonstante* $L > 0$, so dass für jedes $t \in [m,n]$ gilt:

$$\forall y_1, y_2 \in \mathbb{R}^d : \ \|\vec{g}(t,y_1) - \vec{g}(t,y_2)\| \leq L\|y_1 - y_2\| \,. \tag{1.3}$$

Dann gibt es zu jedem $x_0 \in \mathbb{R}^d$ eine differenzierbare Funktion $x : \ \mathbb{R} \to \mathbb{R}^d$, welche das Anfangswertproblem

$$\forall t \in (m,n) : \ \dot{x}(t) = \vec{g}(t, x(t)) \text{ mit } x(m) = x_0$$

eindeutig löst.

Wir werden später noch zeigen, dass alle von uns verwendeten Geschwindigkeitsfelder $\vec{a}$ die Voraussetzungen von Satz 1.1 erfüllen. Daher besitzt das Anfangswertproblem (1.2) eine eindeutige Lösung $z_{t_0}(t) = x(t) = G^x(t)$ für $t \in \mathcal{I}$.

1.2.3 Erhaltungssätze

Wir wollen überprüfen, ob die von uns getesteten Verfahren den Erhaltungssätzen genügen. Dafür betrachten wir die Funktion c in den Lagrangeschen Koordinaten $(t,x) = (t, x(t))$. Wir wählen zur Zeit t_0 ein so genanntes *Kontrollvolumen* $\Omega^0 \subset \Omega$ aus. Dabei ist Ω^0 offen, zusammenhängend und beschränkt. Es bezeichne $\Omega^t = F^t(\Omega^0)$ das weiterbewegte Kontrollvolumen zur Zeit $t \in \mathcal{I}$.

Definition 1.1. Eine Dichtefunktion c genügt einer *integralen Erhaltungsgleichung* bzw. einem *integralen Erhaltungssatz*, wenn für jedes $\Omega^0 \subset \Omega$ die Beziehung

$$\int_{\Omega^0} c(0,x)dx = \int_{\Omega^t} c(t,y)dy = C_{c,\Omega^0} \tag{1.4}$$

gilt für eine von t unabhängige Konstante $C_{c,\Omega^0} \in \mathbb{R}$.
Dies bedeutet, dass die Gesamtmenge der Konzentration konstant bleiben soll. Durch Ableiten von (1.4) nach t erhalten wir

$$\frac{d}{dt}\int_{\Omega^t} c(t,y)dy = 0\,. \tag{1.5}$$

Um hier die Differenziation tatsächlich auszuführen, müssen wir die Zeitabhängigkeit des Integrationsgebietes berücksichtigen. Wir verwenden dafür den

Satz 1.2. (Reynoldsscher Transportsatz): Sei $c : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ eine stetig differenzierbare Dichtefunktion. Sei die Transformation $F^t : \mathbb{R}^d \to \mathbb{R}^d$ ebenfalls stetig differenzierbar. Dann gilt für jedes beschränkte Kontrollvolumen Ω^t die Gleichung

$$\frac{d}{dt}\int_{\Omega^t} c(t,x)dx = \int_{\Omega^t} [\frac{\partial c}{\partial t}(t,x) + \mathrm{div}(c(t,x)\vec{a}(t,x))]dx\,. \tag{1.6}$$

Dabei verwenden wir die Schreibweise $\mathrm{div}(c\,\vec{a})$ für die *Divergenz* von $c\,\vec{a}$, wobei $\mathrm{div}(c\,\vec{a})$ das Skalarprodukt $\nabla_x \cdot (c\,\vec{a})$ darstellt.
Den Beweis für Satz 1.2 liefert Warnecke in [9] für allgemeines $d \in \mathbb{N}$.

Wir setzen den Integranden $\frac{\partial c}{\partial t} + \mathrm{div}(c\,\vec{a})$ als stetig voraus. Da außerdem eine integrale Erhaltungsgleichung für beliebige Kontrollvolumina Ω^0 gelten soll, folgt direkt aus (1.5) und (1.6), dass

$$\int_{\Omega^t} [\frac{\partial c}{\partial t} + \mathrm{div}(c\,\vec{a})]dx = 0 \tag{1.7}$$

und damit auch

$$\frac{\partial c}{\partial t} + \mathrm{div}(c\,\vec{a}) = 0 \tag{1.8}$$

gilt. Wäre dies nicht der Fall und es würde etwa in einem Punkt

$$\frac{\partial c}{\partial t} + \mathrm{div}(c\,\vec{a}) > 0$$

gelten, so müsste dies auch in einer ganzen Umgebung gelten. Wählt man diese dann als Kontrollvolumen, so erhält man einen Widerspruch zu (1.7).

Gleichungen der Form (1.8) bezeichnen wir als *Erhaltungsgleichungen* oder auch *Gleichungen in Divergenzform*.
Wir haben nun die integrale Erhaltungsgleichung (1.4) in eine partielle Differentialgleichung erster Ordnung transformiert, wobei wir das Geschwindigkeitsfeld $\vec{a}$ jeweils als bekannt voraussetzen werden, so dass die Dichtefunktion c als einzige Variable verbleibt.
Wir betrachten nun Gleichung (1.8) etwas genauer. Wir schreiben die Divergenz $\operatorname{div}(c\,\vec{a})$ als

$$\begin{aligned}\operatorname{div}(c\,\vec{a}) &= \sum_{j=1}^{d} \partial_j (c\,\vec{a}_j) \\ &= \sum_{j=1}^{d} (\partial_j c)\vec{a}_j + \sum_{j=1}^{d} c(\partial_j \vec{a}_j) \\ &= \vec{a} \cdot \nabla_x c + c \operatorname{div} \vec{a}\,.\end{aligned} \tag{1.9}$$

Da wir ausschließlich mit divergenzfreien Geschwindigkeitsfeldern arbeiten werden, ist $\operatorname{div} \vec{a} = 0$, woraus sofort

$$\operatorname{div}(c\,\vec{a}) = \vec{a} \cdot \nabla_x c = \dot{x} \cdot \nabla_x c \tag{1.10}$$

folgt. Die Erhaltungsgleichung (1.8) lässt sich nun schreiben als

$$\frac{\partial c}{\partial t} + \vec{a} \cdot \nabla_x c = 0\,. \tag{1.11}$$

Zusammen mit der Ableitung der Dichtefunktion (1.1) ergibt sich

$$\frac{d}{dt}\, c(t, x(t)) = 0\,. \tag{1.12}$$

Dies lässt sich so interpretieren, dass c konstant ist entlang von Bahnen, die vollständig durch das Geschwindigkeitsfeld $\vec{a}$ gegeben sind.

1.3 Zielsetzung

Unser Ziel ist nun die numerische Lösung der Gleichung (1.12) durch die von Behrens, Iske und Käser in [1], [2], [3] und [6] beschriebenen Methoden der *adaptiven Semi-Lagrange-Advektion*, auf die wir später genauer eingehen werden. (Kapitel 2.5).
Fur die dort verwendete Reproduktion von Funktionen aus Punktdaten werden wir zwei unterschiedliche Methoden vergleichen:

- Interpolation mit radialen Basisfunktionen (Behrens, Iske [1] Kap. 3.)
- Approximation mit Moving Least Squares (Wendland [7])

1.4 Motivation

Üblicherweise werden für die Berechnung von $c(t, x)$ *Finite Elemente* verwendet. Diese erfordern jedoch eine Gitterstruktur, die im zeitlichen Ablauf der Simulation gepflegt werden muss. Da die Kosten für die Verwaltung eines solchen Gitters Finiter Elemente in höheren Dimensionen unerschwinglich hoch sind, streben wir die Verwendung einer gitterfreien Advektion an.
In der vorliegenden Arbeit verwenden wir die *adaptive Semi-Lagrange-Methode* (SLM) in Verbindung mit *gitterfreier radialer Basisfunktion-Interpolation* (RBF-Interpolation) und *Moving-Least-Squares-Approximation* (MLS-Approximation) für die Simulation der Dynamik von Systemen.
Ebenfalls unerschwinglich hohe Kosten würde die Berechnung einer globalen Interpolante bzw. Approximante über allen Stützstellen verursachen. Daher werden wir die globale Interpolante durch lokale Patches ersetzen, die wir in vertretbarer Zeit bestimmen können.
Die verwendeten Methoden erfordern eine Reihe von Rahmenbedingungen, deren Evaluierung ein Teil dieser Arbeit ist. Außerdem wird ein Vergleich zwischen den unterschiedlichen Varianten der SLM gezogen und deren Tauglichkeit für das Verfahren überprüft.

1.5 Anfangsbedingungen

Für die experimentellen Untersuchungen dieser Arbeit müssen wir einige Anfangsbedingungen und Voraussetzungen schaffen.
Wir wollen die grundsätzliche Verwendbarkeit der von uns eingesetzten Verfahren überprüfen. Daher werden wir lediglich mit $d = 2$ arbeiten. Wir werden als Gebiet Ω stets kompakte Teilmengen des $\mathbb{R}^2$ wählen. Für die Durchführung unserer Betrachtungen benötigen wir nun Randbedingungen. Um möglichst einfach die Randbedingungen einhalten zu können, werden wir uns bei der Wahl von Ω auf Polygone beschränken. Für den Rand des Gebietes definieren wir eine Funktion $R(t, x)$, die jedem Randpunkt eine Konzentration zuweist. Dabei werden wir uns auf $R \equiv 0$ beschränken, um Störungen der Konzentration von außen zu vermeiden.
Um die Bewegung der Partikel in Ω numerisch erfassbar zu machen, werden wir eine diskrete Menge von *Knoten* $\{x_1, \ldots, x_n\} = X \subset \Omega$ wählen, für die eine Anfangsverteilung der Konzentration $c^0(x) := c(t, x)|_{t=0}$ gegeben ist. Dabei wollen wir stets $c^0(x) \geq 0$ voraussetzen.
Um ein Maß für die Distanz zwischen den Knoten zu haben, verwenden wir die

Definition 1.2. Der *Separationabstand* $q = q_X$ einer diskreten Menge $X \subset \Omega$ ist der Radius der größtmöglichen Kugel, die in Ω liegt und keinen Knoten $x \in X$ enthält. Es gilt

$$q_X := \frac{1}{2} \min_{i \neq j} \|x_i - x_j\|_2 \quad \text{mit } x_i, x_j \in X \text{ für } i, j = 1, \ldots, n\,.$$

Ebenfalls diskretisieren werden wir die Zeit. Dabei führen wir für die Berechnungen diskrete *Zeitschritte* ein. Die Durchführung eines Zeitschritts entspricht dem Übergang von einem Zeitpunkt $t \to t + \tau$, $t \in \mathcal{I}$, $\tau > 0$.
Das Ende der Berechnung ist erreicht, wenn $(t + \tau)$ nicht mehr im Intervall $\mathcal{I}$ liegt. Für einen endlichen Programmablauf wählen wir daher $\mathcal{I}$ kompakt, d.h. wir arbeiten mit endlichem T.
Die Bahnkurve von $x \in X$ besteht nun aus einzelnen Punkten, die durch $x(t_0 + i\tau)$ gegeben werden mit $i \in \mathbb{N}_0$.

1.6 Testreihen

Die in dieser Arbeit vorgestellten Testreihen stellen jeweils signifikante Fälle dar. Dadurch werden die Zusammenhänge und Abhängigkeiten zwischen den verwendeten Methoden und den vorgegebenen Parametern aufgezeigt. Die Parameter sind innerhalb eines Tests konstant gehalten.
Variiert werden

- die Anzahl und Verteilung der Knoten in der Startmenge,
  (Zufällige Verteilung, bzw. auf den Ecken eines Gitters)
- die initiale Konzentration in den Knoten,
  (stetige und unstetige Funktionen)
- die Zeitschrittweite τ,
  (Anzahl der Zeitschritte bis zum Erreichen des Zeitpunkts T)
- die verwendete Methode Für die Funktionsreproduktion,
  (Thin Plate Spline, Wendland-Funktion, Moving Least Squares)
- die Methoden zum Anpassen der Knotenmenge,
  (konstant oder dynamisch)
- die Fehlertoleranz bei dynamisch angepasster Knotenmenge,
  (Grenzwerte für das Einfügen und Löschen von Knoten)
- die Anzahl der Knoten zur Berechnung der lokalen Reproduktion,
  (Für größere Geschwindigkeit bzw. Genauigkeit)
- Clipping.
  (Parameter zur Vermeidung von numerisch bedingten Oszillationen und negativen Werten der Konzentration)

2 Methoden

„Es gibt Dinge, die den meisten Menschen unglaublich erscheinen, die nicht Mathematik studiert haben.“

Archimedes (287 v. Chr. - 212 v. Chr.)

2.1 Upstreampunkt

Da die in dieser Arbeit betrachtete Semi-Lagrange-Advektion ein rückwärts arbeitendes Verfahren ist, interessieren wir uns speziell für den Ursprung der einzelnen Partikel. In Abschnitt 1.2.2 haben wir gezeigt, dass das Anfangswertproblem (1.2) mit der eindeutigen Lösung $z_{t_0}(t) = x(t)$ dargestellt werden kann als

$$\dot{x}(t) = \vec{a}(t, x(t)) \text{ mit } x(t_0) = z_0.$$

Wir setzen nun $t_0 = t - \tau$, um den Zustand vor genau einem Zeitschritt zu erfassen. Die eindeutig bestimmte Stelle, an der sich das Partikel x zur Zeit $t - \tau$ befand, nennen wir den den *Upstreampunkt zu* x, den wir auch als x^- bezeichnen werden. Es gilt also $z_0 = x^-$. Der Upstreampunkt ist das eindeutig bestimmte Partikel in Ω zur Zeit t, das sich unter dem Einfluss des Geschwindigkeitsfeldes entlang seiner Bahn nach einem Zeitschritt τ zur Stelle x bewegt. Die Punkte x und x^- liegen dabei auf derselben Bahnkurve.
Es gilt $x(t) = x^-(t + \tau)$ und $x(t - \tau) = x^-(t)$.
Da wir die exakte Lösung der Differentialgleichung in der Regel nicht geschlossen angeben können, werden wir mit $\tilde{x}$ arbeiten, einer Approximation des Upstreampunktes x^-. Für die Approximation verwenden wir die von Behrens und Iske in [1] angegebene Rekursionsformel

$$\alpha_{n+1} = \tau \cdot \vec{a}(t + \frac{\tau}{2}, x - \frac{\alpha_n}{2}),\ \alpha_0 = 0,\ n = 0, 1, 2, \ldots$$

Wir erhalten dann die Approximation des Upstreampunktes $\tilde{x}$ durch

$$\tilde{x} = x - \alpha_{N_{it}},\ N_{it} \in \mathbb{N}\,.$$

Wir können durch die Wahl von N_{it} die Genauigkeit bestimmen, mit der wir den Upstreampunkt approximieren. Diese hängt jedoch direkt vom verwendeten Geschwindigkeitsfeld $\vec{a}$ ab, so dass wir $N_{it} \equiv N_{it}(\vec{a})$ später festlegen werden (Kapitel 3.1.1). Abbildung 1 zeigt schematisch die approximative Bestimmung von $\tilde{x}$.

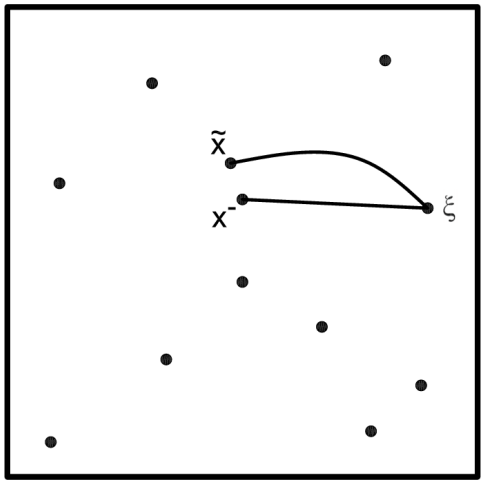


Abb. 1: Upstreampunkt x^- und Approximation $\tilde{x}$

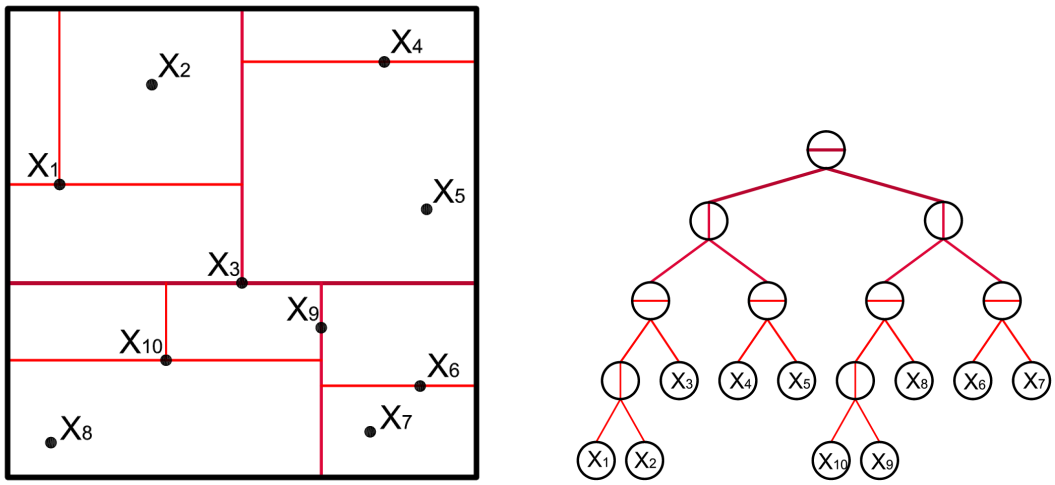
2.2 Bestimmung von Nachbarschaften / KD-Tree

Für die erwähnte Lokalität unserer Berechnungen werden wir uns stets eine Teilmenge von $X = \{x_1, \ldots, x_n\}$ wählen, die uns eine gute Reproduktion der Konzentration an der betrachteten Stelle ξ liefert. Diese Teilmenge nennen wir die *Nachbarschaft von* ξ.

Definition 2.1. Unter einer *k-Nachbarschaft* $\mathcal{N}(\xi) \equiv \mathcal{N}_k(\xi) \subseteq X$, $\xi \in \Omega$ verstehen wir die k Knoten aus X, die zu ξ den geringsten euklidischen Abstand haben und von ξ verschieden sind.
Die Menge $\mathcal{N} \cup \xi$ bezeichnen wir als $\mathcal{N}^*$.

Die einfachste Möglichkeit zur Bestimmung von $\mathcal{N}_k(\xi)$ ist die so genannte *Brute-Force*-Methode. Dabei werden die euklidischen Abstände $\|\xi - x\|_2$ für alle $x \in X$ berechnet. Die k Knoten mit dem geringsten Abstand zu ξ bilden dann $\mathcal{N}$.
Mit diesem Verfahren kostet uns das Ermitteln einer einzigen Nachbarschaft jedoch $O(n)$, so dass wir uns ein effizienteres Verfahren überlegen werden.
Zunächst werden wir der Knotenmenge vor der eigentlichen Suche eine gewisse Struktur geben. Dabei wird eine spezielle Datenstruktur verwendet, der *KD-Tree*. Der KD-Tree ist ein binärer Baum, in dem alle Punkte als Blätter enthalten sind. Wir teilen dazu das Gebiet Ω sukzessive in kleinere Teilgebiete auf. Dabei teilen wir jedes entstandene Teilgebiet geradlinig entlang einer der Raumdimensionen.
Bei jedem Teilvorgang entsteht in der Baumstruktur ein neuer Knoten, der dann in die beiden Unterbäume verzweigt, die zu den beiden neu entstande-

nen Teilgebieten gehören. Dieses Verfahren führen wir nun solange durch, bis jedes Teilgebiet nur noch genau einen Punkt enthält.
Für die Festlegung der Trennlinie gibt es unterschiedliche Auswahlkriterien. Ein Weg ist es, jedes Gebiet, das mehr als einen Punkt enthält, orthogonal zu seiner längsten Seite zu halbieren. Dadurch erreichen wir eine einfache Erzeugung des KD-Tree. Da wir mit einem Separationsabstand $q_X > 0$ arbeiten, ist eine Terminierung des Verfahrens gewährleistet. Diese Methode wird jedoch unseren Bedürfnissen nicht gerecht, da die Verteilung der Punkte über Ω ignoriert wird.
Wie wir später noch sehen werden, liegt in unseren Betrachtungen oft eine sehr ungleichmäßige Verteilung der Knoten vor. Dadurch kann es zu einem sehr ungleich gewichteten KD-Tree kommen, im Extremfall zu einem Baum der Tiefe n für n Punkte. Dies würde die Suche nach Punkten im Baum natürlich stark verlangsamen, da die Kosten für die Suche bei $O(n)$ lägen, statt, wie bei einem binären Baum erwartet, bei $O(\log_2(n))$.
Wir werden daher zum Aufbau des KD-Tree etwas mehr Aufwand betreiben, um die Suche nach Punkten zu beschleunigen. Wir werden in jedem Gebiet zunächst den Median aller Punkte entlang einer Dimension bestimmen. Wenn wir nun entlang des Medians teilen, erhalten wir in beiden Unterbäumen die gleiche Anzahl an Punkten. Letztendlich erhalten wir eine gleichmäßig gewichteten Baum der Tiefe $\lceil \log_2(n) \rceil$, was die Kosten für die Punktsuche auf $O(\log_2(n))$ verringert. Die Erzeugung des KD-Tree bei diesem Verfahren ist mit $O(n \log(n))$ allerdings teurer als bei der zuvor beschriebenen Methode. Die Median-Methode garantiert uns für jede beliebige Verteilung von Punkten die genannten Kosten, so dass wir zugunsten der schnelleren Suche diese Variante verwenden werden.
Abb. 2 zeigt an einem Beispiel den Aufbau eines KD-Trees nach der Median-Methode. In 2(a) ist das Gebiet mit den enthaltenen Punkten abgebildet. Die geraden Linien stellen die Teilungslinien dar. 2(b) zeigt den zugehörigen KD-Tree. In den Knoten des Baums zeigt jeweils eine Linie die Richtung des entsprechenden Teilvorgangs.
Wir verwenden die von Mount und Arya ([13] und [14]) implementierte *Approximate Nearest Neighbor*-Bibliothek (*ANN*) zur Bestimmung der Nachbarschaften.
Die ANN-Bibliothek stellt uns eine Funktion zum Erstellen des KD-Tree zur Verfügung. Außerdem ist dort bereits die Suche nach den nächsten Nachbarn eines Punktes im KD-Tree effizient implementiert, so dass wir an dieser Stelle nicht näher auf diese Routine eingehen wollen. Wir sind nun in der Lage, zu jeder Stelle $\xi \in \Omega$ die Nachbarschaft $\mathcal{N}_k(\xi)$ zu bestimmen.
Dadurch ist es uns möglich, die Konzentration an der Stelle ξ durch Interpo-



(a) Knotenverteilung und Unterteilung des Gebiets

(b) Baumstruktur des fertigen KD-Tree

Abb. 2: Aufbau des KD-Tree nach der Median-Methode

lation auf der Nachbarschaft zu ermitteln.

2.3 Interpolation mit radialen Basisfunktionen

Eine der von uns betrachteten Methoden zur Reproduktion von Funktionen ist die Interpolation mit radialen Basisfunktionen (RBFs).
Basis der Berechnungen ist eine Stelle $\xi \in \Omega$ mit der k-Nachbarschaft $\mathcal{N} \equiv \mathcal{N}_k(\xi) = \{x_1, \ldots, x_k\} \subset \Omega$.
Die Werte $c(t, \cdot)$ seien für die gesamte Nachbarschaft bekannt. Weiterhin sei eine fixe radiale Basisfunktion

$$\begin{aligned} \phi : [0, \infty) &\to \mathbb{R} \\ r &\mapsto \phi(r) \end{aligned}$$

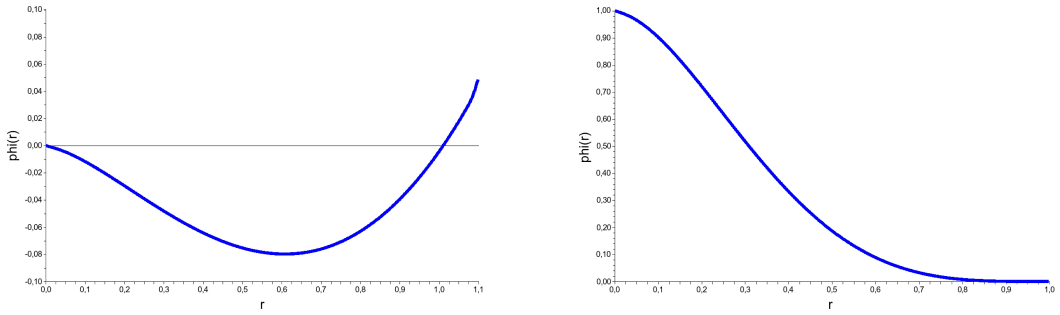
gegeben. In unseren Betrachtungen werden wir mit dem

Thin Plate Spline $\phi(r) := r^2 \log(r)$ (Abb. 3(a)) und der
Wendland-Funktion $\phi(r) := (1-r)_+^4(4r+1)$ (Abb. 3(b))

als radiale Basifunktionen arbeiten.

Dabei sei $a_+ := \begin{cases} 0 & a \leq 0, \\ a & a > 0, \end{cases}$

so dass die Wendland-Funktion außerhalb von $[0, 1]$ stets 0 ist. Wir sprechen vom *kompakten Träger* $[0, 1]$.



(a) Thin Plate Spline
$\phi(r) := r^2 \log(r)$

(b) Wendland-Funktion
$\phi(r) := (1-r)_+^4(4r+1)$

Abb. 3: Funktionsgraphen der verwendeten RBFs

Gesucht wird nun die Interpolante $s : \mathbb{R}^d \to \mathbb{R}$ der Form

$$s = \sum_{j=1}^{k} \lambda_j \phi(\| \cdot - x_j \|) + p\,. \tag{2.1}$$

Hierbei sei $\| \cdot \|$ die euklidische Norm, $p \in \Pi_m^d$ und Π_m^d der lineare Raum aller d-variaten Polynome vom Grad kleiner als m.
Das Hinzufügen eines polynomiellen Teils p bewirkt, dass die Interpolante jedes Polynom aus Π_m^d exakt reproduziert. Dafür wählen wir uns eine Basis $B_Q := \{p_1, \ldots, p_Q\}$ von Π_m^d mit $Q \leq k$, hier also $Q = \binom{m-1+d}{d}$, so dass sich p schreiben lässt als

$$p = \sum_{j=1}^{Q} \mu_j p_j, \quad \mu_j \in \mathbb{R}\,. \tag{2.2}$$

Wir werden als Basis von Π_m^d stets mit der *Monombasis* arbeiten, d.h.

$$\begin{aligned} B_Q = \{p \in \Pi_m^d : p(u) = u^\alpha = u_1^{\alpha_1} \ldots u_d^{\alpha_d}, \\ u \in \mathbb{R}^d,\, \alpha = (\alpha_1, \ldots, \alpha_d), \\ 0 \leq \alpha_i \leq |\alpha| = \sum \alpha_i < m,\, i = 1, \ldots, d\}\,. \end{aligned}$$

Um $s(\xi)$ zu bestimmen, müssen wir nun den Koeffizientenvektor $(\lambda^\top, \mu^\top)^\top \in \mathbb{R}^{k+Q}$ berechnen mit $\lambda = (\lambda_1, \ldots, \lambda_k)^\top \in \mathbb{R}^k$ für den Hauptteil und $\mu = (\mu_1, \ldots, \mu_Q)^\top \in \mathbb{R}^Q$ für den polynomiellen Teil.
Da wir in den vorgegebenen Knoten die Funktion exakt reproduzieren wollen,

muss $s|_{\mathcal{N}} \equiv c(t,\cdot)|_{\mathcal{N}}$ gelten, also

$$c(t,x_i) = s(x_i) = \sum_{j=1}^{k} \lambda_j \phi(\|x_i - x_j\|) + \sum_{l=1}^{Q} \mu_l p_l(x_i), \quad \forall x_i \in \mathcal{N}\,. \tag{2.3}$$

Dies sind die k Hauptbedingungen unseres Problems.
Bei der Reproduktion von Polynomen soll außerdem $\lambda_j = 0$ gelten für $j = 1,\ldots,k$, da wir p offensichtlich bereits durch B_Q exakt reproduzieren können. Außerdem nehmen wir an, $\mathcal{N}$ sei nicht degeneriert bezüglich Π_m^d oder Π_m^d-*unisolvent*.

Definition 2.2. Eine Menge $X = \{x_1,\ldots,x_N\} \subseteq \mathbb{R}^d$ aus paarweise verschiedenen Knoten heißt Π_m^d-*unisolvent*, wenn das Nullpolynom das einzige Polynom aus Π_m^d ist, das in allen x_j verschwindet, d.h. es gilt

$$p(x_j) = 0,\ 1 \le j \le k,\ p \in \Pi_m^d \quad \Rightarrow \quad p \equiv 0. \tag{2.4}$$

Bisher haben wir k Gleichungen in $k+Q$ Unbekannten zu lösen. Um ein eindeutig bestimmtes Gleichungssystem zu erhalten, werden wir noch die Q Nebenbedingungen einführen:

$$\sum_{j=1}^{k} \lambda_j p(x_j) = 0,\ \forall p \in \Pi_m^d\,. \tag{2.5}$$

Außerdem werden wir die Hauptbedingungen des Problems noch anpassen, um eine höhere Stabilität zu erreichen, und wie von Wendland und Rieger in [8] beschrieben, anstelle der exakten Lösung ein Minimierungsproblem betrachten. Das Minimierungsproblem formulieren wir als

$$\begin{aligned}\text{Minimiere } \{\sum_{j=0}^{k}[c_j - s(x_j)]^2 + \rho\|s\|_H^2 :\ & s = \sum_{j=0}^{k} \lambda_j \phi(\|x - x_j\|)\ + P\ , \\ & s \in H + P,\ H := \operatorname{span}\{\phi(\|\cdot - x\|)\}, \\ & x \in \mathbb{R}^d,\ \rho > 0\}\end{aligned} \tag{2.6}$$

mit der Norm $\|\cdot\|_H$ auf dem Hilbertraum H, radialen Basisfunktion ϕ und einem polynomialen Anteil P.
Durch die Verwendung des *Glätteparameters* $\rho > 0$ verlieren wir die Interpolationseigenschaft, werden jedoch weiterhin von der *RBF-Interpolante* sprechen. Für unsere Betrachtungen werden wir mit $\rho = 10^{-5}$ arbeiten.

Dadurch transformieren wir die Hauptbedingungen $\{A\lambda = c\}$ nach $\{(A + \rho I)\lambda = c\}$ mit $\rho > 0$ und der Einheitsmatrix I (Vergleiche [8]).
Insgesamt haben wir also das folgende lineare Gleichungssystem zu lösen:

$$\begin{pmatrix} A_{\phi,\mathcal{N}} + \rho I & P \\ P^\top & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} (c(t, x_j))_{1 \le j \le k} \\ 0 \end{pmatrix}, \text{ wobei}$$
$$A_{\phi,\mathcal{N}} = (\phi(\|x_j - x_l\|))_{1 \le j,\, l \le k} \in \mathbb{R}^{k \times k},$$
$$P = (p_l(x_j))_{1 \le j \le k,\, 1 \le l \le Q} \in \mathbb{R}^{k \times Q} \text{ und}$$
$$x_i \in \mathcal{N} \quad \forall i = 1, \ldots, k\,.$$

Das Gleichungssystem liefert uns die RBF-Interpolante s. Wir erhalten nun den interpolierten Wert an der Stelle ξ durch Einsetzen von ξ, λ und μ in (2.1) und (2.2).
Die von uns angewandte Methode der RBF-Interpolation reproduziert Polynome ausschließlich mit Polynomen.
Für unsere Zwecke reicht es aus, wenn wir höchstens lineare Polynome hinzunehmen, also $Q = 1$ oder $Q = 2$ wählen. Die Monombasis ist dann $B_1 = \{1\}$ bzw. $B_2 = \{1, x, y\}$ für $(x, y) \in \mathbb{R}^2$.
Um die Konzentration an der Stelle ξ mit einer k-Nachbarschaft zu reproduzieren, müssen wir also ein Gleichungssystem der Ordnung $k + Q$ lösen. Da die Anzahl der Polynome konstant 1 oder 3 ist, kostet uns das Lösen des linearen Gleichungssystems ungefähr $O(k^3)$.
Danach müssen wir den Lösungsvektor noch einsetzen, um $s(\xi)$ auszurechnen. Dies geschieht jedoch in Linearzeit $O(k + q)$, so dass wir insgesamt für die RBF-Interpolation ungefähr $O(k^3)$ Operationen für jede Stelle ξ benötigen.
Wir werden zusätzlich eine weitere Methode zur Funktionsreproduktion betrachten, die Approximation mit *Moving Least Squares*.

2.4 Moving Least Squares Approximation

Im Gegensatz zur *Interpolation* mit RBFs wird bei der MLS-*Approximation* die gesuchte Zielfunktion c approximiert. Unter bestimmten Voraussetzungen können wir jedoch auch hier eine Interpolation von c erreichen, wie wir noch sehen werden.
Es sei eine kompakte Menge $\Omega \subseteq \mathbb{R}^d$ gegeben. Eine stetige Funktion $c^t \equiv c \in C(\Omega)$ soll aus einer Knotenmenge $X = \{x_1, \ldots, x_n\} \subset \Omega$ und den Funktionswerten $c(x_1), \ldots, c(x_n)$ an der Stelle $\xi \in \Omega$ rekonstruiert werden. Dann ist die

beste Approximation $p^*(\xi)$ an $c(\xi)$ gegeben durch p^*, die Lösung von

$$\min_{p\in\mathcal{P}}\{\sum_{i=1}^{n}(c(x_i)-p(x_i))^2\omega(\xi,x_i)\}\,,$$

wobei $\mathcal{P}\subseteq C(\Omega)$ ein endlichdimensionaler Unterraum ist, üblicherweise aufgespannt durch Polynome. $\omega:\Omega\times\Omega\to[0,\infty)$ ist eine stetige Funktion mit einer potentiellen Singularität auf der Diagonalen $\omega(\xi,\xi)$.
Wir werden in unseren Betrachtungen $\omega(x,y)=(\|x-y\|_2^{-s})\Phi_\omega(\|x-y\|_2)$ mit festem $s>0$ verwenden. Dabei ist $\Phi_\omega(r)$ eine glatte, stetige Funktion mit kompaktem Träger. Wir verwenden $s=2$ und setzen die Wendland-1-Funktion als Φ_ω ein.
Durch die Singularität erreichen wir eine Interpolation von c durch p^* in den Knoten $x_i\in\mathcal{N}$. Wir können insbesondere eine lokale Reproduktion einer jeden Funktion c durch hinzufügen von c zu $\mathcal{P}$ erreichen.
Unter günstigen Voraussetzungen (Satz 2.1) kann man zeigen, dass die Lösung eindeutig bestimmt ist und dass $p^*(\xi)$ dargestellt werden kann als

$$p^*(\xi)=\sum_{i=1}^{k}a_i^*(x)c(x_i)\,. \tag{2.7}$$

Dabei ist $a_i^*(\xi)$ die Lösung des Minimierungsproblems

$$\begin{aligned}\text{Minimiere}\quad & \frac{1}{2}\sum_{i=1}^{n}a_i^2\theta(\xi,x_i)\\ \text{unter den Nebenbedingungen}\quad & \sum_{i=1}^{n}a_ip(x_i)=p(\xi),\end{aligned}$$

wobei $\theta(x,y)=\frac{1}{\omega(x,y)}$.
Uns interessieren vornehmlich lokale und stetige Gewichtsfunktionen ω, daher wählen wir eine stetige Funktion $\phi:[0,\infty)\to[0,\infty)$ mit $\phi(r)>0$ für $0\le r<1$ bzw. $\phi(r)=0$ für $r\ge 1$ und setzen

$$\theta_\delta(x,y):=\frac{1}{\phi(\frac{\|x-y\|_2}{\delta})},\quad \delta>0\,.$$

Da die Gewichtsfunktion nun eine stetige Funktion ist, führt der Approximationsprozess nicht mehr zur Interpolation der Zielfunktion in den gegebenen Knoten.
Wir definieren zur Knotenmenge $X=\{x_1,\ldots,x_n\}$ eine Indexmenge

$$I(\xi)\equiv I(\xi,\delta,X)=\{j\in\{1,\ldots,k\}:\|\xi-x_j\|_2<\delta\}$$

von k Knoten im Inneren der abgeschlossenen Kugel $B(\xi, \delta)$ mit Radius δ um ξ. I liefert uns gerade die Indizes der Knoten aus der k-Nachbarschaft $\mathcal{N}_k(\xi) \subseteq X$. Wir wählen $\mathcal{P} = \Pi_m^d$ als den Raum der d-variaten Polynome vom Grad kleiner als m. Dann hat die MLS-Approximation die Form

$$s_{c,X}(\xi) = \sum_{j \in I(\xi)} a_j^*(\xi) c(x_j), \tag{2.8}$$

wobei die Koeffizienten $a_j^*(\xi)$ bestimmt werden durch

$$\begin{aligned} &\text{Minimiere } \frac{1}{2} \sum_{i \in I(\xi)} a_j(\xi)^2 \theta_\delta(\xi, x_j) \\ &\text{unter den Nebenbedingungen } \sum_{i \in I(\xi)} a_j(\xi) p(x_j) = p(\xi), \quad p \in \Pi_m^d \,. \end{aligned} \tag{2.9}$$

In der Praxis werden wir δ und k in Abhängigkeit voneinander setzen, indem wir bei vorgegebenen k den Radius δ dynamisch an die $\mathcal{N}_k(\xi)$ umfassende Kugel anpassen.

Satz 2.1. Es sei $\mathcal{N}_k(\xi) = \{x_j \in X : j \in I(\xi, \delta, X)\}$ gegeben mit einer Π_m^d-unisolventen Untermenge. Sei $\{p_1, \ldots, p_Q\}$ eine Basis von Π_m^d. Dann hat das Minimierungsproblem (2.9) die eindeutig bestimmte Lösung a_j^*, $j \in I(\xi)$ mit der Darstellung

$$a_j^*(\xi) = \phi(\frac{\|\xi - x_j\|_2}{\delta}) \sum_{i=1}^{Q} \lambda_i p_i(x_j), \tag{2.10}$$

wobei die λ_i die eindeutigen Lösungen des Gleichungssystems

$$\sum_{i=1}^{Q} \lambda_i \sum_{j \in I(\xi)} \phi(\frac{\|\xi - x_j\|_2}{\delta}) p_i(x_j) p_l(x_j) = p_l(\xi), \quad 0 < l \leq Q \tag{2.11}$$

sind.
Den Beweis für diese Behauptung liefert Wendland in [7].
Wir müssen zum Berechnen der MLS-Approximante das lineare Gleichungssystem (2.11) der Ordnung Q lösen, was uns $O(Q^3)$ kostet. Das Aufstellen der Matrix des LGS kostet uns $O(Q^2 \cdot k)$. Das Einsetzen der Lösungen λ_i in (2.10) erledigen wir in $O(Q)$, die Auswertung der Approximante an der Stelle ξ durch (2.8) erfordert $O(k)$ Operationen.
Insgesamt haben wir also Kosten von $O(k+k \cdot Q^2+Q^3)$ für die MLS-Approximante für jede Reproduktion der Konzentration an einer Stelle $\xi \in \Omega$. Im Gegensatz

zur RBF-Interpolation, deren Kosten in erster Linie von der Anzahl k der Punkte in den Nachbarschaften abhängen, ist bei der MLS-Approximation der Grad Q der verwendeten Polynome entscheidend. Wir werden später auf die Wahl von k und Q zurückkommen.

2.5 Semi-Lagrange-Advektion

In dieser Arbeit verwenden wir die so genannte *Rückwärts-Semi-Lagrange-Advektion* (SLM).
Die Verwendung des rückwärts arbeitenden Algorithmus ermöglicht uns die Verwendung größerer Zeitschrittweiten τ bei der SLM. Dieses Verfahren kommt insbesondere bei der Erstellung von Wettervorhersagen zum Einsatz, wie Falcone und Ferretti in [11] beschreiben. Die SLM integriert die Advekionsgleichung entlang von Bahnen. Als Ausgangspunkt dient uns daher für das spezielle Advektionsschema die Diskretisierung

$$\frac{c(t+\tau, x) - c(t, x^-)}{\tau} = 0,\ x \in \Omega,$$

wobei $t \in I$ die aktuelle Zeit darstellt, $\tau > 0$ die Zeitschrittweite und x^- den Upstreampunkt zu x.
Für eine endliche Menge aktueller Knoten $X = \{x_1, ..., x_n\} \subset \Omega$ müssen wir für jeden Zeitschritt $t \to t + \tau$ in der SLM einen Vektor

$$c_X^{t+\tau} = (c(t+\tau, x_1), ..., c(t+\tau, x_n))^\top \in \mathbb{R}^n \text{ berechnen.}$$

Dieser Vektor enthält die aktuellen Konzentrationen in den Knoten $x_1, \ldots, x_n$ zur Zeit $t + \tau$ entsprechend den Werten des Vektors c_X^t des vorherigen Zeitschritts. Für die Zeit $t = 0$ ist die Verteilung der Konzentration c_X^0 gegeben durch die Startbedingungen. Die Berechnung für die neuen Konzentrationen $c(t+\tau), x \in X$ erfolgt dann nach folgendem Schema:

1. Berechne die Approximation $\tilde{x}$ des Upstreampunktes x^-,
2. Reproduziere $c(t, \tilde{x})$ mit den Werten c_X^t,
3. Führe die Advektion durch, indem $c(t+\tau, x) = c(t, \tilde{x})$ gesetzt wird.

Die Qualität des Verfahrens hängt im Wesentlichen von der Qualität der Interpolation in Schritt 2 ab. Da die Kosten für die bisher üblichen Methoden mit Finiten Elementen in höheren Dimensionen sehr hoch sind, bieten sich für solche Probleme gitterfreie Methoden an.

Unsere Betrachtungen befassen sich mit der generellen Durchführbarkeit und Parametrisierung der gitterfreien Methoden. Daher werden wir uns auf zweidimensionale Probleme beschränken, so dass $x_1, ..., x_n \in \mathbb{R}^2$.
Eine gitterfreie SLM basiert im Wesentlichen auf einer gitterfreien Reproduktion von Funktionen. Für unsere Versuche werden wir einerseits mit RBF-Interpolation, andererseits mit MLS-Approximation arbeiten. Diese Arten der Reproduktion eignen sich gut für die Reproduktion von Funktionen auf verstreuten multivariaten Daten.
Um die Komplexität des Verfahrens gering zu halten, ohne jedoch an Genauigkeit einzubüßen, führen wir in jedem Zeitschritt eine dynamische Anpassung der Knotenmenge durch die so genannte *Adaption*.

2.6 Adaption

Die Qualität der Advektion wird wesentlich von der lokalen Reproduktion der Konzentrationswerte bestimmt. In Bereichen mit hohen Schwankungen der Konzentration benötigen wir ein hinreichend dichtes Netz von Stützstellen für eine ausreichende Genauigkeit. Da wir das Verfahren jedoch möglichst effizient halten wollen, wäre eine gleichmäßig dichte Knotenverteilung über Ω eher ungünstig. In Gebieten mit geringen oder keinen Konzentrationschwankungen könnten wir mit sehr weit auseinanderliegenden Knoten dennoch die Konzentration sehr gut reproduzieren.
Daher werden wir vor jeder Advektion zunächst die Knotenmenge dynamisch anpassen, um sowohl die Geschwindigkeit als auch die Genauigkeit zu erhöhen. Wir werden die Knotenmenge vor jeder Advektion solange anpassen, bis die Adaption *stationär* wird. Dies bedeutet, dass sich die Knotenmenge sowohl in der Anzahl der Knoten als auch in der Verteilung nicht oder nur noch sehr wenig verändert. Wir wollen hiermit den Interpolations- bzw Approximationsfehler bei der Advektion möglichst gering halten.
Wir benötigen zunächst ein Maß, um zu entscheiden, an welchen Stellen von Ω wir mehr oder weniger Knoten benötigen.
Wir betrachten den Fehlerindikator $\eta : X^t \to [0, \infty)$, wobei $X \equiv X^t$ die Knotenmenge zur Zeit t darstellt.
Zu jedem $x \in X$ gibt es einen *Signifikanzwert* $\eta(x)$, der die Qualität der lokalen Approximation in x misst. Wir definieren $\eta(x) := |c(x) - s(x)|$, wobei $c \equiv c(t, \cdot)$ ist und die Reproduzierende $s \equiv s_{\mathcal{N}}$ mit der k-Nachbarschaft $\mathcal{N} \equiv \mathcal{N}_k(x) \subset X$. Wir verwenden für die lokale Reproduktion s die RBF-Interpolation bzw. die MLS-Approximation.
Bei der RBF-Interpolation gilt für c und s insbesondere $c(\nu) = s(\nu)$ für alle $\nu \in \mathcal{N}$. Die Eigenschaft von s, lineare Polynome zu rekonstruieren, bedingt

$\eta(x) = 0$, falls c linear ist in der Nachbarschaft von x. Kann c durch s gut reproduziert werden, so erhalten wir für $\eta(x)$ kleine Werte, hingegen bedeuten große η, dass c in der Umgebung von x nicht gut reproduziert wird.
Für die Adaption der Knotenmenge verwenden wir daher die

Definition 2.3. Sei $\eta^* = \max_{x \in X} \eta(x)$ und seien $\sigma_{crs}, \sigma_{ref}$ Parameter mit $0 < \sigma_{crs} < \sigma_{ref} < 1$.
Dann muss ein Knoten $x \in X$ *verfeinert* werden, wenn $\eta(x) > \sigma_{ref} \cdot \eta^*$.
η wird *ausgedünnt*, wenn $\eta(x) < \sigma_{crs} \cdot \eta^*$.

Kann die Konzentration in einem Knoten durch die umliegenden gut genug reproduziert werden, so wird dieser Knoten nicht länger benötigt.

Definition 2.4. Ein Knoten $x \in X$ wird *ausgedünnt*, indem er aus der aktuellen Knotenmenge X gelöscht wird, X wird ersetzt durch $X \setminus \{x\}$.
Schwieriger gestaltet sich die Suche nach einem sinnvollen Verfahren für das Einfügen neuer Knoten. Wir müssen einerseits den Ort der neuen Knoten festlegen, andererseits die Konzentration in den neuen Knoten bestimmen. Zunächst werden wir uns ein Verfahren zur Ortsbestimmung überlegen. Wir möchten das Gebiet, in dem schlecht reproduziert wird, mit einem möglichst gleichmäßigen feineren Knotengitter überdecken und trotzdem eine gewisse Distanz zwischen den Knoten erhalten, um ein Entarten der Knotenmenge zu verhindern. Daher erscheint es sinnvoll, zusätzliche Knoten möglichst weit entfernt von bereits bestehenden Knoten einzufügen, d.h. möglichst mitten in bisher knotenfreien Bereichen. Diese Bedingung erfüllen gerade die Ecken der *Voronoizelle*.

2.6.1 Voronoipunkte / Delaunay-Triangulierung

Definition 2.5. Die *Voronoizelle* $V(x)$ zu einem Knoten $x \in X := \{x_1, \ldots, x_n\} \subset \mathbb{R}^2$ ist definiert als

$$V(x) := \{\xi \in \mathbb{R}^2 : \|x - \xi\|_2 \leq \|x_j - \xi\|_2, \forall x_j \in X,\, j = 1, \ldots, n\}\,.$$

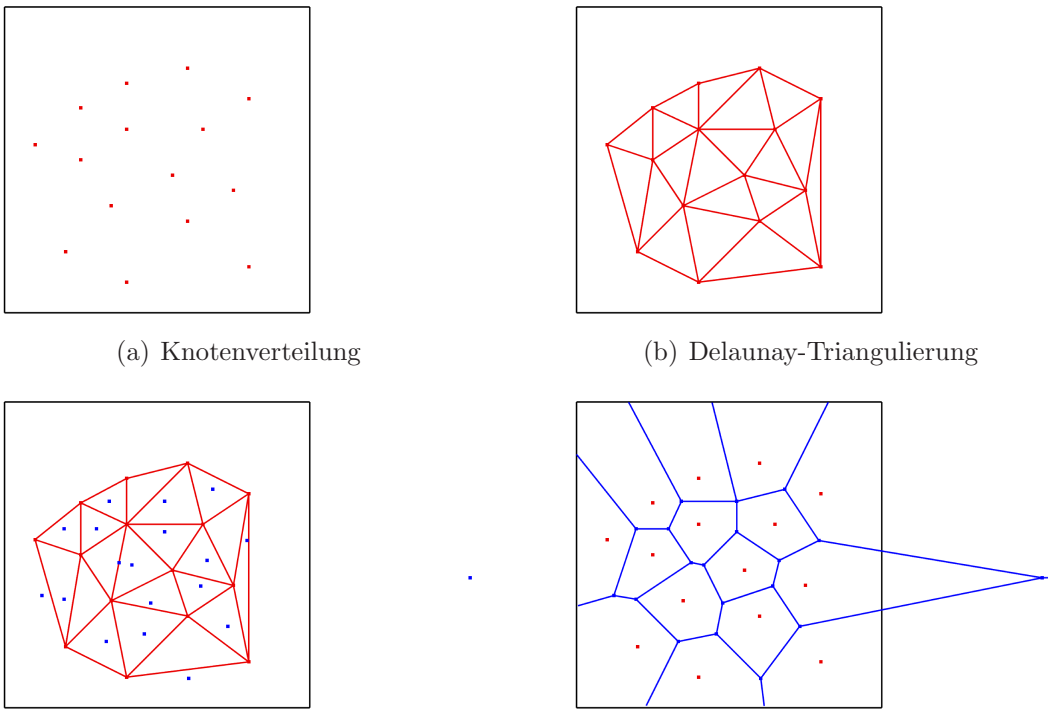
Die Voronoizelle wird stets von geraden Linien begrenzt, die jeweils auf der Mittelsenkrechten zwischen zwei Punkte liegen.
Für unsere Betrachtungen interessieren uns lediglich die *Voronoi-Punkte* von x. Dies sind die Ecken der Voronoizelle, welche durch den Schnitt von jeweils zwei Kanten entstehen. Dadurch wird jeder Voronoipunkt zum Mittelpunkt eines Kreises, der durch genau drei Punkte aus X geht und in dessen Inneren sich keine Punkte aus X befinden. Diese spezielle Eigenschaft der Voronoizelle

werden wir später noch benötigen.
Die Voronoizelle ist nicht immer komplett geschlossen, am Rand der Knotenmenge kann es zu halboffenen Zelle kommen (siehe Abb. 4(d)). Für den Fall, dass X degeneriert ist, d.h. alle Punkte aus X auf einer Geraden liegen, gibt es lediglich offene Voronoizellen ohne Ecken. Wir werden jedoch X als nicht degeneriert voraussetzen, so dass zu jedem $x \in X$ mindestens ein Voronoipunkt existiert.
Die Menge aller Voronoipunkte von X mit den sie verbindenden Kanten bezeichnen wir als *Voronoidiagramm* von X.
Um die Voronoipunkte zu berechnen, betrachten wir zunächst den zum Voronoidiagramm dualen Graphen, die *Delaunay-Triangulierung*.

Definition 2.6. Als *Triangulierung* einer Knotenmenge X bezeichnen wir einen Graphen, bei dem jeweils drei Knoten $x, y, z \in X$ mit den (ungerichteten) Kanten (x, y), (x, z) und (y, z) verbunden sind. Dabei darf es zu keinen Überschneidungen von Kanten kommen. Die drei so verbundenen Knoten bezeichnen wir als *Dreieck* (x, y, z) $(\triangle xyz)$.

Definition 2.7. Eine Triangulierung heißt *Delaunay*-Triangulierung, wenn sich innerhalb des Umkreises von $\triangle xyz$ keine Knoten befinden. x, y und z liegen als Ecken des Dreiecks stets auf dem Umkreis.
Aus den Eigenschaften der Voronoizelle folgt direkt, dass es in jedem Fall eine Delaunay-Triangulierung von X gibt, falls X nicht degeneriert ist.
Wir bestimmen nun zunächst die Delaunay-Triangulierung $T_D(X)$ zu X (Abb. 4(a) und 4(b)). Jeder Mittelpunkt (des Umkreises) eines Dreicks aus $T_D(X)$ bildet dann eine Ecke im *Voronoidiagramm* (Abb. 4(c) und 4(d)).
Um die Voronoipunkte zu x zu bestimmen, betrachten wir $T_x \equiv T_D(X)|_x$, die Menge aller Dreiecke aus T_D, die x enthalten. Nun berechnen wir zu jedem $t_i \in T_x$ den Schnittpunkt M_i der Mittelsenkrechten.
Dies liefert uns die Voronoipunkte $V^*(x)$, die wir nun für die dynamische Anpassung der Knotenmenge verwenden.

Definition 2.8. Ein Knoten $x \in X$ wird *verfeinert*, indem die *Voronoipunkte* von x zu X hinzugefügt werden, X wird ersetzt durch $X \cup V_x^*$.
In der numerischen Simulation wird uns eine exakte Einhaltung dieser Verfeinerungsvorschrift vor einige Probleme stellen. Wie wir in Abb. 4(c) und 4(d) sehen, kann es unter ungünstigen Umständen zu sehr großen Voronoizellen kommen. Dies geschieht, wenn die Delaunay-Triangulierung am Rande der Knotenmenge lange schmale Dreiecke liefert. Dann liegt der Schnittpunkt der Mittelsenkrechten sehr weit von x entfernt. Dies kann am Rand zu schlechten Ergebnisse bei der Rekonstruktion der Konzentration führen. Daher werden wir grundsätzlich nur Voronoipunkte zur Knotenmenge hinzufügen, wenn diese



(a) Knotenverteilung

(b) Delaunay-Triangulierung

(c) Delaunay-Triangulierung mit Voronoipunkten

(d) Voronoidiagramm mit Knoten

Abb. 4: Delaunay-Triangulierung & Voronoi-Diagramm

innerhalb von Ω liegen. Daher werden wir die Ersetzungsvorschrift der Definition (2.8) ändern in $X \mapsto X \cup (V_x^* \cap \Omega)$.
Für die Konzentration in den neu eingefügten Knoten liegt es nahe, diese aus den bekannten umliegenden Knoten zu berechnen. Wir werden also einfach die Konzentrationen für $V^*(x)$ durch die lokale Reproduktion s bestimmen. Für jedes neue $\xi \in \Omega$ wird die Konzentration definiert als $c(t, \xi) := s_{\mathcal{N}(\xi)}(\xi)$.

2.6.2 Praktische Anwendung

Bei der praktischen Umsetzung der Adaption stehen wir vor einer wichtigen Entscheidung. Wir haben zwei Möglichkeiten, die beschriebenen Methoden anzuwenden:
(a) Wir verwenden stets die Menge X^t als Basis für die Adaption,
(b) Wir verwenden die schon modifizierte Menge $\bar{X}^t$.
Der Fall (a) ist recht einfach zu handhaben. Wir werden zunächst alle Punkte ermitteln, die wir neu einfügen müssen. Die Konzentrationen in diesen Punkten bestimmen wir durch lokale Reproduktion auf X^t. Wir werden außerdem alle Punkte markieren, die auszudünnen sind. Danach führen wir in einem einzigen Schritt das Löschen der überflüssigen sowie das Einfügen der neuen Punkte durch. Wir können vor der Adaption einmal alle Nachbarschaften sowie das Voronoidiagramm zu X^t bestimmen und zwischenspeichern.
Methode (b) verspricht eine höhere Genauigkeit der Adaption, da wir die schon vollzogenen Änderungen sofort mit einbeziehen.
Hier müssen wir jedoch erheblich mehr Aufwand betreiben. Mit jedem gelöschten oder eingefügten Knoten ändert sich natürlich die Gesamtstruktur (Triangulierung, Voronoidiagramm, Nachbarschaften) der Daten und damit auch die lokalen Fehler. Wir können daher nicht die benötigten Daten zwischenspeichern und bei Bedarf wieder abrufen, sondern müssen diese Informationen sehr aufwändig pflegen. Dieser Aufwand liegt deutlich höher als die zu erwartetende qualitative Verbesserung, so dass wir uns schon aus Kostengründen auf Methode (a) konzentrieren werden.

2.7 Clipping

Sowohl bei der Advektion als auch bei der Adaption arbeiten wir mit Methoden zur Funktionsreproduktion, um die Konzentrationen zwischen den Knoten zu berechnen. Da wir bei unseren Betrachtungen eine initiale Knotenverteilung

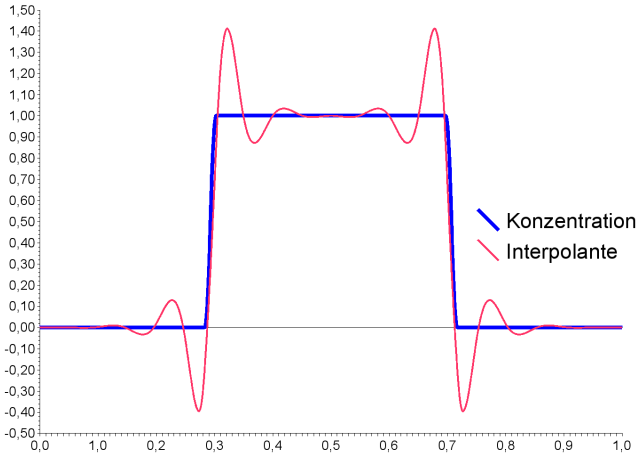


Abb. 5: Interpolationsfehler

mit zugeordneten Konzentrationen verwenden, gilt

$$c^0(\xi) \in [c_{\min}, c_{\max}] =: C \text{ mit } 0 \leq c_{\min} \leq c_{\max} \quad \forall \xi \in \Omega\,,$$
$$c_{\min} = \min_{\xi \in \Omega} c^0(\xi)\,,$$
$$c_{\max} = \max_{\xi \in \Omega} c^0(\xi)\,.$$

Bedingt durch die Eigenschaften der RBF-Interpolante bzw. der MLS-Approximante kann es jedoch vorkommen, dass $c(t, x)$ nicht in C liegt und insbesondere auch negativ werden kann. Dies tritt besonders in Bereichen auf, in denen die Konzentration starken Schwankungen unterliegt. Abb. 5 zeigt das mögliche Verhalten der RBF-Interpolante verglichen mit der tatächlichen Konzentration im Querschnitt. Die Abbildung zeigt die zur vorgegebenen Konzentration gehörige RBF-Interpolante mit der Wendland-1-Funktion und $Q = 2$.
Wir werden in unseren Testreihen Ausgangsverteilungen mit sehr starken Konzentrationsschwankungen betrachten. Um ein Aufschaukeln des dargestellten Effekts zu verhindern, werden wir die möglichen Konzentrationswerte auf C reduzieren. Das *Clipping* definiert das Ermitteln der neuen Konzentration an der Stelle $\xi \in \Omega$ als

$$c(t, \xi) \equiv c := \begin{cases} c_{\min} & c < c_{\min}, \\ c_{\max} & c > c_{\max}, \\ c & \text{sonst.} \end{cases}$$

Bei der lokalen Reproduktion auf Nachbarschaften von x werden wir entsprechend mit lokalen und zeitabhängigen $c_{\min}$ und $c_{\max}$ arbeiten, also mit $c_{\min} = \min_{\xi \in \mathcal{N}(x)} c(t, \xi)$ und $c_{\max} = \max_{\xi \in \mathcal{N}(x)} c(t, \xi)$.

3 Numerische Ergebnisse

„Man darf nicht das, was uns unwahrscheinlich und unnatürlich erscheint, mit dem verwechseln, was absolut unmöglich ist.“
Carl Friedrich Gauß (1777 - 1855)

Ein wesentlicher Teil dieser Arbeit beschäftigt sich mit der praktischen Erprobung der beschrieben Methoden. Um eine möglichst optimale Konfiguration zu finden, werden wir zunächst die einzelnen Verfahren unabhängig voneinander testen. Dadurch haben wir die Möglichkeit, genauer auf die Vor- und Nachteile der Komponenten einzugehen. Zudem können wir die Programmparameter genauer auf die Erfordernisse abstimmen. Wir werden die für die Versuche verwendeten Parameter nach und nach einführen und ihre Relevanz erläutern. Eine Übersicht über alle verwendeten Parameter werden wir im Programmteil in Kapitel C.3 geben.

3.1 Anfangsbedingungen

Um die einzelnen Verfahren gut vergleichen zu können, werden wir lediglich mit einer geringen Zahl von Anfangsbedingungen arbeiten.
Diese wurden repräsentativ ausgewählt, um eine gute optische Darstellung zu gewährleisten.

3.1.1 Geschwindigkeitsfeld

Wir werden stets dasselbe Gebiet $\Omega = [-1,1] \times [-1,1] \subset \mathbb{R}^2$ betrachten. Wir verwenden das Geschwindigkeitsfeld
$\vec{a}(t,x) \equiv \vec{a}(x) = \omega \cdot (-x_2, x_1)$, $x = (x_1, x_2)$, $\omega \in \mathbb{R}$.
$\vec{a}$ beschreibt eine Rotation um den Ursprung in mathematisch positiver Richtung (Abb. 6). Das Geschwindigkeitsfeld ist offensichtlich *divergenzfrei*, da

$$\operatorname{div}(\vec{a}) = \frac{\partial(-x_2)}{\partial x_1} + \frac{\partial x_1}{\partial x_2} \equiv 0\,.$$

Ebenfalls leicht nachrechnen lässt sich die Lipschitz-Stetigkeit von $\vec{a}$, denn durch direktes Einsetzen erhalten wir

$$\|\vec{a}(x) - \vec{a}(y)\| = |\omega| \cdot \|x - y\|\,.$$

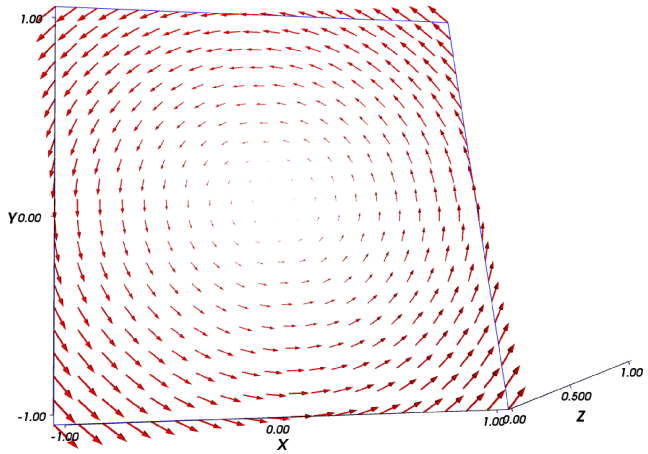


Abb. 6: Geschwindigkeitsfeld $\vec{a}$

Nun können wir mit der Lipschitzkonstanten $L \geq \frac{1}{|\omega|}$ die Voraussetzungen für den Satz von Picard-Lindelöf (Kapitel 1.2.2) erfüllen.
Der Skalar ω bestimmt zusammen mit der Zeitschrittweite τ die Geschwindigkeit, mit der sich in unseren Testreihen die Konzentration fortbewegt. Für das tangentiale Geschwindigkeitsfeld $\vec{a}$ setzen wir $\omega = \frac{2 \cdot \pi}{100 \cdot 12^3} = 0{,}3636 \times 10^{-4}$.
Um mit dem Geschwindigkeitsfeld die Upstreampunkte durch die in Kapitel 2.1 beschriebene Iteration approximieren zu können, müssen wir die Iterationstiefe N_{it} festlegen. Für den von uns eingesetzten Wert $\omega = 0{,}3636 \times 10^{-4}$ erreichen wir mit $N_{it} = 5$ bereits eine sehr gute Approximation des Upstreampunktes, es ist $\|x^- - \tilde{x}\|_2 < 10^{-12}$.

3.2 Zeitschrittweite

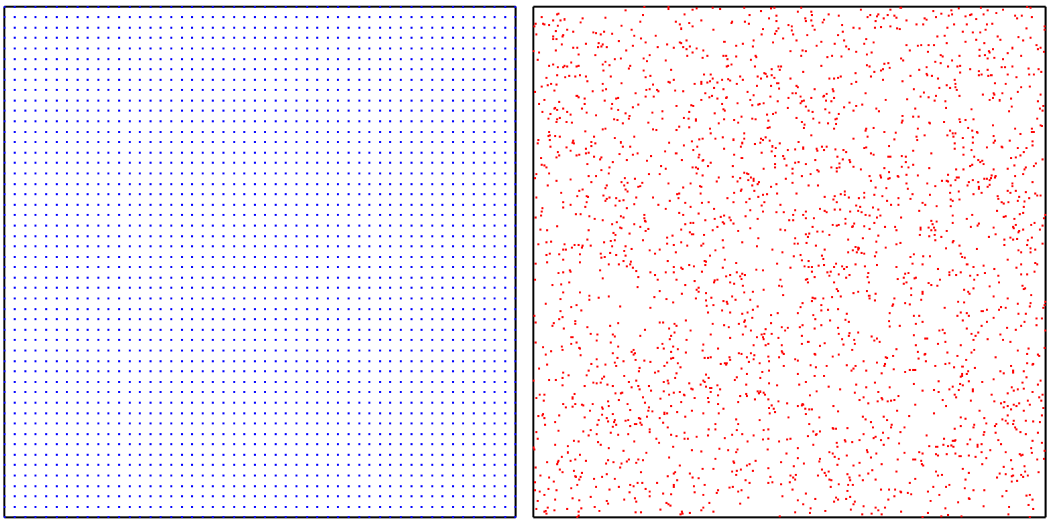
Wir können nun durch die Wahl von τ für das tangentiale Geschwindigkeitsfeld festlegen, nach wie vielen Zeitschritten z alle Partikel eine vollständige Rotation um den Ursprung vollziehen. Es gilt $z = \lceil \frac{2\pi}{\omega \cdot \tau} \rceil$.

τ	$\backsim z$	Rotation pro Zeitschritt
960	180	$\frac{\pi}{90}$
480	360	$\frac{\pi}{180}$
80	2.160	$\frac{\pi}{1.080}$

3.2.1 Knotenverteilung

Um die Abhängigkeit der Verfahren von der Anzahl der Knoten zu dokumentieren, verwenden wir Verteilungen mit

$N \in \{2.500; 10.000; 40.000; 90.000; 250.000; 490.000\}$ Knoten. Für jede der Knotenzahlen werden wir eine Ausgangsverteilung mit zufällig über Ω verteilten Knoten betrachten und eine mit Knoten, die jeweils auf den Ecken eines symmetrischen Gitters über Ω liegen. Die Abbildung 7 zeigt beispielhaft die initialen Knotenverteilungen für jeweils 2.500 Knoten. Wir werden Testreihen mit zufälliger Anfangsverteilung mit *rnd* kennzeichnen, bei zu Beginn gleichmäßig angeordneter Knotenmenge verwenden wir die Bezeichnung *grid*.



(a) gleichmäßige Verteilung. $N = 2.500 grid$ (b) zufällige Verteilung. $N = 2.500 rnd$

Abb. 7: Anfangsverteilungen

3.2.2 Konzentrationen

Wir werden zwei repräsentative Fälle von Konzentrationen betrachten, um die getesteten Verfahren auf ihr Verhalten bei glatten Funktionen und bei unstetigen Funktionen zu untersuchen. Dabei verwenden wir ausschließlich auf $[0, 1]$ skalierte Konzentrationen.
Als glatte Ausgangsverteilung werden wir die Wendland-2-Funktion $\phi(r) = (1 - r)^4(4r + 1)$ verwenden. Wir werden die Funktion, die ihr Maximum in (0/0) annimmt und einen Support-Radius von 1 hat, skalieren, um sie an unsere Erfordernisse anzupassen. Dazu verschieben wir die Funktion um (0,5/0) und verringern den Support-Radius auf 0,45, so dass der Funktionskegel bei einer Rotation um den Ursprung vollständig im Gebiet $\Omega = [-1, 1]^2$ liegt.
Die Funktionsgleichung der von uns verwendeten Ausgangsverteilung ist also $\phi_W(r) = (1 - \frac{20}{9}r)^4(1, 8r + 1)$ mit $r = \|x_i - x_0\|_2$ und $x_0 = (0{,}5/0)$.

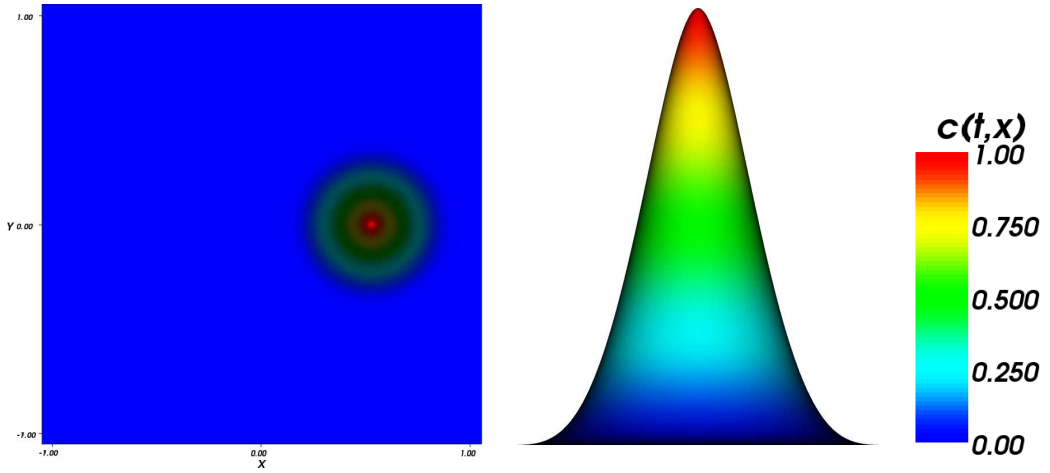


Abb. 8: Wendland-Funktion ϕ_W

Abbildung 8 zeigt die skalierte Wendland-Funktion. Von der Wendland-Funktion erwarten wir wegen der hohen Glätte eine gute Approximierbarkeit.
Konträr dazu verwenden wir die sogenannte *Slotted Disc*, die eine sehr stark unstetige Konzentrationsverteilung darstellt. Die *Disc* besteht aus einer kreisförmigen Scheibe, in deren Inneren die Konzentration 1, außerhalb hingegen 0 ist.
Der *Slot* ist ein rechteckiger Bereich, der die Disc überlagert und in dessen Inneren die Konzentration ebenfalls 0 ist.
Wir verwenden eine Slotted Disc (SD) mit dem Radius 0,325 um das Zentrum (0,5/0). Der Slot hat eine Ausdehnung von $0{,}08 \times 0{,}3$ und zeigt in Richtung (0,1).
Die zur Slotted Disc gehörige Funktion (Abb. 9) beschreiben wir als

$$\phi_{SD}(\xi) = \begin{cases} 0 & \|\xi - x_0\|_2 > 0{,}325,\ x_0 = (0{,}5/0)\,, \\ 0 & \xi \in [0{,}46; 0{,}54] \times [0{,}025; 0{,}325]\,, \\ 1 & \text{sonst.} \end{cases} \quad \text{mit } \xi \in [-1,1]^2$$

Die Slotted Disc wird uns zeigen, ob die angewandten Verfahren die scharfen Kanten und Spitzen erhalten werden.

3.3 Clipping

Wie bereits in Abschnitt 2.7 erwähnt, kann es bei der Reproduktion der Konzentration zu Fehlern kommen, die das Clipping der Werte erfordern.

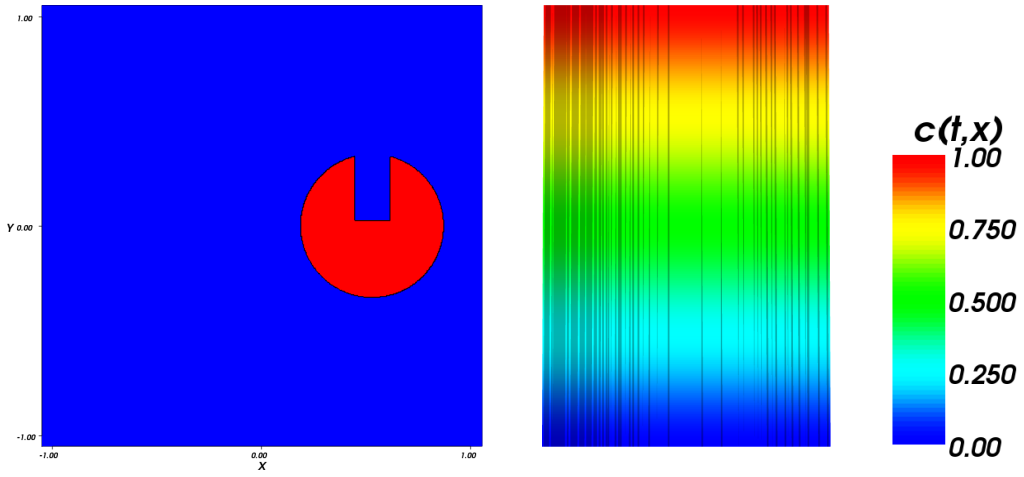
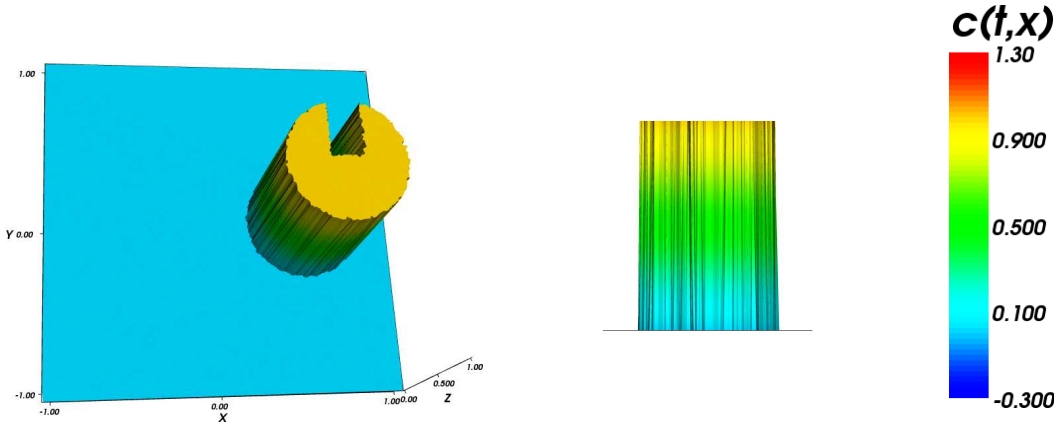


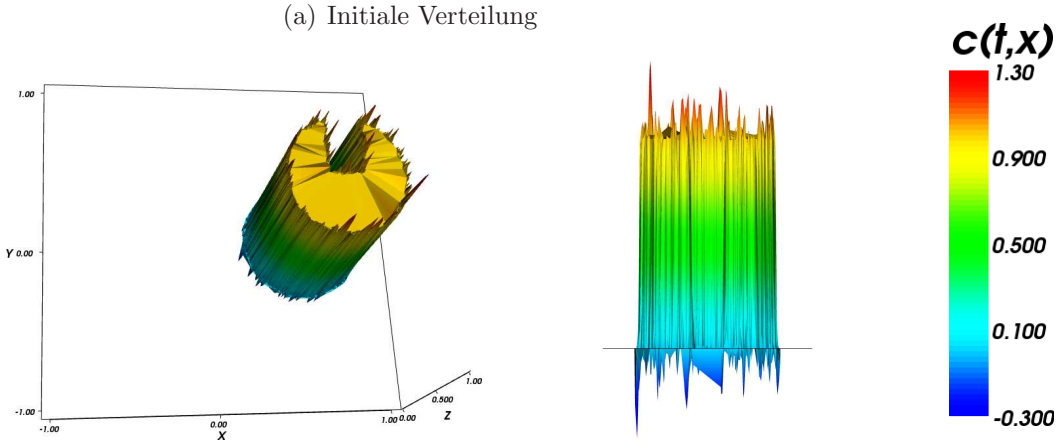
Abb. 9: Slotted Disc ϕ_{SD}

Abbildung 10 zeigt am Beispiel der Slotted Disc die Problematik bei deaktiviertem Clipping auf. Wir sehen dort zunächst die initiale Verteilung der Konzentration zur Zeit $t = 0$ bei zufällig verteilter Knotenmenge mit $N = 40.000$ Knoten (Abb. 10(a)). Es folgt die Konzentrationsverteilung nach einem durchgeführten Adaptionsschritt (Ohne Advektion!). Unten ist die Konzentration nach einer kompletten Rotation um den Ursprung abgebildet, wobei die Knotenmenge invariant gehalten wurde.
Um die deutlich sichtbaren Ausreißer zu vermeiden, werden wir in allen folgenden Testreihen das Clipping aktivieren.

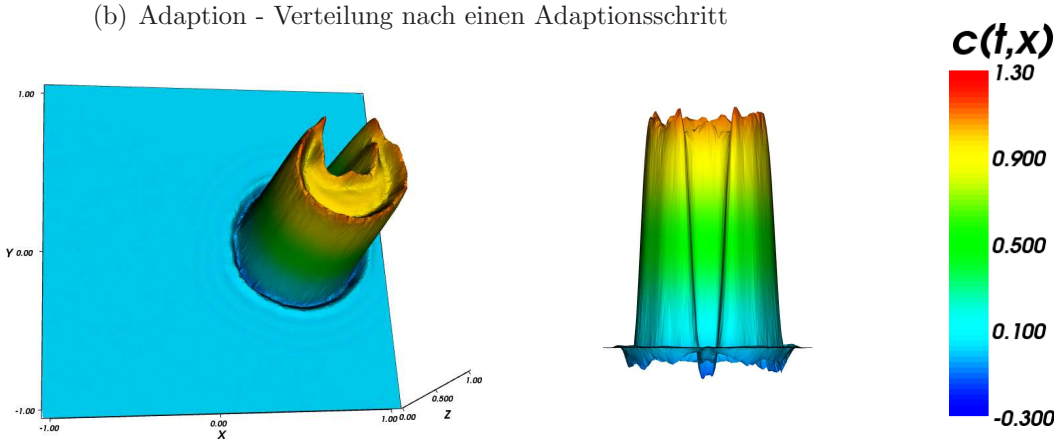
`Parameter Abb. 10:`	
`Initiale Knotenzahl:`	40.000***rnd***
`Nachbarschaften:`	$n = 20$
`Nur 10(b):`	
`Adaption:`	`RBF-Interpolation`
`RBF (Adaption):`	`Wendland-Funktion`
`Nur 10(c):`	
`Advektion:`	`RBF-Interpolation`
`RBF (Advektion):`	`Wendland-Funktion`
`Rotation:`	$2 \cdot \pi$
`Zeitschrittweite:`	$\tau = 960$
`Zeitintervall:`	$\mathcal{I} = [0, 180 \cdot \tau]$



(a) Initiale Verteilung



(b) Adaption - Verteilung nach einen Adaptionsschritt



(c) Advektion - Verteilung nach einer Rotation

Abb. 10: Konzentrationen bei deaktiviertem Clipping

3.4 Advektion

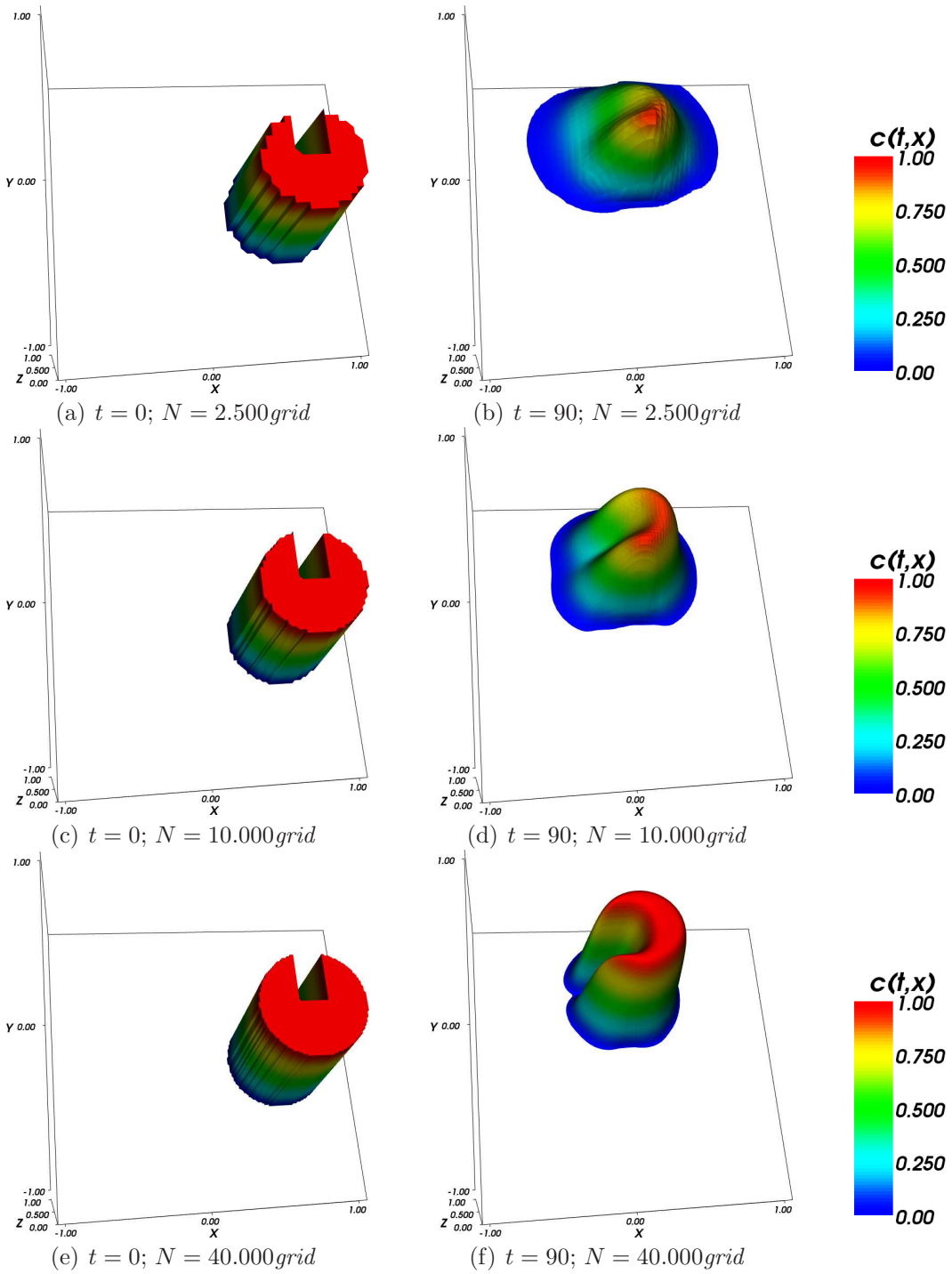
Wir betrachten nun die adaptionsfreie Advektion. Dabei werden wir als Ausgangsmenge sowohl eine zufällig verteilte Knotenmenge als auch eine Verteilung der Knoten auf den Ecken eines Gitters betrachten. Dies ermöglicht uns eine Betrachtung der Abhängigkeit des Verfahrens von der initialen Knotenverteilung.
Für eine sinnvolle dreidimensionale Darstellung der Testreihen werden wir die berechneten Knotenmengen optisch aufbereiten. Dazu legen wir ein symmetrisches Gitter aus 40.000 Punkten über das kleinste rechteckige Gebiet, das von den Knoten überdeckt wird. Wir berechnen dann die Interpolante in den Gitterpunkten mit den für die Testreihe verwendeten Parametern. Dadurch vermeiden wir Darstellungsfehler, die durch die Triangulierung einer sehr ungleichmäßig verteilten Knotenmenge entstehen. Außerdem werden wir ein manuelles Clipping für Konzentrationen $< 0{,}01$ durchführen, so dass wir geringe Unterschiede in der Konzentration besser erkennen können.
Für die Darstellung der einzelnen Knoten sowie der Delaunay-Triangulierung der Knotenmenge verwenden wir die stets die berechneten Werte ohne Clipping.

3.4.1 Advektion mit MLS

Wir beginnen unsere Betrachtung der adaptionsfreien Advektion mit der MLS-Approximation. Dabei betrachten wir zunächst den Einfluß der Knotenzahl auf das MLS-Verfahren. Wir verwenden die Slotted Disc mit dem tangentialen Geschwindigkeitsfeld. Abbildung 11 zeigt die Rotation mit den Knotenzahlen 2.500, 10.000 und 40.000. Erwartungsgemäß liefert eine höhere Anzahl Knoten ein genaueres Ergebnis.

`Parameter Abb. 11:`	
`Advektion:`	`MLS-Approximation`
`Nachbarschaften:`	$n = 5$
`Rotation:`	$\frac{\pi}{2}$
`Zeitschrittweite:`	$\tau = 480$
`Zeitintervall:`	$\mathcal{I} = [0, 90 \cdot \tau]$
`Konzentration:`	`Slotted Disc`
`Initiale Knotenzahl:`	$N = 2.500$ *grid* `(11(a), 11(b))`
	$N = 10.000$ *grid* `(11(c), 11(d))`
	$N = 40.000$ *grid* `(11(e), 11(f))`

Für die weiteren Tests ohne durchgeführte Adaption werden wir uns auf eine



(a) $t = 0$; $N = 2.500 grid$ (b) $t = 90$; $N = 2.500 grid$

(c) $t = 0$; $N = 10.000 grid$ (d) $t = 90$; $N = 10.000 grid$

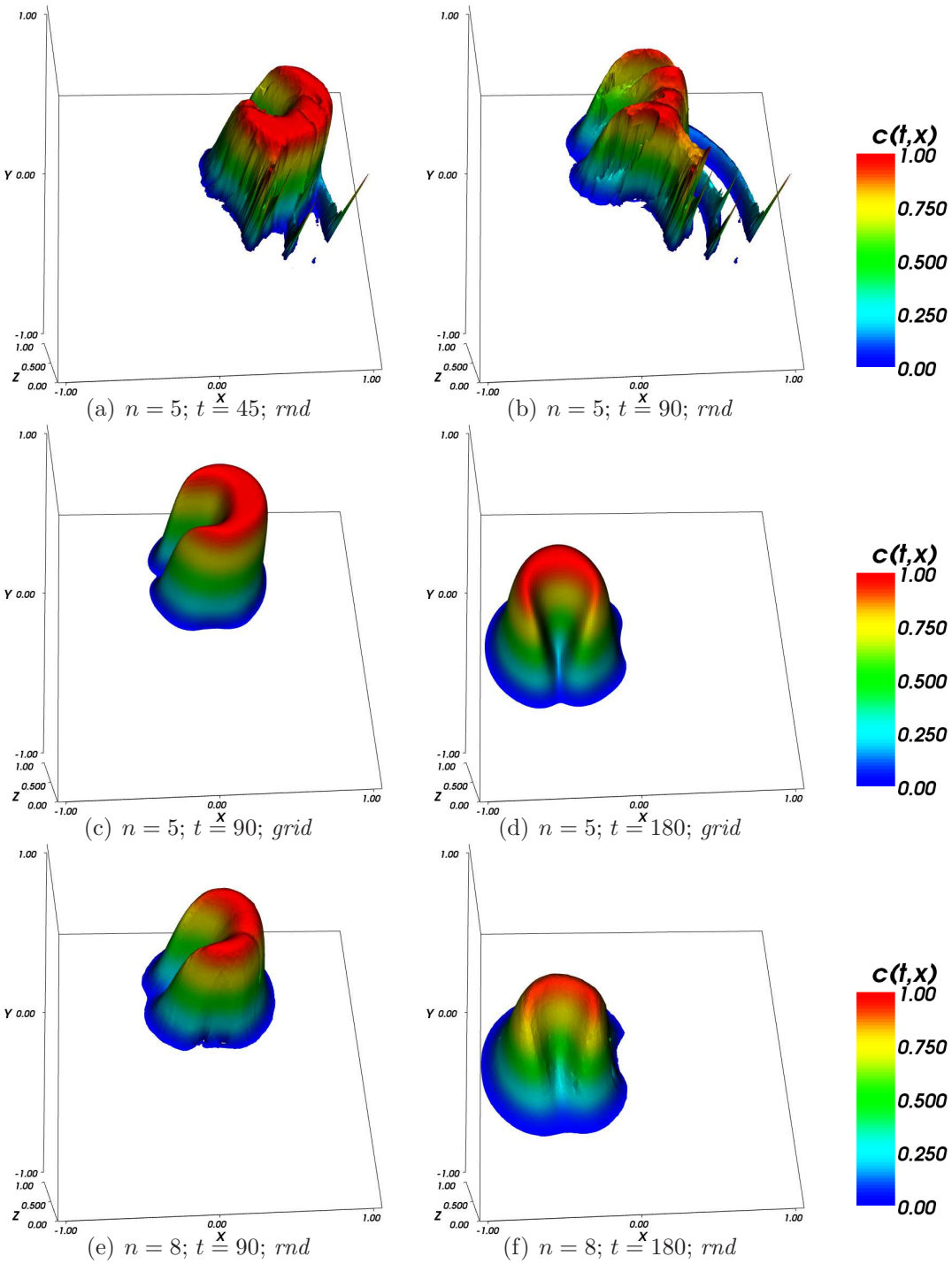
(e) $t = 0$; $N = 40.000 grid$ (f) $t = 90$; $N = 40.000 grid$

Abb. 11: Rotation der Slotted Disc mit unterschiedlichen Knotenzahlen

initiale Knotenzahl von 40.000 beschränken.
Wir werden nun untersuchen, wie die MLS-Approximation von der Verteilung der Knoten bzw. der Anzahl n der Punkte in den Nachbarschaften abhängt. Bei der Verwendung von nur 5 Punkten zur Berechnung der Approximante kommt es bei einer zufallsverteilten Knotenmenge zu einer ungewünschten Schattenbildung, wie Abb. 12(a) und 12(b) zeigen. Diese kann durch die Verwendung einer gleichmäßig verteilten Knotenmenge verhindert werden (Abb. 12(c), 12(d)). Da wir jedoch später die Knotenmenge dynamisch verändern möchten, ist diese Lösung nicht praktikabel, denn eine durch Adaption veränderte Knotenmenge ähnelt eher einer Zufallsverteilung als einer Gitterstruktur.
Wir versuchen daher, das Ergebnis durch die Hinzunahme von mehr Punkten in die Nachbarschaften, das Ergebnis auch für die zufällig gestreuten Knoten zu verbessern. Die Abbildungen 12(e) und 12(f) zeigen die Slotted Disc nach einer halben Rotation. Wie wir sehen, ist das Verfahren bezüglich der Knotenverteilung mit $n = 8$ wesentlich weniger anfällig für Störungen als mit $n = 5$.

Parameter Abb. 12:	
Advektion:	MLS-Approximation
Nachbarschaften:	$n = 5$ (12(a), 12(b), 12(c), 12(d))
	$n = 8$ (12(e), 12(f))
Rotation:	$2 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 180 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	40.000*rnd* (12(a), 12(b), 12(e), 12(f))
	40.000*grid* (12(c), 12(d))

Es zeigen sich auch bei der Verwendung von $n = 8$ noch Einflüsse der ungleichen Knotenverteilung, die sich im Laufe der Zeit verstärken. Abbildung 13 zeigt die Slotted Disc nach einer Viertel-Rotation unter Verwendung unterschiedlicher Knotenzahlen in den Nachbarschaften. Hierbei fällt auf, dass die Disc bei der Verwendung von mehr Punkten stärker abflacht, dafür jedoch weniger von Störungen in der Knotenverteilung beeinflusst wird.

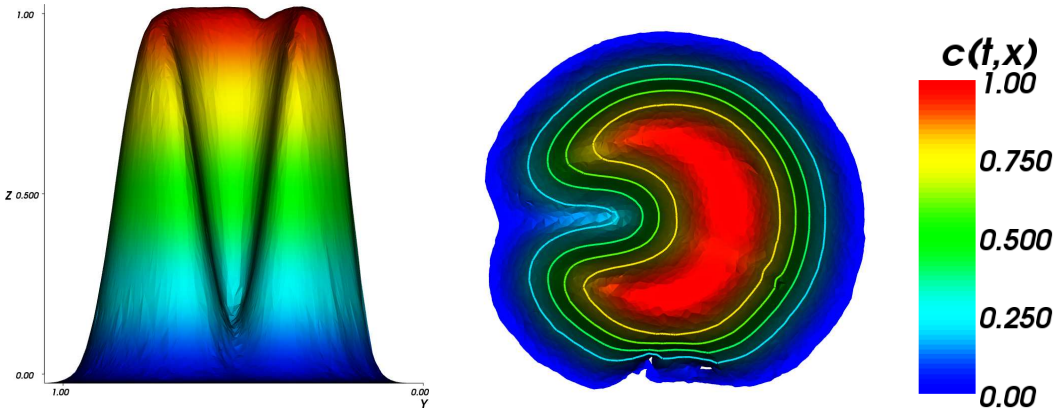


(a) $n = 5$; $t = 45$; *rnd* (b) $n = 5$; $t = 90$; *rnd*

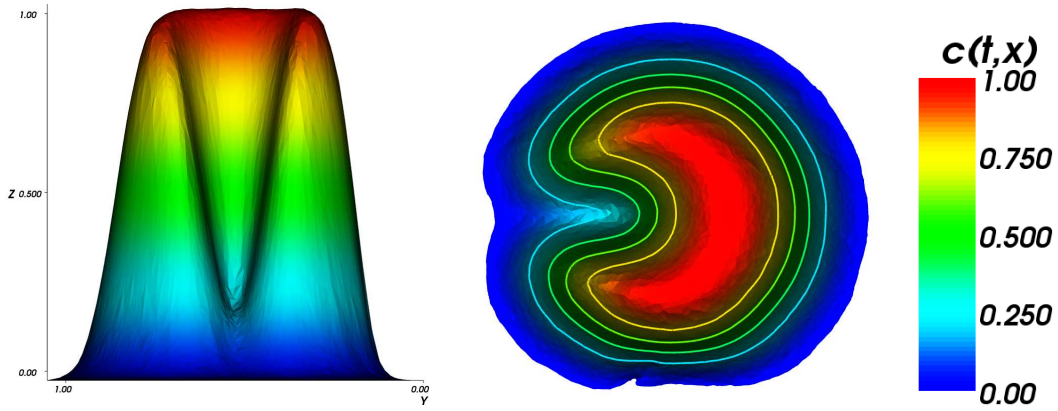
(c) $n = 5$; $t = 90$; *grid* (d) $n = 5$; $t = 180$; *grid*

(e) $n = 8$; $t = 90$; *rnd* (f) $n = 8$; $t = 180$; *rnd*

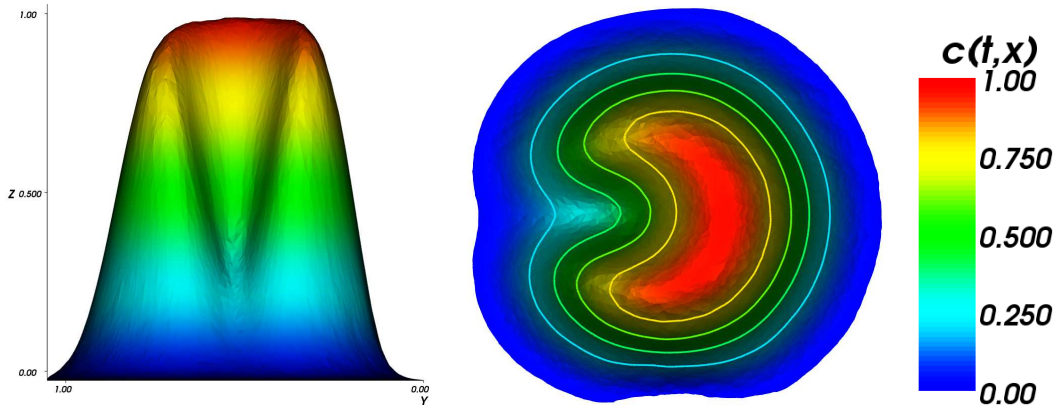
Abb. 12: Advektion mit unterschiedlicher Ausgangsverteilung und unterschiedlichen Nachbarschaften. $N = 40.000$



(a) $n = 8$



(b) $n = 10$



(c) $n = 20$

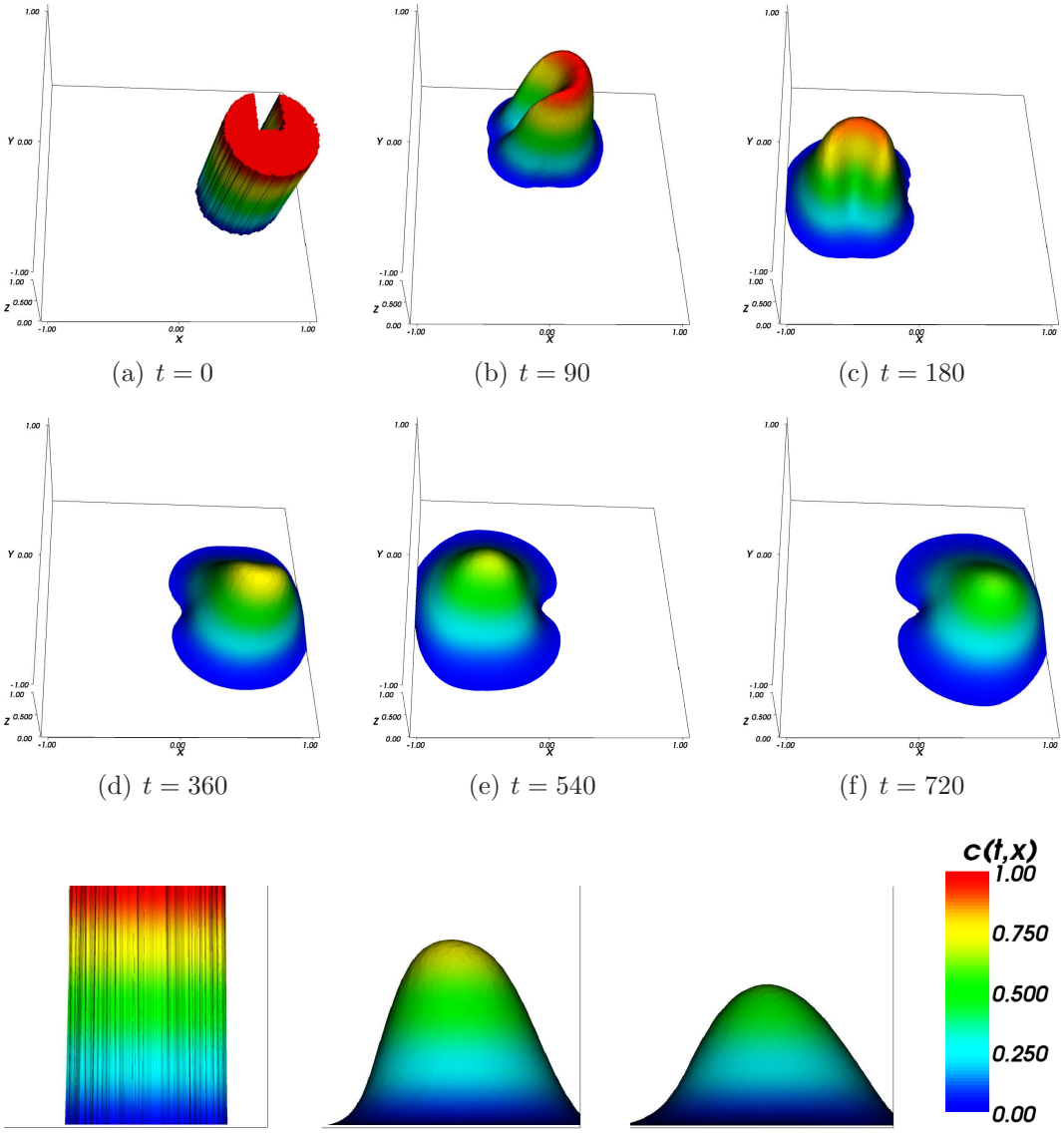
Abb. 13: Rotation um $\frac{\pi}{2}$ mit unterschiedlichen Nachbarschaften. $N = 40.000 rnd$

Parameter Abb. 13:	
Advektion:	MLS-Approximation
Nachbarschaften:	$n = 8$ (13(a))
	$n = 10$ (13(b))
	$n = 20$ (13(c))
Rotation:	$\frac{\pi}{2}$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 90 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	40.000_{rnd}

Wir müssen uns daher entscheiden, ob wir in den Nachbarschaften mehr Punkte für eine geringere Störanfälligkeit oder weniger Punkte für eine geringere Abflachung verwenden. Wir betrachten daher die Abflachung bei einer längeren Testreihe. Wie Abbildung 14 für $n = 15$ zeigt, beträgt bereits nach zwei Rotationen um den Ursprung die Höhe der Slotted Disc nur noch rund 60% der Ausgangshöhe. Dadurch werden die Konturen stark verwischt, so dass die Grundform verloren geht. Wir müssen daher, um später bei der Kombination von Adaption und Advektion brauchbare Ergebnisse erzielen zu können, n möglichst gering halten.

Parameter Abb. 14:	
Advektion:	MLS-Approximation
Nachbarschaften:	$n = 15$
Rotation:	$4 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 720 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	40.000_{rnd}

Wir werden nun noch überprüfen, wie sich die Advektion mit MLS bei einer glatten Ausgangsverteilung verhält. Abbildung 15 zeigt in einer Übersicht, dass dieselben Effekte wie bei der Slotted Disc auch bei der Wendland-Funktion auftreten. Außerdem wird deutlich, dass bei zufälliger Verteilung die Abflachung stärker ist als bei einer regelmäßigen.

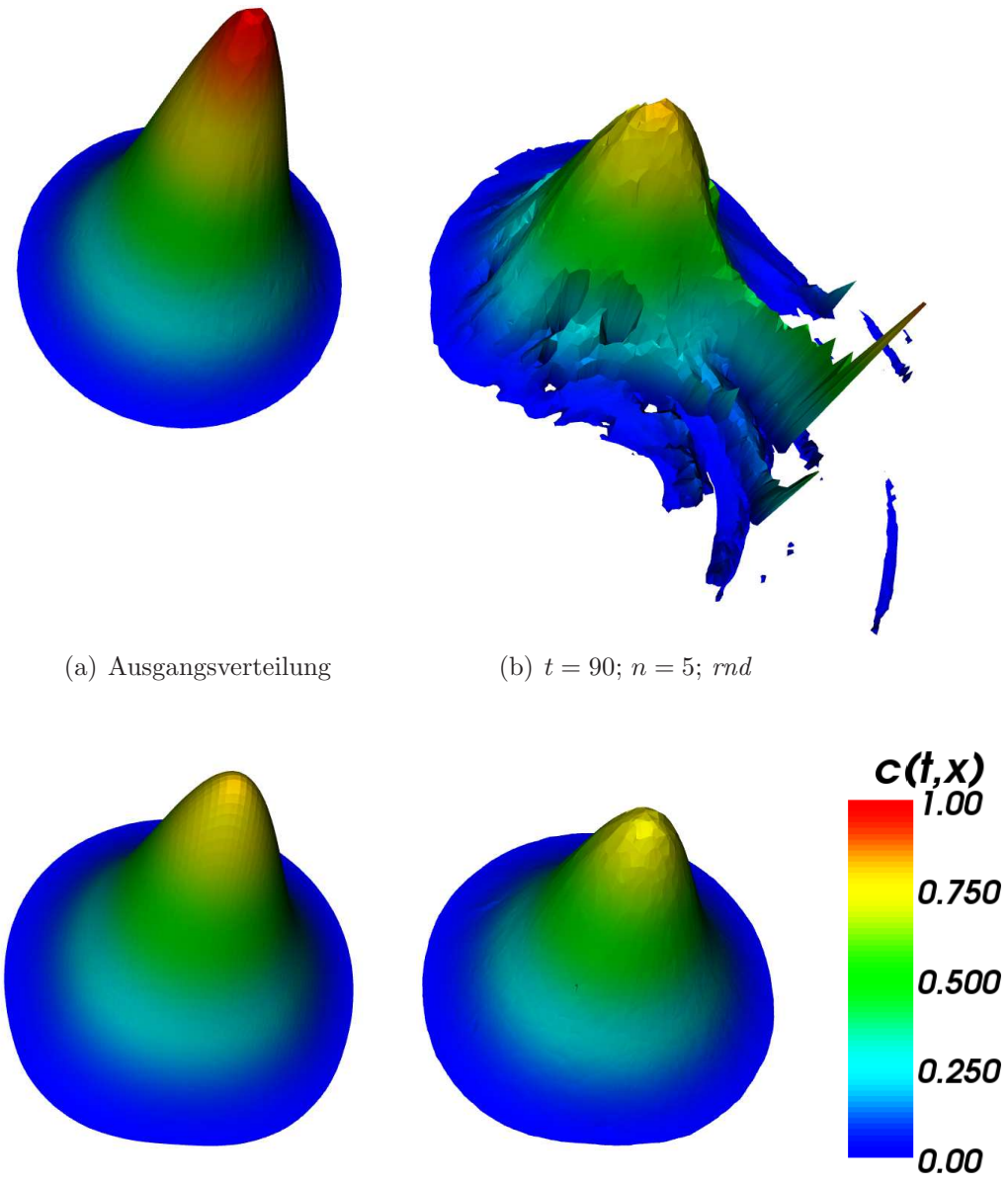


(a) $t = 0$ (b) $t = 90$ (c) $t = 180$

(d) $t = 360$ (e) $t = 540$ (f) $t = 720$

(g) $t = 0$ (h) $t = 360$ (i) $t = 720$

Abb. 14: Rotation der Slotted Disc mit Abflachung. $N = 40.000 rnd$



(a) Ausgangsverteilung (b) $t = 90$; $n = 5$; *rnd*

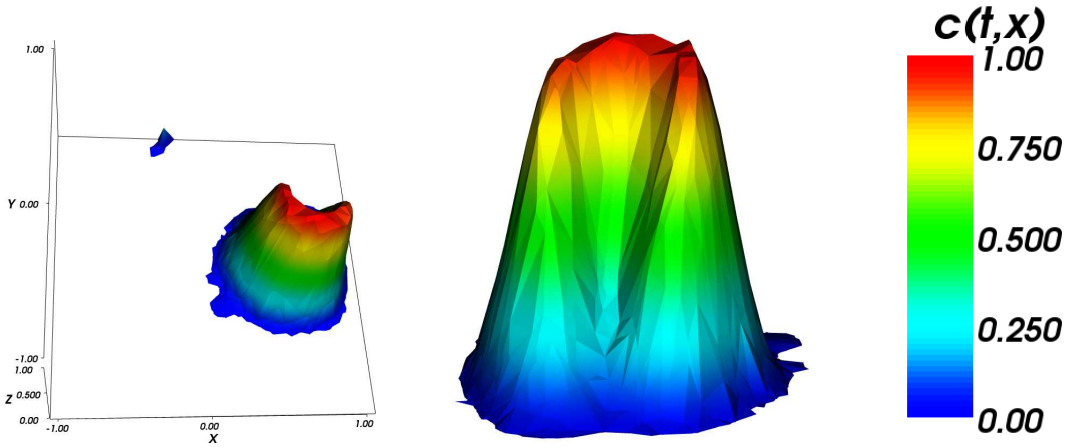
(c) $t = 90$; $n = 10$; *grid* (d) $t = 90$; $n = 10$; *rnd*

Abb. 15: Advektion der Wendland-Funktion. $N = 40.000$

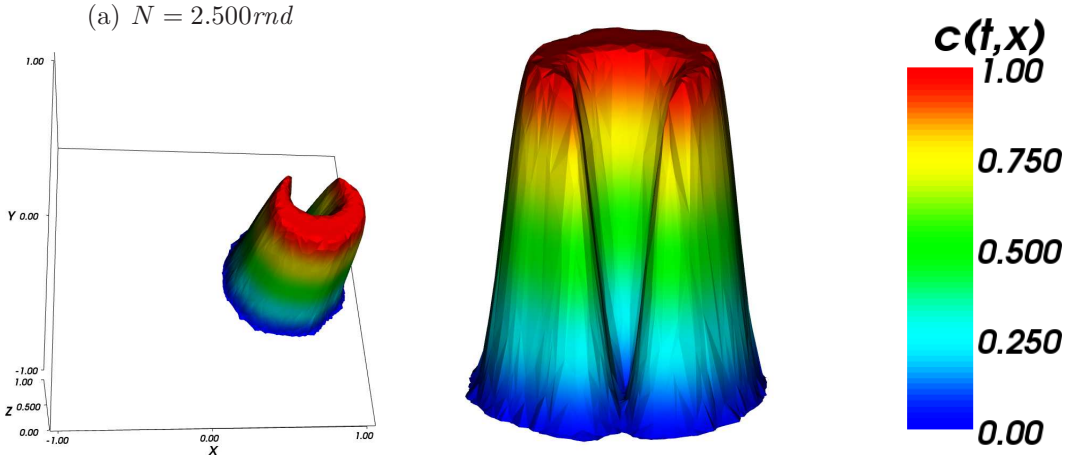
Parameter Abb. 15:	
Advektion:	MLS-Approximation
Nachbarschaften:	$n = 5$ (15(b))
	$n = 10$ (15(c), 15(d))
Rotation:	$\frac{\pi}{2}$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 90 \cdot \tau]$
Konzentration:	Wendland-Funktion
Initiale Knotenzahl:	40.000*rnd* (15(a), 15(b), 15(d))
	40.000*grid* (15(c))

3.4.2 Advektion mit RBF

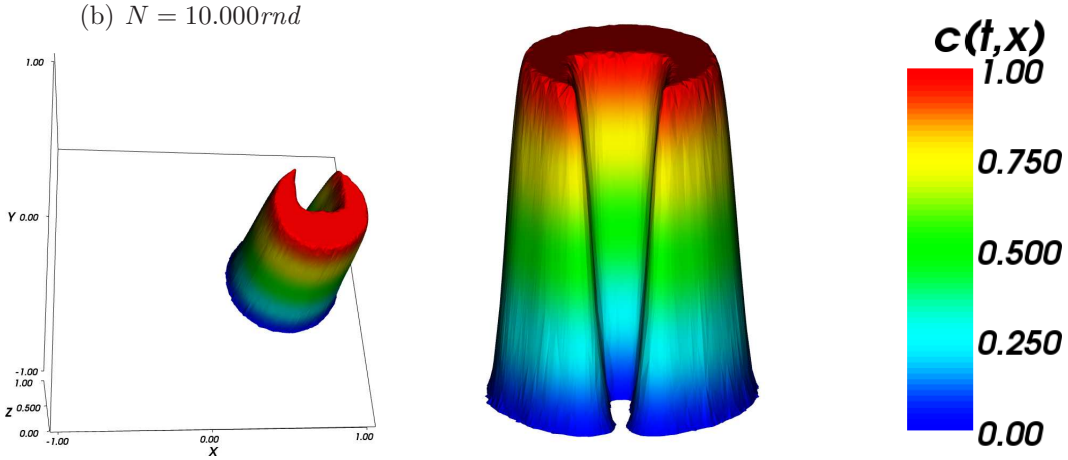
Auch für die RBF-Interpolation werden wir zunächst die Abhängigkeit von der Knotenanzahl und -verteilung bei der adaptionsfreien Advektion überprüfen. Wir vergleichen zunächst für eine regelmäßig verteilte Knotenmenge die RBF-Interpolation für beide von uns eingesetzten RBFs.
Die Abbildungen 16 und 17 zeigen jeweils die Slotted Disc nach einer vollständigen Rotation. Dabei variiert die Anzahl von Knoten zwischen 2.500, 10.000 und 40.000 mit einer Zufallsverteilung. Die Anzahl der Knoten in den Nachbarschaften liegt konstant bei $n = 20$. Es zeigt sich, dass bei Verwendung des Thin Plate Spline (Abb. 17) eine stärkere Kantenglättung erfolgt als bei der Wendland-Funktion (Abb. 16). Dieser Effekt wird insbesondere bei geringeren Knotenzahlen offensichtlich.
Die Unterschiede zwischen den verwendeten RBFs werden bei einer gleichmäßig verteilten Knotenmenge deutlich geringer. Während hier die Kanten der Slotted Disc etwas weniger geglättet werden, entsteht bei geringen Knotenzahlen ein schwacher Oszillationseffekt, so dass die Disc einen leichten Schatten nachzieht. Bemerkenswert ist, dass der Thin Plate Spline (Abb. 19) bei kleineren Knotenzahlen der Wendland-Funktion (Abb. 18) überlegen ist. Der Thin Plate Spline scheint also eher von der Lage der Knoten als von deren Anzahl abhängig zu sein, während die Wendland-Funktion robuster bei ungleich verteilten Knoten ist.
Wir vermuten daher eine bessere Eignung der Wendland-Funktion im Zusammenspiel mit der Adaption der Knotenmenge.



(a) $N = 2.500 rnd$

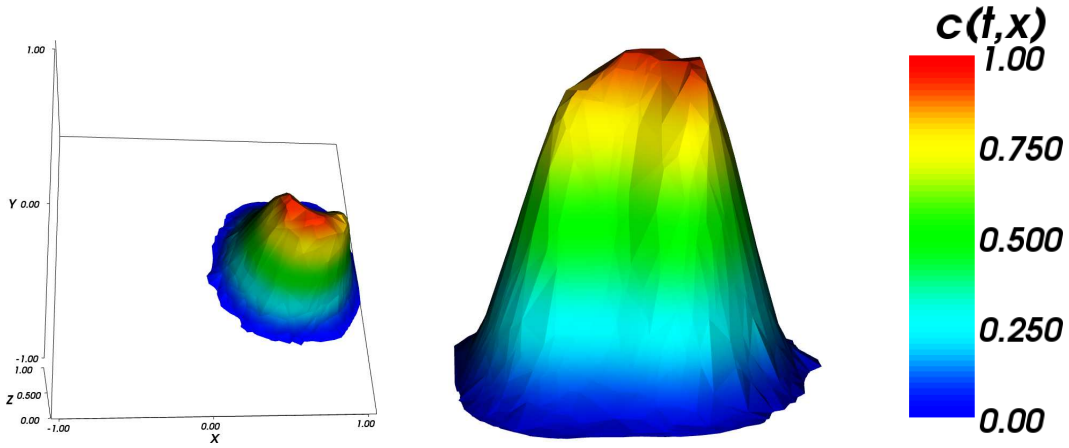


(b) $N = 10.000 rnd$

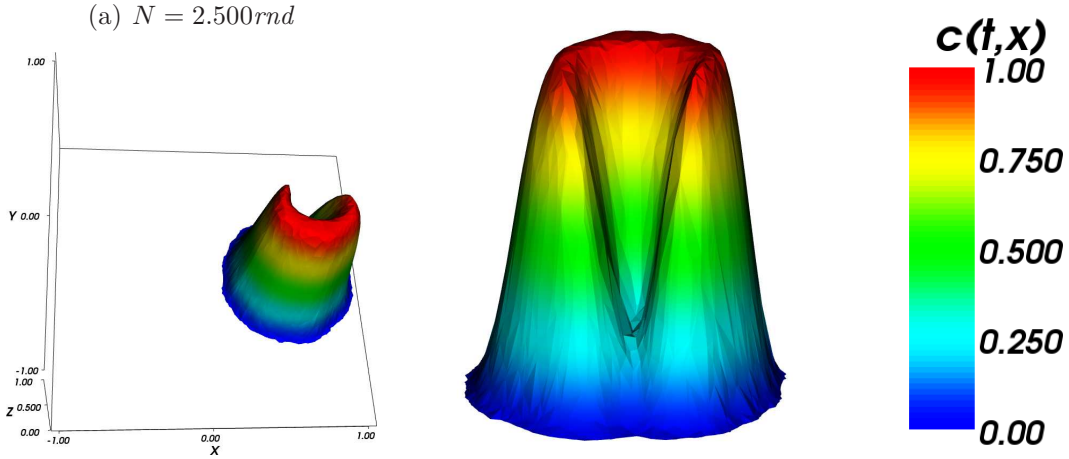


(c) $N = 40.000 rnd$

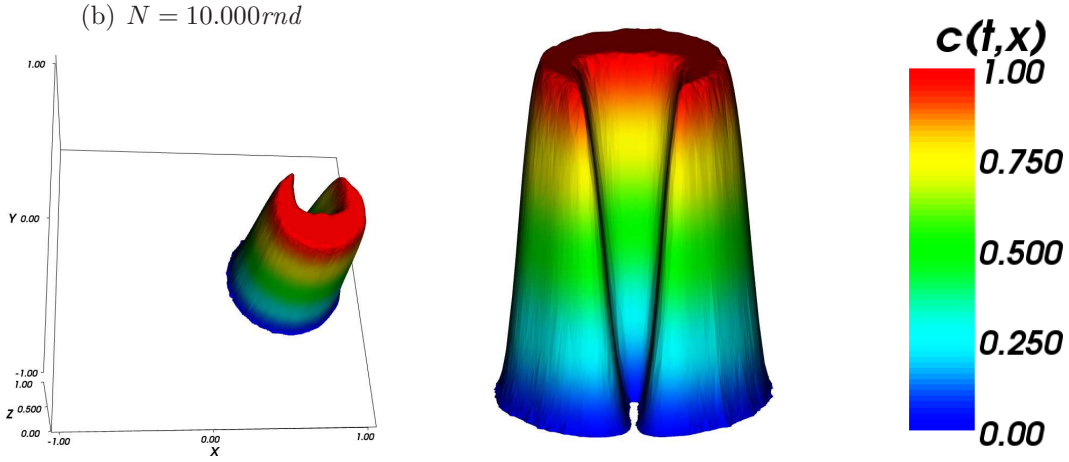
Abb. 16: Advektion mit der Wendland-Funktion



(a) $N = 2.500rnd$

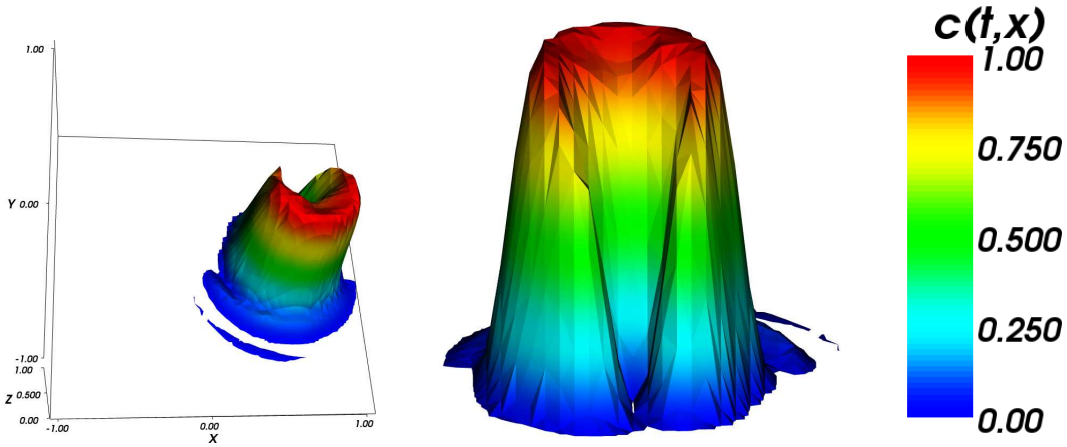


(b) $N = 10.000rnd$

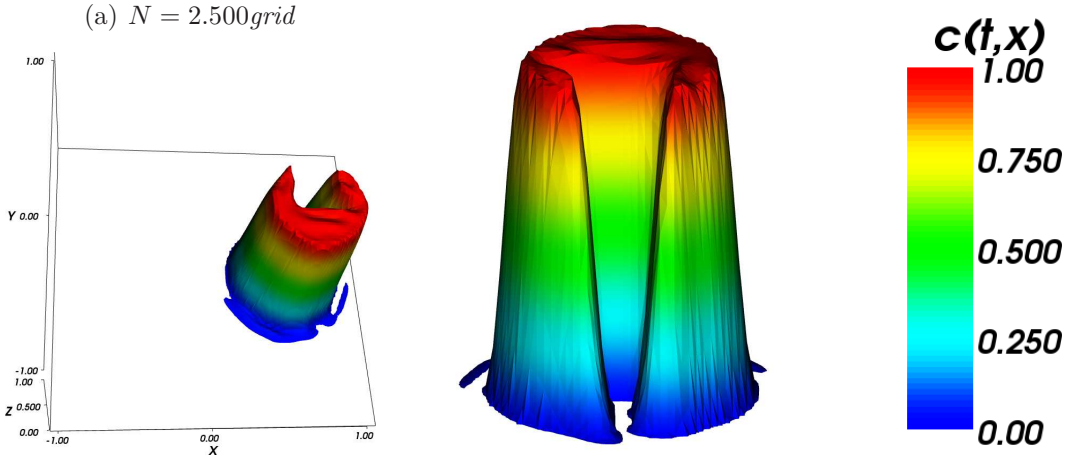


(c) $N = 40.000rnd$

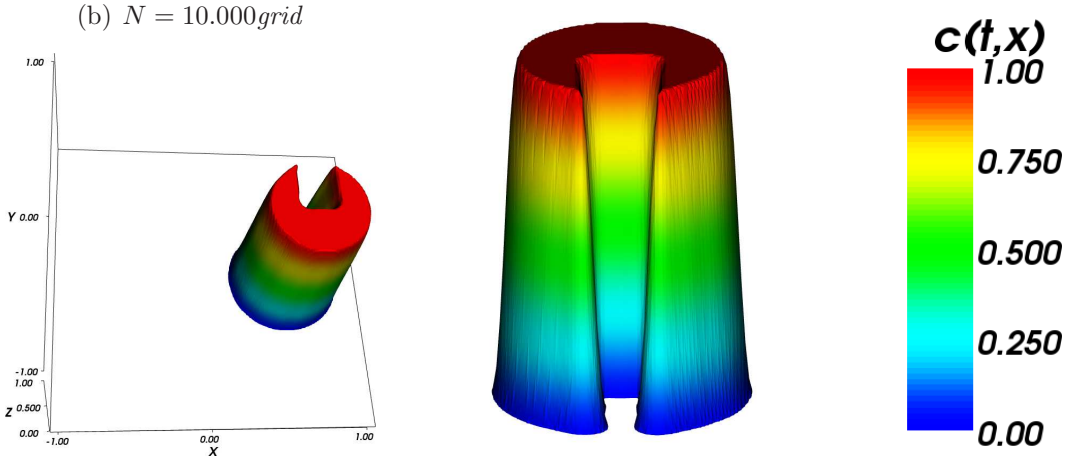
Abb. 17: Advektion mit dem Thin Plate Spline



(a) $N = 2.500 grid$

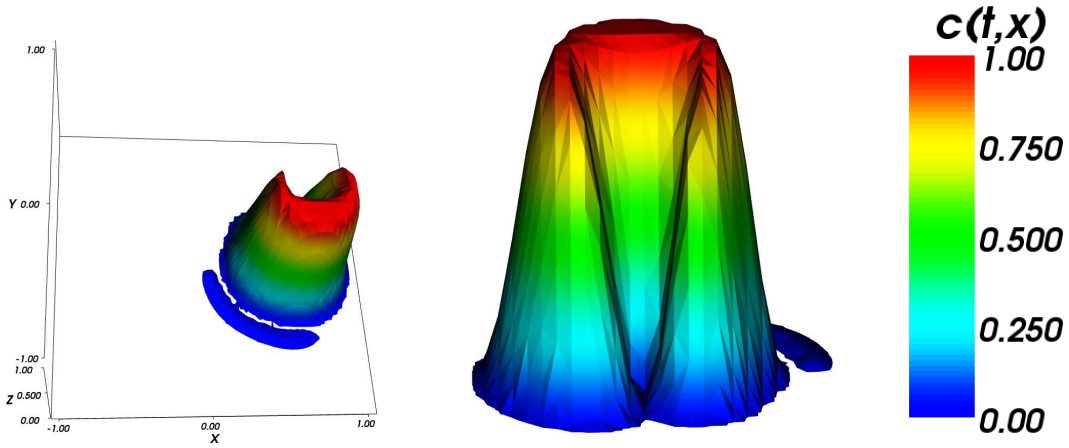


(b) $N = 10.000 grid$

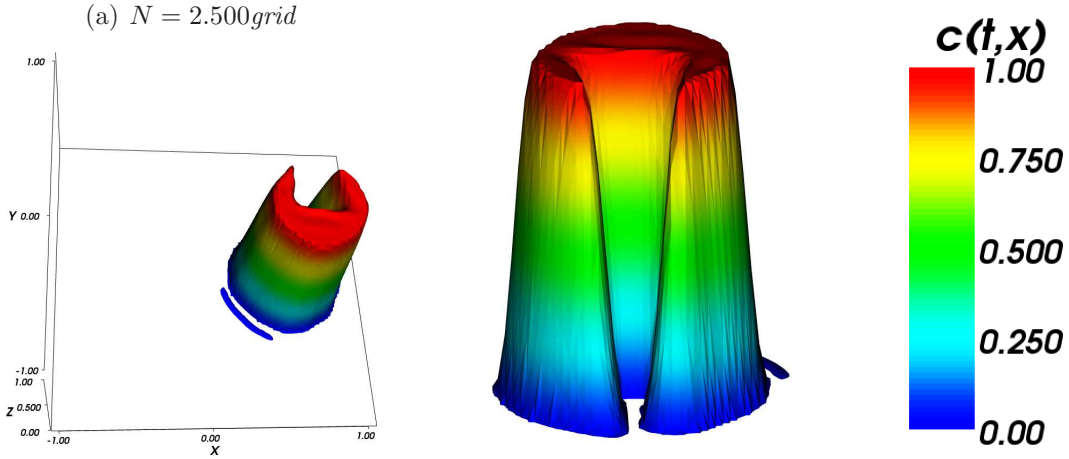


(c) $N = 40.000 grid$

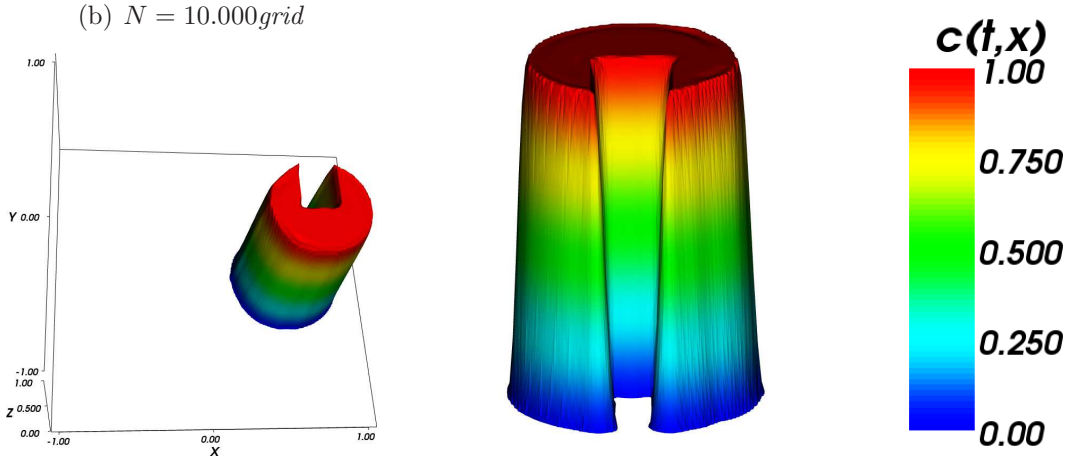
Abb. 18: Advektion mit der Wendland-Funktion



(a) $N = 2.500 grid$



(b) $N = 10.000 grid$



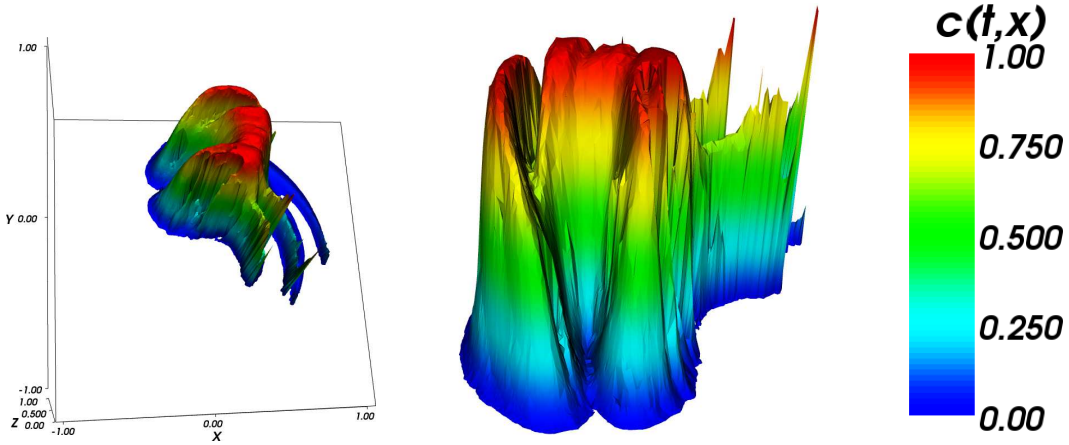
(c) $N = 40.000 grid$

Abb. 19: Advektion mit dem Thin Plate Spline

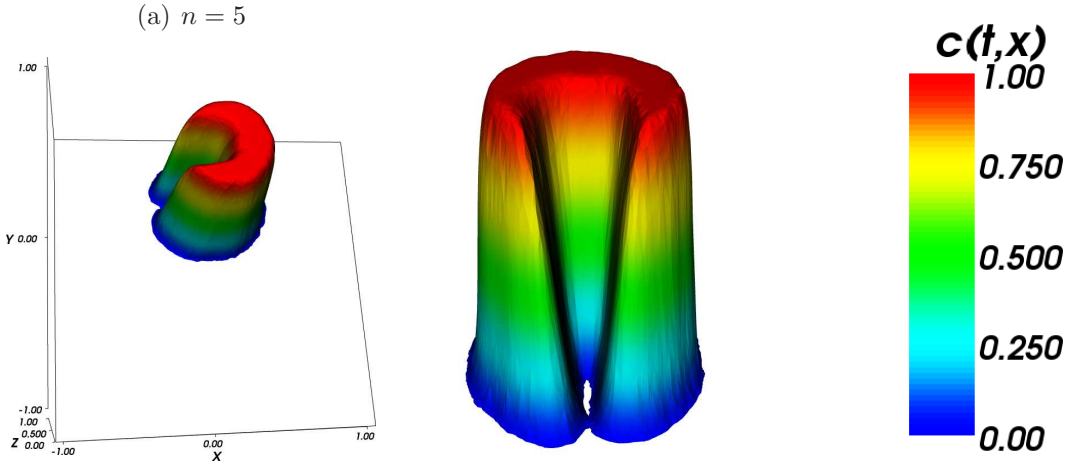
Parameter Abb. 16, 17, 18 und 19:	
Advektion:	RBF-Interpolation
RBF:	Wendland-Funktion (16, 18)
	Thin Plate Spline (17, 19)
Nachbarschaften:	$n = 20$
Rotation:	$2 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 360 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	2.500*rnd* (16(a), 17(a))
	10.000*rnd* (16(b), 17(b))
	40.000*rnd* (16(c), 17(c))
	2.500*grid* (18(a), 19(a))
	10.000*grid* (18(b), 19(b))
	40.000*grid* (18(c), 19(c))

Um die Abhängigkeit der RBF-Interpolation von der Anzahl der Knoten in den Nachbarschaften zu bestimmen, werden wir mit 40.000 Knoten rechnen. Hiervon erwarten wir eine recht gute Darstellungsqualität. Wir werden in Hinblick auf die später erfolgende Anpassung der Knotenmenge mit einer Zufallsverteilung arbeiten.
Abbildung 20 zeigt die Slotted Disc nach einer Viertel-Rotation mit der Wendland-Funktion als Basisfunktion. Wie bei der MLS-Approximation bilden sich bei zu kleinen Nachbarschaften Schatten, wie Abbildung 20(a) für $n = 5$ zeigt. Bei der Verwendung von $n = 10$ bzw. $n = 20$ verbessert sich jeweils das Ergebnis. Dabei kommt es bei wachsendem n im Gegensatz zur MLS-Approximation jedoch nicht zu einer stärkeren Abflachung, sondern die Kanten bleiben wesentlich besser erhalten. Derselbe Effekt tritt bei Verwendung des Thin Plate Spline als Basisfunktion auf.

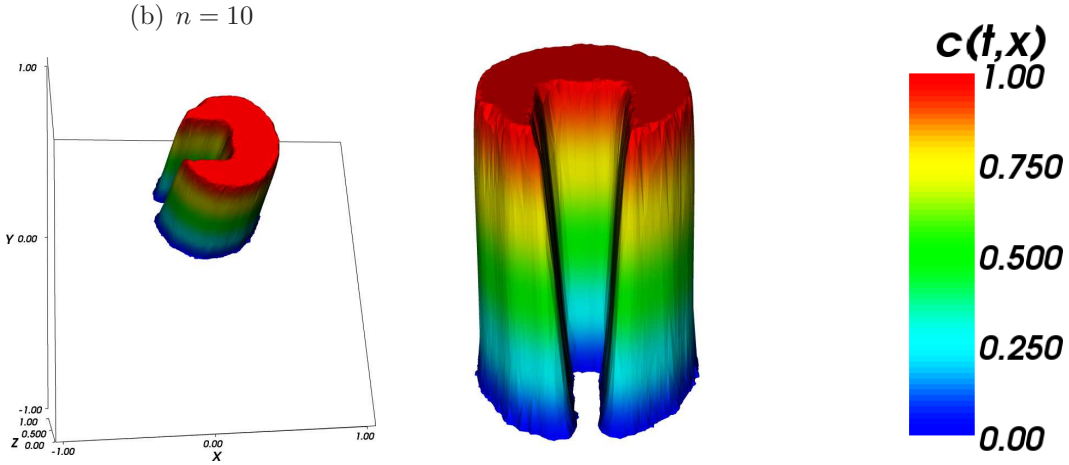
Parameter Abb. 20:	
Advektion:	RBF-Interpolation
RBF:	Wendland-Funktion
Nachbarschaften:	$n = 5$ (20(a))
	$n = 10$ (20(b))
	$n = 20$ (20(c))
Rotation:	$\frac{\pi}{2}$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 90 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	40.000*rnd*



(a) $n = 5$



(b) $n = 10$



(c) $n = 20$

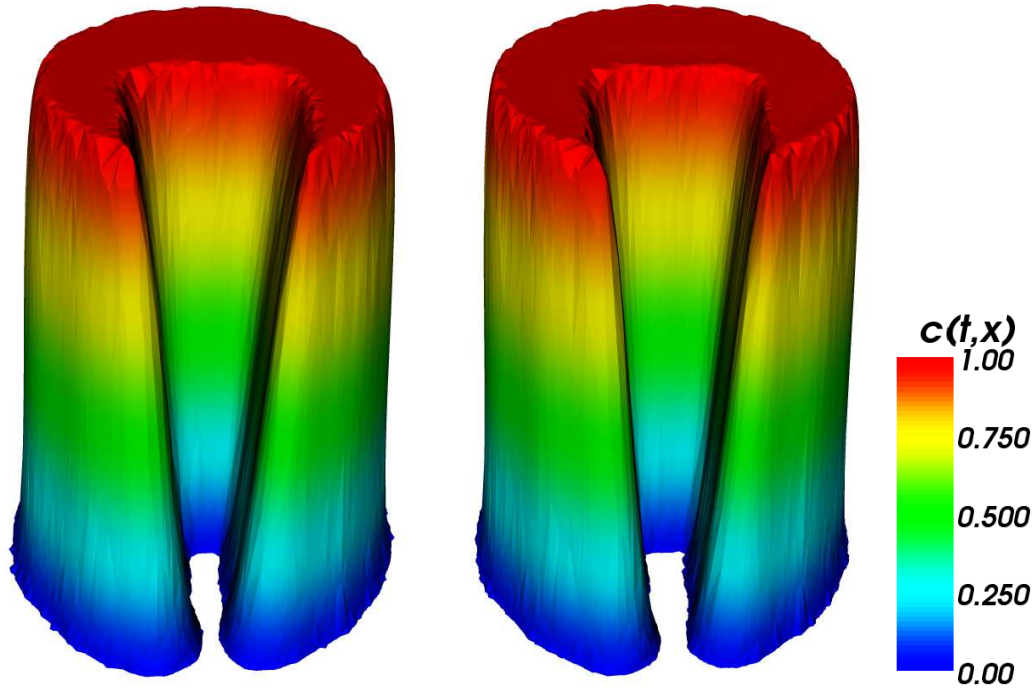
Abb. 20: Advektion mit der Wendland-Funktion und unterschiedlichen Nachbarschaften. $N = 40.000 rnd$

Durch eine weitere Erhöhung der Knotenzahl in den Nachbarschaften können wir das Ergebnis weiter verbessern. Abbildung 21 zeigt die Slotted Disc nach einer vollständigen Rotation mit $n = 20$ und $n = 30$. Wir sehen, dass die Kanten der Disc bei Verwendung des Thin Plate Spline (Abb. 21(c) und 21(d)) etwas stärker geglättet werden als bei Verwendung der Wendland-Funktion (Abb. 21(a) und 21(b)). Die Wendland-Funktion liefert hier mit $n = 20$ in etwa dasselbe Ergebnis wie der Thin Plate Spline mit $n = 30$.
Diese Qualitätsverbesserung durch größere Werte für n ist jedoch mit hohen Kosten verbunden, da bei der RBF-Interpolation die Laufzeit wesentlich von der Anzahl n der Knoten in den Nachbarschaften abhängt. Wir erwarten von der Adaption der Knotenmenge eine Verbesserung der Datenqualität, daher werden wir uns auf $n = 20$ beschränken mit der Möglichkeit, durch Hinzunahme weiterer Punkte das Ergebnis zu verbessern.

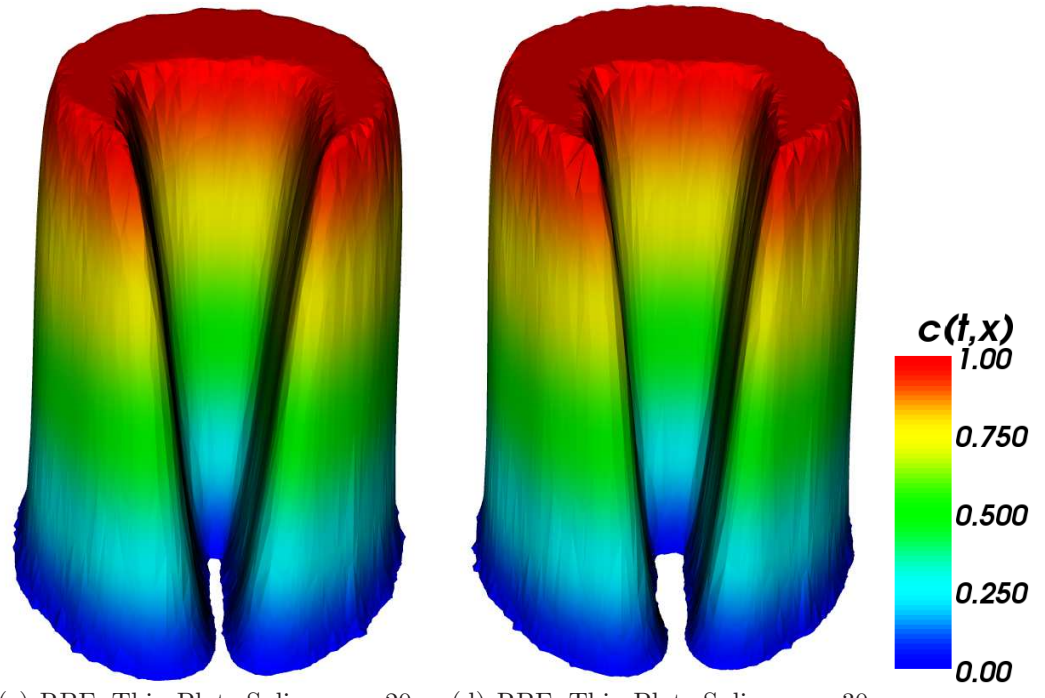
Parameter Abb. 21:	
Advektion:	RBF-Interpolation
RBF:	Wendland-Funktion (21(a), 21(b))
	Thin Plate Spline (21(c), 21(d))
Nachbarschaften:	$n = 20$ (21(a), 21(c))
	$n = 30$ (21(b), 21(d))
Rotation:	$2 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 360 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$40.000_{\boldsymbol{rnd}}$

Für die Durchführung von längeren Testreihen überprüfen wir das Verhalten der Advektion nach einigen kompletten Rotationen. Abb. 22 zeigt, dass auch nach fünf Umläufen die Slotted Disc noch sehr gut abgebildet wird. Es findet lediglich eine Abrundung der Konturen statt, die zu Beginn stärker ist und sich nach einigen Rotationen abschwächt. Eine Abflachung der Disc wie bei der MLS-Approximation findet nicht statt.

Parameter Abb. 22:	
Advektion:	RBF-Interpolation
RBF:	Wendland-Funktion
Nachbarschaften:	$n = 20$
Rotation:	$10 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 1.800 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$40.000_{\boldsymbol{rnd}}$

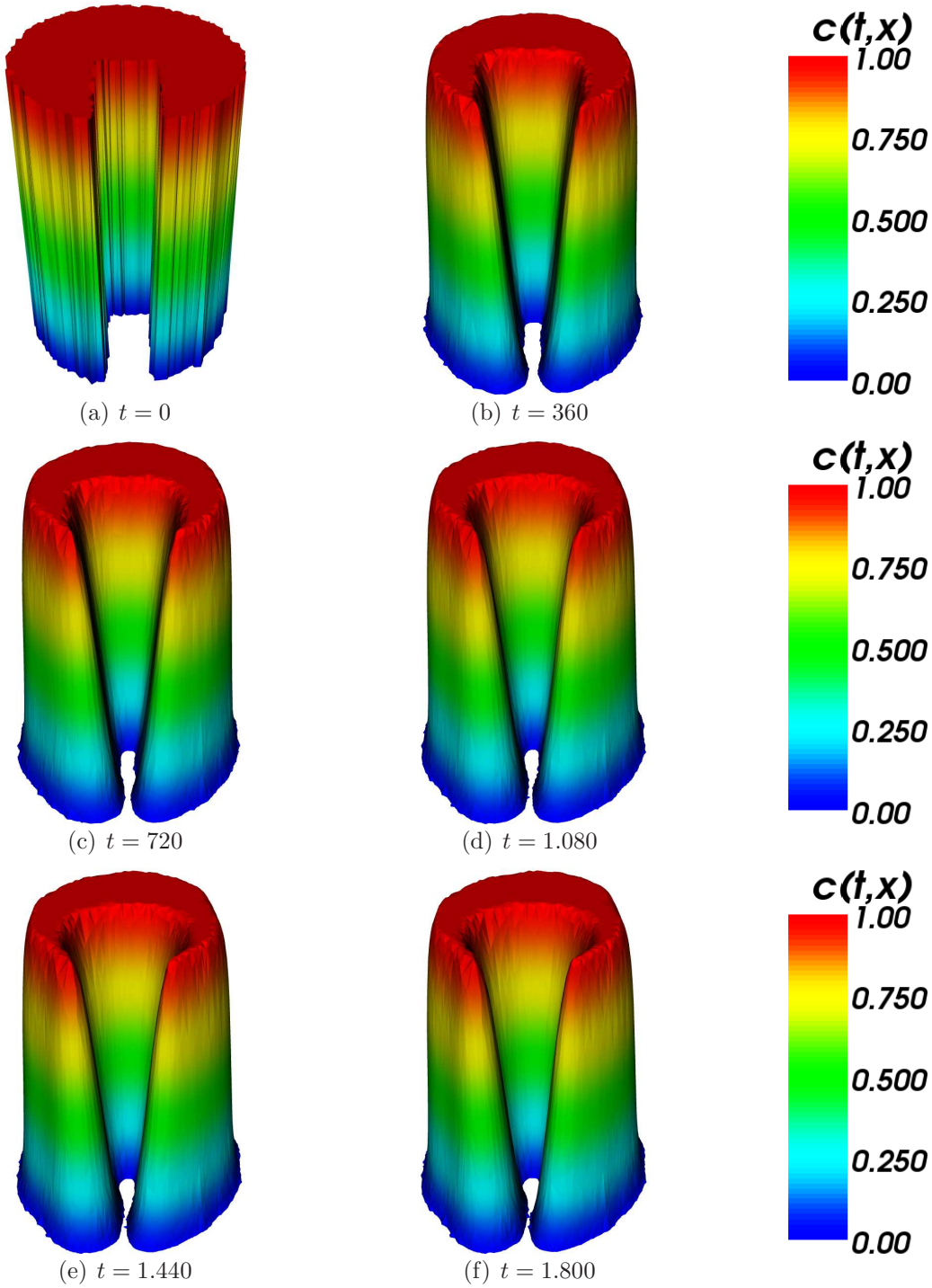


(a) RBF: Wendland-Funktion, $n = 20$ (b) RBF: Wendland-Funktion, $n = 30$



(c) RBF: Thin Plate Spline, $n = 20$ (d) RBF: Thin Plate Spline, $n = 30$

Abb. 21: Advektion nach einer Rotation mit verschiedenen RBFs und Nachbarschaften. $N = 40.000 rnd$



(a) $t = 0$ (b) $t = 360$

(c) $t = 720$ (d) $t = 1.080$

(e) $t = 1.440$ (f) $t = 1.800$

Abb. 22: Fünf Rotationen (RBF: Wendland-Funktion). $N = 40.000 rnd$

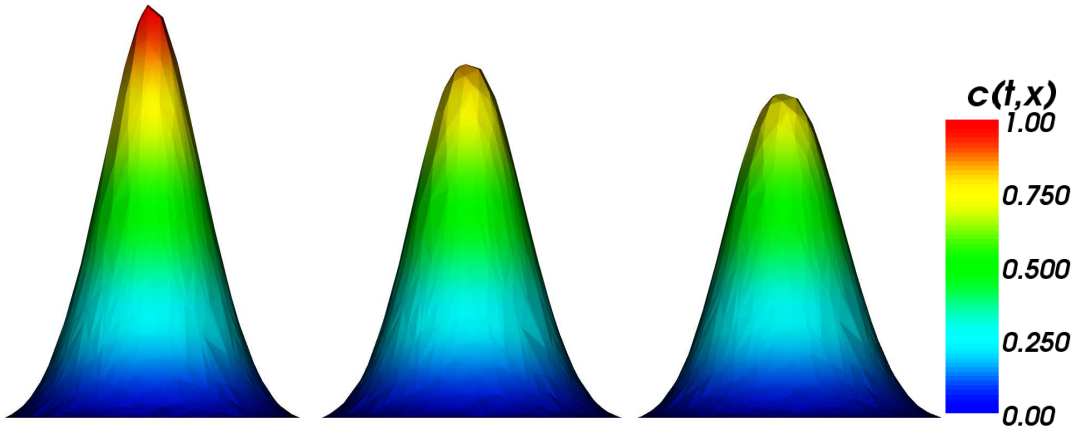
Bei der Verwendung einer glatten Ausgangsverteilung der Konzentration liefert die Advektion mit RBF-Interpolation sehr gute Ergebnisse. Selbst bei zufälliger Verteilung der Knoten ist lediglich im Bereich der Spitze eine Abflachung zu beobachten. Durch Erhöhung von N können wir diese Abflachung reduzieren. Abbildung 23 zeigt zwei Rotationen der Wendland-Funktion mit 10.000 bzw. 40.000 zufallsverteilten Knoten und $n = 20$. Bei der Verwendung des Thin Plate Spline als Basisfunktion (Abb. 23(d) - 23(f)) ist eine leichte Abflachung zu beobachten. Beid er Verwendung der Wendland-Funktion als Basisfunktion ist auch nach zwei Rotationen keine Abflachung zu erkennen, der Funktionskegel ist am oberen Ende lediglich etwas breiter geworden.

Parameter Abb. 23:	
Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline (23(a) - 23(f))
	Wendland-Funktion (23(g) - 23(i))
Nachbarschaften:	$n = 20$
Rotation:	$4 \cdot \pi$
Zeitschrittweite:	$\tau = 480$
Zeitintervall:	$\mathcal{I} = [0, 720 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	10.000*rnd* (23(a) - 23(c))
	40.000*rnd* (23(d) - 23(i))

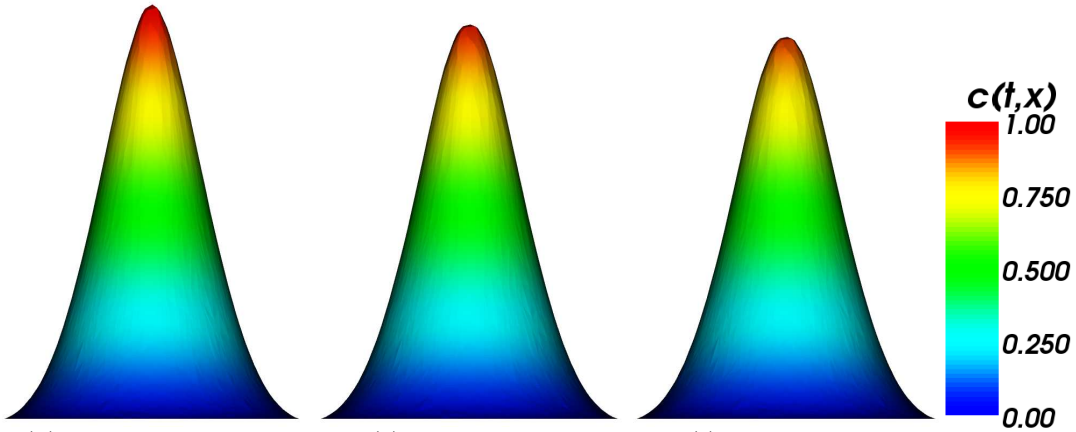
Im Vergleich zur MLS-Approximation liefert die RBF-Interpolation wesentlich bessere Ergebnisse bei der adaptionsfreien Advektion. Durch die geringere Abflachung des betrachteten Objekts erwarten wir in Verbindung mit der Adaption ebenfalls bessere Resultate bei der RBF-Interpolation.

3.5 Adaption

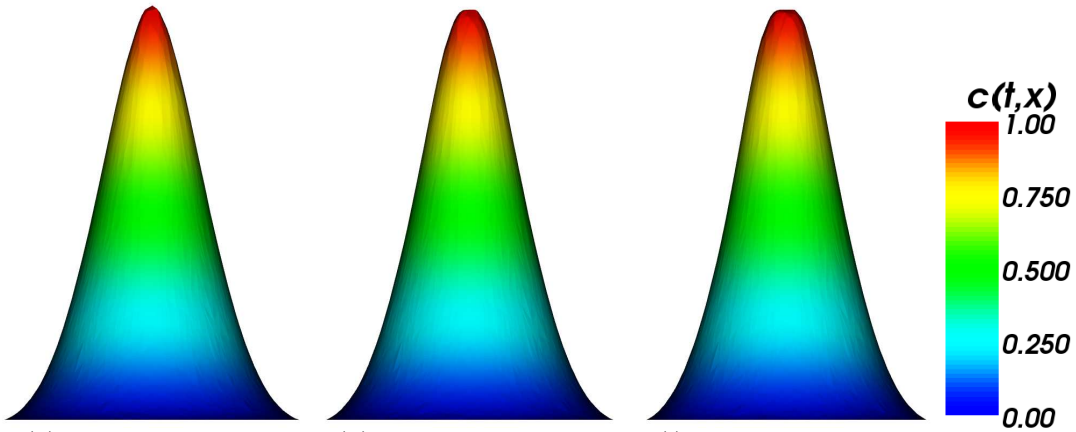
Wir betrachten nun die dynamische Anpassung der Knotenmenge, ohne ein Geschwindigkeitsfeld einwirken zu lassen. Wir haben so die Möglichkeit, die eingesetzten Verfahren auf Ihre Tauglichkeit für die Adaption zu testen. Da wir während der Adaption Knoten sowohl löschen als auch hinzufügen werden, müssen wir neben einer guten Qualität der Reproduktion auch auf die Anzahl der Knoten achten. Wir untersuchen zunächst die eingesetzen Verfahren auf die Qualität der Reproduktion. Dafür verwenden wir die Slotted Disc jeweils mit zufälliger und gleichmäßiger Knotenverteilung mit 10.000 und 40.000 Knoten. Dadurch können wir brauchbare Werte für die Parameter σ_{crs}, σ_{ref} und die Größe der Nachbarschaften n bestimmen.



(a) $t = 0$; $N = 10.000$ (b) $t = 360$; TPS (c) $t = 720$; TPS



(d) $t = 0$; $N = 40.000$ (e) $t = 360$; TPS (f) $t = 720$; TPS



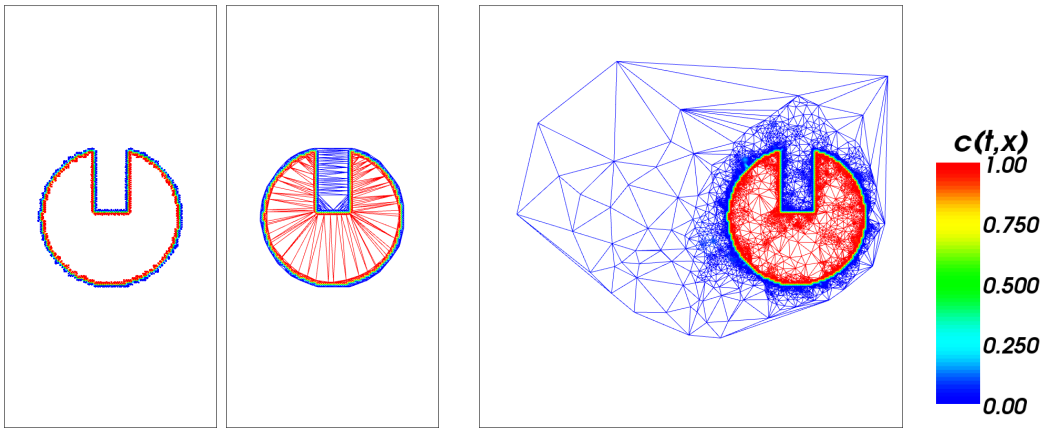
(g) $t = 0$; $N = 40.000$ (h) $t = 360$; Wendland (i) $t = 720$; Wendland

Abb. 23: Advektion der Wendland-Funktion mit der RBF-Interpolation bei zufälliger Ausgangsverteilung

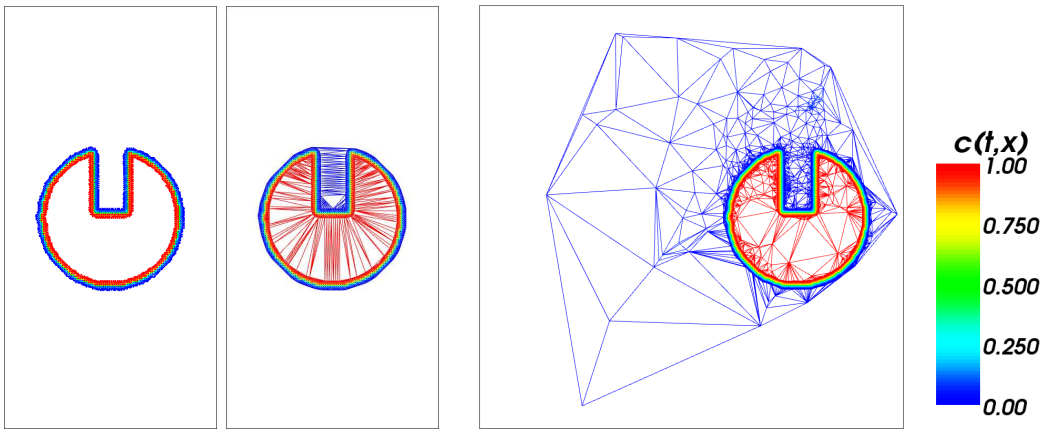
Anschließend testen wir die Verfahren auf Stabilität, indem wir für die Entwicklung der Knotenzahl nach mehreren Adaptionsschritten betrachten. Wir verlangen von der Adaption, dass sie nach einer gewissen Anzahl von Schritten *stationär* wird, d.h. sowohl in der Verteilung der Konzentration als auch in der Anzahl der Knoten keinen großen Veränderungen mehr unterliegt.

3.5.1 Adaption mit MLS

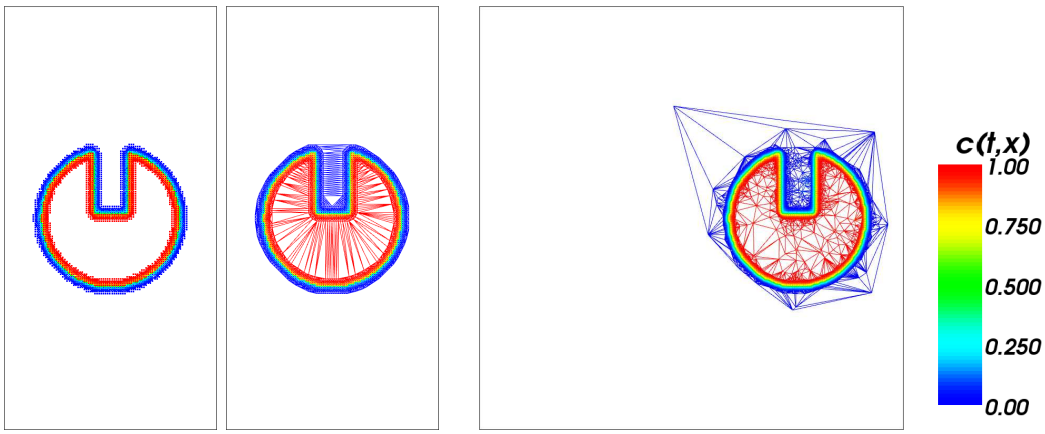
Wir werden zunächst die optimale Anzahl n von Knoten in den Nachbarschaften bestimmen. Dazu betrachten wir die Adaption nach 10 durchgeführten Schritten bei festen Parametern $\sigma_{crs} = 0{,}001$ und $\sigma_{ref} = 0{,}1$. Als Ausgangsverteilung werden wir mit 40.000 Knoten arbeiten und jeweils eine Zufallsverteilung und eine gleichmäßige Verteilung betrachten. Da wir die Adaption vor jedem Advektionsschritt solange ausführen werden, bis die Knotenmenge stationär wird, untersuchen wir die Adaption auf ihr Verhalten nach mehreren durchgeführten Adaptionsschritten.
Die Abbildungen 24 und 25 zeigen die Knotenverteilung während der Adaption nach einem bzw. zehn Schritten.
Die linke Spalte zeigt dabei die Knotenverteilung und die Triangulierung nach dem ersten Schritt, rechts ist die Triangulierung nach zehn Schritten abgebildet. In beiden Abbildungen ist jeweils in der ersten Zeile die Adaption mit Nachbarschaften von $n = 5$ dargestellt, in den folgenden Zeilen ist $n = 20$ bzw. $n = 50$.
Bei allen verwendeten Werten fur n liefert der erste Adaptionsschritt eine sehr gute Darstellung der Slotted Disc. Die Knotenanzahl wird dabei um mehr als 90% reduziert, für $n = 50$ bleiben jedoch wesentlich mehr Knoten erhalten als für $n = 5$ bzw. $n = 20$. Dies resultiert nach zehn Schritten in einer sehr schlechten Abbildung der Slotted Disc bei zu geringem n. Die Knotenmenge ist hier nicht mehr auf die unmittelbare Umgebung der Disc beschränkt, sondern relativ weit über Ω verteilt. Es kommt im Bereich außerhalb der Disc zu großen Fehlern bei der Reproduktion der Konzentration. Dieser Effekt tritt besonders stark bei zufallsverteilten Anfangsknoten auf. Für die Adaption mit der MLS-Approximation kommen daher nur Nachbarschaften mit hinreichend vielen Punkten in Frage.
Abb. 26 zeigt die Slotted Disc als Detailaufnahme nach zehn Schritten für beide Anfangsverteilungen und alle drei verwendeten Nachbarschaften. Ein akzeptables Ergebnis liefert die Adaption nur für $n = 50$. Wir werden fortan ausschließlich mit Nachbarschaften von jeweils 50 Knoten arbeiten.



(a) $t_A = 1$; $n = 5$; $N_1 = 2.077$ (b) $t_A = 10$; $n = 5$; $N_{10} = 160.880$

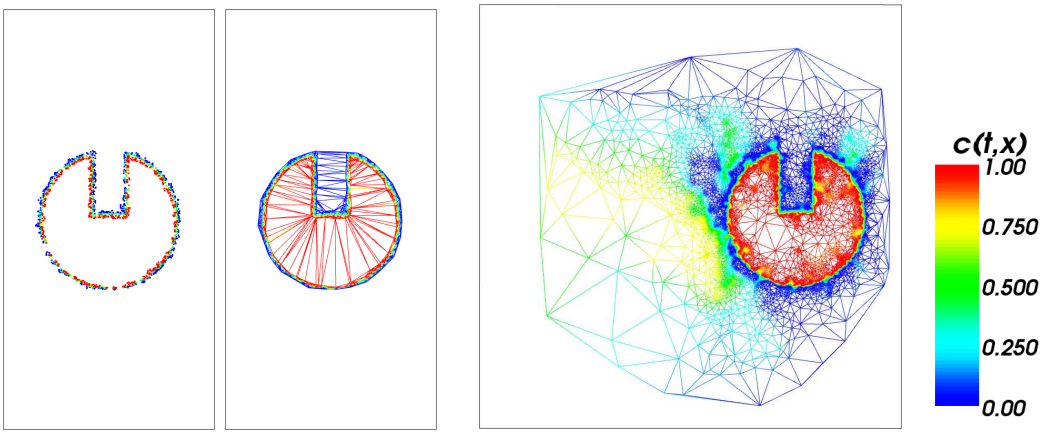


(c) $t_A = 1$; $n = 20$; $N_1 = 3.446$ (d) $t_A = 10$; $n = 20$; $N_{10} = 66.029$

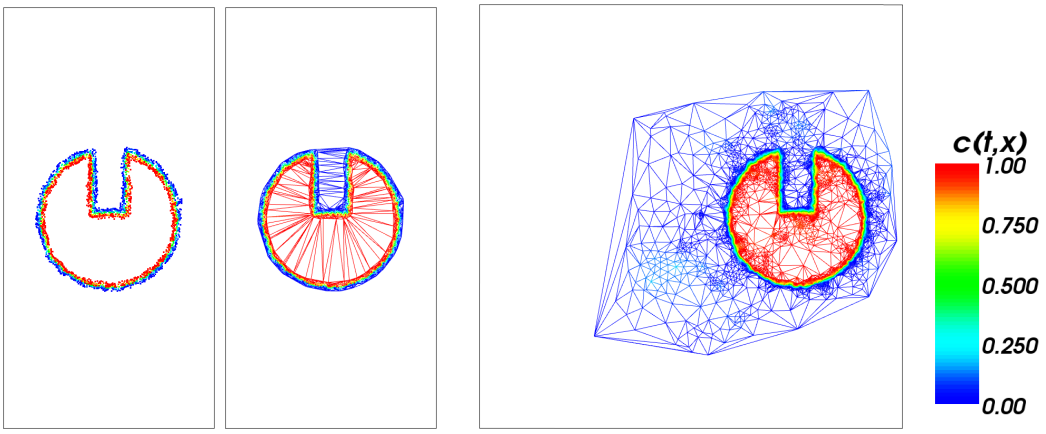


(e) $t_A = 1$; $n = 50$; $N_1 = 4.755$ (f) $t_A = 10$; $n = 50$; $N_{10} = 81.998$

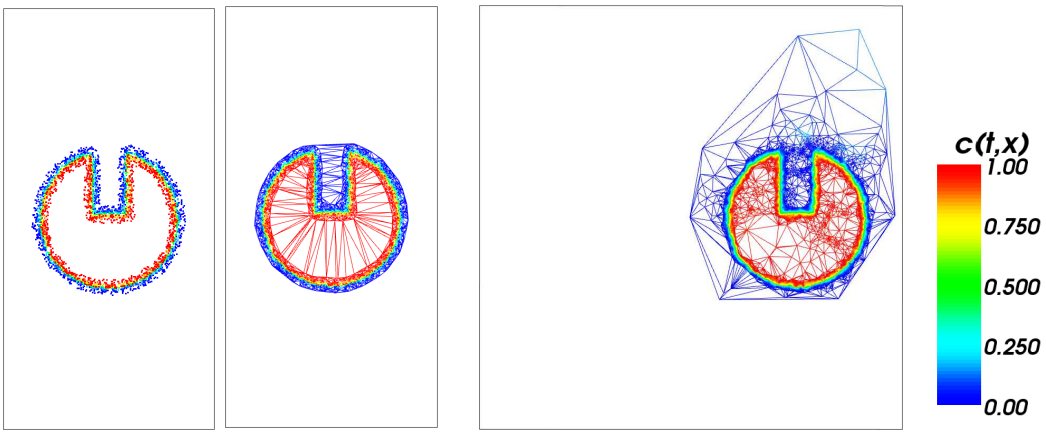
Abb. 24: Adaption nach 10 Schritten. $N = 40.000 grid$



(a) $t_A = 1$; $n = 5$; $N_1 = 1.557$ (b) $t_A = 10$; $n = 5$; $N_{10} = 64.009$

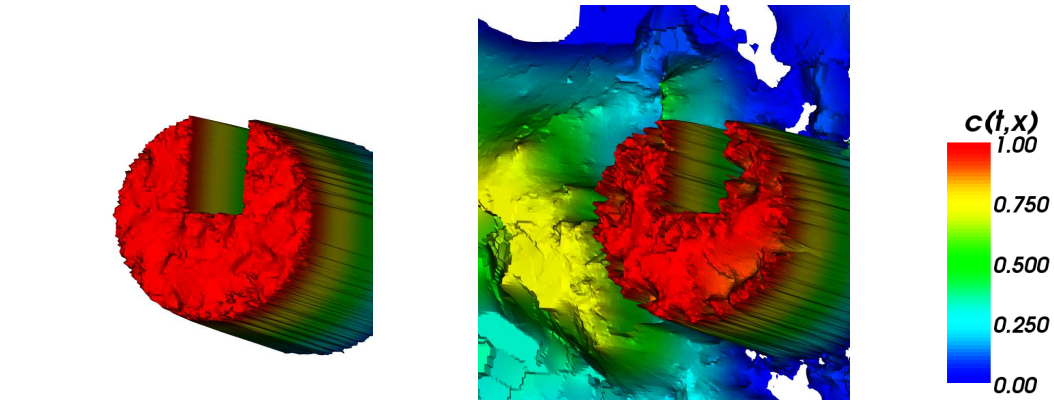


(c) $t_A = 1$; $n = 20$; $N_1 = 2.766$ (d) $t_A = 10$; $n = 20$; $N_{10} = 51.932$

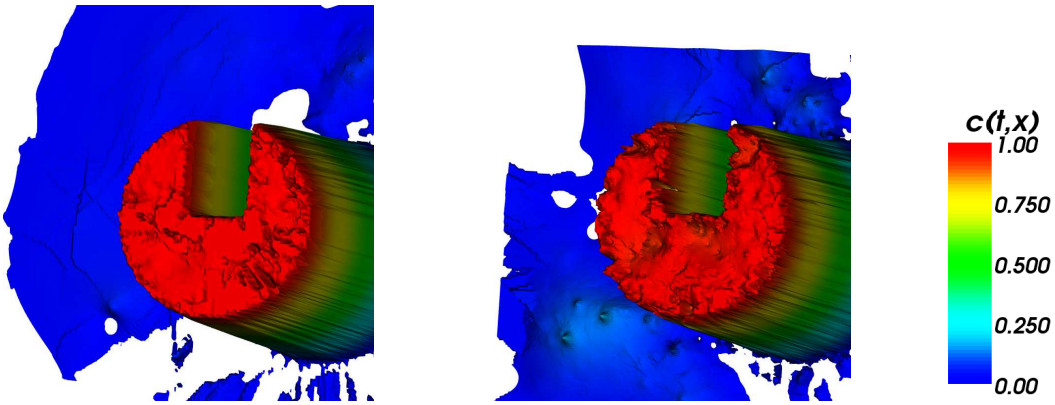


(e) $t_A = 1$; $n = 50$; $N_1 = 4.026$ (f) $t_A = 10$; $n = 50$; $N_{10} = 68.991$

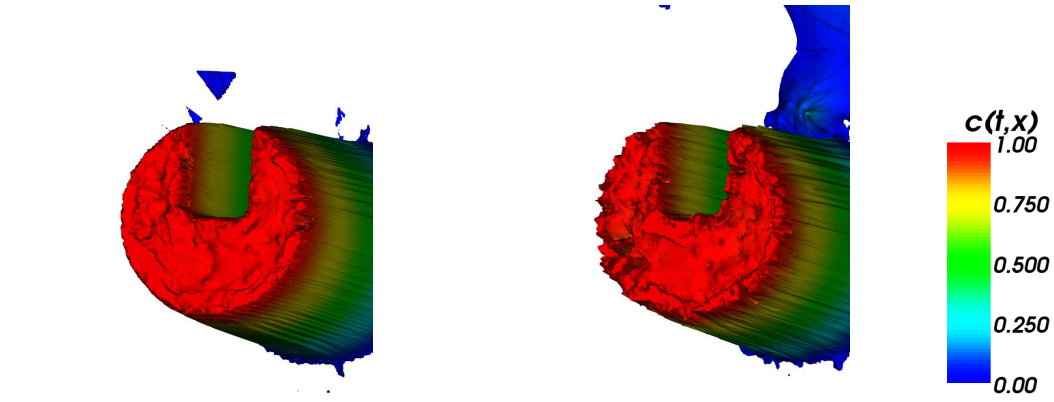
Abb. 25: Adaption nach 10 Schritten. $N = 40.000 rnd$



(a) $n = 5$; $N_{10} = 160.880$; *grid* (b) $n = 5$; $N_{10} = 64.009$; *rnd*



(c) $n = 20$; $N_{10} = 66.029$; *grid* (d) $n = 20$; $N_{10} = 51.932$; *rnd*

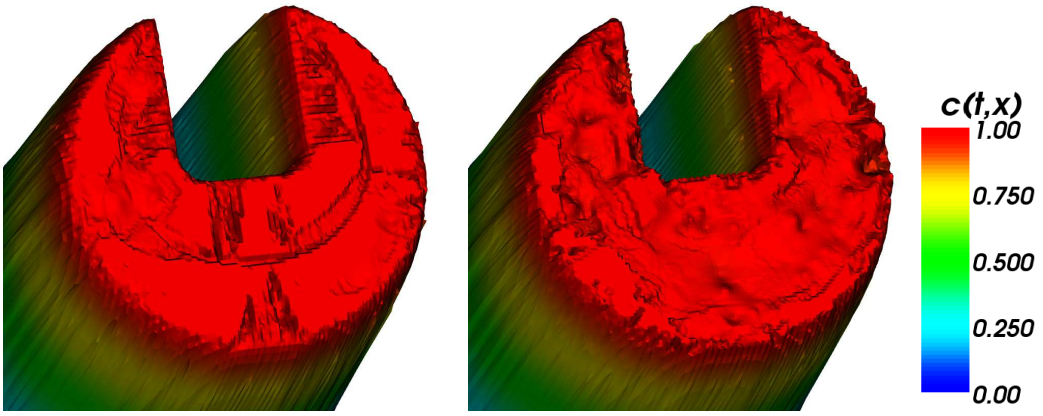


(e) $n = 50$; $N_{10} = 81.998$; *grid* (f) $n = 50$; $N_{10} = 68.991$; *rnd*

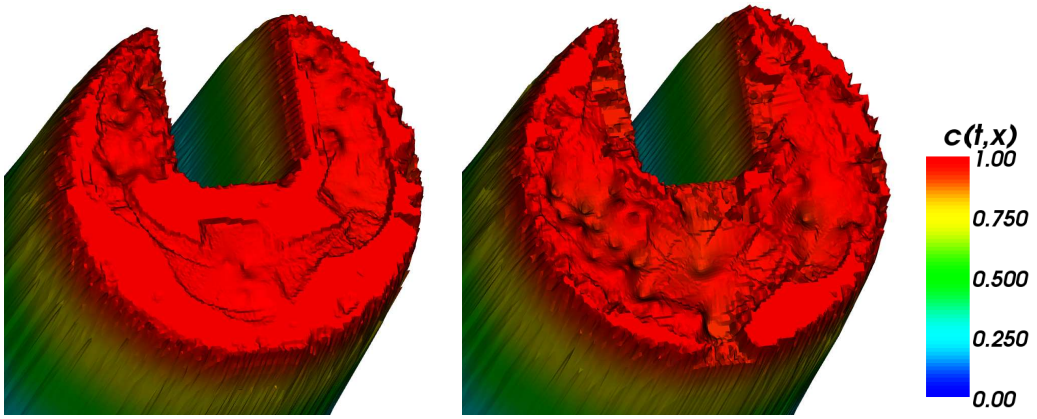
Abb. 26: Adaption nach 10 Schritten. $N = 40.000$

Parameter Abb. 24, 25 und 26:	
Adaption:	MLS-Approximation
Nachbarschaften:	$n = 5$ (24(a), 24(b), 25(a), 25(b), 26(a), 26(b))
	$n = 20$ (24(c), 24(d), 25(c), 25(d), 26(c), 26(d))
	$n = 50$ (24(e), 24(f), 25(e), 25(f), 26(e), 26(f))
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$t_A = 1$ (24(a), 24(c), 24(e), 25(a), 25(c), 25(e))
	$t_A = 10$ (24(b), 24(d), 24(f), 25(b), 25(d), 25(f), 26)
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (24, 26(a), 26(c), 26(e))
	$N = 40.000$ *rnd* (25, 26(b), 26(d), 26(f))

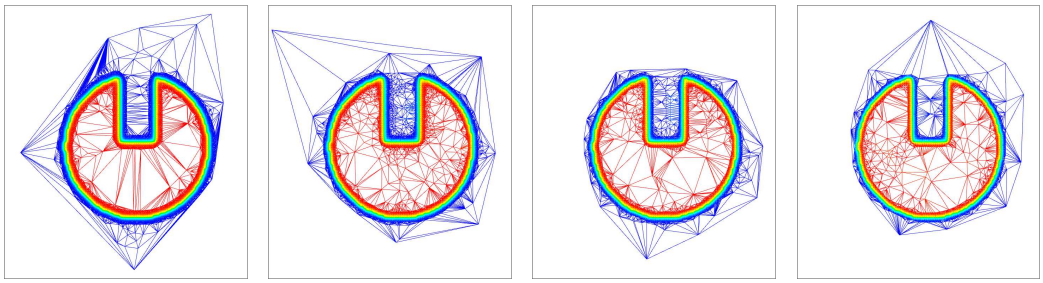
Um die Abhängigkeit des Verfahrens vom Verfeinerungsparameter σ_{ref} zu testen, werden wir die Adaption für $\sigma_{crs} = 0{,}001$ erneut auf den zufalls- und gleichmäßig verteilten Knotenmengen mit $N = 40.000$ durchführen. Für kleinere Werte von σ_{ref} erwarten wir mehr verfeinerte Punkte und daher eine größere Anzahl von Knoten. Die Abbildungen 27 und 28 zeigen die Adaption nach zehn Schritten für $\sigma_{ref} \in \{0{,}05; 0{,}10; 0{,}15; 0{,}20\}$. Für die gleichmäßig verteilte Knotenmenge liefern $\sigma_{ref} = 0{,}05$ und $\sigma_{ref} = 0{,}15$ ein ähnlich gutes Ergebnis, wobei die Knoten für $\sigma_{ref} = 0{,}15$ besser in der Nähe der Disc konzentriert sind. Bei der Zufallsverteilung liefert jedoch $\sigma_{ref} = 0{,}05$ eine deutlich bessere Reproduktion der Disc.
Diese Verbesserung der Qualität ist jedoch sehr teuer, wie uns Abb. 29 zeigt. Hier ist die Entwicklung der Knotenzahlen für 50 ausgeführte Adaptionsschritte dargestellt. Abb. 29(a) stellt die Entwicklung für eine gleichmäßig verteilte Knotenmenge dar, Abb. 29(b) für eine Zufallsverteilung. Für alle eingesetzten Werte $\sigma_{ref} \in \{0{,}05; 0{,}10; 0{,}15\}$ verläuft die Entwicklung ähnlich, jedoch auf unterschiedlichem Niveau. Im ersten Schritt wird die Zahl der Knoten von 40.000 auf unter 5.000 reduziert, um danach rapide anzusteigen. Nach ca. 15 Schritten bleibt die Knotenzahl dann nahezu konstant, die Adaption wird stationär. Für $\sigma_{ref} = 0{,}05$ geschieht dies erst bei einer Knotenzahl von > 325.000 für die gleichmäßige Knotenverteilung, bzw. rund 240.000 bei der Zufallsverteilung. Ob wir diese Werte durch die Verwendung von mehr Knoten in der Ausgangsverteilung senken können, werden wir später betrachten.



(a) $\sigma_{ref} = 0{,}05$ (b) $\sigma_{ref} = 0{,}10$

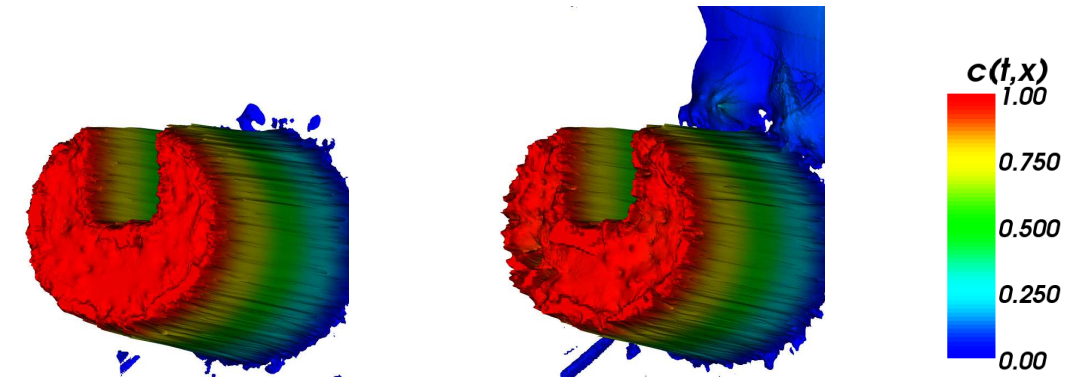


(c) $\sigma_{ref} = 0{,}15$ (d) $\sigma_{ref} = 0{,}20$

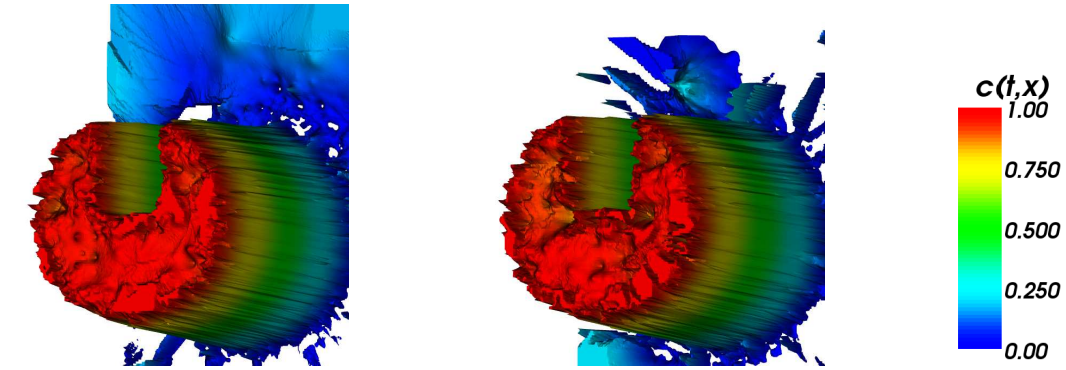


(e) $\sigma_{ref} = 0{,}05$, $N_{10} = 159.115$ (f) $\sigma_{ref} = 0{,}10$, $N_{10} = 81.998$ (g) $\sigma_{ref} = 0{,}15$, $N_{10} = 65.031$ (h) $\sigma_{ref} = 0{,}20$, $N_{10} = 49.772$

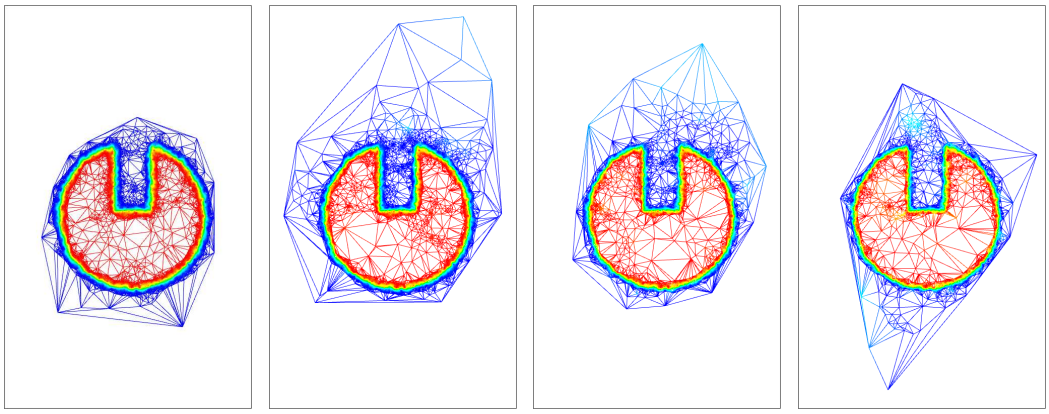
Abb. 27: Adaption nach 10 Schritten. $\sigma_{crs} = 0{,}001$; $N = 40.000 grid$



(a) $\sigma_{ref} = 0{,}05$ (b) $\sigma_{ref} = 0{,}10$

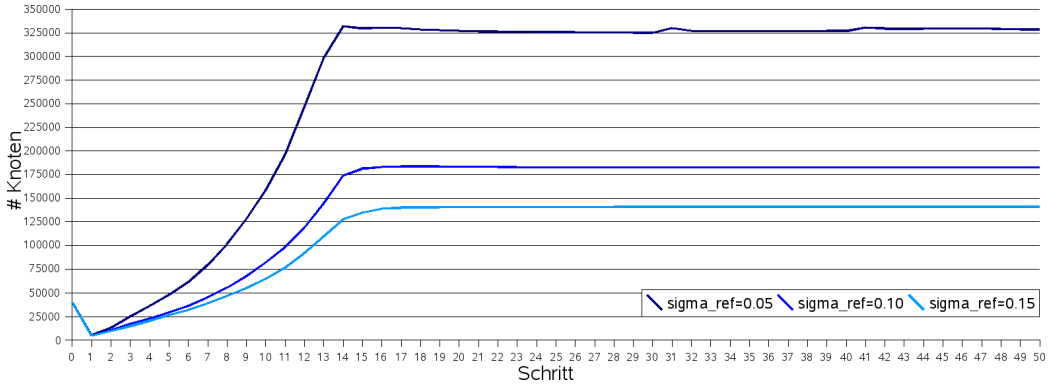


(c) $\sigma_{ref} = 0{,}15$ (d) $\sigma_{ref} = 0{,}20$

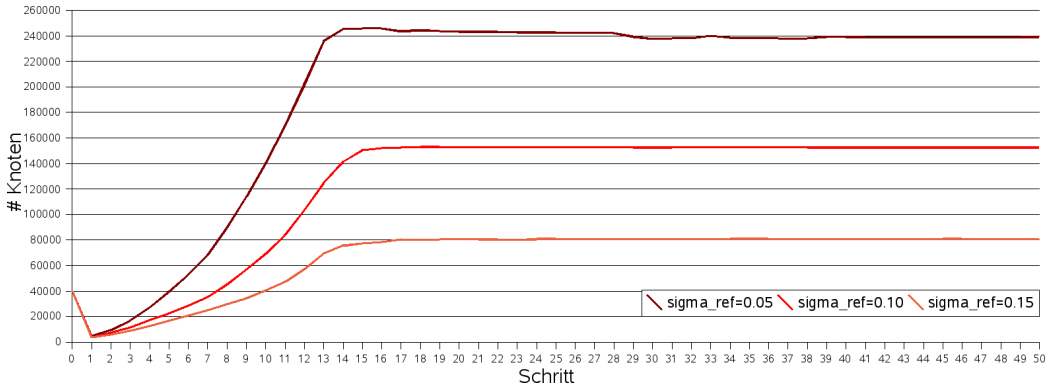


(e) $\sigma_{ref} = 0{,}05$, $N_{10} = 140.213$ (f) $\sigma_{ref} = 0{,}10$, $N_{10} = 68.991$ (g) $\sigma_{ref} = 0{,}15$, $N_{10} = 40.545$ (h) $\sigma_{ref} = 0{,}20$, $N_{10} = 30.576$

Abb. 28: Adaption nach 10 Schritten. $\sigma_{crs} = 0{,}001$; $N = 40.000rnd$



(a) Knotenzahl nach 50 Adaptionsschritten. $N = 40.000 grid$



(b) Knotenzahl nach 50 Adaptionsschritten. $N = 40.000 rnd$

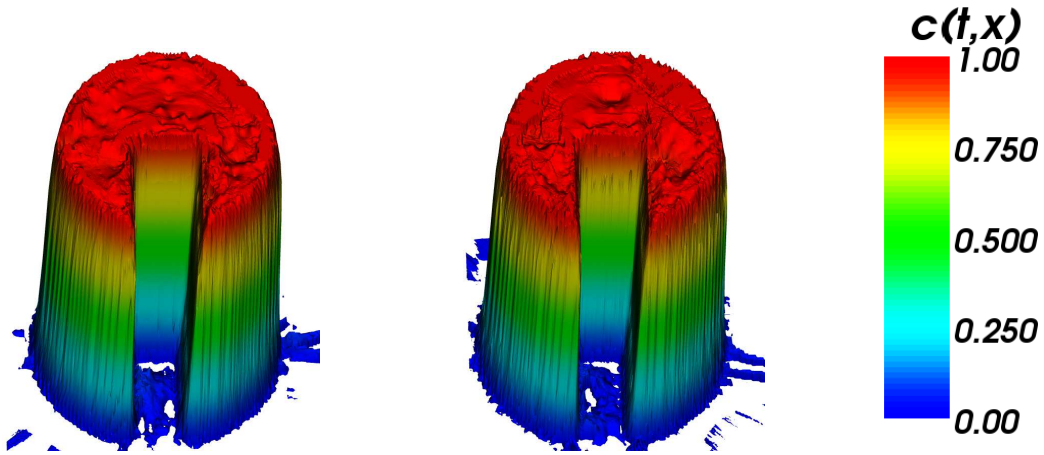
Abb. 29: Adaption nach 50 Schritten, $\sigma_{crs} = 0{,}001$

Parameter Abb. 27, 28 und 29:	
Adaption:	MLS-Approximation
Nachbarschaften:	$n = 50$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}05$ (27(a), 28(a))
	$\sigma_{ref} = 0{,}10$ (27(b), 28(b))
	$\sigma_{ref} = 0{,}15$ (27(c), 28(c))
	$\sigma_{ref} = 0{,}20$ (27(d), 28(d))
Adaptionsschritte:	$t_A = 10$ (27, 28)
	$t_A = 50$ (29)
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$*grid* (27, 29(a))
	$N = 40.000$*rnd* (28, 29(b))

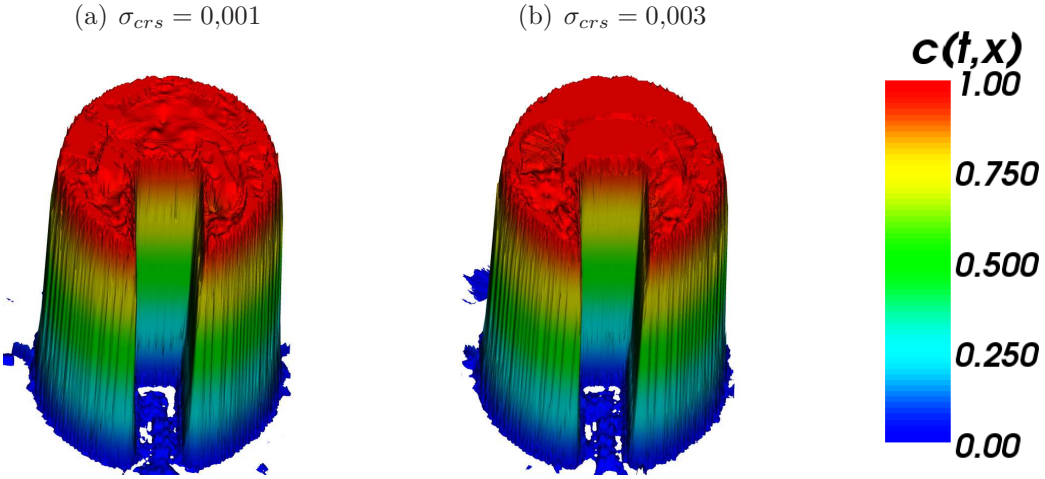
Wir werden nun σ_{crs} festlegen. Dafür halten wir $\sigma_{ref} = 0{,}10$ fest und betrachten die Adaption für $\sigma_{crs} \in [0{,}001, 0{,}01]$.
Die Abbildungen 30 und 31 zeigen die Disc im Detail nach zehn Adaptionsschritten. Die beste Reproduktion erreichen wir mit $\sigma_{crs} \in [5 \cdot 10^{-3}, 7 \cdot 10^{-3}]$. Betrachten wir zusätzlich noch die Verteilung der Knoten bei der Adaption (Abb. 32), so wird deutlich, dass bei $\sigma_{crs} = 7 \cdot 10^{-3}$ Die Knotenverteilung eher ungünstig ist, so dass wir uns in den weiteren Betrachtungen auf $\sigma_{crs} \in [5 \cdot 10^{-3}, 6 \cdot 10^{-3}]$ beschränken werden.

Parameter Abb. 30, 31 und 32:	
Adaption:	MLS-Approximation
Nachbarschaften:	$n = 50$
Verfeinern:	$\sigma_{ref} = 0{,}10$
Ausdünnen:	$\sigma_{crs} = 0{,}001$ (30(a), 31(a))
	$\sigma_{crs} = 0{,}003$ (30(b), 31(b))
	$\sigma_{crs} = 0{,}005$ (30(c), 31(c), 32(a), 32(b))
	$\sigma_{crs} = 0{,}006$ (30(d), 31(d), 32(c), 32(d))
	$\sigma_{crs} = 0{,}007$ (30(e), 31(e), 32(e), 32(f))
	$\sigma_{crs} = 0{,}010$ (30(f), 31(f))
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$*grid* (30, 32(a), 32(c), 32(e))
	$N = 40.000$*rnd* (31, 32(b), 32(d), 32(f))

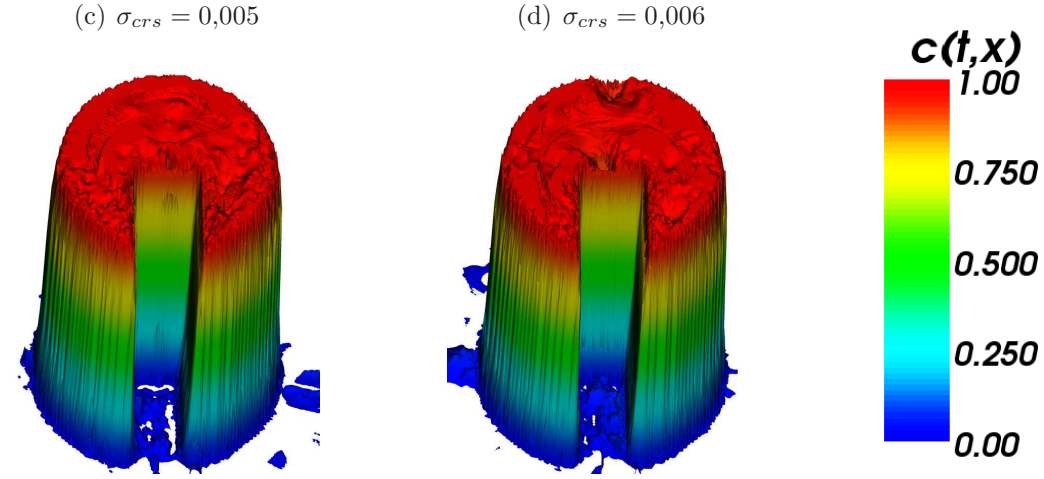
Die Abbildungen 33 und 34 zeigen die Adaption in stationärem Zustand nach 25 Schritten. Dabei ist jeweils $\sigma_{crs} = 0{,}006$, als Ausgangsverteilung dient die Slotted Disc mit $40.000grid$ bzw. $40.000rnd$ Knoten. Für die Verfeinerung der Knotenmenge wird $\sigma_{ref} = 0{,}05$ bzw. $\sigma_{ref} = 0{,}1$ verwendet. Es fällt auf, dass



(a) $\sigma_{crs} = 0{,}001$ (b) $\sigma_{crs} = 0{,}003$

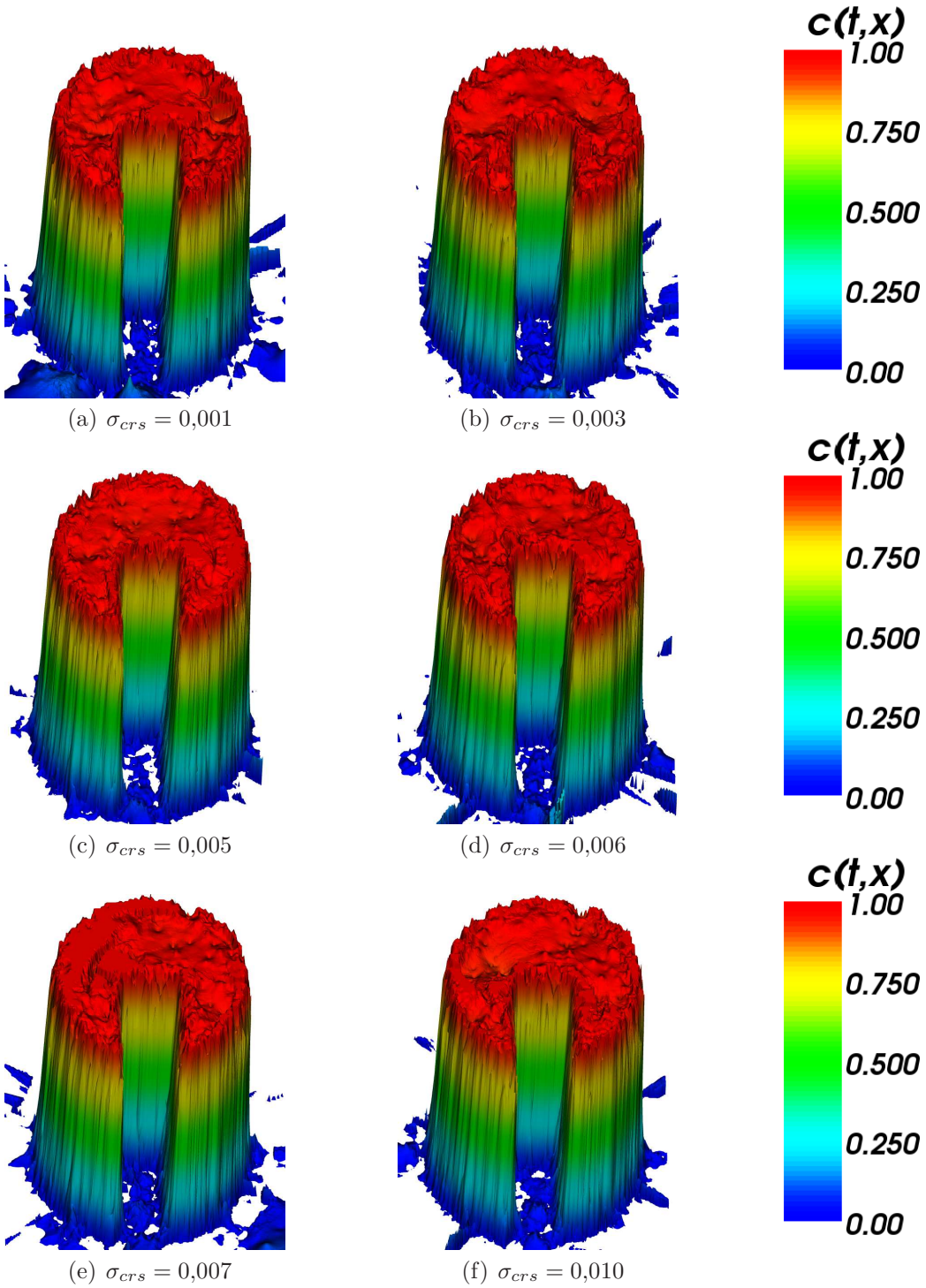


(c) $\sigma_{crs} = 0{,}005$ (d) $\sigma_{crs} = 0{,}006$



(e) $\sigma_{crs} = 0{,}007$ (f) $\sigma_{crs} = 0{,}010$

Abb. 30: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}10$; $N = 40.000 grid$

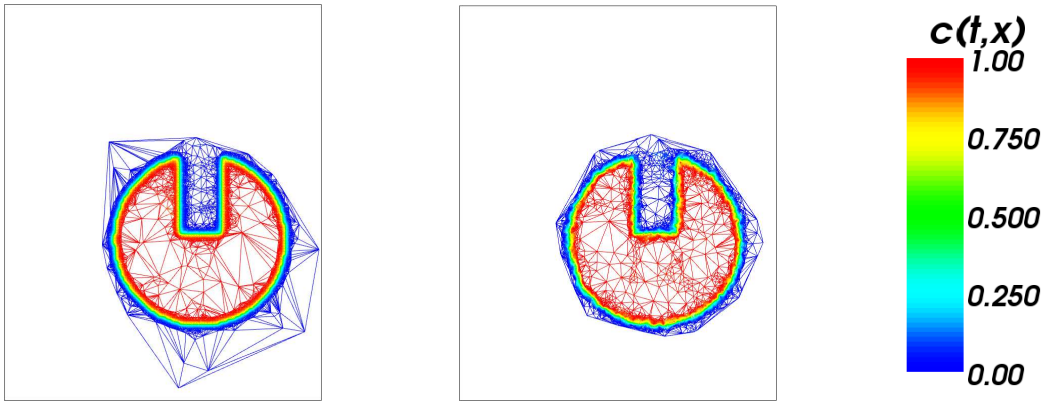


(a) $\sigma_{crs} = 0{,}001$ (b) $\sigma_{crs} = 0{,}003$

(c) $\sigma_{crs} = 0{,}005$ (d) $\sigma_{crs} = 0{,}006$

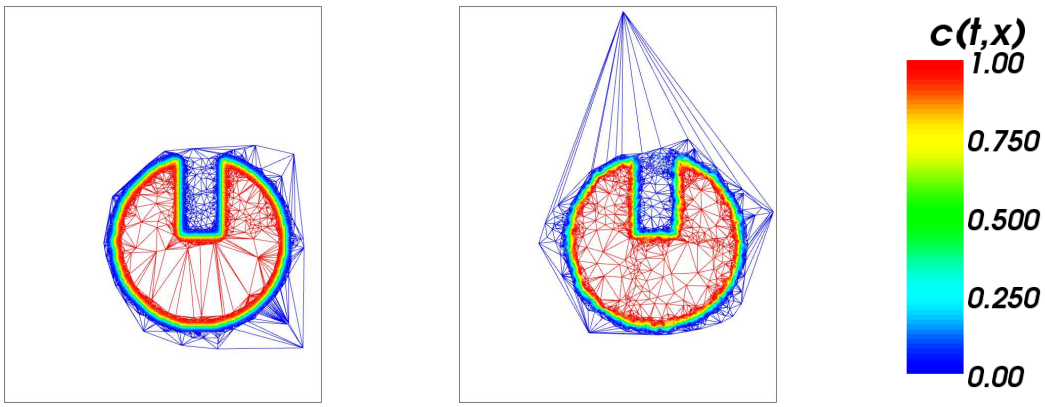
(e) $\sigma_{crs} = 0{,}007$ (f) $\sigma_{crs} = 0{,}010$

Abb. 31: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}10$; $N = 40.000 rnd$



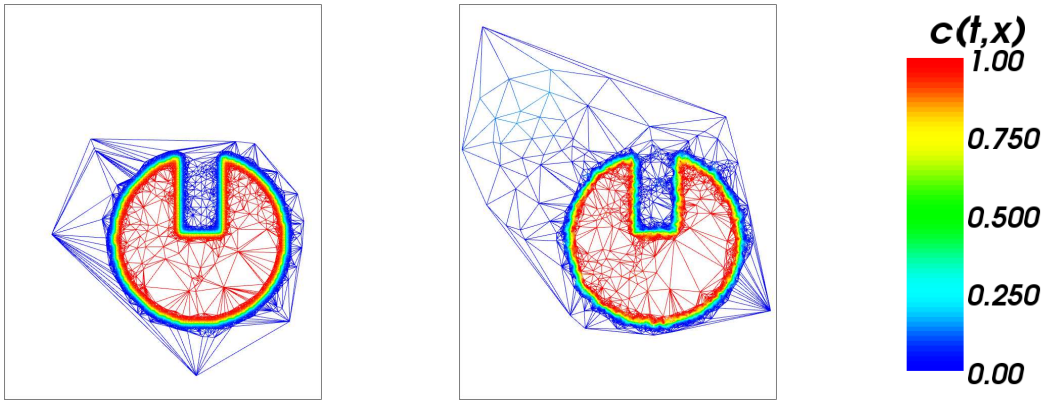
(a) $\sigma_{crs} = 0{,}005$
$N_{10} = 63.074$; *grid*

(b) $\sigma_{crs} = 0{,}005$
$N_{10} = 79.496$; *rnd*



(c) $\sigma_{crs} = 0{,}006$
$N_{10} = 56.560$; *grid*

(d) $\sigma_{crs} = 0{,}006$
$N_{10} = 77.217$; *rnd*



(e) $\sigma_{crs} = 0{,}007$
$N_{10} = 58.326$; *grid*

(f) $\sigma_{crs} = 0{,}007$
$N_{10} = 78.841$; *rnd*

Abb. 32: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}10$; $N = 40.000$

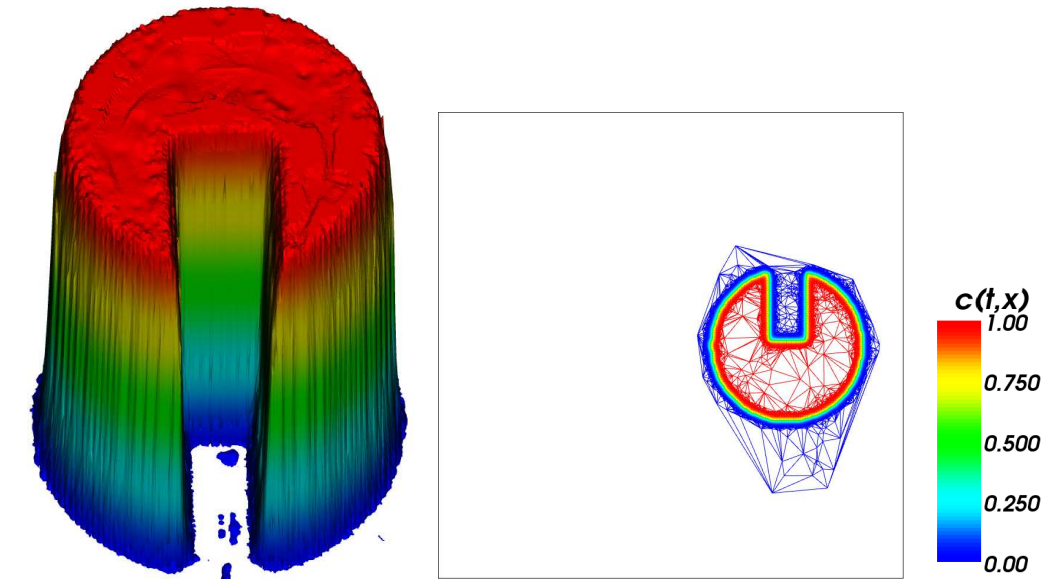
im ersten Fall zwar die Disc besser reproduziert wird, der Bereich außerhalb der Disc jedoch eher schlecht. Umgekehrt wird im zweiten Fall die Knotenmenge weniger zerstreut, die Disc jedoch schlechter reproduziert. Die Anzahl der Knoten ist für $\sigma_{ref} = 0{,}05$ wesentlich höher als für $\sigma_{ref} = 0{,}10$.

Parameter Abb. 33 und 34:	
Adaption:	MLS-Approximation
Nachbarschaften:	$n = 50$
Verfeinern:	$\sigma_{ref} = 0{,}05$ (33(a), 34(a))
	$\sigma_{ref} = 0{,}10$ (33(b), 34(b))
Ausdünnen:	$\sigma_{crs} = 0{,}06$
Adaptionsschritte:	$t_A = 25$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (33)
	$N = 40.000$ *rnd* (34)

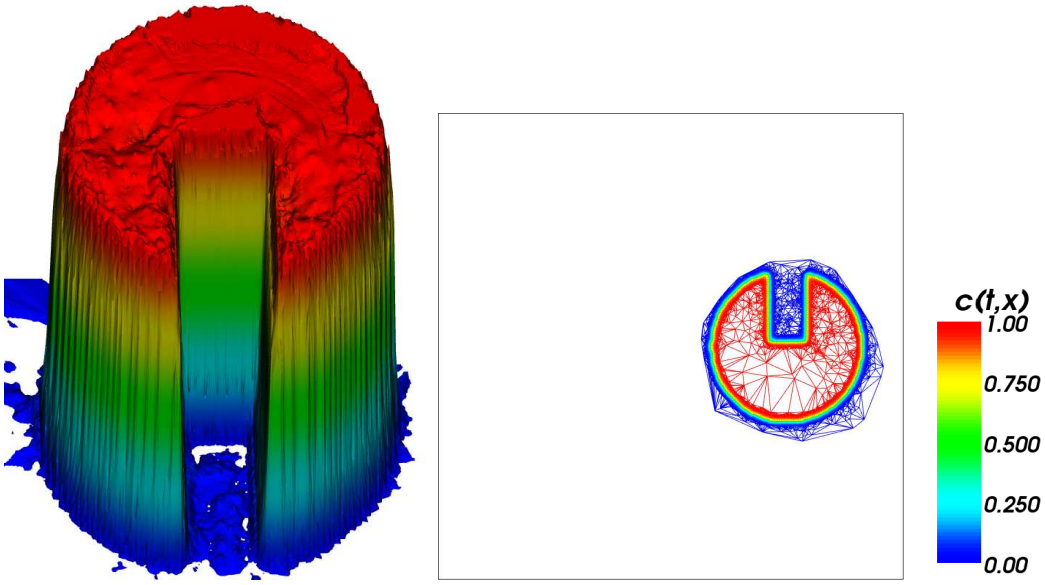
Die folgende Tabelle zeigt die gerundeten Knotenzahlen für den stationären Zustand der Adaption für $\sigma_{crs} = 0{,}006$ und $N = 40.000$.

σ_{ref}	*grid*	*rnd*
0,05	302.000	208.000
0,10	186.000	115.000

Wir überprüfen nun den Einfluß der Anzahl N von Knoten in der Ausgangsverteilung auf die Qualität sowie die Knotenzahl bei der Adaption. Dafür verwenden wir für N unterschiedliche Werte zwischen 10.000 und 500.000 Punkten. Für die Verfeinerung setzen wir $\sigma_{crs} = 0{,}006$ und $\sigma_{ref} = 0{,}10$. Abbildung 35 zeigt für die intialen Knotenmengen 10.000, 40.000 und 490.000 die Slotted Disc nach 20 Adaptionsschritten im Detail. Dabei ist auf der linken Seite das Ergebnis bei gleichmäßiger Knotenverteilung dargestellt, rechts das Ergebnis bei zufälliger Verteilung. Abb. 36 zeigt die zugehörigen Knotenverteilungen mit der Delaunay-Triangulierung. Die Qualität der Reproduktion ist generell bei einer gitterbasierten Ausgangsmenge besser als bei einer Zufallsverteilung. Durch die Verwendung von mehr Punkten erreichen wir eine bessere Reproduktion der Außenkanten der Disc, die Oberfläche der Disc wird jedoch unregelmäßiger abgebildet. Zudem erhöht sich zusammen mit der Ausgangspunktzahl auch die Zahl der Knoten bei der Adaption.

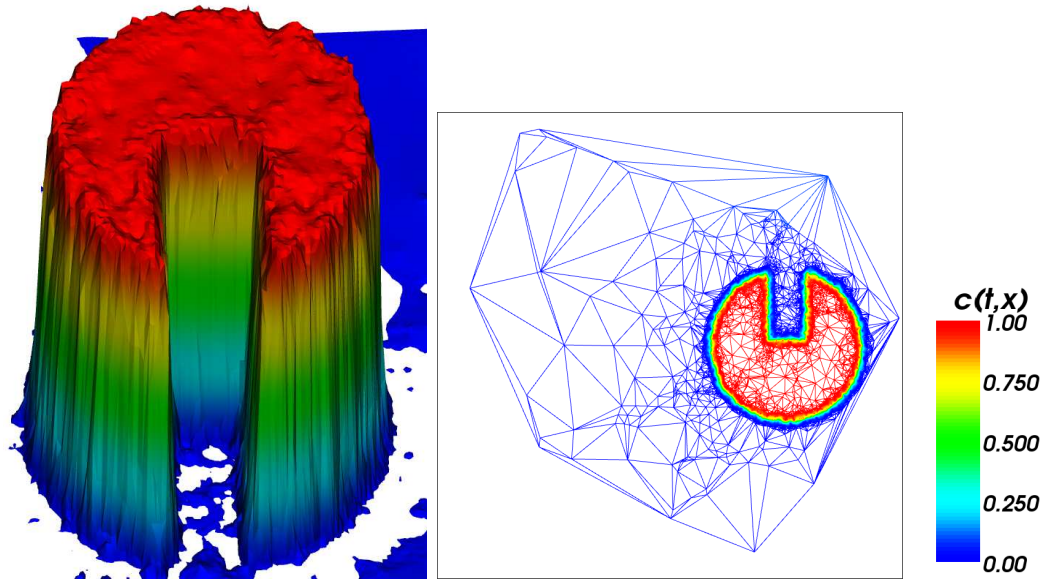


(a) $\sigma_{ref} = 0{,}05$; $N_{25} = 301.920$

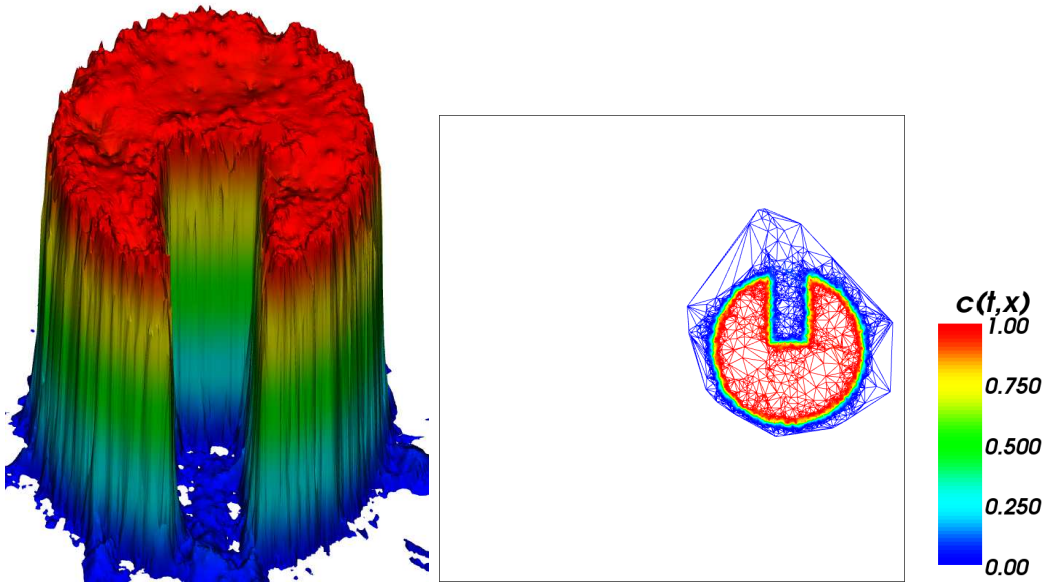


(b) $\sigma_{ref} = 0{,}10$; $N_{25} = 186.443$

Abb. 33: Stationärer Zustand der Adaption. $\sigma_{crs} = 0{,}006$; $N = 40.000 grid$

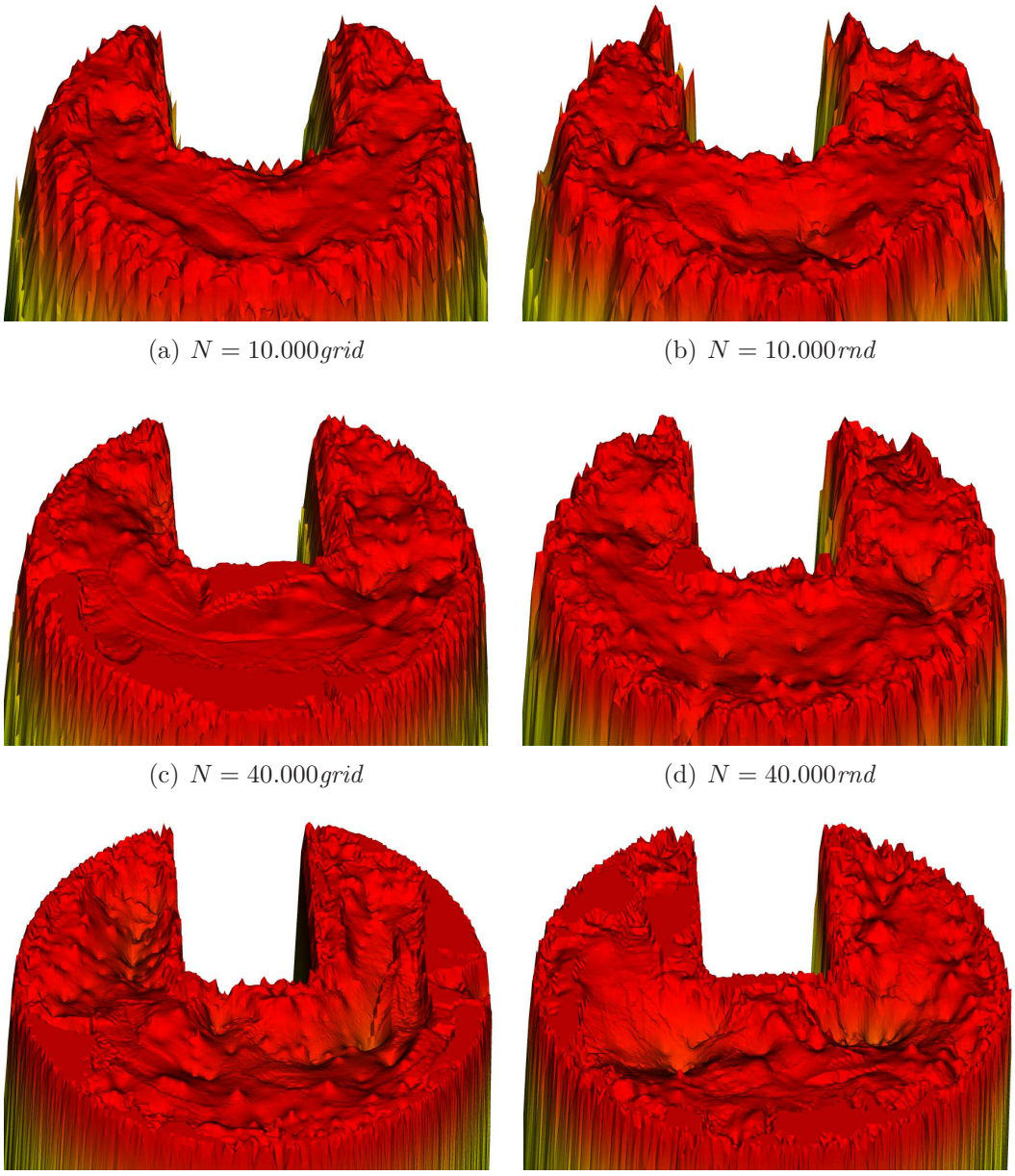


(a) $\sigma_{ref} = 0{,}05$; $N_{25} = 208.229$



(b) $\sigma_{ref} = 0{,}10$; $N_{25} = 114.760$

Abb. 34: Stationärer Zustand der Adaption. $\sigma_{crs} = 0{,}006$; $N = 40.000 rnd$

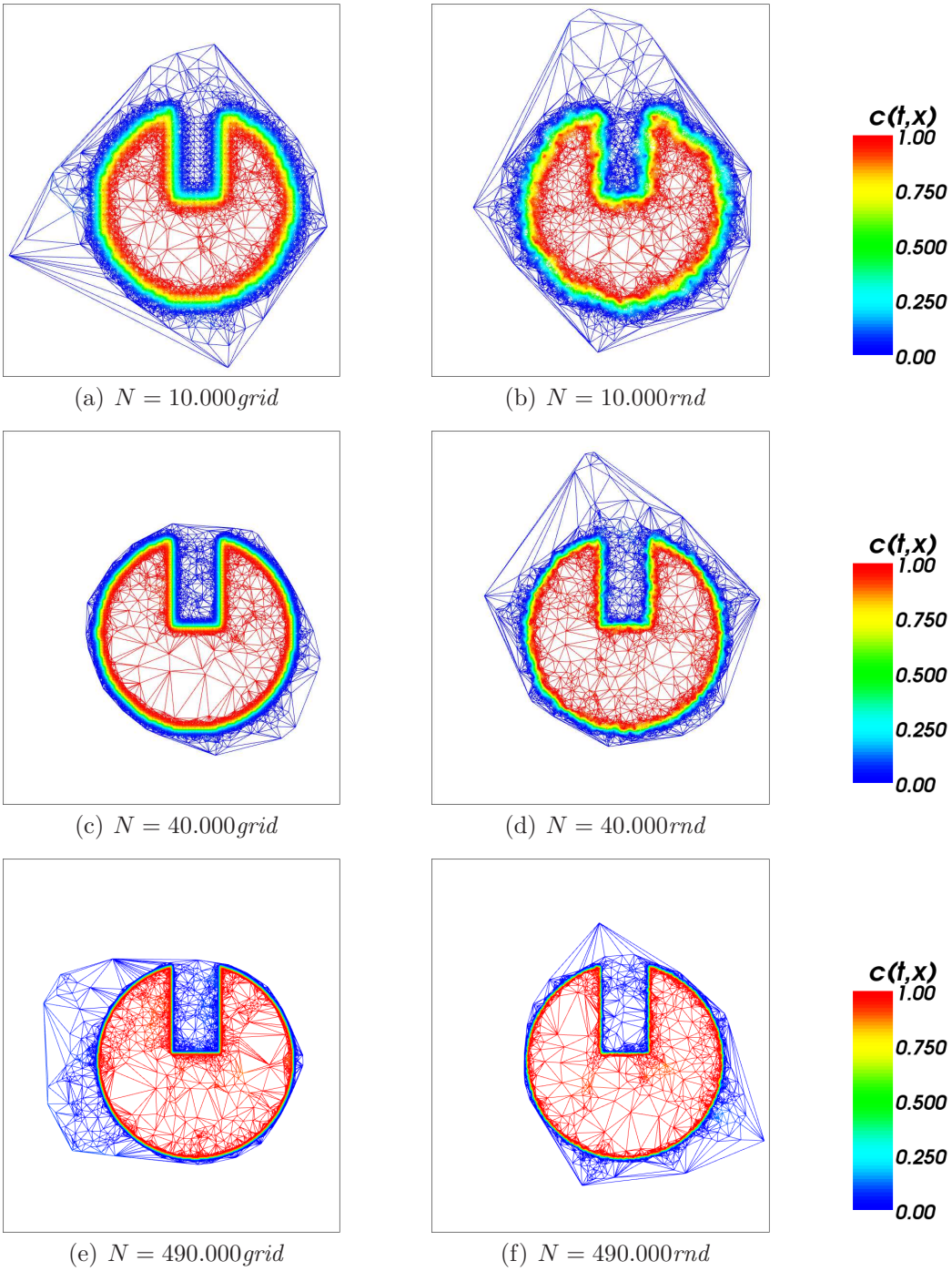


(a) $N = 10.000 grid$ (b) $N = 10.000 rnd$

(c) $N = 40.000 grid$ (d) $N = 40.000 rnd$

(e) $N = 490.000 grid$ (f) $N = 490.000 rnd$

Abb. 35: Adaption nach 20 Schritten. $\sigma_{crs} = 0{,}006$; $\sigma_{ref} = 0{,}10$



(a) $N = 10.000 grid$ (b) $N = 10.000 rnd$

(c) $N = 40.000 grid$ (d) $N = 40.000 rnd$

(e) $N = 490.000 grid$ (f) $N = 490.000 rnd$

Abb. 36: Adaption nach 20 Schritten. $\sigma_{crs} = 0{,}006$; $\sigma_{ref} = 0{,}1$

Parameter Abb. 35 und 36:	
Adaption:	MLS-Approximation
Nachbarschaften:	$n = 50$
Verfeinern:	$\sigma_{ref} = 0{,}10$
Ausdünnen:	$\sigma_{crs} = 0{,}006$
Adaptionsschritte:	$t_A = 20$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 10.000$ *grid* (35(a), 36(a))
	$N = 10.000$ *rnd* (35(b), 36(b))
	$N = 40.000$ *grid* (35(c), 36(c))
	$N = 40.000$ *rnd* (35(d), 36(d))
	$N = 490.000$ *grid* (35(e), 36(e))
	$N = 490.000$ *rnd* (35(f), 36(f))

Die folgende Tabelle zeigt die Knotenzahlen, bei der die Adaption stationär wird, für die unterschiedlichen Ausgangsverteilungen.

N	*grid*	*rnd*
10.000	109.000	97.000
40.000	186.000	115.000
90.000	252.000	153.000
250.000	322.000	181.000
490.000	411.000	250.000

Eine grafische Übersicht über die Tabelle gibt uns Abbildung 37. Die Zahl der Knoten bei der Adaption wächst langsamer als die Knotenzahl der Ausgangsverteilung. Da die absoluten Zahlen jedoch sehr hoch sind, werden wir uns bei den weiteren Betrachtungen auf 40.000 Punkte bei der anfänglichen Knotenverteilung beschränken.
Um die Eignung der MLS-Approximation für glatte Funktionen zu überprüfen, verwenden wir als Ausgangskonzentration die Wendland-Funktion. Wir werden dabei die oben festgelegten Parameter beibehalten. Die Qualität der Reproduktion ist sowohl bei der zufallsverteilten Ausgangsmenge als auch bei der gitterbasierten sehr gut. Abbildung 38 zeigt uns die gleichmäßig verteilte Anfangskonzentration sowie die Konzentration nach 20 Adaptionsschritten. Nach der Adaption ist lediglich im Bereich der Spitze des Funktionskegels eine leichte Abflachung erfolgt, wie wir in der dargestellten Vergrößerung sehen. Außerdem dargestellt sind die Knoten vor und nach der Adaption sowie die Triangulierung der angepassten Knotenmenge. Entsprechend zeigt Abbildung 39 die Adaption bei zufällig verteilten Knoten.

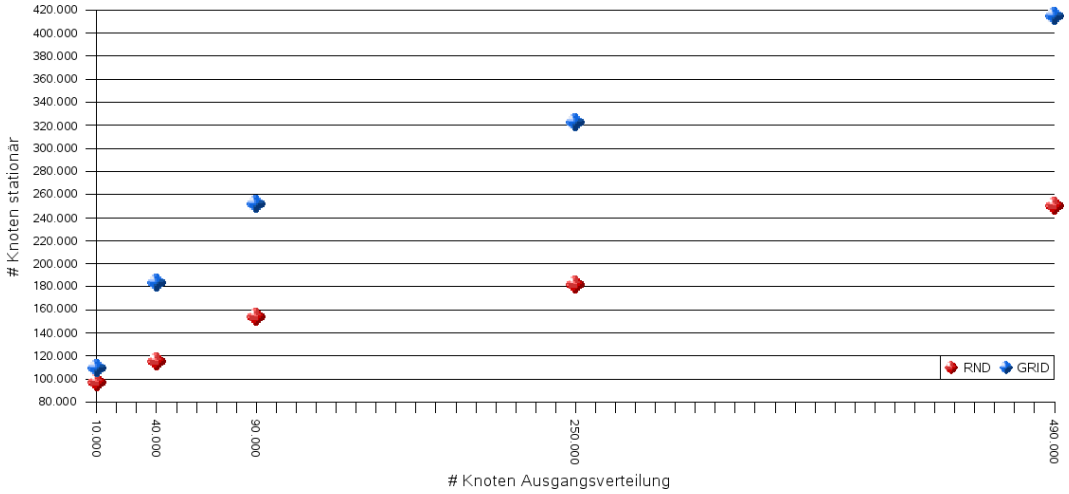
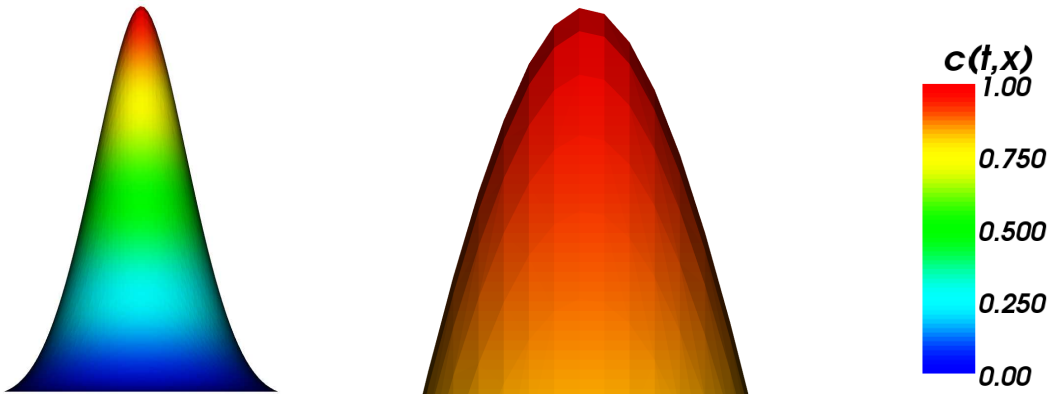


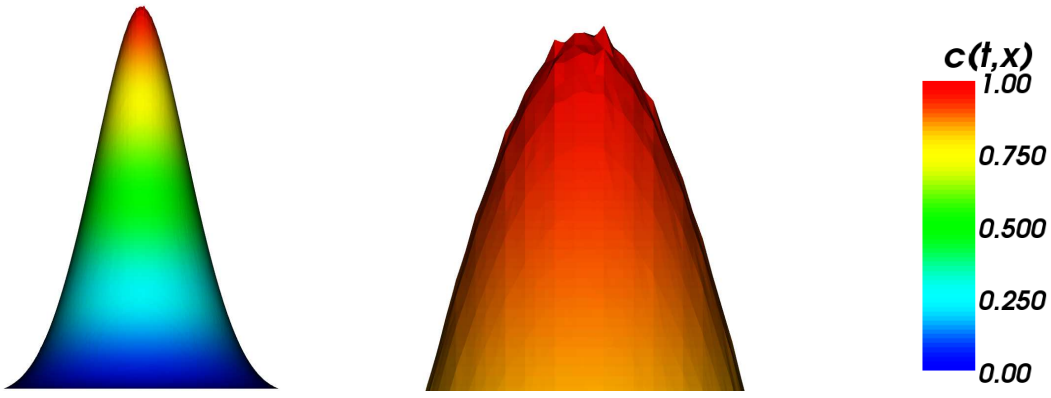
Abb. 37: Anzahl der Knoten bei der stationären Adaption für unterschiedliche Ausgangsverteilungen.

`Parameter Abb. 38, 39 und 40:`	
`Adaption:`	`MLS-Approximation`
`Nachbarschaft:`	$n = 50$
`Verfeinern:`	$\sigma_{ref} = 0{,}10$
`Ausdünnen:`	$\sigma_{crs} = 0{,}006$
`Adaptionsschritte:`	$t_A = 20$ `(38, 39)`
	$t_A = 50$ `(40)`
`Konzentration:`	`Wendland-Funktion`
`Initiale Knotenzahl:`	$N = 40.000$ *`grid`* `(38)`
	$N = 40.000$ *`rnd`* `(39)`

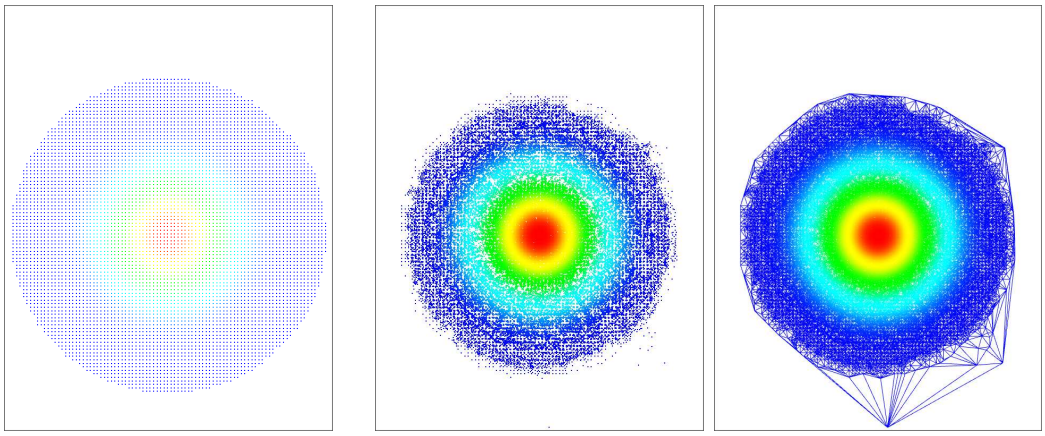
Wir betrachten nun die Entwicklung der Knotenzahl bei der Ausführung mehrerer Adaptionsschritte. Bei der Slotted Disc wurde die Knotenmenge nach ca. 15 Schritten stationär, d.h. die Anzahl von Knoten näherte sich einem gewissen Grenzwert. Abbildung 40 zeigt, dass bei der Adaption der glatten Ausgangsverteilung größere Sprünge in der Knotenzahl vorkommen. Die Annäherung an einen Grenzwert findet hier wesentlich weniger gleichmäßig statt als bei der Slotted Disc.
Diese Schwankungen in der Knotenmenge haben so gut wie keine Auswirkungen auf die optische Darstellung. Da wir aber für die später eingesetzte Advektion eine stationäre Adaption voraussetzen, benötigen wir Kriterien, ab wann die Knotenmenge als stationär gelten soll. Diese Kriterien werden wir später genauer erläutern und zunächst einmal die RBF-Interpolation auf ihre Tauglichkeit für die Adaption überprüfen.



(a) $t_A = 0$

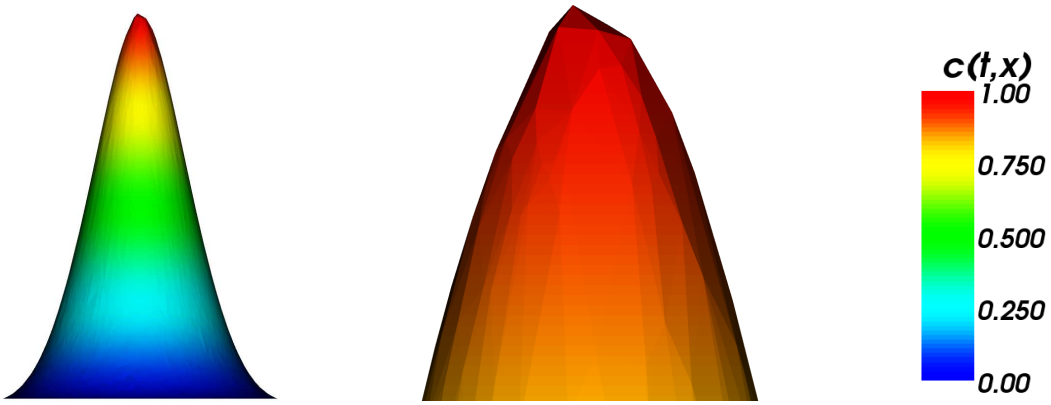


(b) $t_A = 20$

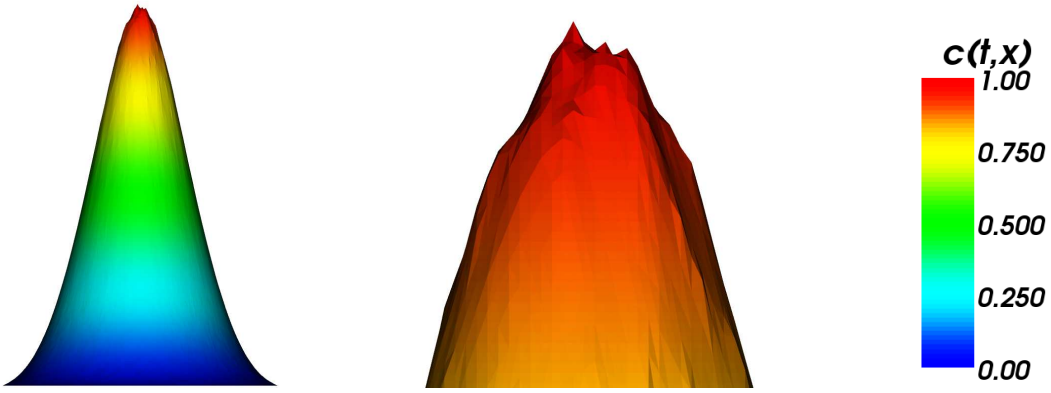


(c) $t_A = 0$ (d) $t_A = 20$

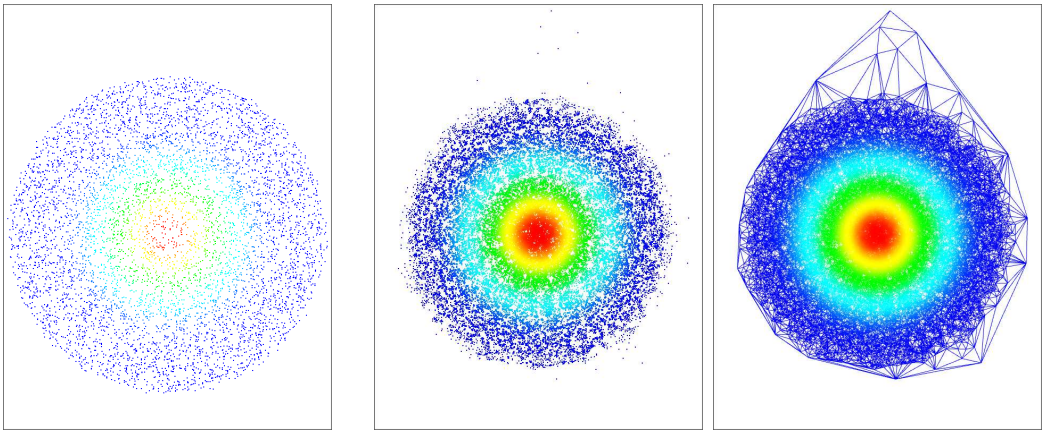
Abb. 38: Adaption der Wendland-Funktion. $N = 40.000 grid$



(a) $t_A = 0$



(b) $t_A = 20$



(c) $t_A = 0$ (d) $t_A = 20$

Abb. 39: Adaption der Wendland-Funktion. $N = 40.000 rnd$

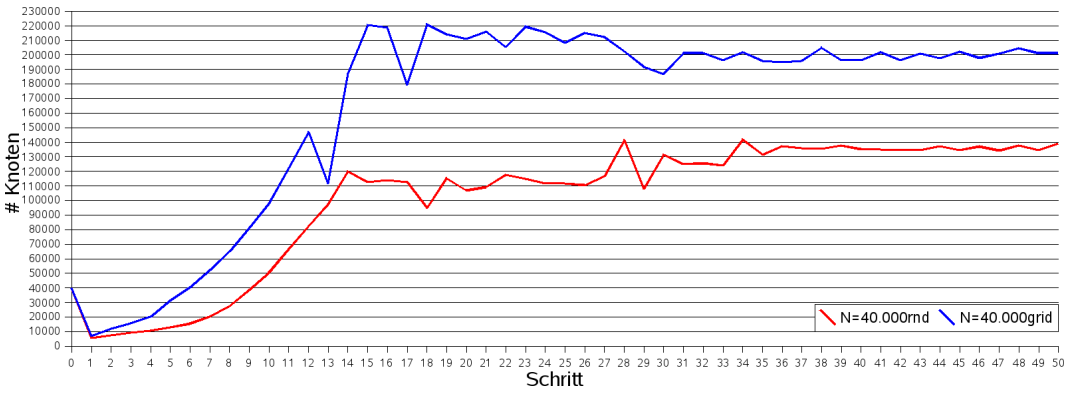
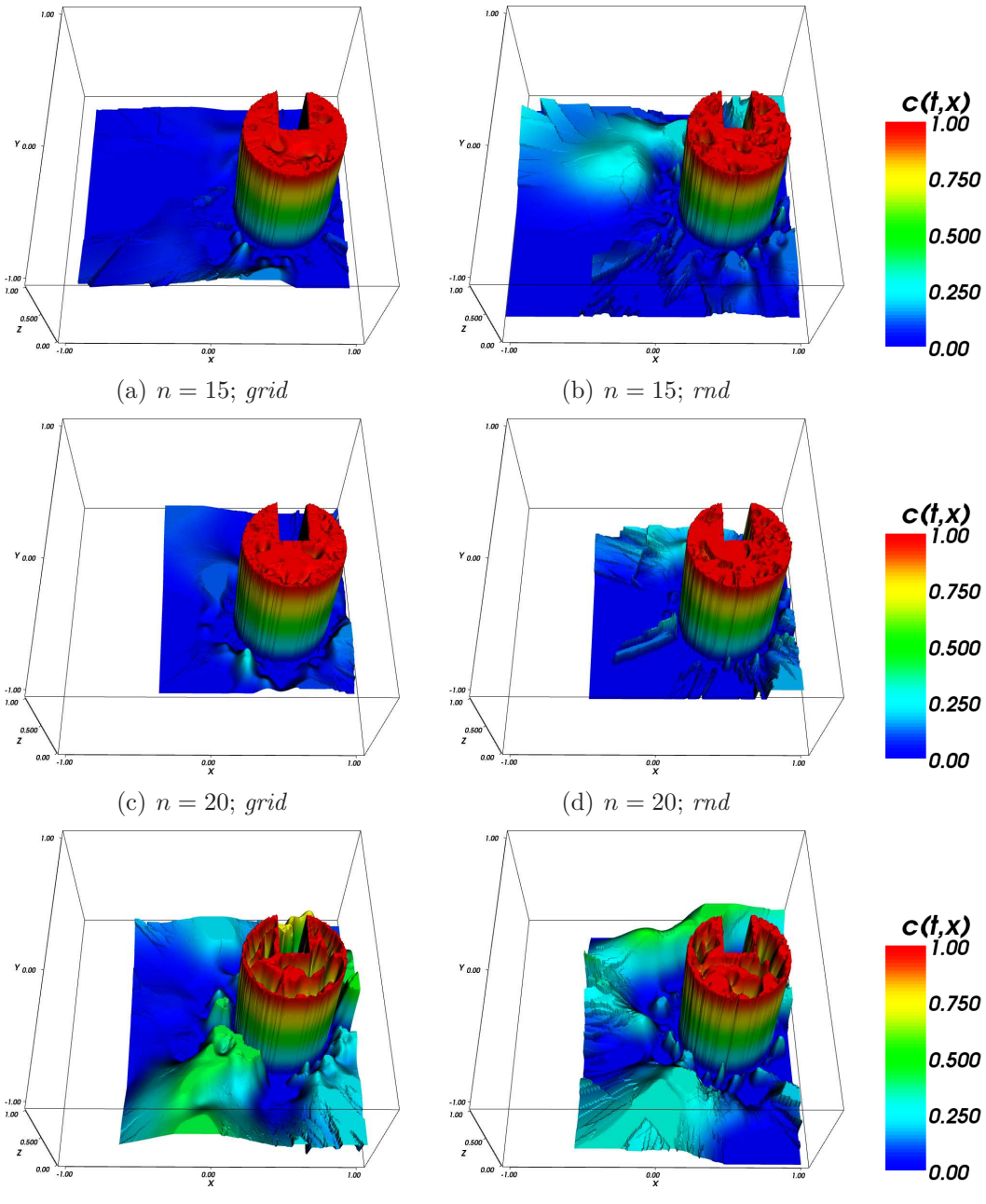


Abb. 40: Entwicklung der Knotenzahl bei der Adaption

3.5.2 Adaption mit RBF - Wendland-Funktion

Für die Adaption mit der RBF-Interpolation beginnen wir unsere Betrachtungen mit der Wendland-Funktion $\phi(r) = (1-r)_+^4(4r+1)$ als Basisfunktion. Wir werden zunächst die optimale Anzahl von Knoten in den Nachbarschaften bestimmen. Dafür setzen wir $\sigma_{crs} = 0{,}006$ und $\sigma_{ref} = 0{,}1$. Als Ausgangsverteilung verwenden wir die Slotted Disc mit 40.000 Knoten. Abbildung 41 zeigt die Adaption nach $t_A = 10$ Adaptionsschritten für Nachbarschaften mit 10, 15 und 20 Knoten. Bei der links dargestellten gitterbasierten Anfangsverteilung erzielen wir mit $n = 15$ und $n = 20$ noch akzeptable Ergebnisse, während für $n = 25$ sehr starke Oszillationen der Konzentration zu erkennen sind. Bei einer Zufallsverteilung treten für alle Größen von Nachbarschaften ebenfalls deutliche Oszillationen auf, die im Bereich von $n = 20$ noch am Geringsten sind. Durch die Wahl von σ_{crs} bzw. σ_{ref} werden wir versuchen, die Oszillationen zu minimieren.

Parameter Abb. 41:	
Adaption:	RBF-Interpolation
RBF:	Wendland-Funktion
Nachbarschaften:	$n = 15$ (41(a), 41(b))
	$n = 20$ (41(c), 41(d))
	$n = 25$ (41(e), 41(f))
Ausdünnen:	$\sigma_{crs} = 0{,}006$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$***grid*** (41(a), 41(c), 41(e))
	$N = 40.000$***rnd*** (41(b), 41(d), 41(f))



(a) $n = 15$; *grid* (b) $n = 15$; *rnd*

(c) $n = 20$; *grid* (d) $n = 20$; *rnd*

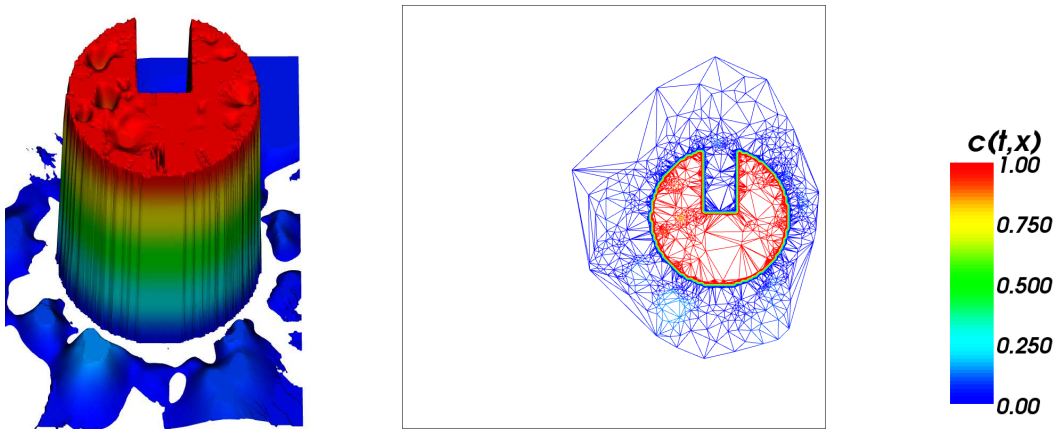
(e) $n = 25$; *grid* (f) $n = 25$; *rnd*

Abb. 41: Adaption mit der Wendland-Funktion bei unterschiedlichen Nachbarschaften. $t_A = 10$; $N = 40.000$

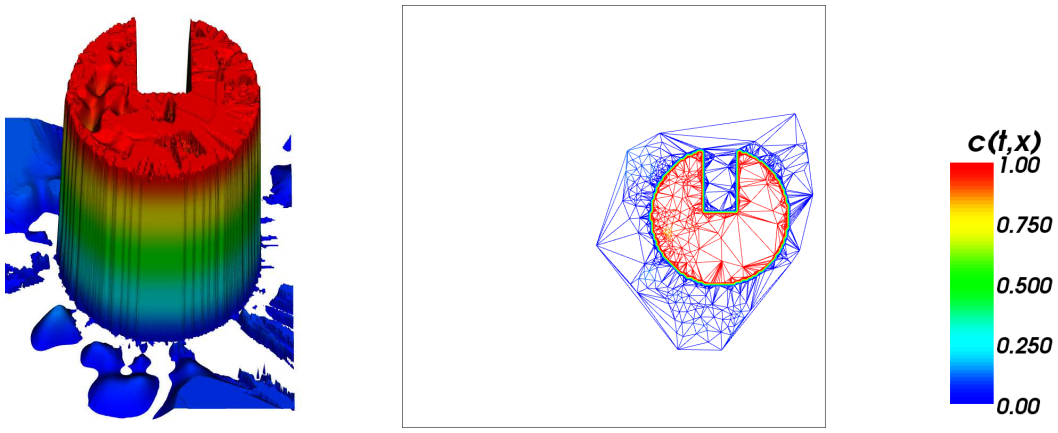
Wir betrachten nun für festes $n = 20$ die optimalen Werte für σ_{crs} und σ_{ref}. Abbildung 42 stellt für eine regelmäßig verteilte Knotenmenge die Adaption nach zehn Schritten dar. Dabei erzielen wir das beste Ergebnis für $\sigma_{crs} = 0{,}001$ und $\sigma_{ref} = 0{,}05$. In diesem Fall benötigen wir auch mit rund 180.000 die mit Abstand meisten Knoten. Bei einer Erhöhung von σ_{crs} oder σ_{ref} wird jedoch die Qualität der Reproduktion schlechter, so dass wir beide Parameter möglichst gering halten müssen.

Parameter Abb. 42:	
Adaption:	RBF-Interpolation
RBF:	Wendland-Funktion
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$ (42(a), 42(b))
	$\sigma_{crs} = 0{,}003$ (42(c))
Verfeinern:	$\sigma_{ref} = 0{,}05$ (42(a), 42(c))
	$\sigma_{ref} = 0{,}15$ (42(b))
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000 grid$

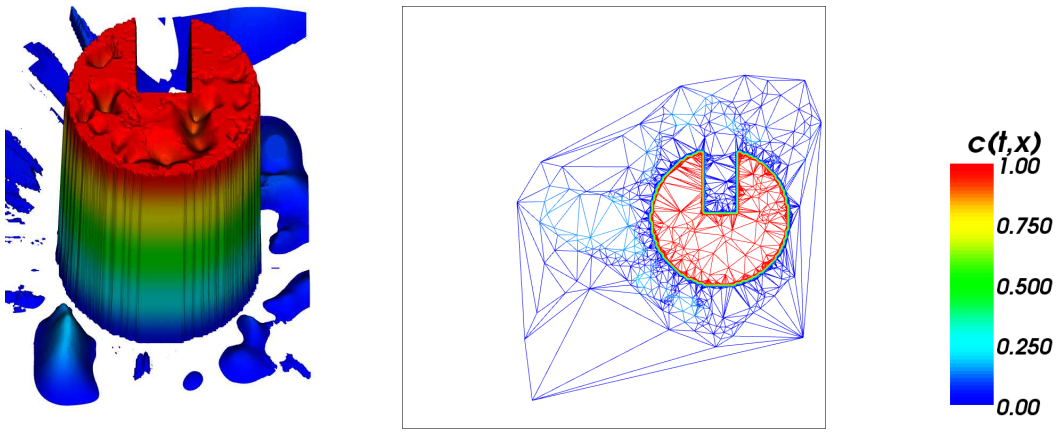
Abbildung 43 zeigt die Adaption einer zufallsverteilten Knotenmenge für unterschiedliche Werte von σ_{crs} und σ_{ref}. Es ist deutlich sichtbar, dass die Oszillationen der Konzentration bei allen Werten für σ_{crs} und σ_{ref} mehr oder weniger stark auftreten. Für sehr große σ_{ref} werden die Schwankungen außerhalb der Disc zwar geringer, jedoch haben wir selbst für $\sigma_{ref} = 0{,}2$ noch kein akzeptables Ergebnis. Da wir hier zudem erst ab einem relativen Fehler von 20% verfeinern, erscheint eine solche Wahl von σ_{ref} nicht praxisnah.



(a) $\sigma_{crs} = 0{,}001$; $\sigma_{ref} = 0{,}05$; $N_{10} = 179.987$

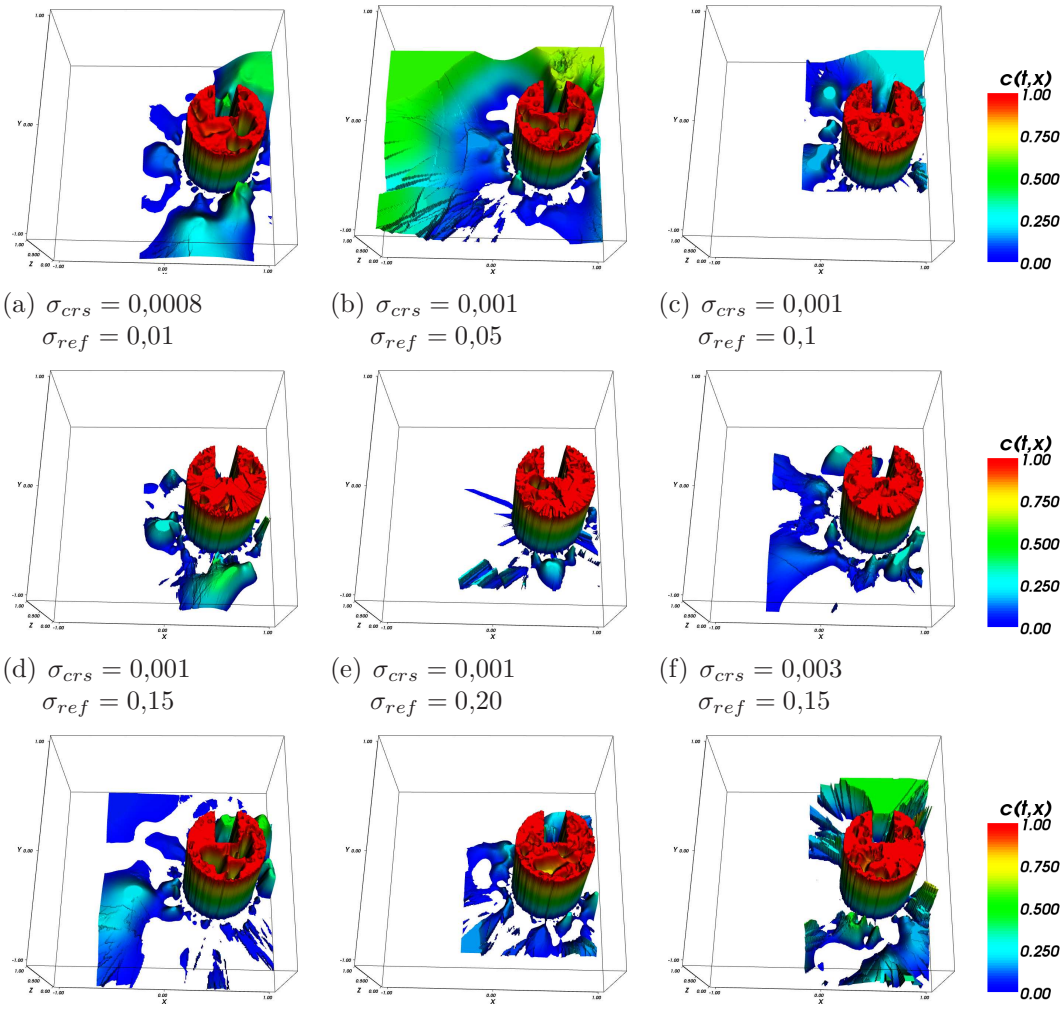


(b) $\sigma_{crs} = 0{,}001$; $\sigma_{ref} = 0{,}15$; $N_{10} = 37.372$



(c) $\sigma_{crs} = 0{,}003$; $\sigma_{ref} = 0{,}05$; $N_{10} = 31.029$

Abb. 42: Adaption mit der Wendland-Funktion. $t_A = 10$; $N = 40.000 grid$



(a) $\sigma_{crs} = 0{,}0008$ $\sigma_{ref} = 0{,}01$ (b) $\sigma_{crs} = 0{,}001$ $\sigma_{ref} = 0{,}05$ (c) $\sigma_{crs} = 0{,}001$ $\sigma_{ref} = 0{,}1$

(d) $\sigma_{crs} = 0{,}001$ $\sigma_{ref} = 0{,}15$ (e) $\sigma_{crs} = 0{,}001$ $\sigma_{ref} = 0{,}20$ (f) $\sigma_{crs} = 0{,}003$ $\sigma_{ref} = 0{,}15$

(g) $\sigma_{crs} = 0{,}005$ $\sigma_{ref} = 0{,}05$ (h) $\sigma_{crs} = 0{,}007$ $\sigma_{ref} = 0{,}1$ (i) $\sigma_{crs} = 0{,}007$ $\sigma_{ref} = 0{,}2$

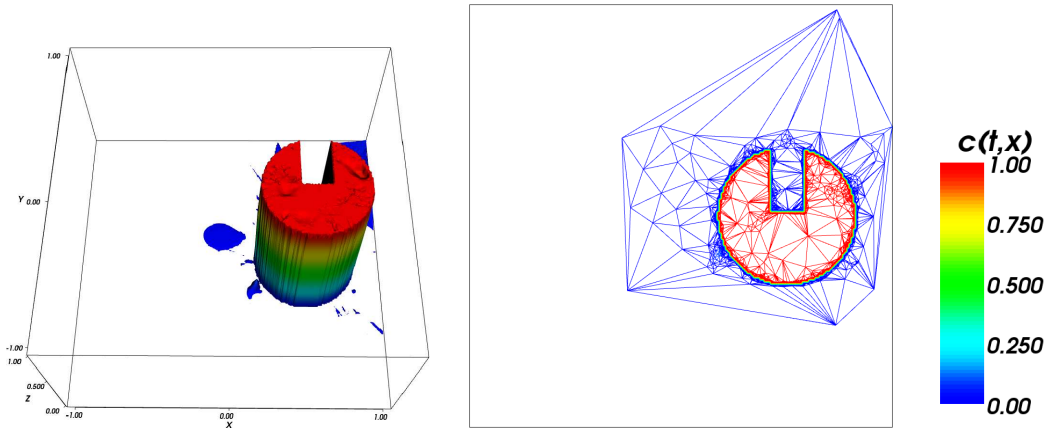
Abb. 43: Adaption mit der Wendland-Funktion. $t_A = 10$; $N = 40.000 rnd$

Parameter Abb. 43:	
Adaption:	RBF-Interpolation
RBF:	Wendland-Funktion
Nachbarschaften:	$n = 20$
Ausdünnen & Verfeinern:	$\sigma_{crs} = 0{,}0008;\ \sigma_{ref} = 0{,}01$ (43(a))
	$\sigma_{crs} = 0{,}0010;\ \sigma_{ref} = 0{,}05$ (43(b))
	$\sigma_{crs} = 0{,}0010;\ \sigma_{ref} = 0{,}10$ (43(c))
	$\sigma_{crs} = 0{,}0010;\ \sigma_{ref} = 0{,}15$ (43(d))
	$\sigma_{crs} = 0{,}0010;\ \sigma_{ref} = 0{,}20$ (43(e))
	$\sigma_{crs} = 0{,}0030;\ \sigma_{ref} = 0{,}15$ (43(f))
	$\sigma_{crs} = 0{,}0050;\ \sigma_{ref} = 0{,}05$ (43(g))
	$\sigma_{crs} = 0{,}0070;\ \sigma_{ref} = 0{,}10$ (43(h))
	$\sigma_{crs} = 0{,}0070;\ \sigma_{ref} = 0{,}20$ (43(i))
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$*rnd*

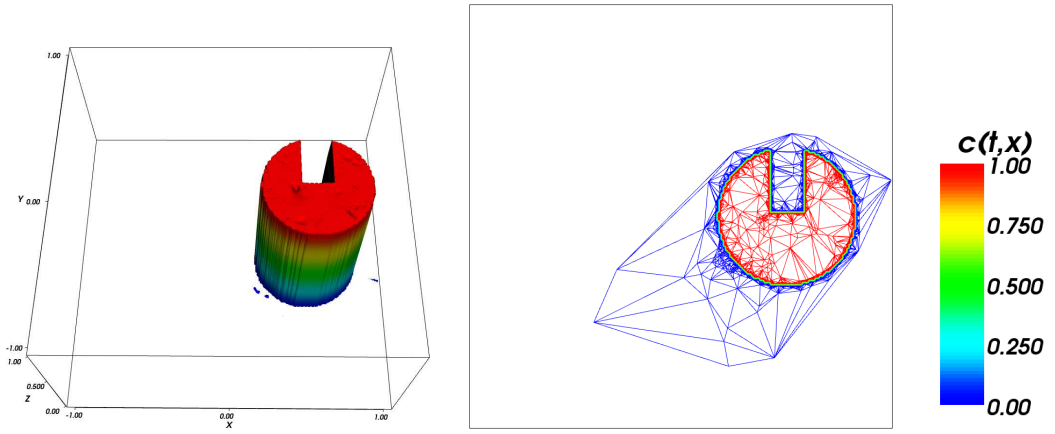
Wir werden uns daher für die Adaption mit der RBF-Interpolation auf den Thin Plate Spline als Basisfunktion konzentrieren und die Wendland-Funktion nicht weiter betrachten.

3.5.3 Adaption mit RBF - Thin Plate Spline

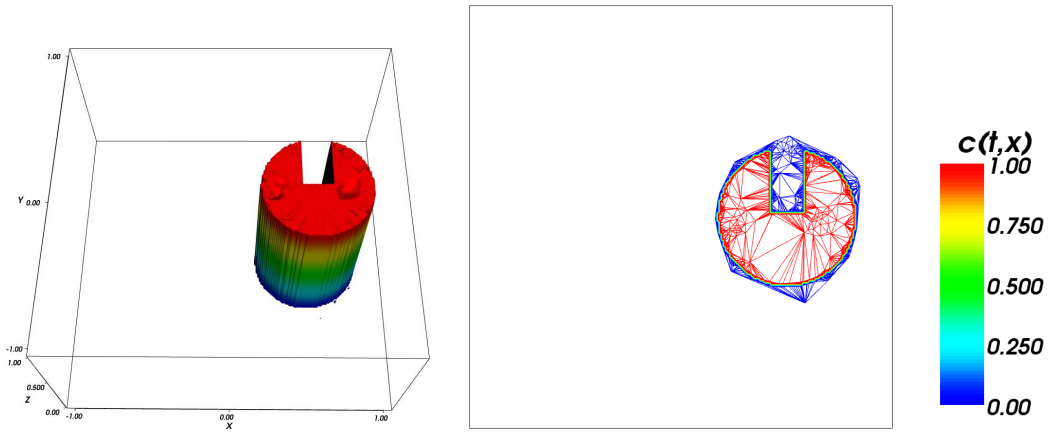
Der Thin Plate Spline erweist sich als wesentlich besser für die Adaption geeignet als die Wendland-Funktion. Wir ermitteln auch hier zunächst die optimale Anzahl von Punkten in den Nachbarschaften und halten $\sigma_{crs} = 0{,}006$ sowie $\sigma_{ref} = 0{,}1$ fest. Abbildung 44 zeigt die Adaption bei gleichmäßiger Anfangsverteilung mit $N \in \{10; 20; 30\}$. Dabei ist die Lage der Knoten nach zehn Adaptionsschritten bei $n = 30$ am besten auf die Disc konzentriert, die Reproduktion der Disc selber ist jedoch bei $n = 20$ am besten. Für $n = 10$ kommt es zu einer schlechteren Reproduktion außerhalb der Disc.
Dieser Effekt wird bei zufälliger Anordnung der Knotenmenge noch verstärkt, wie Abbildung 45 zeigt. Hier kommt es für $n = 10$ und $n = 30$ zu Oszillationen in der Konzentration neben der Disc. Daher werden wir mit 20 Knoten in den Nachbarschaften weiterarbeiten.



(a) $n = 10$; $N_{10} = 115.347$

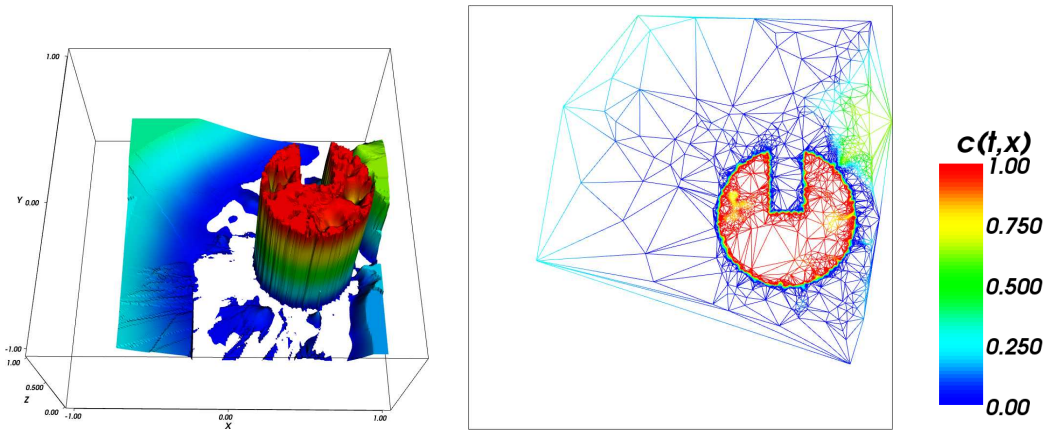


(b) $n = 20$; $N_{10} = 130.066$

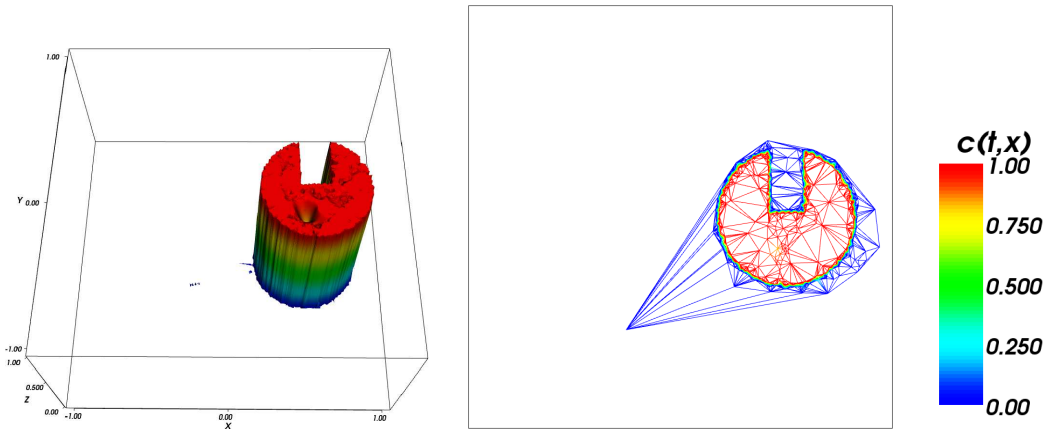


(c) $n = 30$; $N_{10} = 77.174$

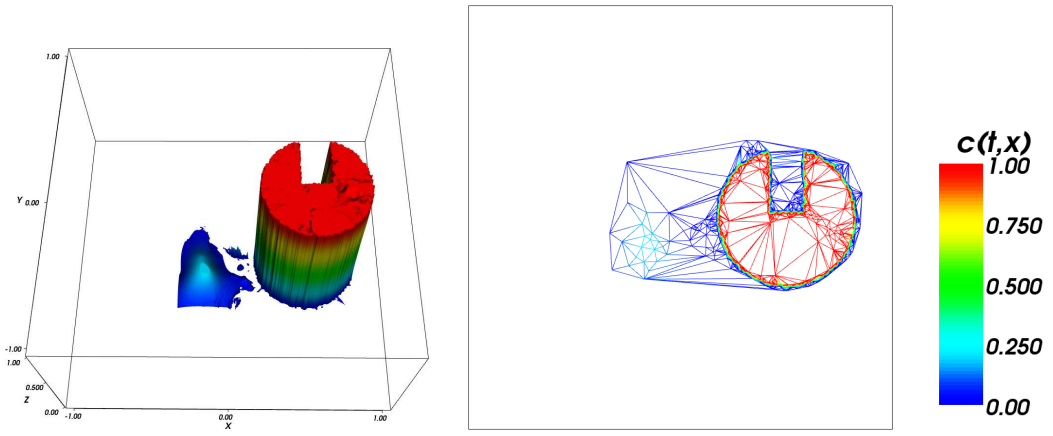
Abb. 44: Adaption mit dem Thin Plate Spline mit unterschiedlichen Nachbarschaften. $N = 40.000 grid$



(a) $n = 10$; $N_{10} = 31.383$



(b) $n = 20$; $N_{10} = 9.919$



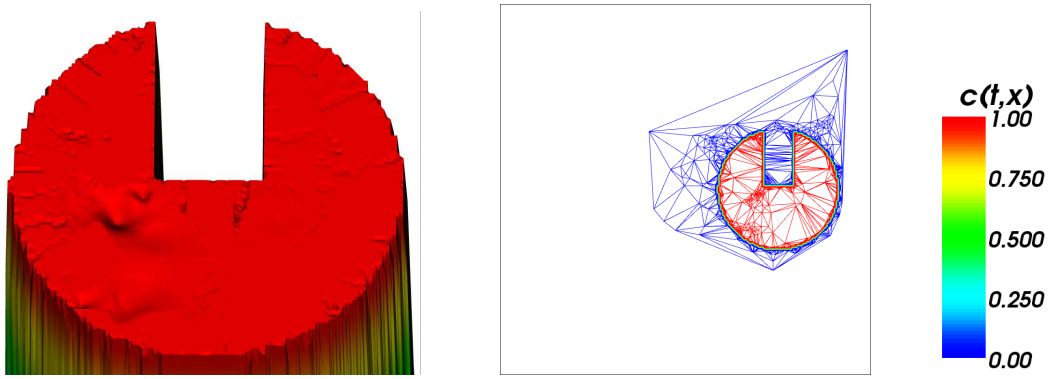
(c) $n = 30$; $N_{10} = 6.846$

Abb. 45: Adaption mit dem Thin Plate Spline mit unterschiedlichen Nachbarschaften. $N = 40.000 rnd$

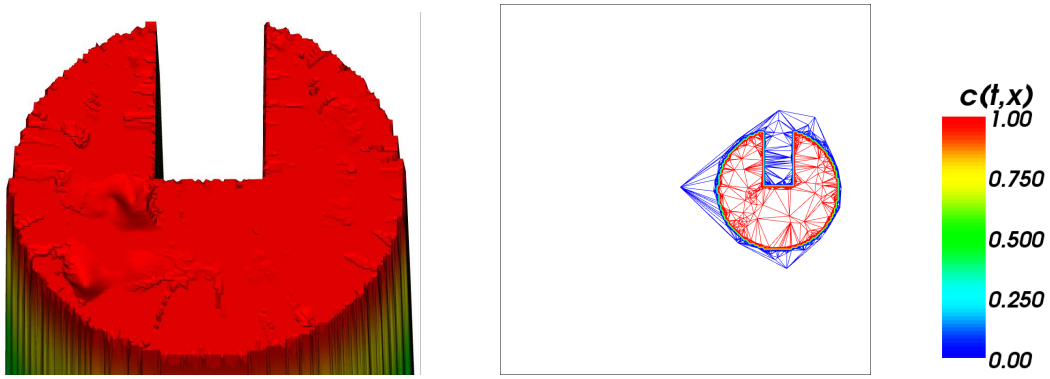
Parameter Abb. 44 und 45:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 10$ (44(a), 45(a))
	$n = 20$ (44(b), 45(b))
	$n = 30$ (44(c), 45(c))
Ausdünnen:	$\sigma_{crs} = 0{,}006$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (44)
	$N = 40.000$ *rnd* (45)

Bemerkenswert ist, dass die Adaption der zufällig verteilten Knotenmenge mit wesentlich weniger Punkten auskommt. Für $n = 20$ liefert das Verfahren nach zehn Schritten rund 130.000 Punkte bei der gleichmäßigen, jedoch weniger als 10.000 Punkte bei der zufälligen Verteilung. Wir werden später genauer auf die Entwicklung der Knotenzahl bei der Adaption eingehen und zunächst σ_{crs} und σ_{ref} festlegen.
Die Abbildungen 46 und 47 zeigen die Adaption nach zehn Schritten mit unterschiedlichen Werten für σ_{ref}. Die Reproduktion der Slotted Disc ist für niedrigere Werte von σ_{ref} besser als für höhere. Dafür werden jedoch wesentlich mehr Punkte benötigt. Für $\sigma_{ref} = 0{,}05$ liefert die Adaption rund dreimal so viele Punkte wie für $\sigma_{ref} = 0{,}1$. Da hier die Adaption die Disc nicht sehr viel schlechter reproduziert, können wir für eine bessere Performance $\sigma_{ref} = 0{,}1$ wählen. Wir werden im Folgenden beide Werte weiter betrachten, um die Adaption zu optimieren.

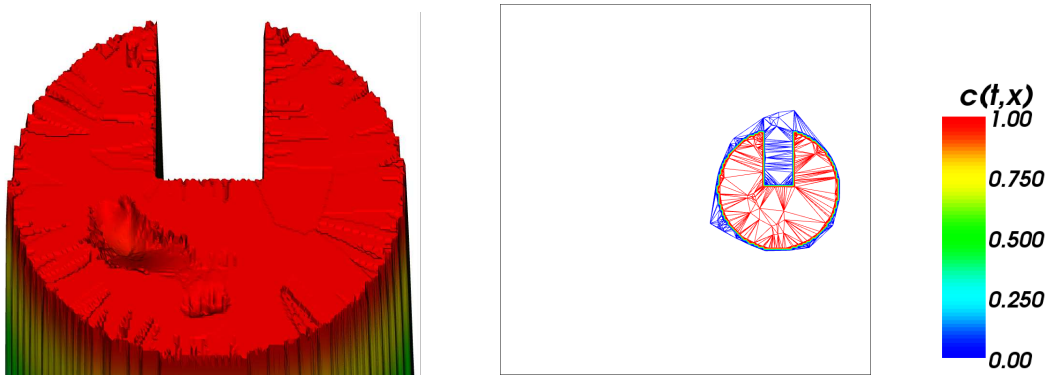
Parameter Abb. 46 und 47:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}05$ (46(a), 47(a))
	$\sigma_{ref} = 0{,}10$ (46(b), 47(b))
	$\sigma_{ref} = 0{,}15$ (46(c), 47(c))
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (46)
Initiale Knotenzahl:	$N = 40.000$ *rnd* (47)



(a) $\sigma_{ref} = 0{,}05$; $N_{10} = 498.653$

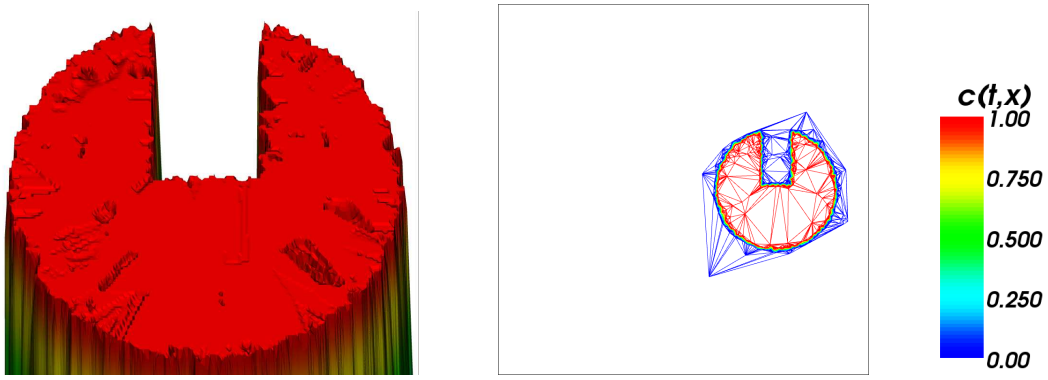


(b) $\sigma_{ref} = 0{,}10$; $N_{10} = 155.797$

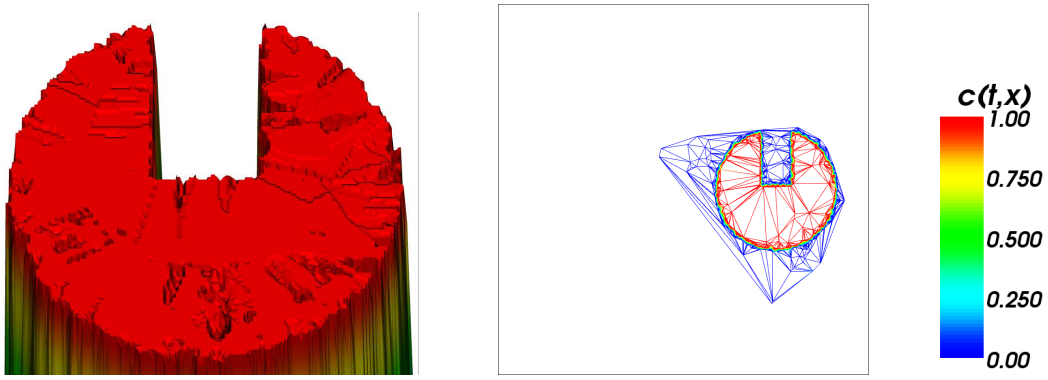


(c) $\sigma_{ref} = 0{,}15$; $N_{10} = 39.482$

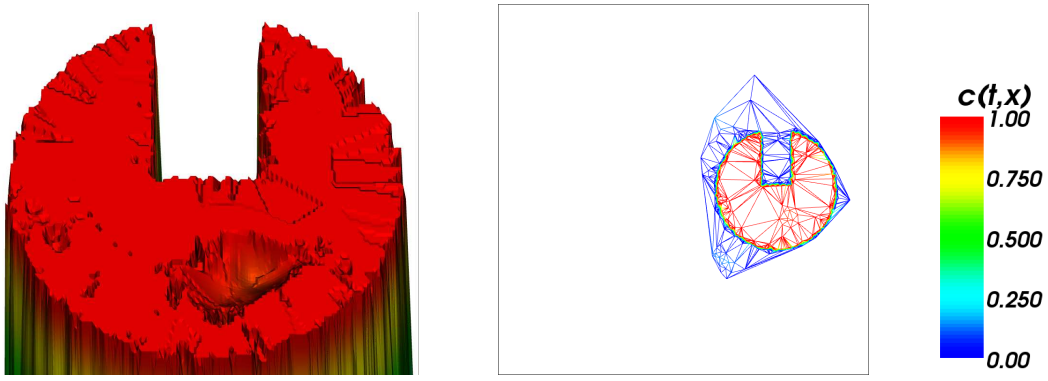
Abb. 46: Adaption mit dem Thin Plate Spline. $N = 40.000 grid$



(a) $\sigma_{ref} = 0{,}05$; $N_{10} = 45.609$



(b) $\sigma_{ref} = 0{,}10$; $N_{10} = 18.066$



(c) $\sigma_{ref} = 0{,}15$; $N_{10} = 11.653$

Abb. 47: Adaption mit dem Thin Plate Spline. $N = 40.000rnd$

Auf Veränderungen von σ_{crs} reagiert das Adaptionsverfahren relativ empfindlich. Wir erreichen die besten Resultate im Bereich von $\sigma_{crs} = 0{,}001$. Die Abbildungen 50 und 51 zeigen die Slotted Disc nach zehn Adaptionsschritten für eine gleichmäßige Ausgangsverteilung (Abb. 48) mit $\sigma_{ref} \in \{0{,}1; 0{,}05\}$, $\sigma_{crs} \in \{0{,}0009; 0{,}001; 0{,}0011\}$. Die Unterschiede in der Qualität der Reproduktion sind hierbei nicht allzu groß, die Knotenzahlen sind jedoch sehr unterschiedlich hoch.

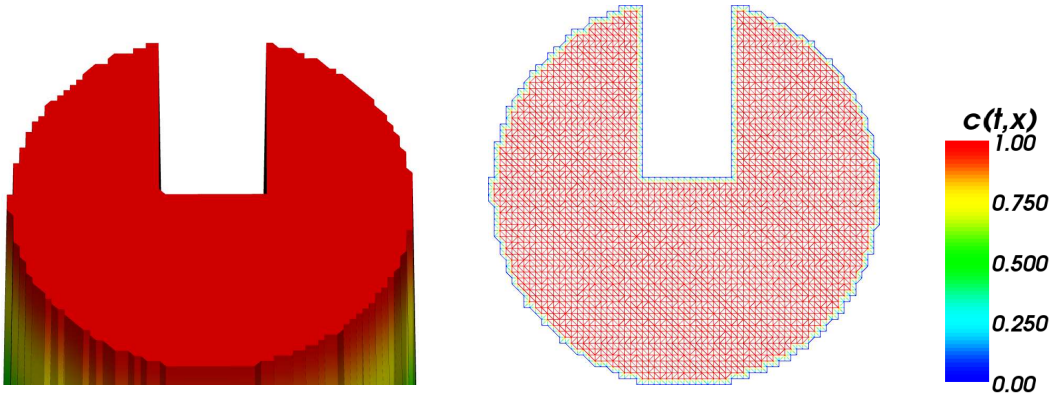


Abb. 48: Gleichmäßige Ausgangsverteilung. $N = 40.000 grid$

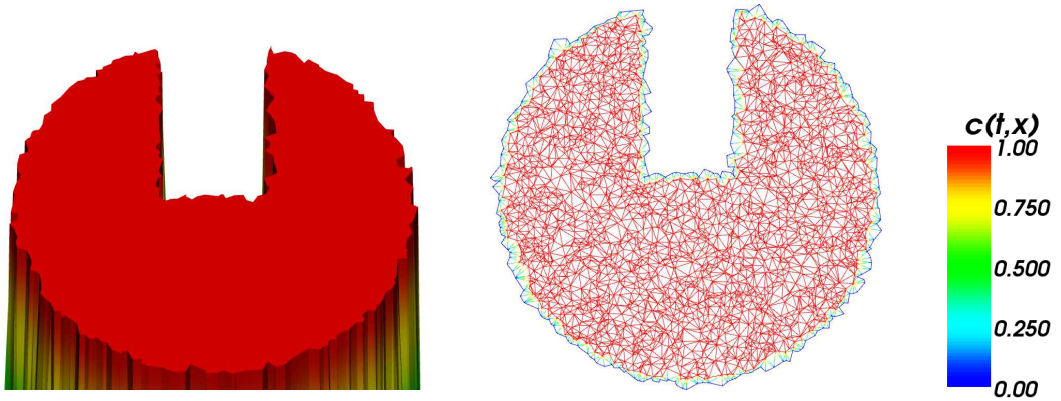
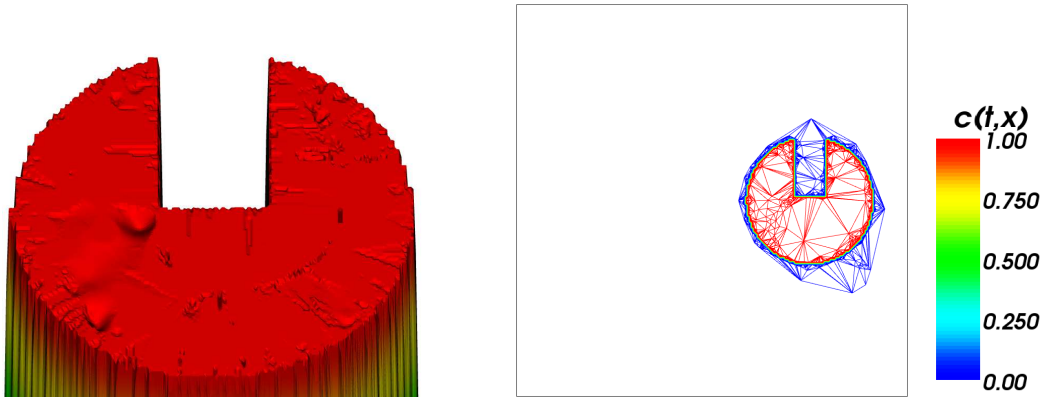
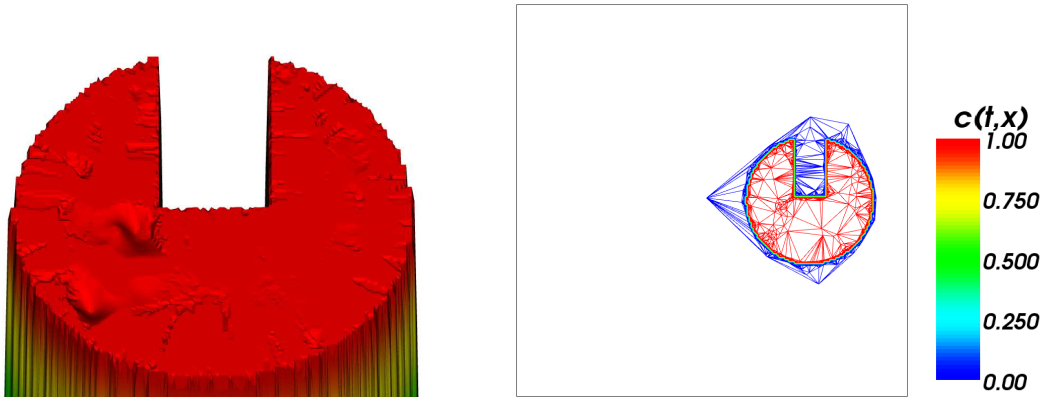


Abb. 49: Zufällige Ausgangsverteilung. $N = 40.000 rnd$

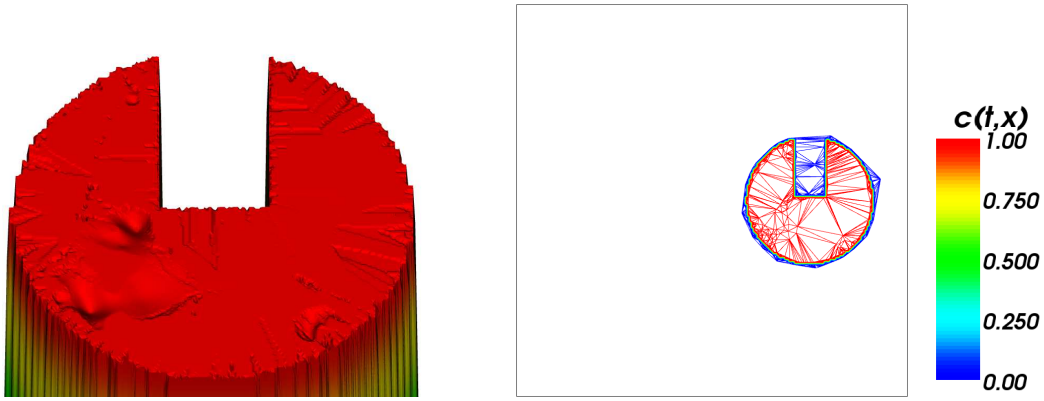
Bei einer zufälligen Ausgangsverteilung (Abb. 49) liefert das Verfahren mit denselben Parametern hingegen relativ konstante Knotenzahlen für die unterschiedlichen Werte von σ_{crs}. Dabei treten allerdings deutlich sichtbare Unterschiede in der Reproduktion der Disc auf, wie die Abbildungen 52 und 53 zeigen.



(a) $\sigma_{crs} = 0{,}0009$; $N_{10} = 120.281$

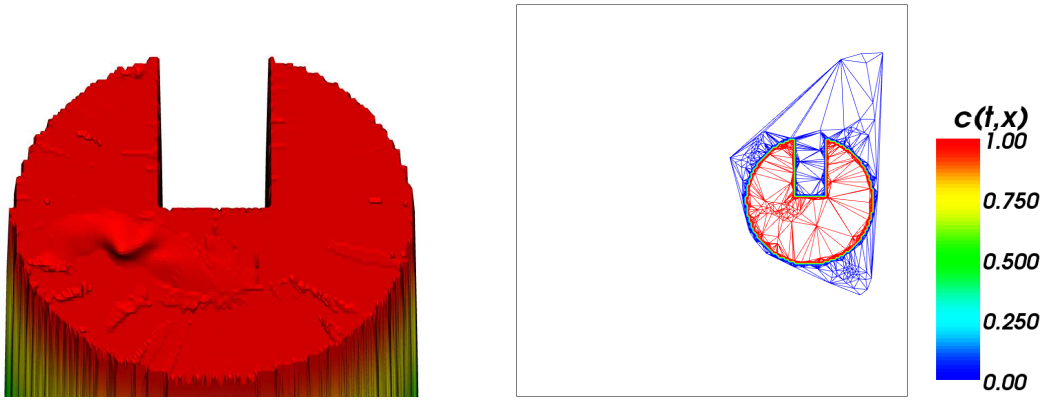


(b) $\sigma_{crs} = 0{,}0010$; $N_{10} = 155.797$

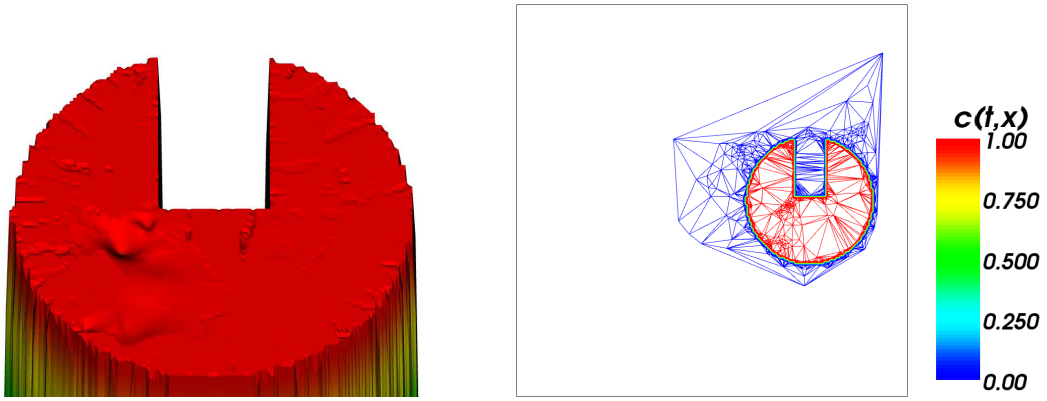


(c) $\sigma_{crs} = 0{,}0011$; $N_{10} = 108.639$

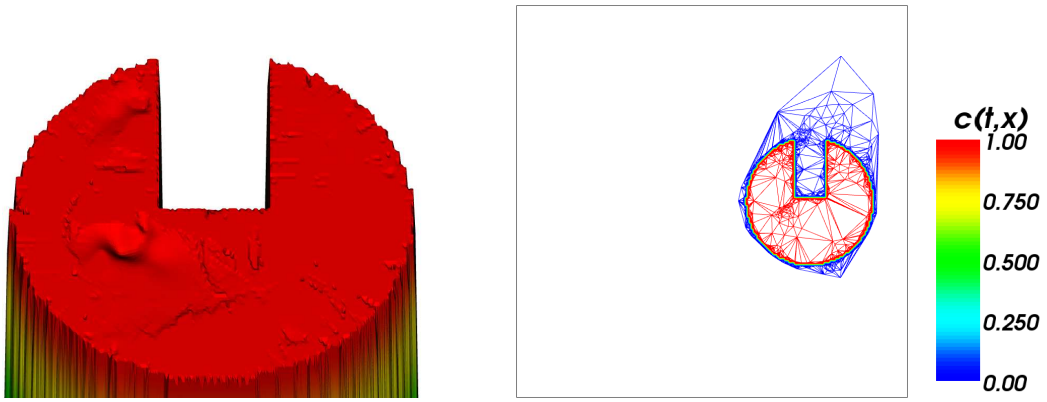
Abb. 50: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}10$; $N = 40.000 grid$



(a) $\sigma_{crs} = 0{,}0009$; $N_{10} = 600.571$

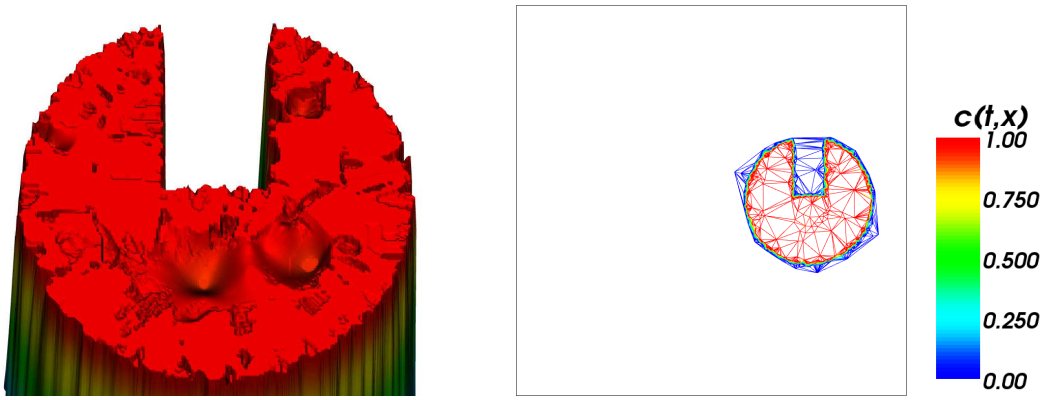


(b) $\sigma_{crs} = 0{,}0010$; $N_{10} = 498.653$

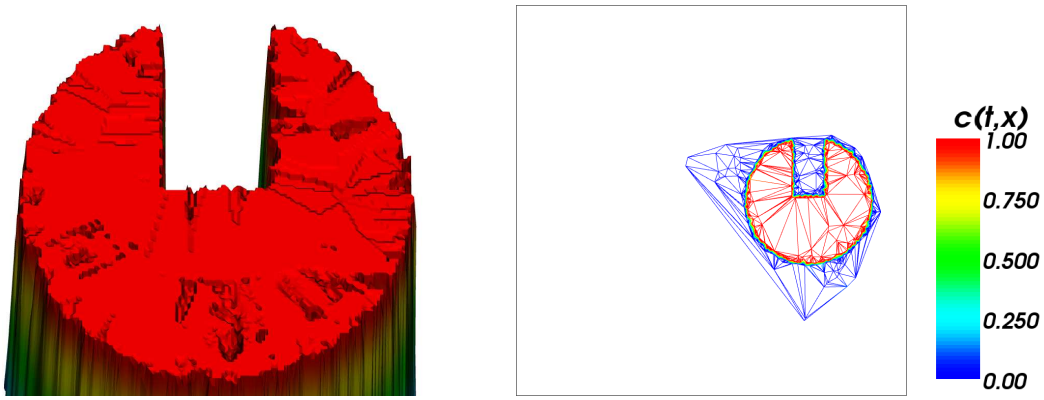


(c) $\sigma_{crs} = 0{,}0011$; $N_{10} = 434.711$

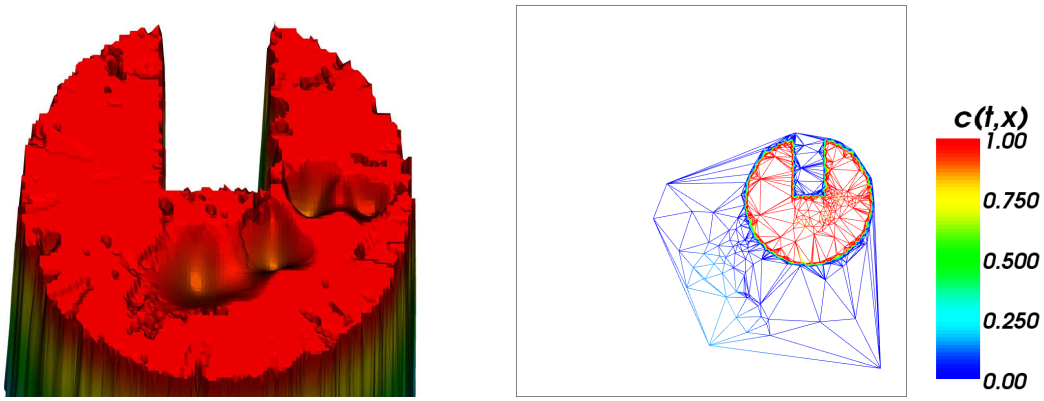
Abb. 51: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}05$; $N = 40.000 grid$



(a) $\sigma_{crs} = 0{,}0009$; $N_{10} = 17.966$

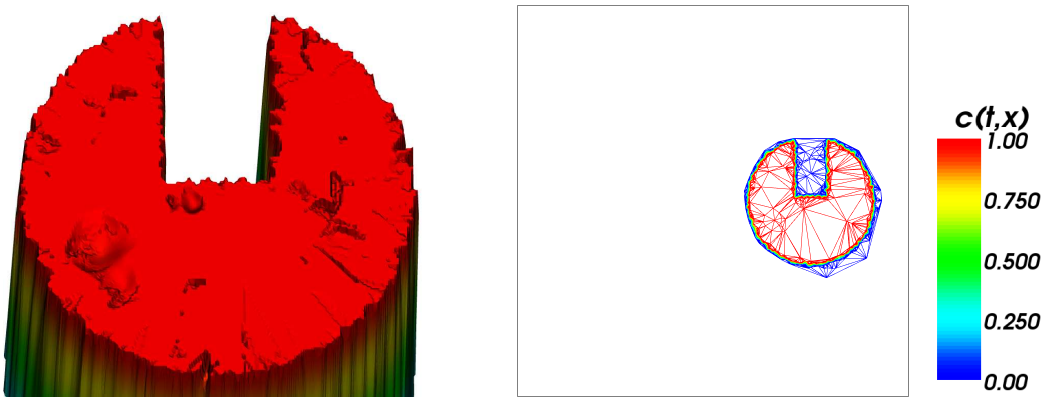


(b) $\sigma_{crs} = 0{,}0010$; $N_{10} = 18.066$

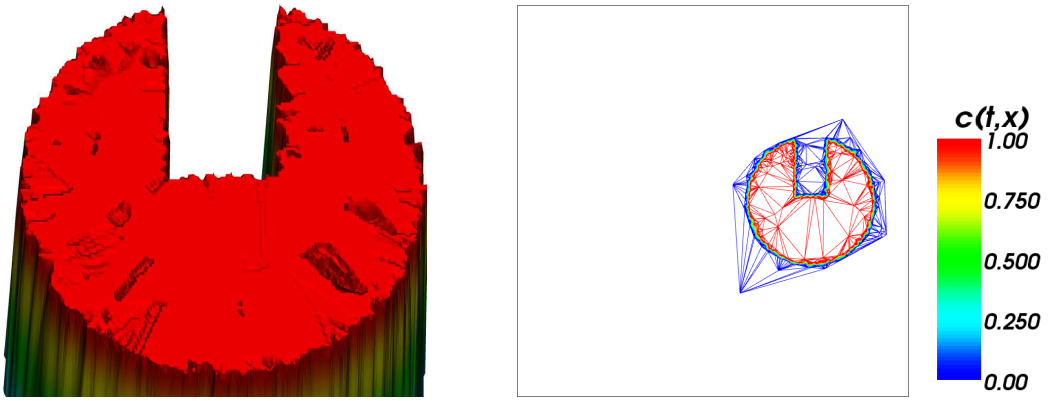


(c) $\sigma_{crs} = 0{,}0011$; $N_{10} = 17.502$

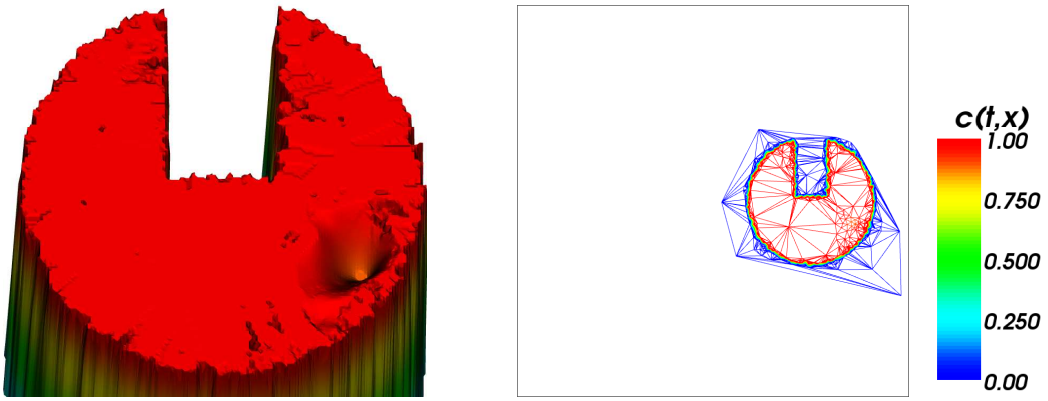
Abb. 52: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}10$; $N = 40.000 rnd$



(a) $\sigma_{crs} = 0{,}0009$; $N_{10} = 45.866$



(b) $\sigma_{crs} = 0{,}0010$; $N_{10} = 45.609$



(c) $\sigma_{crs} = 0{,}0011$; $N_{10} = 44.310$

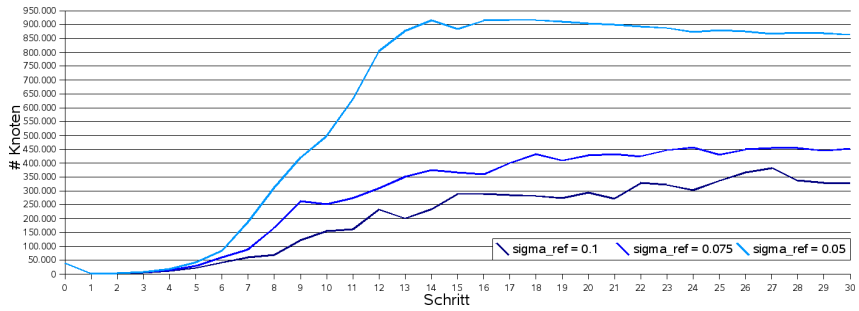
Abb. 53: Adaption nach 10 Schritten. $\sigma_{ref} = 0{,}05$; $N = 40.000 rnd$

Parameter Abb. 50, 51, 52 und 53:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}0009$ (50(a), 51(a), 52(a), 53(a))
	$\sigma_{crs} = 0{,}0010$ (50(b), 51(b), 52(b), 53(b))
	$\sigma_{crs} = 0{,}0011$ (50(c), 51(c), 52(c), 53(c))
Verfeinern:	$\sigma_{ref} = 0{,}10$ (50, 52)
	$\sigma_{ref} = 0{,}05$ (51, 53)
Adaptionsschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$*grid* (50, 51)
Initiale Knotenzahl:	$N = 40.000$*rnd* (52, 53)

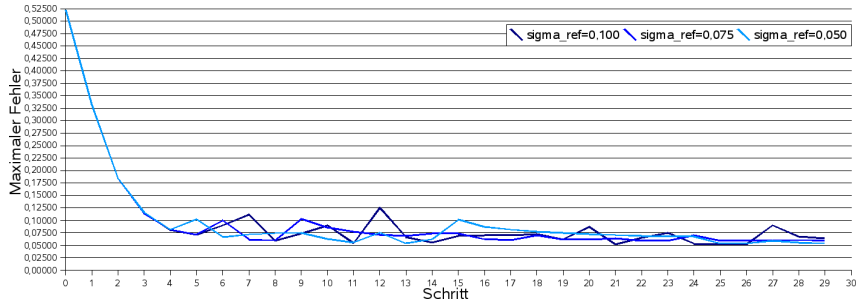
Insgesamt die beste Reproduktion liefert uns $\sigma_{crs} = 0{,}001$, so dass wir uns auf diesen Wert beschränken werden.
Betrachten wir nun die Entwicklung der Knotenzahlen während der Adaption. Dafür werden wir 30 Adaptionsschritte beobachten und sowohl die Knotenzahl als auch den maximalen Fehler für jeden Schritt notieren. Wir wählen als Ausgangsverteilung die Slotted Disc mit $40.000grid$ und $40.000rnd$ Knoten. Wir halten $\sigma_{crs} = 0{,}001$ fest und betrachten jeweils $\sigma_{ref} \in \{0{,}05; 0{,}075; 0{,}1\}$. Abbildung 54 zeigt uns die entsprechenden Zahlen grafisch.

Parameter Abb. 54:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}100$
	$\sigma_{ref} = 0{,}075$
	$\sigma_{ref} = 0{,}050$
Adaptionsschritte:	$t_A = 30$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$*grid* (54(a), 54(b))
	$N = 40.000$*rnd* (54(c), 54(d))

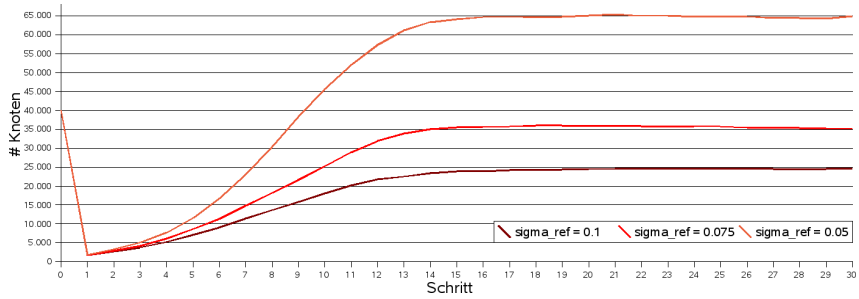
Wie wir sehen, wird für alle eingesetzten Parameter die Adaption nach einer Anzahl von Schritten stationär. Eine sehr gleichmäßige Konvergenz erfolgt dabei bei der zufallsverteilten Ausgangsmenge, wobei hier auch sehr viel geringere



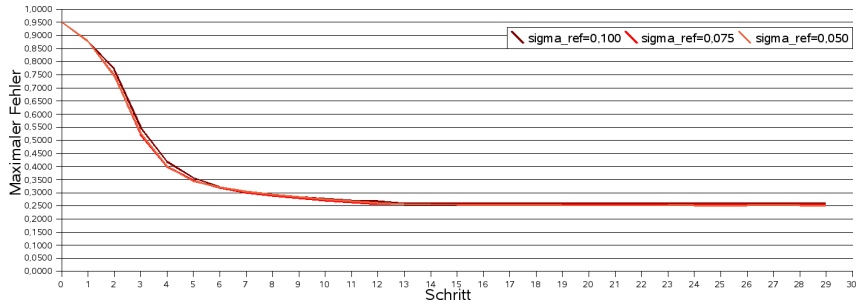
(a) Knotenzahl bei gleichmäßiger Ausgangsverteilung



(b) Maximaler Fehler bei gleichmäßiger Ausgangsverteilung

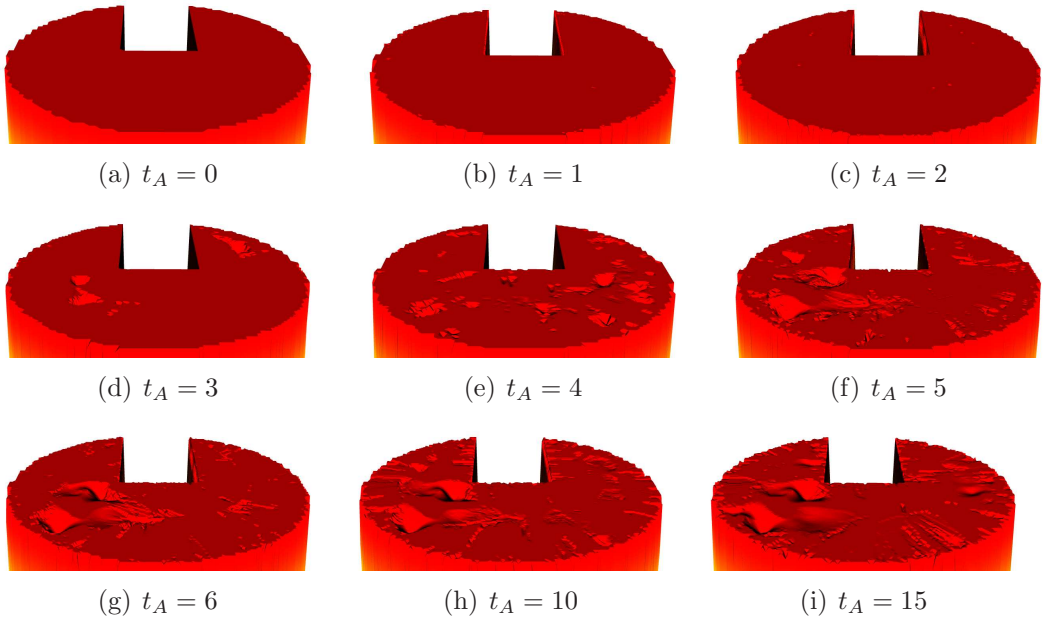


(c) Knotenzahl bei zufälliger Ausgangsverteilung



(d) Maximaler Fehler bei zufälliger Ausgangsverteilung

Abb. 54: Knotenzahl und maximaler Fehler bei der Adaption mit dem Thin Plate Spline. $\sigma_{crs} = 0{,}001$; $N = 40.000$



(a) $t_A = 0$ (b) $t_A = 1$ (c) $t_A = 2$

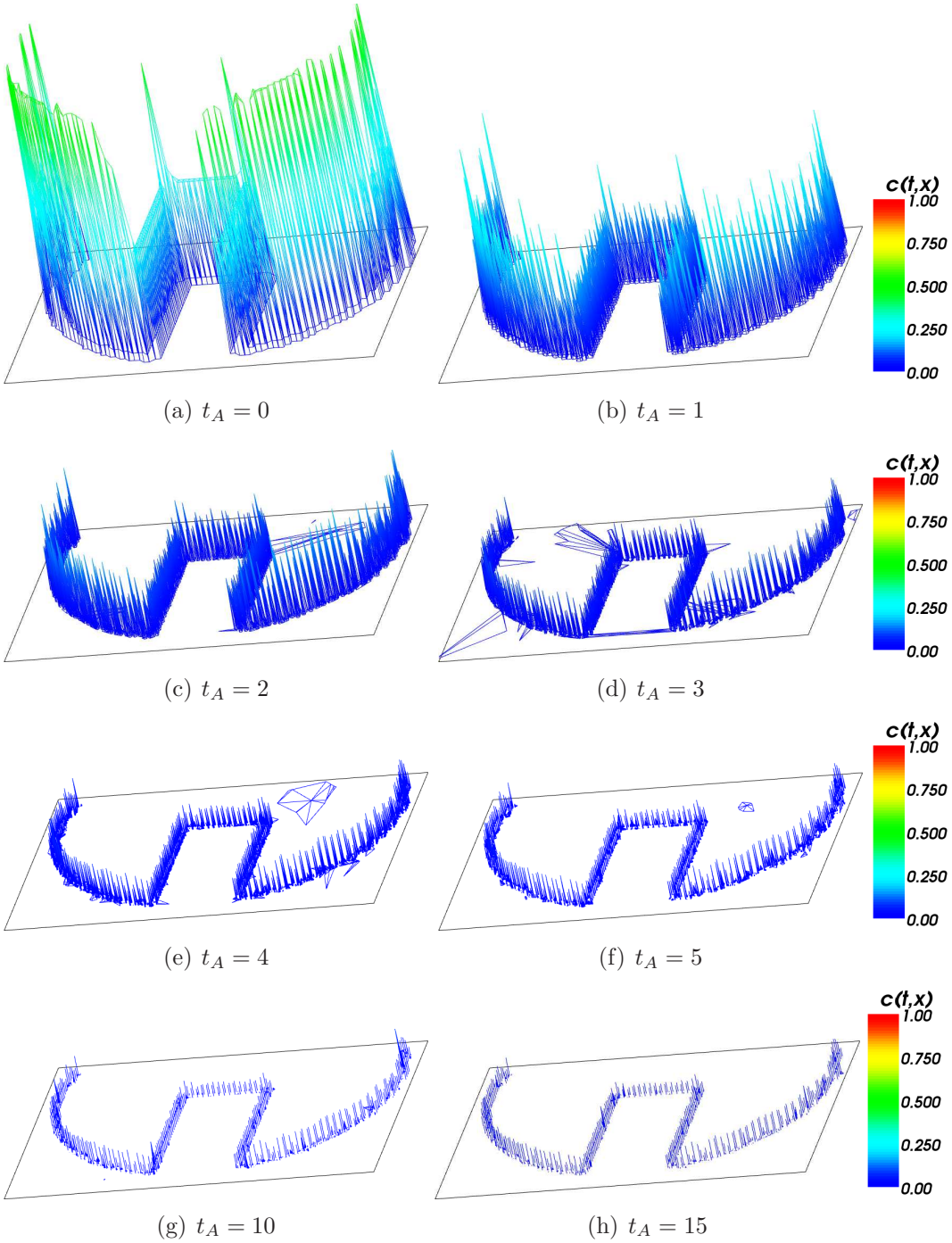
(d) $t_A = 3$ (e) $t_A = 4$ (f) $t_A = 5$

(g) $t_A = 6$ (h) $t_A = 10$ (i) $t_A = 15$

Abb. 55: Adaption der Slotted Disc. $N = 40.000 grid$

Knotenzahlen auftreten. Der maximale Fehler fällt dabei ebenfalls gleichmäßig bis auf ein Niveau von 0,25. Bei gleichmäßig verteilten Knoten wird der maximale Fehler mit $< 0{,}1$ wesentlich geringer. Hier ist jedoch das Verhalten der Adaption weniger stabil, es treten stärkere Schwankungen sowohl in der Knotenzahl als auch korrelierend im maximalen Fehler auf. Dabei verändert sich nach einigen Schritten jedoch lediglich die Anzahl der Knoten. Die Reproduktion der Disc bleibt im Wesentlichen unverändert.
Abbildung 55 zeigt dies für $\sigma_{ref} = 0{,}001$ und die ersten 15 Adaptionsschritte. Die zugehörigen Interpolationsfehler (Dargestellt werden nur die Werte mit $\eta > 0{,}01$) zeigt Abbildung 56 für einen Ausschnitt der Disc.

`Parameter Abb. 55 und 56:`	
`Adaption:`	`RBF-Interpolation`
`RBF:`	`Thin Plate Spline`
`Schritte:`	$t_A = 15$
`Nachbarschaften:`	$n = 20$
`Ausdünnen:`	$\sigma_{crs} = 0{,}001$
`Verfeinern:`	$\sigma_{ref} = 0{,}100$
`Adaptionsschritte:`$t_A = 15$	
`Konzentration:`	`Slotted Disc`
`Initiale Knotenzahl:`	$N = 40.000 grid$



(a) $t_A = 0$ (b) $t_A = 1$

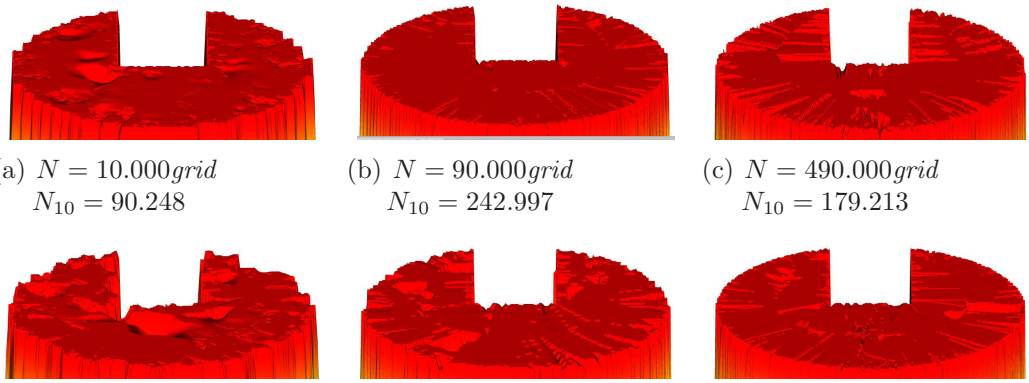
(c) $t_A = 2$ (d) $t_A = 3$

(e) $t_A = 4$ (f) $t_A = 5$

(g) $t_A = 10$ (h) $t_A = 15$

Abb. 56: Fehler bei der Adaption der Slotted Disc. $N = 40.000 grid$

Die RBF-Interpolation mit dem Thin Plate Spline ist bei der Adaption relativ robust bezüglich der Anzahl von Knoten in der Ausgangsverteilung. Abbildung 57 zeigt die Adaption nach zehn Schritten für $N \in \{10.000; 90.000; 490.000\}$. Lediglich bei 10.000 Punkten sind deutliche Einbußen in der Qualität der Reproduktion festzustellen. Ein direkter Zusammenhang zwischen der Anzahl der Knoten zu Beginn und während der Adaption lässt sich nicht mit Bestimmtheit ermitteln. Wir werden im Folgenden für die Slotted Disc mit einer Anfangsknotenzahl von 40.000 arbeiten.



(a) $N = 10.000 grid$ $N_{10} = 90.248$ (b) $N = 90.000 grid$ $N_{10} = 242.997$ (c) $N = 490.000 grid$ $N_{10} = 179.213$

(d) $N = 10.000 rnd$ $N_{10} = 28.800$ (e) $N = 90.000 rnd$ $N_{10} = 22.455$ (f) $N = 490.000 rnd$ $N_{10} = 67.816$

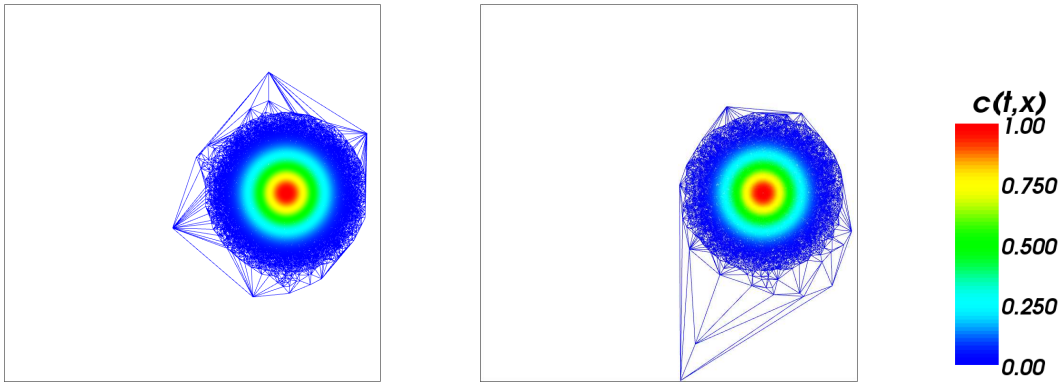
Abb. 57: Adaption der Slotted Disc mit unterschiedlichen Knotenzahlen.

Parameter Abb. 57:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionssschritte:	$t_A = 10$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 10.000 grid$ (57(a))
	$N = 10.000 rnd$ (57(b))
	$N = 90.000 grid$ (57(c))
	$N = 90.000 rnd$ (57(d))
	$N = 490.000 grid$ (57(e))
	$N = 490.000 rnd$ (57(f))

Wir werden nun die ermittelten Parameter bei der Adaption der Wendland-Funktion als Anfangskonzentration betrachten.
Zunächst betrachten wir den Einfluß von N auf die Adaption. Dabei setzen wir $N = 10.000$, $N = 40.000$ und $N = 250.000$ sowohl für eine gleichmäßige als auch für eine Zufallsverteilung. Das Ergebnis der Adaption nach 30 Schritten ist dabei für alle Tests optisch nicht von der Ausgangskonzentration unterscheidbar. Lediglich in der Knotenanzahl und -verteilung lassen sich Unterschiede feststellen. Abbildung 58 zeigt die Triangulierungen der entsprechenden Knotenmengen.
Wir betrachten nun die Entwicklung der Knotenzahlen sowie den maximalen Fehler. Wie Abbildung 59 zeigt, ist im Gegensatz zur Adaption der Slotted Disc bei der Wendland-Funktion eine Konvergenz der Knotenmenge nicht offensichtlich.

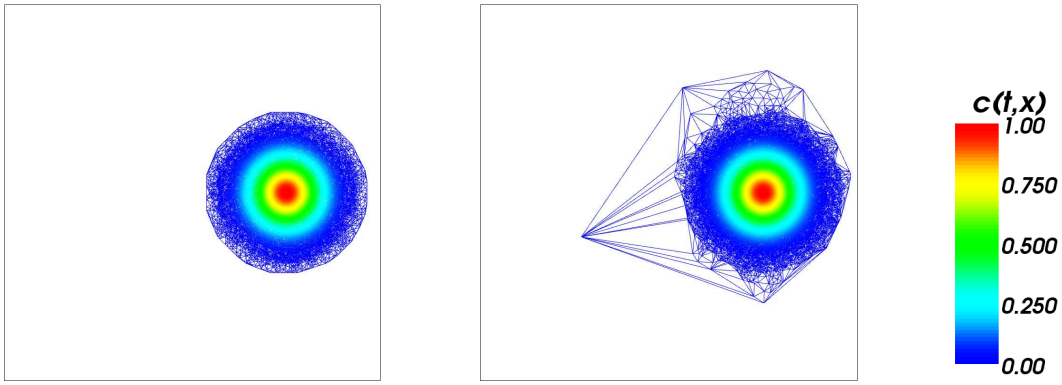
Parameter Abb. 58 und 59:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}010$
Adaptionsschritte:	$t_A = 30$
Konzentration:	Wendland-Funktion
Initiale Knotenzahl:	$N = 10.000$ *grid* (58(a), 59(a), 59(b))
	$N = 40.000$ *grid* (58(b), 59(a), 59(b))
	$N = 250.000$ *grid* (58(c), 59(a), 59(b))
	$N = 10.000$ *rnd* (58(d), 59(c), 59(d))
	$N = 40.000$ *rnd* (58(e), 59(c), 59(d))
	$N = 250.000$ *rnd* (58(f), 59(c), 59(d))

Dies resultiert in erster Linie aus einem extrem niedrigen maximalen Fehler, für den $\eta < 0{,}003$ bei gleichmäßiger Verteilung und immerhin noch $\eta < 0{,}005$ bei zufällig verteilter Knotenmenge erreicht wird. Dadurch sind die relativen Fehler in den einzelnen Knoten recht hoch, was zu einer Verfeinerung zahlreicher Knoten führt. Die Folge ist wiederum ein Anwachsen des maximalen Fehlers, gefolgt von einer Verringerung der relativen Fehler und einer starken Ausdünnung der Knotenmenge.
Wir betrachten die Reproduktion der Wendland-Funktion für eine zufällige Ausgangsverteilung mit 250.000 Knoten etwas genauer. Dafür wählen wir die Adaptionsschritte 17 - 20 aus, da wir bei diesen besonders hohe Unter-



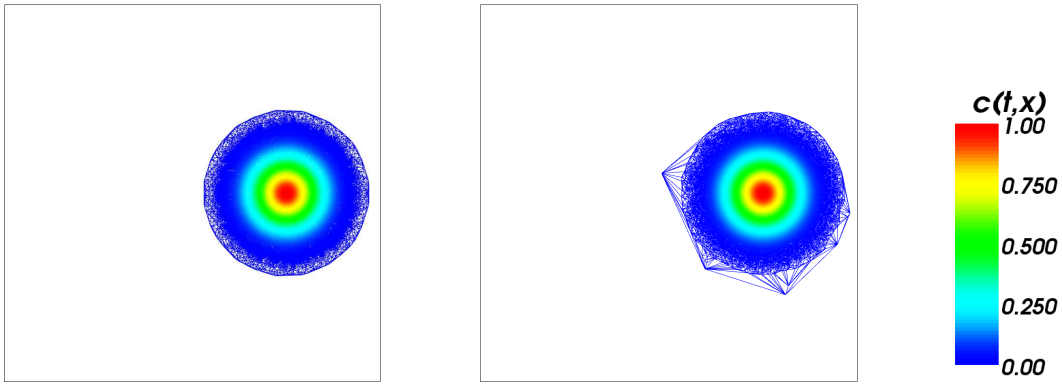
(a) $N = 10.000 grid$
$N_{30} = 363.884$

(b) $N = 10.000 rnd$
$N_{30} = 134.740$



(c) $N = 40.000 grid$
$N_{30} = 162.956$

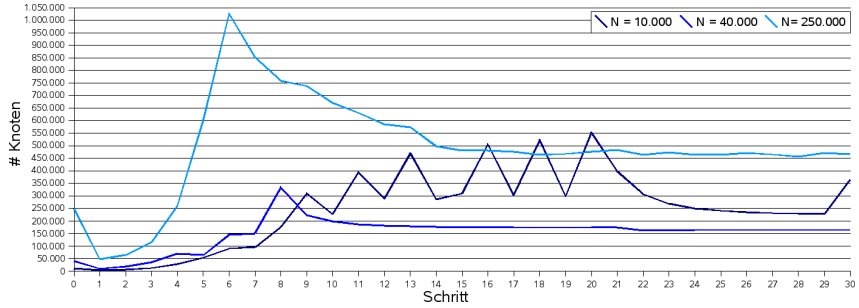
(d) $N = 40.000 rnd$
$N_{30} = 172.410$



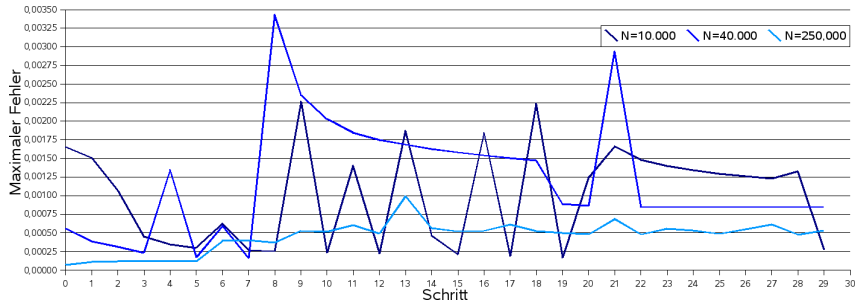
(e) $N = 250.000 grid$
$N_{30} = 465.255$

(f) $N = 250.000 rnd$
$N_{30} = 378.192$

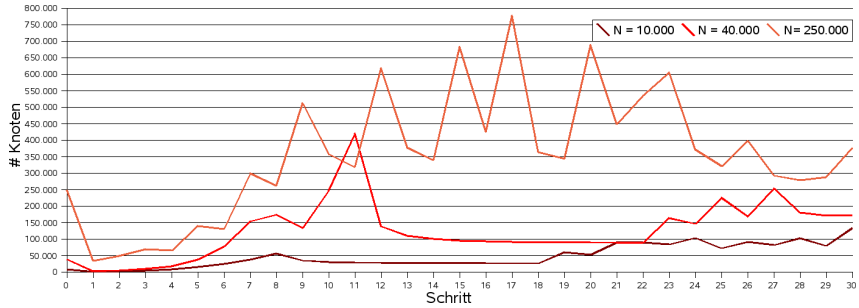
Abb. 58: Triangulierung bei unterschiedlichen Knotenzahlen. $t_A = 30$



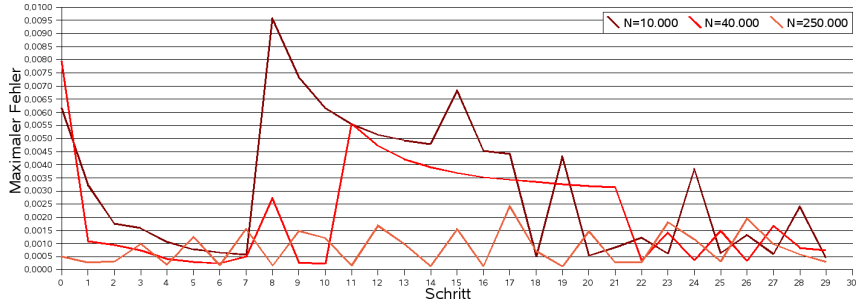
(a) Knotenzahl bei gleichmäßiger Ausgangsverteilung



(b) Maximaler Fehler bei gleichmäßiger Ausgangsverteilung



(c) Knotenzahl bei zufälliger Ausgangsverteilung



(d) Maximaler Fehler bei zufälliger Ausgangsverteilung

Abb. 59: Knotenzahlen und maximaler Fehler. $\sigma_{crs} = 0{,}001$; $\sigma_{ref} = 0{,}01$

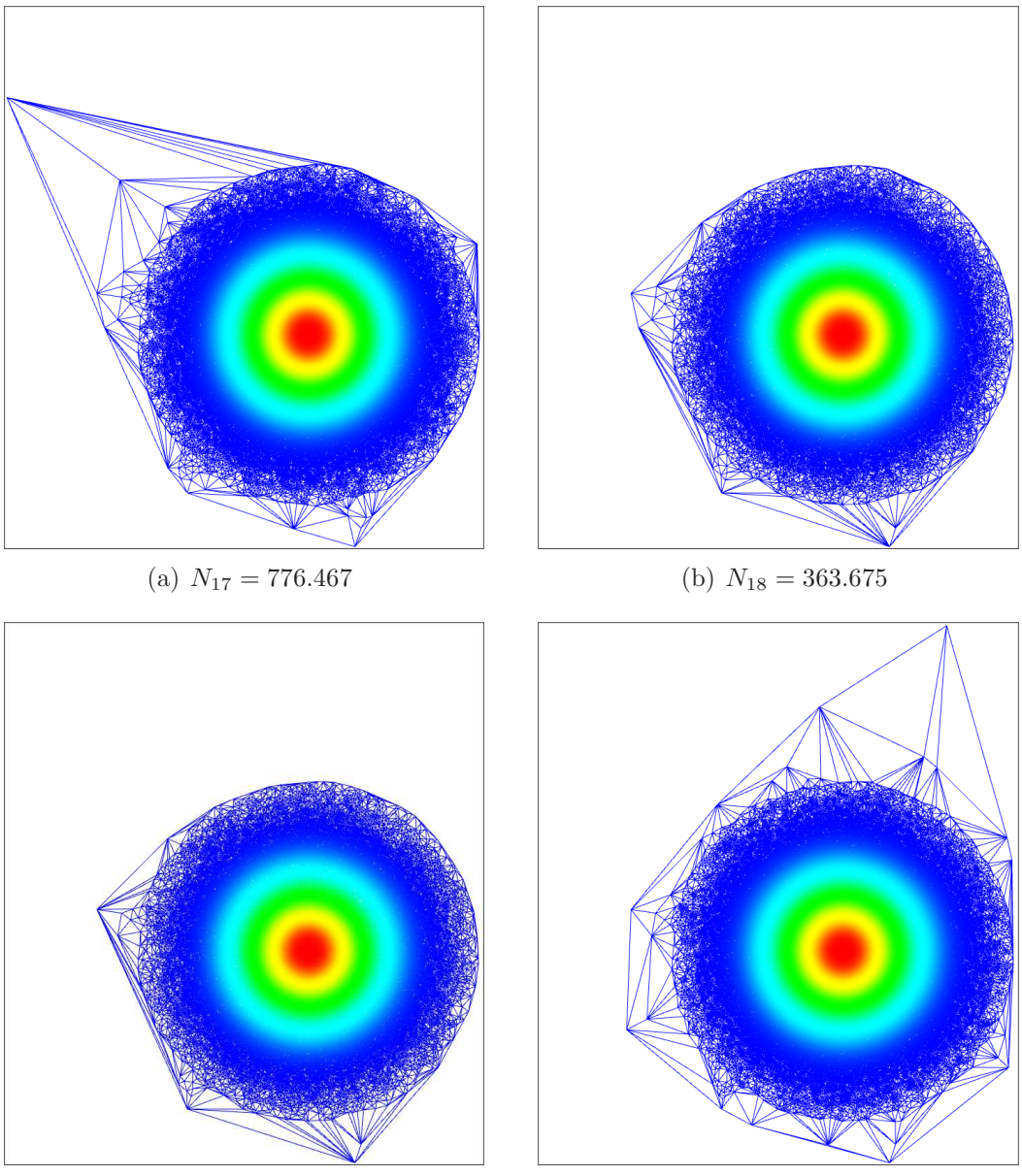
schiede in den Knotenzahlen vorfinden. Abbildung 60 zeigt die Delaunay-Triangulierungen der entsprechenden Knotenmengen. Die Reproduktion des Funktionskegels ist nach allen Schritten optisch nicht unterscheidbar von der Ausgangskonzentration, wir verzichten daher auf eine Darstellung.

`Parameter Abb. 60:`	
`Adaption:`	`RBF-Interpolation`
`RBF:`	`Thin Plate Spline`
`Nachbarschaft:`	$n = 20$
`Ausdünnen:`	$\sigma_{crs} = 0{,}001$
`Verfeinern:`	$\sigma_{ref} = 0{,}100$
`Adaptionsschritte:`	$t_A = 17$ `(60(a))`
	$t_A = 18$ `(60(b))`
	$t_A = 19$ `(60(c))`
	$t_A = 20$ `(60(d))`
`Konzentration:`	`Wendland-Funktion`
`Initiale Knotenzahl:`	$N = 250.000$***rnd***

3.6 Advektion & Adaption

Bei der Kombination von Adaption und Advektion erfolgt die Ausführung eines Zeitschrittes stets nach der Adaption der Knotenmenge.
Wir werden im Folgenden von Adaptionsschritten auch als *Iterationen* sprechen.
Einen durchgeführten Zeit- oder Advektionsschritt werden wir als *Schritt* bezeichnen.
Um die Adaption und die Advektion sinnvoll zu einem Verfahren zusammenzuführen, müssen wir sicherstellen, dass die Adaption vor der Durchführung eines Advektionsschrittes stationär wird. Da wir besonders bei glatten Ausgangskonzentrationen starke Schwankungen in der Knotenzahl beobachten konnten, werden wir einige Kriterien festlegen, ab wann wir die Adaption als stationär betrachten.
Die Adaption gelte als stationär, falls mindestens IT_{min} Iterationen ausgeführt worden sind und mindestens eines der folgenden Kriterien erfüllt ist:

- der maximale Fehler unterschreitet den Wert σ_{stat}.
- Die maximalen Fehler zweier aufeinander folgender Iterationen unterscheiden sich um weniger als σ_{diff}.
- die Adaption ist nach IT_{max} Iterationen noch nicht stationär.



(a) $N_{17} = 776.467$ (b) $N_{18} = 363.675$

(c) $N_{19} = 343.610$ (d) $N_{20} = 687.997$

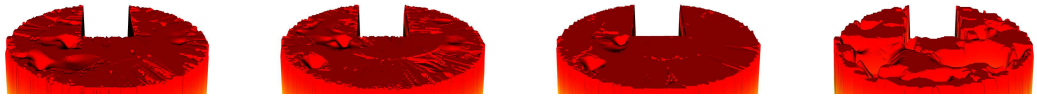
Abb. 60: Adaption der Wendland-Funktion. $N = 250.000 rnd$

Das letzte Kriterium garantiert uns die Terminierung des Verfahrens, falls keine der anderen Bedingungen erfüllt wird. Die Parameter für die stationäre Adaption werden wir später genauer betrachten.
Bei den bisher durchgeführten Testläufen für die Adaption fällt auf, dass das Verfahren sowohl für die MLS-Approximation als auch für die RBF-Interpolation sehr hohe Knotenzahlen liefert. Da wir eine möglichst hohe Performance der Semi-Lagrange-Advektion erreichen wollen, sollten wir versuchen, die Anzahl der Knoten niedrig zu halten. Betrachten wir die Anordnung der Knoten nach der Adaption etwas genauer, so stellen wir fest, dass sich Häufungspunkte bilden, in deren Umgebung die Knotendichte extrem hoch ist. Abbildung 38(d) (Seite 71) zeigt diesen Effekt für die MLS-Approximation. Um ein Entarten der Knotenmenge zu verhindern, überprüfen wir nach jedem Adaptionsschritt die Knotenmenge auf Duplikate. Wir stellen sicher, dass zwischen zwei Knoten mindestens der Abstand q_{min} eingehalten wird, indem wir alle Knoten entfernen, die diesen Mindestabstand unterschreiten. Dabei entfernen wir stets den später hinzugefügten Knoten. Durch die Wahl von q_{min} gilt für den Separationsabstand der gesamten Knotenmenge stets $q_X \geq \frac{q_{min}}{2}$. In allen bisherigen Testreihen haben wir mit $q_{min} = 10^{-5}$ gearbeitet. Wir untersuchen nun, inwieweit sich eine Erhöhung von q_{min} auf die Adaption auswirkt. Die folgende Tabelle stellt die gerundeten Knotenzahlen und den maximalen Fehler nach zwanzig Adaptionsschritten in Bezug zu q_{min}.

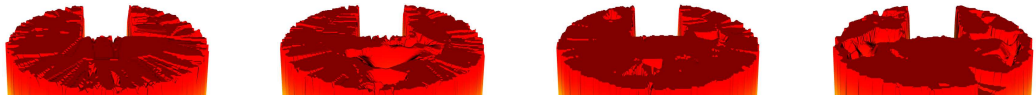
$N = 40.000 grid$	$q_{min} = 10^{-5}$	$q_{min} = 10^{-4}$	$q_{min} = 10^{-3}$	$q_{min} = 10^{-2}$
# Knoten	290.000	185.000	20.000	1.200
max. Fehler	0,065	0,060	0,08	0,300
$N = 40.000 rnd$	$q_{min} = 10^{-5}$	$q_{min} = 10^{-4}$	$q_{min} = 10^{-3}$	$q_{min} = 10^{-2}$
# Knoten	65.000	14.000	5.000	900
max. Fehler	0,25	0,30	0,44	0,45

Wir sehen, dass wir die Knotenzahl durch höhere Werte von q_{min} deutlich verringern können. Dabei wächst der maximale Fehler an, jedoch relativ gering im Vergleich zur Verringerung der Knotenmenge. Abbildung 61 zeigt uns die Qualität der Reproduktion für die in der Tabelle aufgeführten Werte von q_{min}. Für die Adaption ohne durchgeführte Advektion liefern uns Werte bis maximal $q_{min} = 10^{-3}$ akzeptable Ergebnisse. Ob bei der Durchführung der Advektion die Qualität der Reproduktion erhalten bleibt, werden wir später sehen.

Parameter Abb. 61:	
Adaption:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}10$
Adaptionsschritte:	$t_A = 20$
Separationsabstand:	$q_{min} = 10^{-5}$ (61(a), 61(e))
	$q_{min} = 10^{-4}$ (61(b), 61(f))
	$q_{min} = 10^{-3}$ (61(c), 61(g))
	$q_{min} = 10^{-2}$ (61(d), 61(h))
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (61(a) - 61(d))
	$N = 40.000$ *rnd* (61(e) - 61(h))



(a) $q_{min} = 10^{-5}$ (b) $q_{min} = 10^{-4}$ (c) $q_{min} = 10^{-3}$ (d) $q_{min} = 10^{-2}$



(e) $q_{min} = 10^{-5}$ (f) $q_{min} = 10^{-4}$ (g) $q_{min} = 10^{-3}$ (h) $q_{min} = 10^{-2}$

Abb. 61: Adaption mit unterschiedlichen Mindest-Separationsabständen.

Um ein optimales Ergebnis bei der Kombination von Adaption und Advektion zu erzielen, werden wir versuchen, die bestmöglichen Werte für q_{min} sowie für die Kriterien der stationären Adaption zu ermitteln.

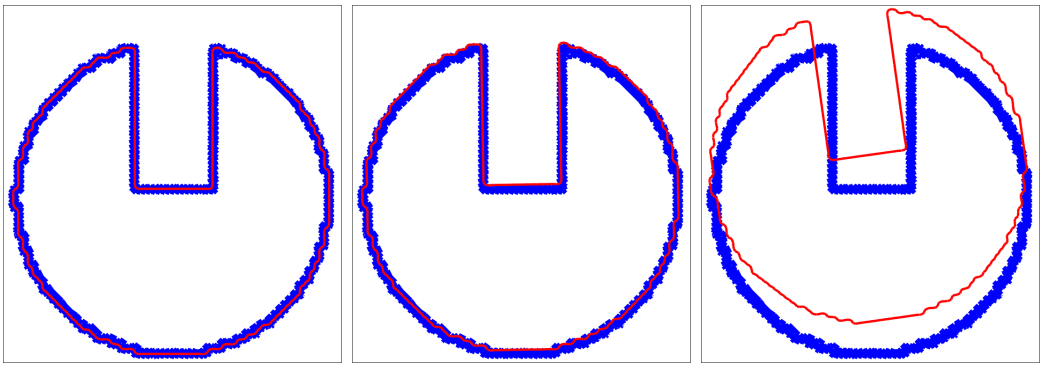
3.6.1 Advektion & Adaption mit MLS

Die Approximation mit Moving Least Squares lieferte uns sowohl bei der reinen Advektion als auch bei der Adaption schlechtere Resultate als die Interpolation mit radialen Basisfunktionen. Bei der Durchführung der Advektion wurde das abzubildende Objekt jeweils stark abgeflacht. Zudem erwies sich die Methode als relativ empfindlich gegenüber Ungleichverteilungen in der Knotenmenge. Bei der hier zu betrachtenden Kombination aus Adaption und Advektion treten beide Probleme verstärkt auf. Dadurch ist es uns nicht möglich, akzeptable Ergebnisse mit der Kombination aus beiden Verfahren zu erzielen.

Wir verzichten daher an dieser Stelle auf die Darstellung von Testläufen mit der MLS-Approximation und konzentrieren unsere Betrachtungen auf die Interpolation mit radialen Basisfunktionen.

3.6.2 Advektion & Adaption mit RBF

Ein entscheidender Parameter für die Advektion ist die Zeitschrittweite τ. Bei der reinen Advektion ohne Anpassung der Knotenmenge haben wir zu jedem Zeitpunkt t eine relativ gleichmäßige Knotenverteilung über Ω. Dadurch ist sichergestellt, dass wir die Konzentration zur Zeit t stets durch die vorhandenen Knoten gut darstellen können. Durch die Adaption enstehen nun knotenfreie Bereiche in Ω. Da wir mit der *rückwärts*-SLM arbeiten, wandern die Knoten nicht mit dem Geschwindigkeitsfeld mit. Wir müssen daher sicherstellen, dass der Bereich, in den das betrachtete Objekt vom Geschwindigkeitsfeld transportiert wird, genügend Knoten für die Reproduktion enthält. Im Fall der Slotted Disc befindet sich nach der Adaption vor dem ersten Zeitschritt der Großteil der Knoten in einem schmalen Bereich um den Rand der Disc. Um die Disc nach dem Zeitschritt noch gut reproduzieren zu können, müssen wir τ hinreichend klein wählen. Abb. 62 zeigt schematisch die Knotenverteilung in blau sowie den Rand der Disc in rot. In Abb. 62(b) ist τ klein genug, so dass der Rand der Disc auch nach dem dargestellten Zeitschritt im Bereich der Knoten liegt. Abb. 62(c) stellt ein zu groß gewähltes τ dar. Hier würde der Rand der Disc nicht reproduziert werden können, da an den entsprechenden Stellen keine Knoten vorhanden sind.



(a) Ausgangsverteilung (b) Kleines τ (c) Zu großes τ

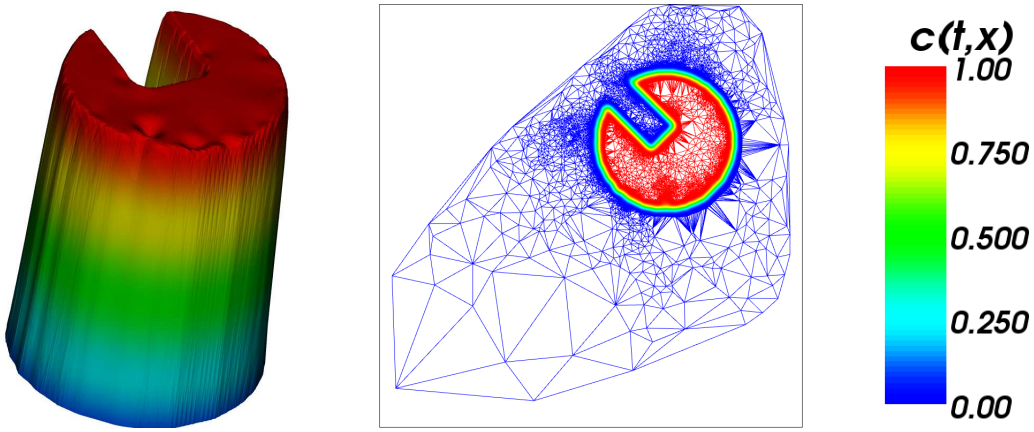
Abb. 62: Schematische Darstellung eines Zeitschrittes

Die Wahl von τ wird direkt vom verwendeten Geschwindigkeitsfeld beeinflusst.

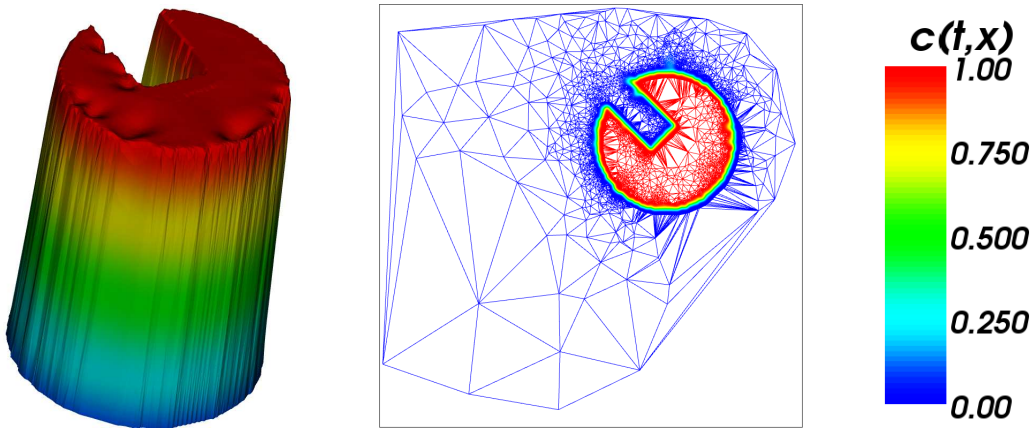
Die Abbildungen 63 und 64 zeigen die adaptive Advektion für das von uns verwendete tangentiale Geschwindigkeitsfeld. Dargestellt ist jeweils eine Rotation um $\frac{\pi}{2}$ für unterschiedlche Werte von τ. Eine gute Reproduktion erhalten wir für $\tau \leq 80$, höhere Werte bedingen die in Abbildung 62(c) dargestellte Problematik. Wir werden bei unseren weiteren Betrachtungen mit $\tau = 80$ arbeiten, da wir bei geringeren Werten mehr Zeitschritte berechnen müssen, um dieselbe Bewegung darzustellen. Dies bedeutet bei der Verwendung des tangentialen Geschwindigkeitsfeldes a mit $\omega_1 = 0{,}3636 \times 10^{-4}$, dass wir in jedem Zeitschritt eine Rotation um $\frac{\pi}{1.080}$ erreichen.

Parameter Abb. 63 und 64:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$IT_{min} = IT_{max} = 1$
Separationsabstand:	$q_{min} = 0{,}001$
Zeitschritte:	$t = 270$ (63(a), 64(a))
	$t = 90$ (63(b), 64(b))
	$t = 30$ (63(c), 64(c))
Zeitschrittweite:	$\tau = 80$ (63(a), 64(a))
	$\tau = 240$ (63(b), 64(b))
	$\tau = 720$ (63(c), 64(c))
Zeitintervall:	$\mathcal{I} = [0, t \cdot \tau]$
Rotation:	$\frac{\pi}{4}$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (63)
	$N = 40.000$ *rnd* (64)

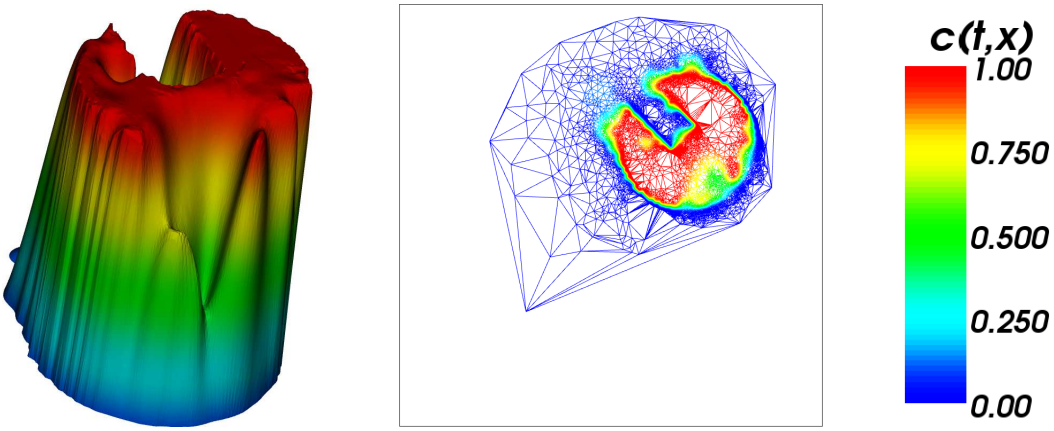
Niedrigere Werte für τ liefern mehr Knoten bei der Adaption. Wir haben daher bei der Durchführung der gerade betrachteten Testreihen den Mindestseparationsabstand $q_{min} = 0{,}001$ gesetzt. Für $\tau = 80$ erreichen wir dadurch nach $t = 270$ Schritten eine Knotenzahl von rund 130.000. Durch ein Herabsetzen von q_{min} erwarten wir höhere Knotenzahlen. Abbildung 65 zeigt die Entwicklung der Knotenzahl für die ersten 30 Schritte bei einer zufälligen Ausgangsverteilung. Dabei wurde vor jedem Zeitschritt genau ein Adaptionsschritt ausgeführt.



(a) $\tau = 80$; $t = 270$; $N_t = 129.487$

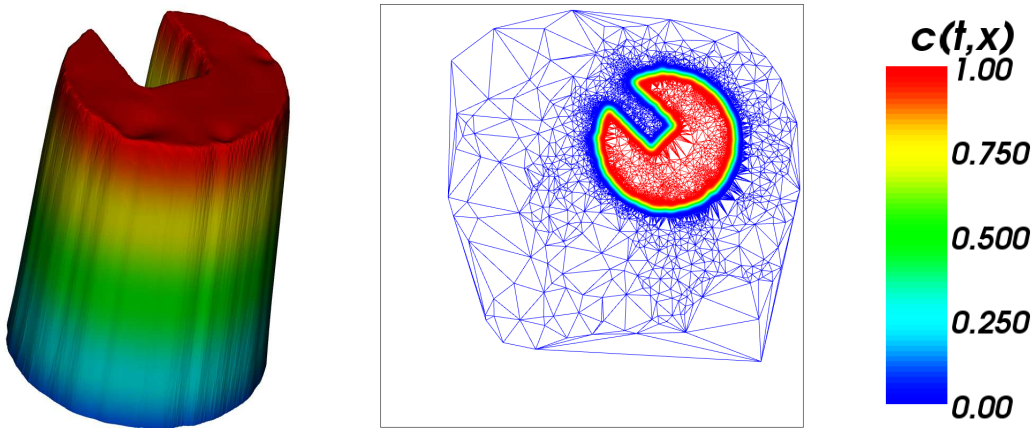


(b) $\tau = 240$; $t = 90$; $N_t = 77.075$

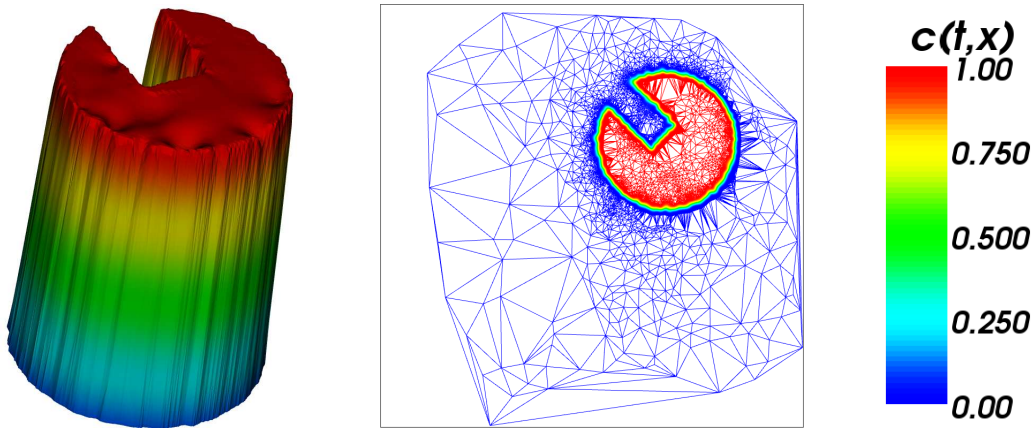


(c) $\tau = 720$; $t = 30$; $N_t = 20.318$

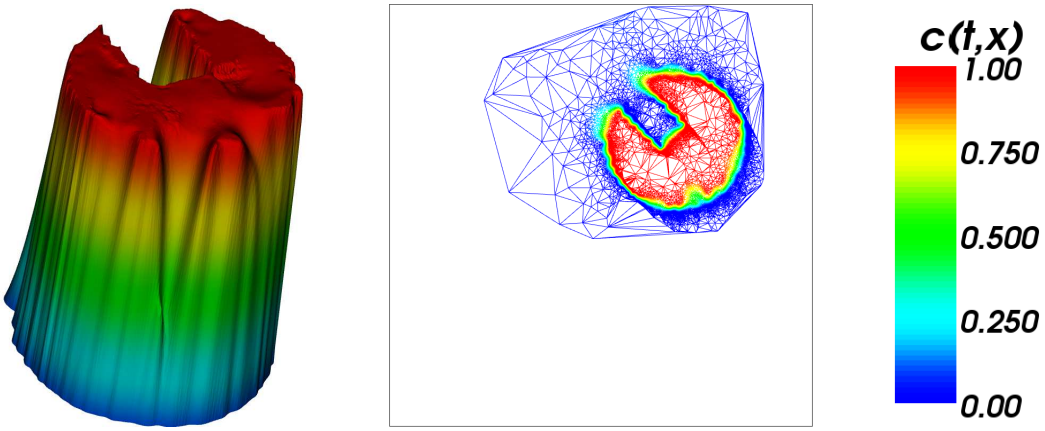
Abb. 63: Rotation mit unterschiedlichen Zeitschrittweiten. $N_0 = 40.000 grid$



(a) $\tau = 80$; $t = 270$; $N_t = 131.096$

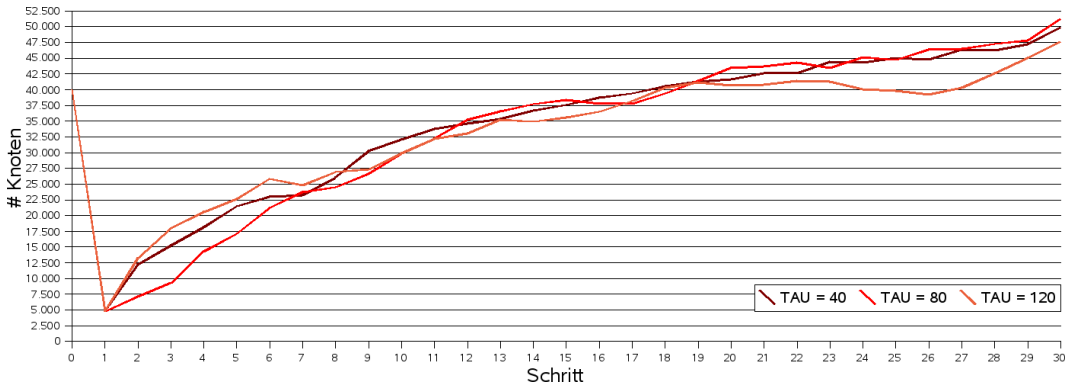


(b) $\tau = 240$; $t = 90$; $N_t = 81.174$

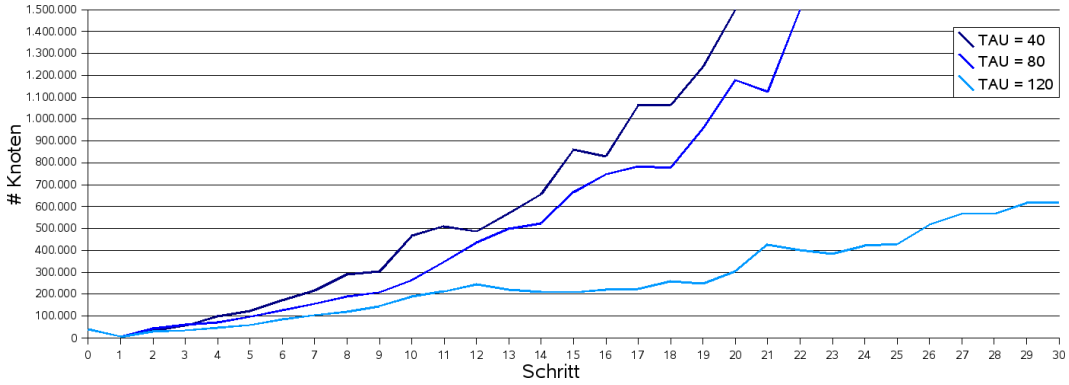


(c) $\tau = 720$; $t = 30$; $N_t = 21.245$

Abb. 64: Rotation mit unterschiedlichen Zeitschrittweiten. $N_0 = 40.000 rnd$



(a) $q_{min} = 0{,}001$



(b) $q_{min} = 0{,}0001$

Abb. 65: Knotenzahlen bei der adaptiven Advektion mit unterschiedlichen Mindest-Separationsabständen

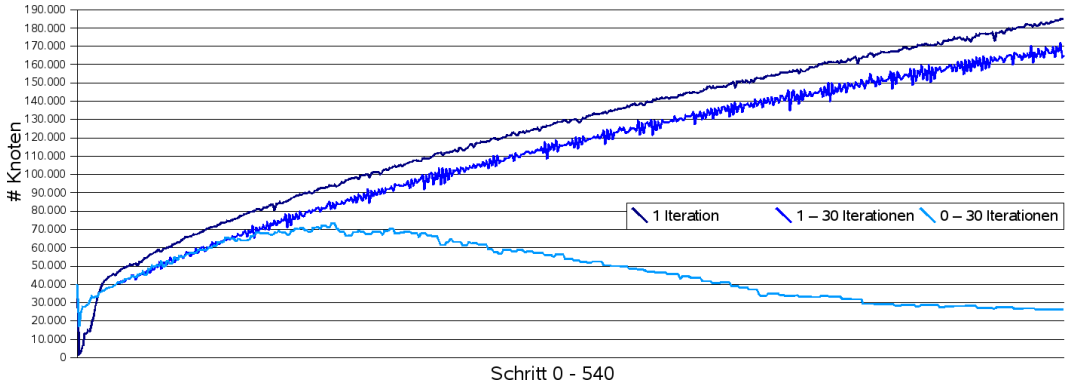
Parameter Abb. 65:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$IT_{min} = IT_{max} = 1$
Separationsabstand:	$q_{min} = 0{,}001$ 65(a)
	$q_{min} = 0{,}0001$ 65(b)
Zeitschritte:	$t = 30$
Zeitschrittweite:	$\tau = 80$
Zeitintervall:	$\mathcal{I} = [0, 30 \cdot \tau]$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$***rnd***

Bei einem Separationsabstand von $q_{min} = 0{,}0001$ zeigt sich, dass bei kleinen Werten für τ die Knotenzahl über 1,5 Millionen steigt. Dies ist im Hinblick auf die Rechenzeit inakzeptabel. Wir werden daher $q_{min} = 0{,}001$ festlegen.
Wir betrachten nun die Kriterien für die stationäre Adaption. Für eine bessere Performance werden wir versuchen, mit möglichst wenigen Iterationen auszukommen. Dabei müssen wir jedoch auch die Knotenzahl betrachten. Wir werden zunächst die Anzahl der durchgeführten Iterationen betrachten und dabei σ_{stat} und σ_{diff} festhalten. Wir setzen $\sigma_{stat} = 0{,}01$ und $\sigma_{diff} = 0{,}005$. Wir werden also solange eine Adaption durchführen, bis der maximale Fehler unter 1% fällt, bzw. der maximale Fehler in zwei nachfolgenden Iterationen sich um weniger als 0,5% unterscheidet.
Wir betrachten zunächst den Fall $IT_{min} = IT_{max} = 1$, so dass wir vor jedem Zeitschritt genau einen Adaptionsschritt ausführen. Danach werden wir IT_{max} auf 30 erhöhen und so ermitteln, wie viele Iterationen benötigt werden, um den maximalen Fehler unter 1% zu drücken.
Es zeigt sich, dass der maximale Fehler nach einigen Zeitschritten unter ein gewisses Niveau sinkt. Daher werden wir außerdem den Fall $IT_{min} = 0$ in unsere Betrachtungen einbeziehen. Dadurch vermeiden wir eine Adaption, falls der maximale Fehler nach einem Zeitschritt bereits unter σ_{stat} liegt.
Abbildung 66 zeigt die Entwicklung der Knotenzahl bei zufälliger und gleichmäßiger Ausgangsverteilung. Wenn wir $IT_{min} = 1$ betrachten, so steigt die Knotenzahl bei gleichmäßiger Ausgangsverteilung in etwa logarithmisch an. Dies geschieht in ähnlichem Maße für $IT_{max} = 1$ und $IT_{max} = 30$. Bei einer Zufallsverteilung geschieht der Anstieg ungleichmäßiger, jedoch mit vergleichbaren Knotenzahlen. Nach einer Viertelrotation der Slotted Disc hat das Ver-

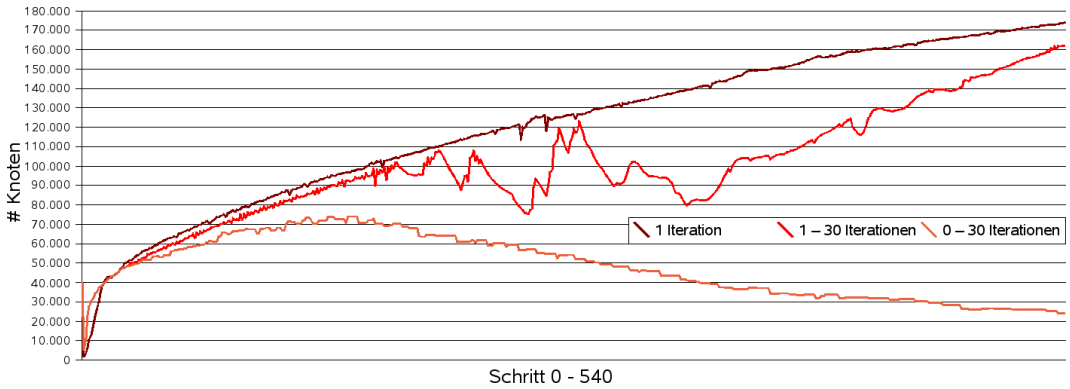
fahren in beiden Fällen bereits mehr als 160.000 Knoten erzeugt. Der Anstieg der Knotenzahl setzt sich weiter fort, so dass nach einer kompletten Rotation rund 300.000 Knoten enstanden sind.

Parameter Abb.66, 67, 68 und 69:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Maximaler Fehler:	$\sigma_{stat} = 0{,}01$
	$\sigma_{diff} = 0{,}005$
Adaptionsschritte:	$IT_{min} = 0$ (68(c), 69(c))
	$IT_{min} = 1$ (68(a), 68(b), 69(a), 69(b))
	$IT_{max} = 1$ (68(a), 69(a))
	$IT_{max} = 30$ (68(b), 68(c), 69(b), 69(c))
Separationsabstand:	$q_{min} = 0{,}001$
Zeitschrittweite:	$\tau = 80$
Zeitschritte:	$t = 540$
Zeitintervall:	$\mathcal{I} = [0, 540 \cdot \tau]$
Rotation:	$\frac{\pi}{2}$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$***grid*** (66(a), 67(a), 67(b), 68)
	$N = 40.000$***rnd*** (66(b), 67(c), 67(d), 69)

Wir betrachten daher den Fall $IT_{min} = 0$ und $IT_{max} = 30$. Auch hier steigt die Knotenzahl zunächst an bis rund 75.000, danach fällt sie jedoch wieder ab. Abbildung 67 zeigt das Verhalten des maximalen Fehlers sowie die Anzahl der durchgeführten Iterationen pro Zeitschritt. Der maximale Fehler ist bei genau einer Iteration pro Zeitschritt während der ersten Schritte schlechter als bei mehreren möglichen Iterationen. Nach einigen Schritten liegen die maximalen Fehler bei allen Varianten jedoch in derselben Größenordnung. Erlauben wir das Auslassen der Adaption durch $IT_{min} = 0$, so unterliegt der maximale Fehler stärkeren Schwankungen.
Beschränken wir die Anzahl von Iterationen auf $1 - 30$, so findet nach ca. 30 Zeitschritten stets nur noch ein einziger Adaptionsschritt statt, da der maximale Fehler sehr gering ist. Für den Fall $IT_{min} = 0$ wird bei höheren Zeitschritten immer häufiger die Adaption ausgelassen. Hier kommt es bei einigen Schritten zu mehreren Iterationen in einem Zeitschritt.
Das Ergebnis der Reproduktion ist bei einer Viertelrotation für alle drei Varian-

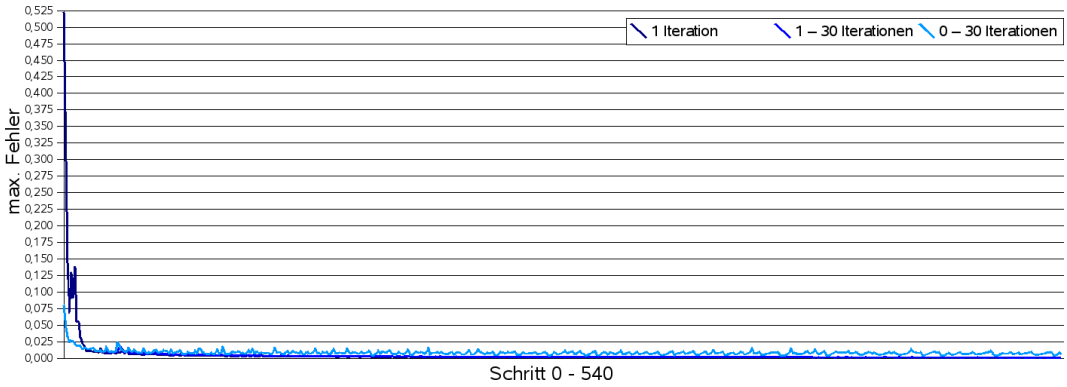


(a) $N_0 = 40.000 grid$

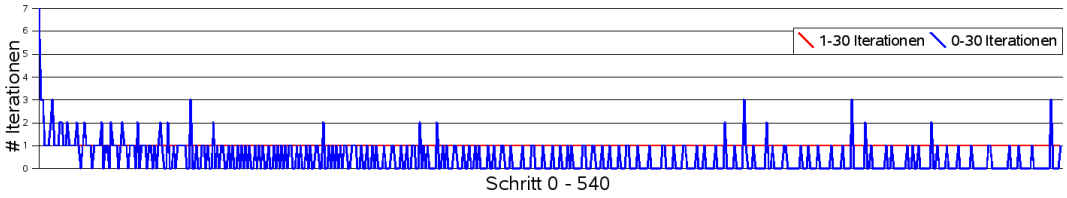


(b) $N_0 = 40.000 rnd$

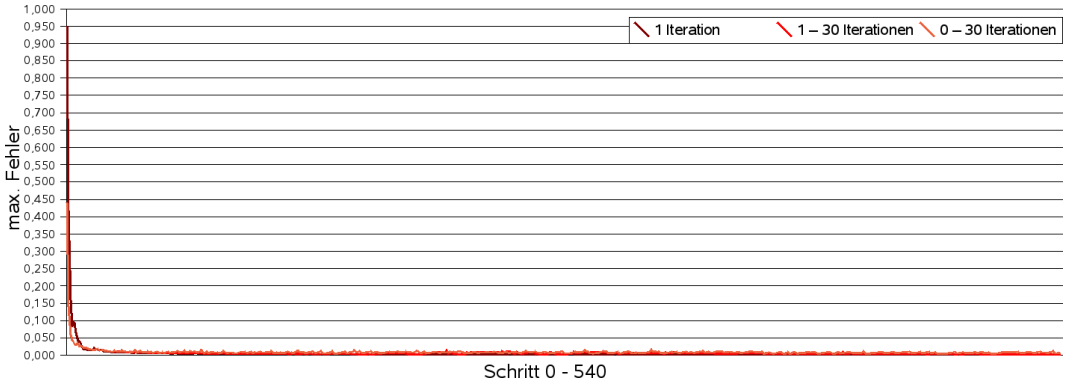
Abb. 66: Knotenzahlen bei unterschiedlicher Anzahl von Iterationen



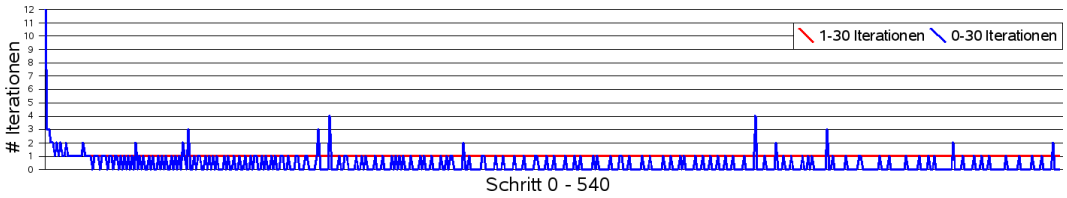
(a) Maximaler Fehler. $N_0 = 40.000 grid$



(b) Anzahl durchgeführter Iterationen. $N_0 = 40.000 grid$



(c) Maximaler Fehler. $N_0 = 40.000 rnd$

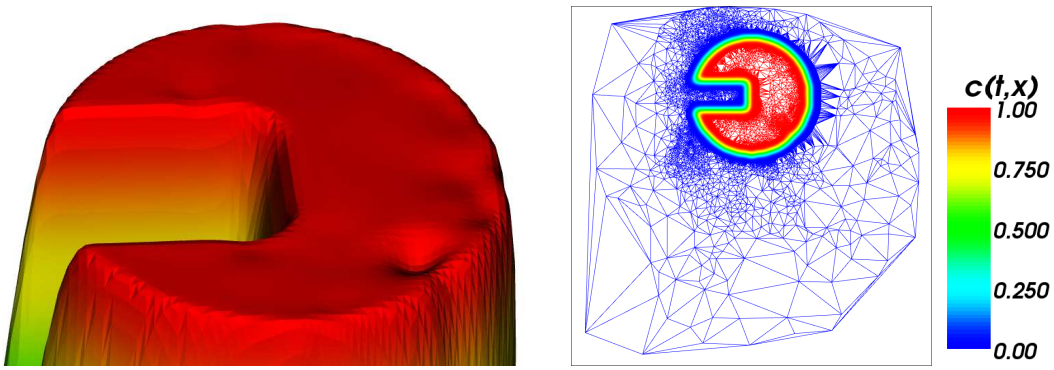


(d) Anzahl durchgeführter Iterationen. $N_0 = 40.000 rnd$

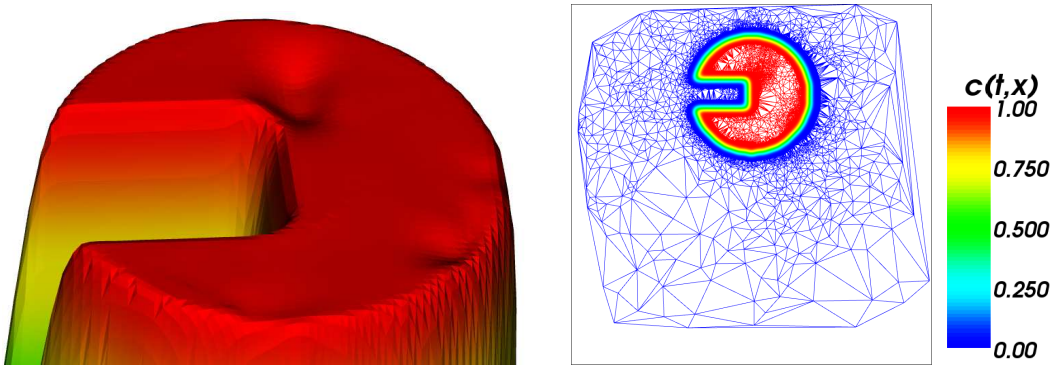
Abb. 67: Maximaler Fehler und Anzahl Iterationen

ten in etwa vergleichbar, wie die Abbildungen 68 und 69 zeigen. Die Verteilung und die Anzahl der Knoten in Ω sind jedoch bei $0 - 30$ Iteration am günstigsten. Hier sind die Knoten besser auf die Disc konzentriert. Bei $IT_{min} = 1$ enstehen auch außerhalb der Disc Häufungspunkte von Knoten.
Da wir aufgrund der niedrigen Wahl von τ sehr viele Schritte berechnen müssen, ist die Anzahl der Knoten ein sehr entscheidender Faktor. Diese wächst für $IT_{min} = 1$ jedoch sehr stark an. Daher werden wir $IT_{min} = 0$ setzen, um eine akzeptable Performance zu erreichen.
Wir müssen nun noch die optimalen Werte für σ_{stat} und σ_{diff} bestimmen. Dafür betrachten wir das Verhalten der adaptiven Advektion bei mehreren Umläufen der Slotted Disc. Brauchbare Ergebnisse erzielen wir für $\sigma_{diff} \leq \sigma_{stat} \leq 0{,}01$. Für höhere Werte wird zu selten eine Adaption ausgeführt, so dass die Reproduktion der Disc nach einigen Schritten zu schlecht wird. Wir werden σ_{diff} in der Größenordnung von $\frac{\sigma_{stat}}{10}$ wählen. Wir betrachten zunächst $\sigma_{stat} = 0{,}01$ und $\sigma_{diff} = 0{,}001$ für fünf volle Rotationen der Slotted Disc.
Die Abbildungen 70 und 71 zeigen die Disc in der Seitenansicht mit Blick auf den Slot. Dabei sind jeweils komplette Rotationen dargestellt. Bemerkenswert ist, dass es außerhalb der Disc nur zu sehr geringen Anhebungen der Konzentration kommt. Lediglich bei einer zufälligen Ausgangsverteilung entsteht eine leichte Erhebung direkt vor der Disc (Abb. 71(b)). Diese verstärkt sich jedoch im Testverlauf nicht weiter, sondern flacht im Gegenteil wieder ab.

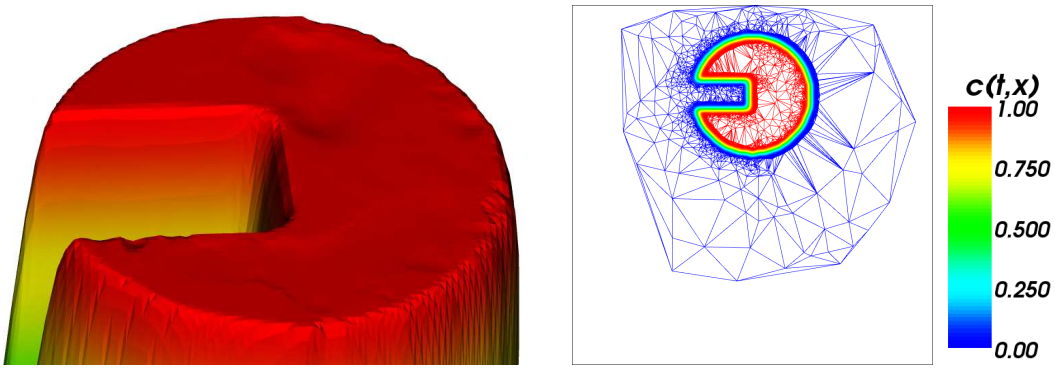
Parameter Abb.70, 71, 72, 73, 74, 75 und 76:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$IT_{min} = 0$
	$IT_{max} = 30$
Separationsabstand:	$q_{min} = 0{,}001$
Maximaler Fehler:	$\sigma_{stat} = 0{,}01$
	$\sigma_{diff} = 0{,}001$
Zeitschritte:	$t = 10.800$
Zeitschrittweite:	$\tau = 80$
Zeitintervall:	$\mathcal{I} = [0, 10.800 \cdot \tau]$
Rotation:	$10 \cdot \pi$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (70, 72, 74)
	$N = 40.000$ *rnd* (71, 73, 75)



(a) $IT_{min} = 1$; $IT_{max} = 1$; $N_{540} = 185.239$

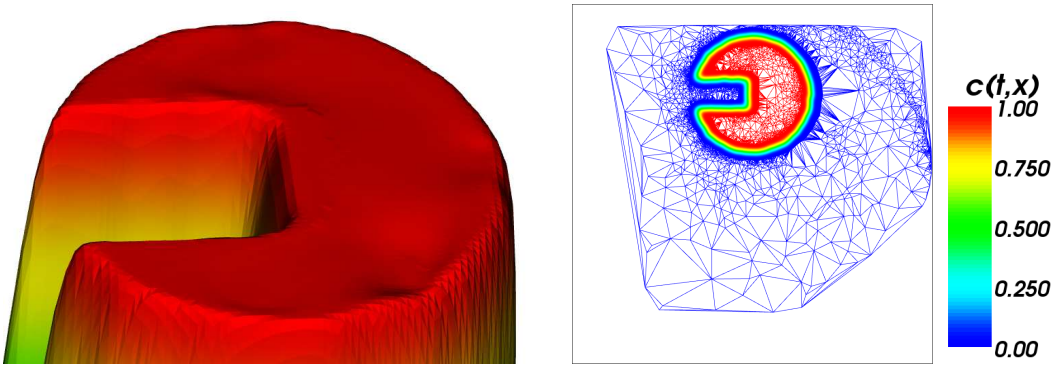


(b) $IT_{min} = 1$; $IT_{max} = 30$; $N_{540} = 172.762$

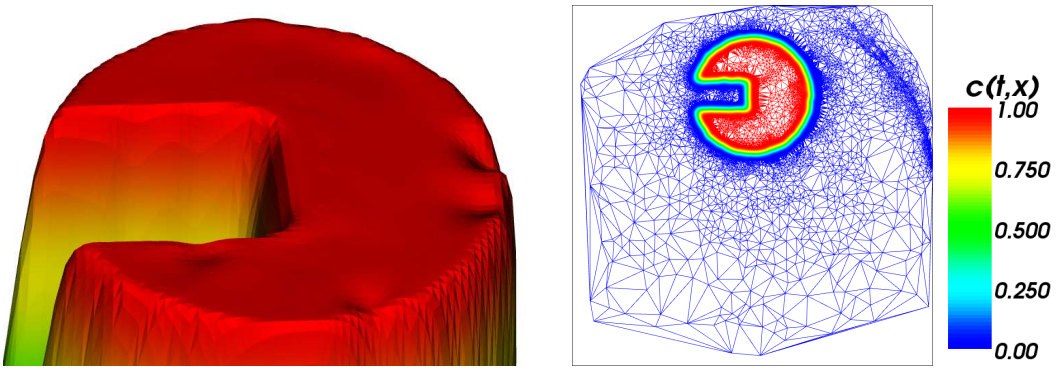


(c) $IT_{min} = 0$; $IT_{max} = 30$; $N_{540} = 25.837$

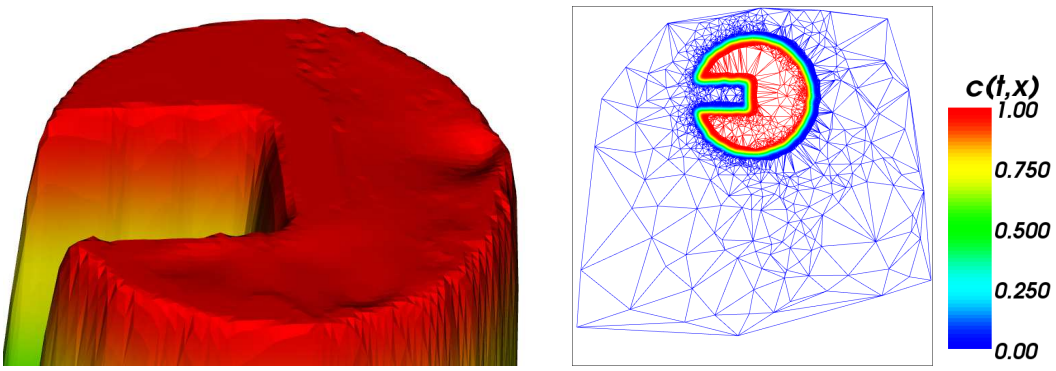
Abb. 68: Rotation mit unterschiedlicher Anzahl von Adaptionsschritten. $N = 40.000 grid$



(a) $IT_{min} = 1$; $IT_{max} = 1$; $N_{540} = 173.774$

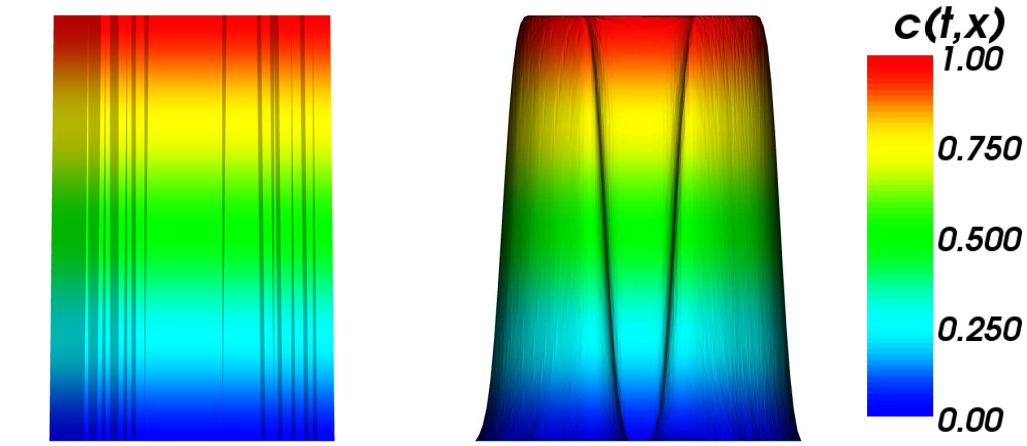


(b) $IT_{min} = 1$; $IT_{max} = 30$; $N_{540} = 162.840$

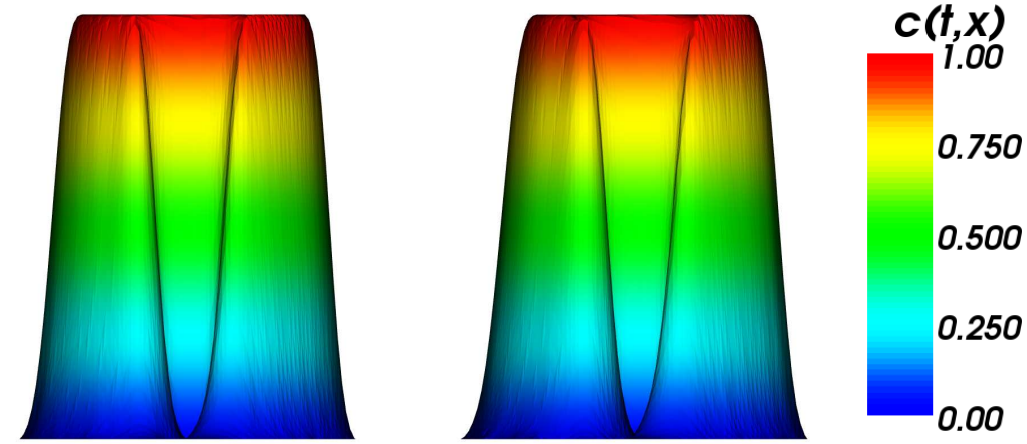


(c) $IT_{min} = 0$; $IT_{max} = 30$; $N_{540} = 23.883$

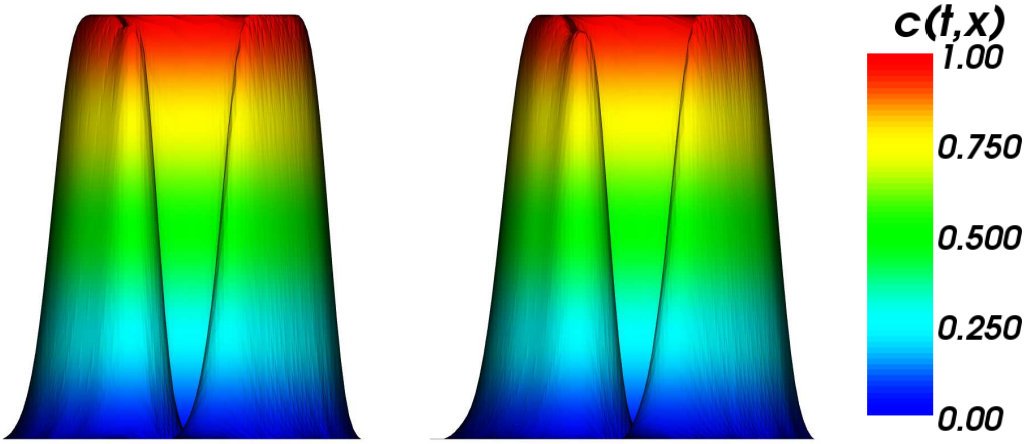
Abb. 69: Rotation mit unterschiedlicher Anzahl von Adaptionsschritten. $N = 40.000 rnd$



(a) $t = 0$ (b) $t = 2.160$



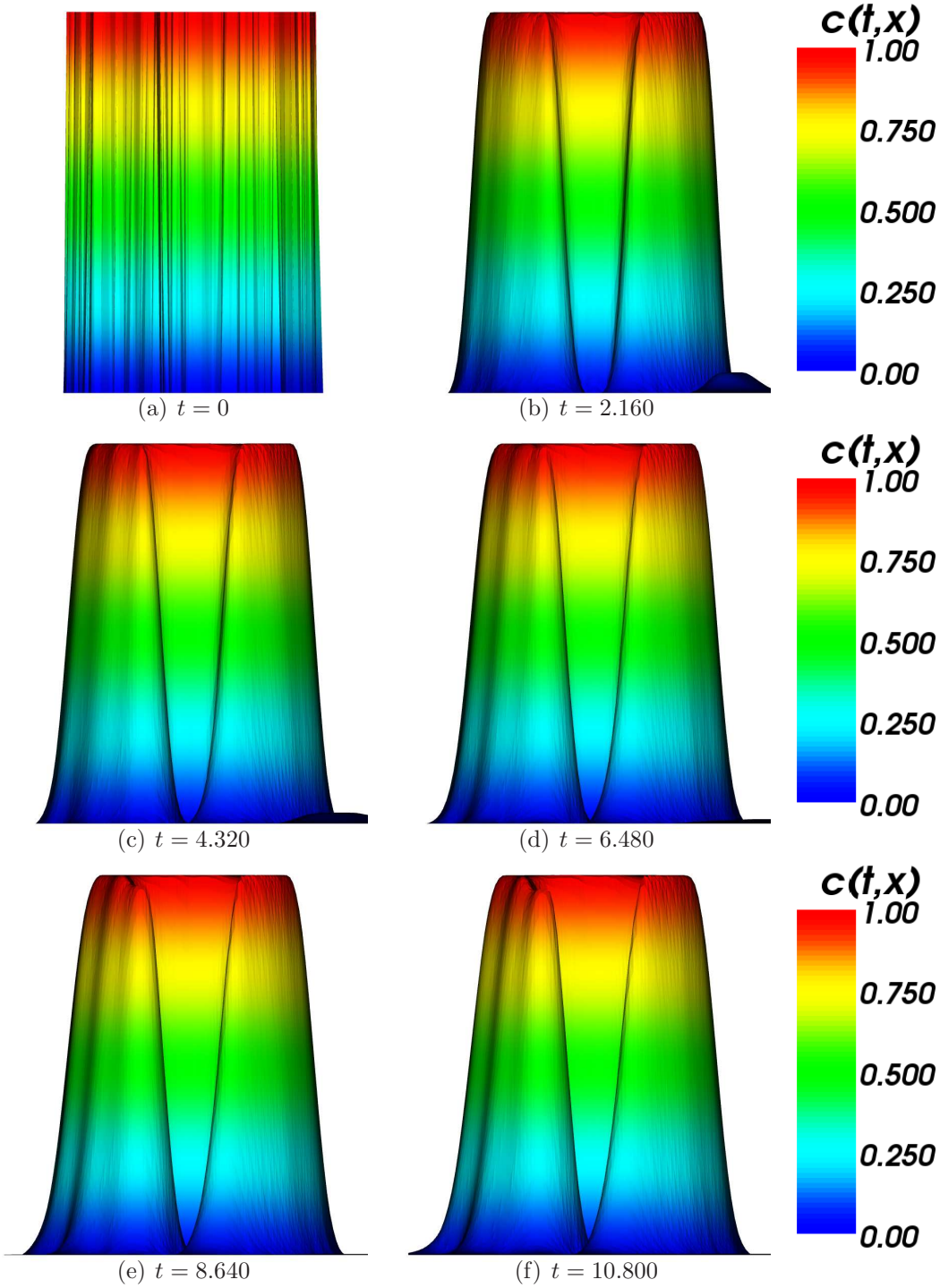
(c) $t = 4.320$ (d) $t = 6.480$



(e) $t = 8.640$ (f) $t = 10.800$

Abb. 70: Fünf vollständige Rotationen. $N = 40.000 grid$

Sowohl bei der gleichmäßigen Ausgangsverteilung als auch bei der Zufallsverteilung zeigt sich, dass die Kanten der Disc im Laufe der Zeit abgerundet werden.
Die Abbildungen 72 und 73 zeigen die Disc von oben in einer Detailansicht, die diesen Effekt verdeutlicht.
Die Knotenmenge ist auch nach 5 Rotationen noch sehr gut auf die Disc konzentriert. Die Abbildungen 74 und 75 zeigen, dass die Knotenmenge von einer Rotation zur nächsten im Wesentlichen unverändert wieder hergestellt wird.
Die Anzahl der Knoten bleibt bei dieser Testreihe ebenfalls sehr stabil. Nach ca. 150 Zeitschritten erreicht die Knotenzahl ihr Maximum von rund 75.000 Knoten. Danach fällt sie bis auf rund 10.000, um dann relativ konstant zu bleiben. Abbildung 76 zeigt die Knotenzahlen während der fünf Rotationen der Slotted Disc für beide Anfangsverteilungen.
Eine Erhöhung von σ_{diff} auf $\frac{\sigma_{stat}}{2} = 0{,}005$ verringert die Anzahl durchgeführter Iterationen um rund 5%. Das Ergebnis der Reproduktion ist hierbei nahezu identisch. Um das Kriterium des niedrigen maximalen Fehlers stärker zu gewichten, werden wir $\sigma_{diff} = \frac{\sigma_{stat}}{10}$ festlegen.
Wir werden nun den Wert für σ_{stat} auf 0,005 senken. Dies führt im Schnitt zu mehr Adaptionen pro Zeitschritt, wie die Abbildungen 77(a) und 78(a) zeigen. Dadurch werden mehr Knoten erzeugt. Die Abbildungen 77(b) und 77(b) zeigen die Entwicklung der Knotenzahlen im Vergleich für eine Rotation bei unterschiedlichen Ausgangsverteilungen.
Die Kanten der Slotted Disc sind bei der Verwendung von $\sigma_{stat} = 0{,}005$ bereits nach einer Rotation wesentlich stärker geglättet als bei $\sigma_{stat} = 0{,}01$. Die Abbildungen 77(c), 77(d), 78(c) und 78(d) zeigen die Disc nach jeweils einer Rotation im Detail.

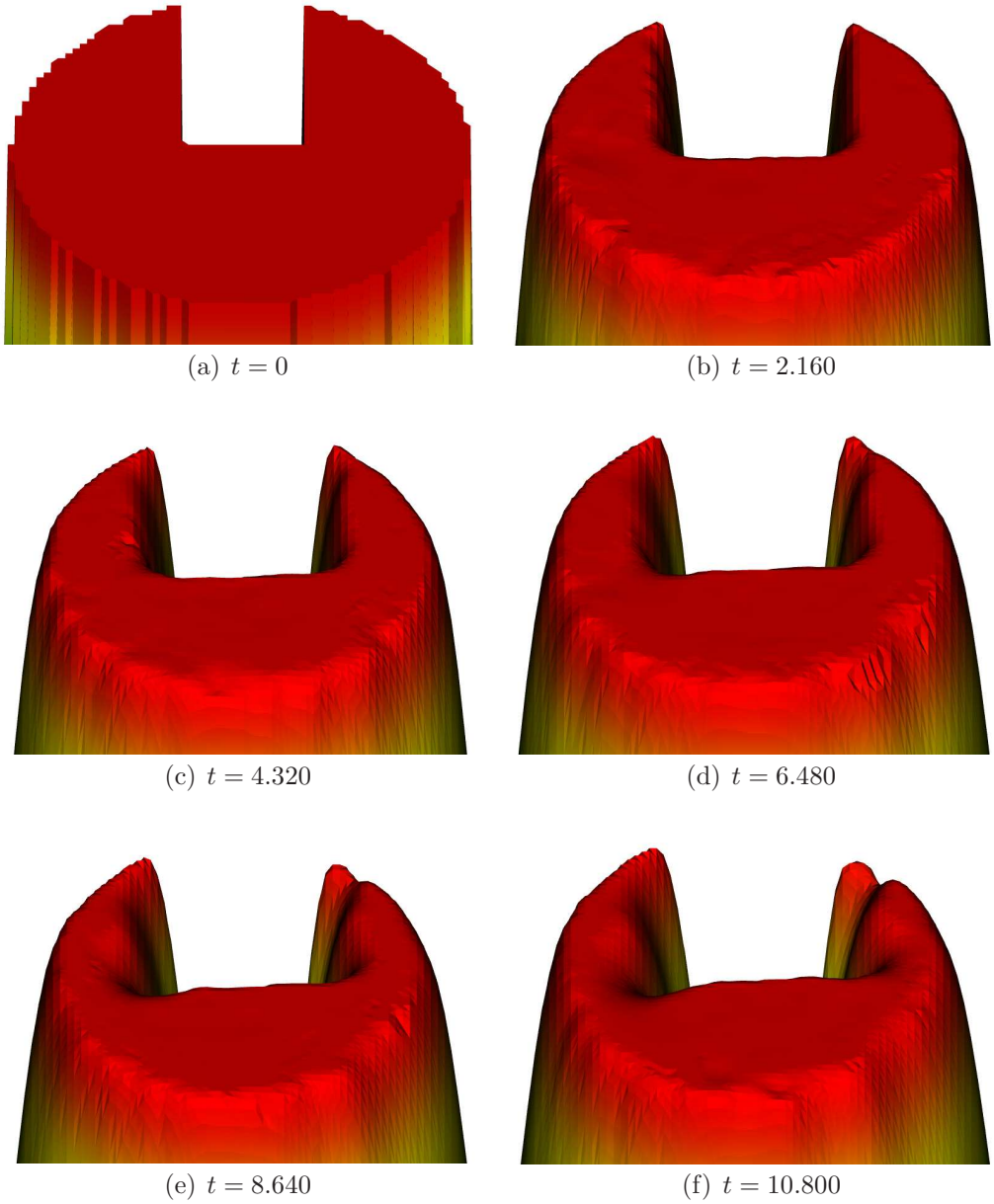


(a) $t = 0$ (b) $t = 2.160$

(c) $t = 4.320$ (d) $t = 6.480$

(e) $t = 8.640$ (f) $t = 10.800$

Abb. 71: Fünf vollständige Rotationen. $N = 40.000 rnd$

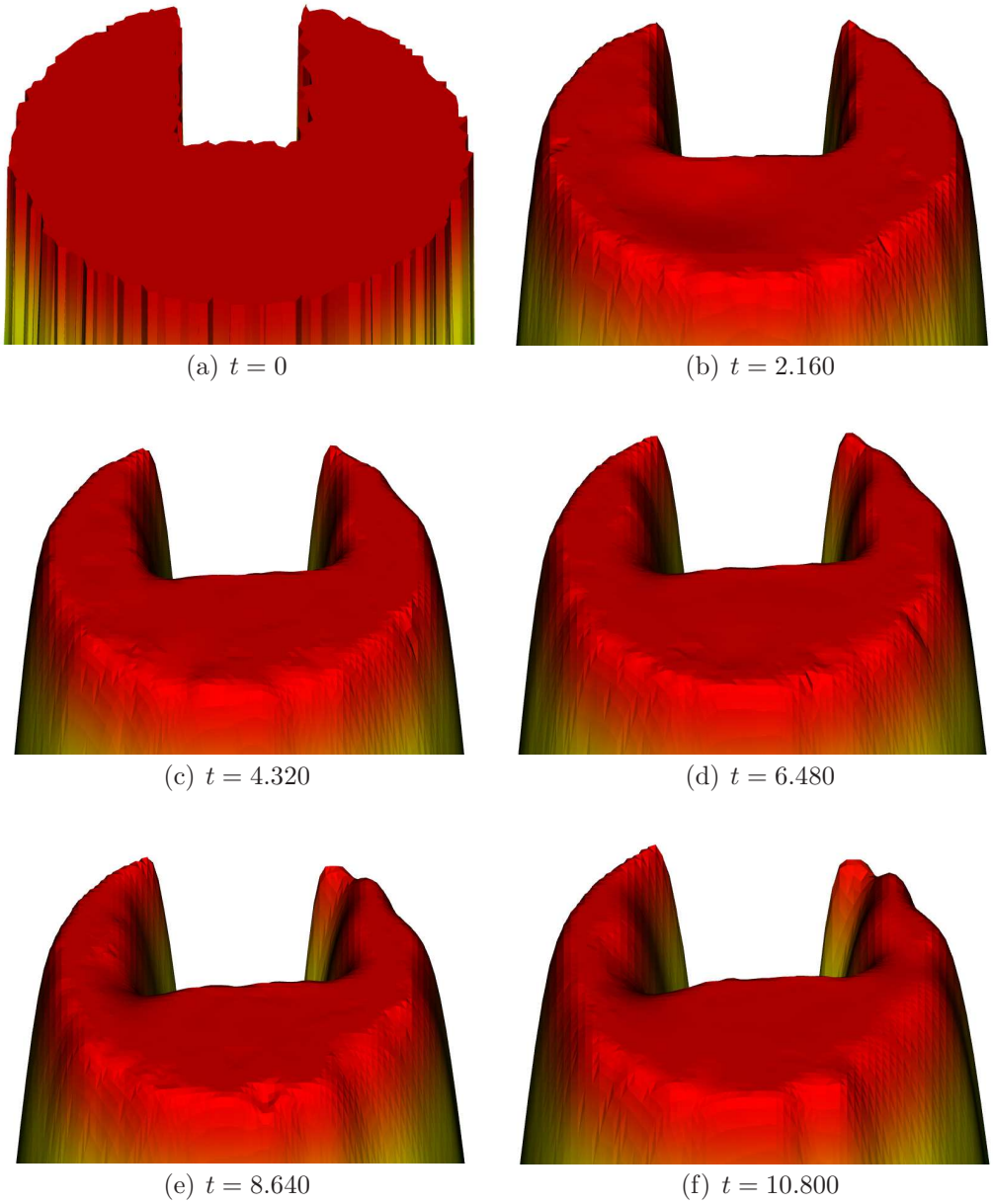


(a) $t = 0$ (b) $t = 2.160$

(c) $t = 4.320$ (d) $t = 6.480$

(e) $t = 8.640$ (f) $t = 10.800$

Abb. 72: Detailansicht nach fünf Rotationen. $N = 40.000 grid$

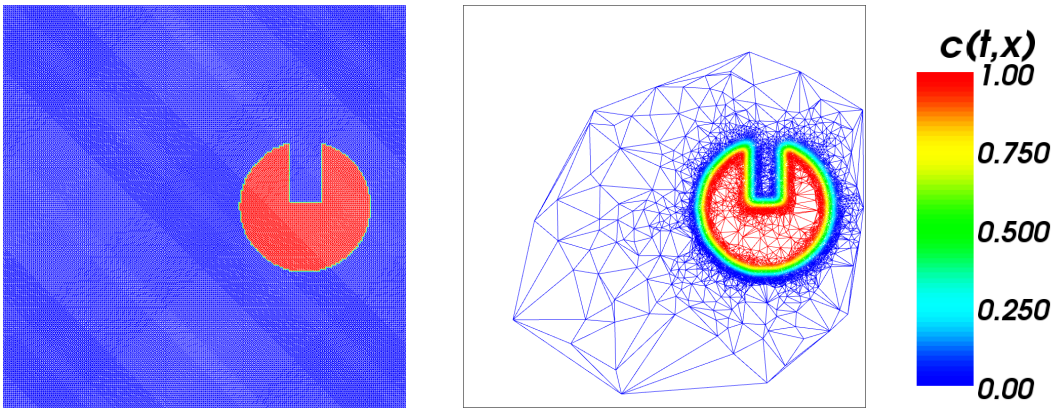


(a) $t = 0$ (b) $t = 2.160$

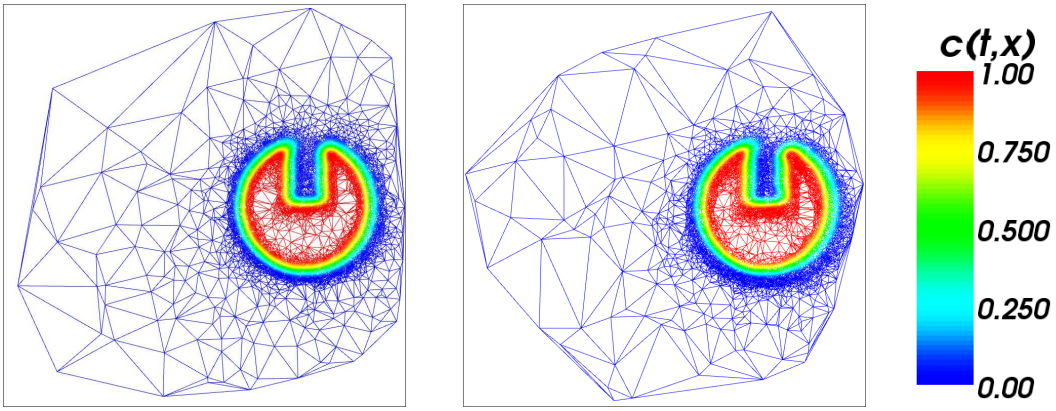
(c) $t = 4.320$ (d) $t = 6.480$

(e) $t = 8.640$ (f) $t = 10.800$

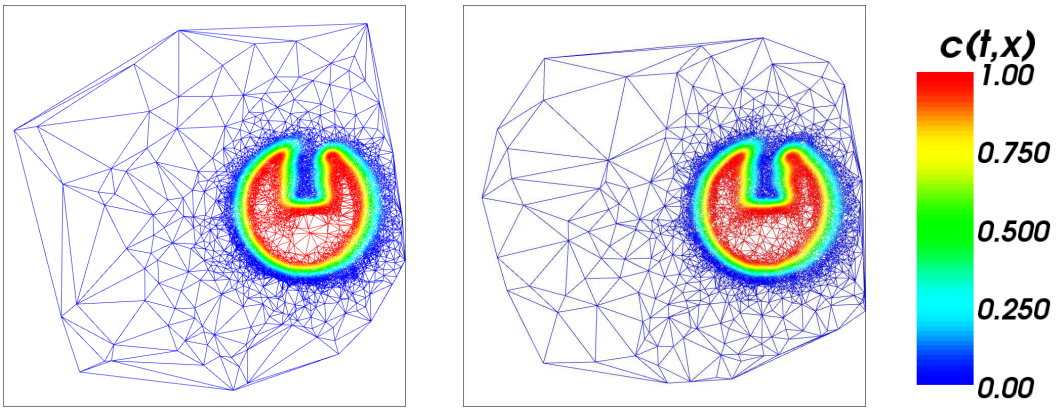
Abb. 73: Detailansicht nach fünf Rotationen. $N = 40.000 rnd$



(a) $t = 0$ (b) $t = 2.160$

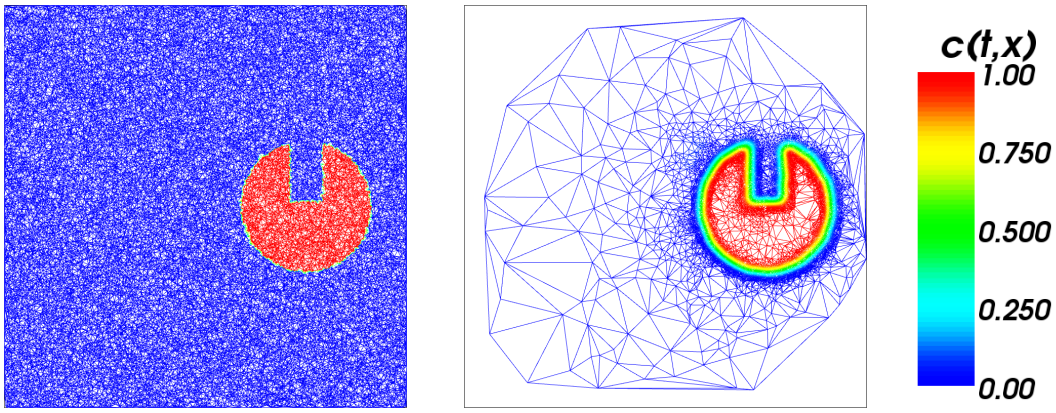


(c) $t = 4.320$ (d) $t = 6.480$

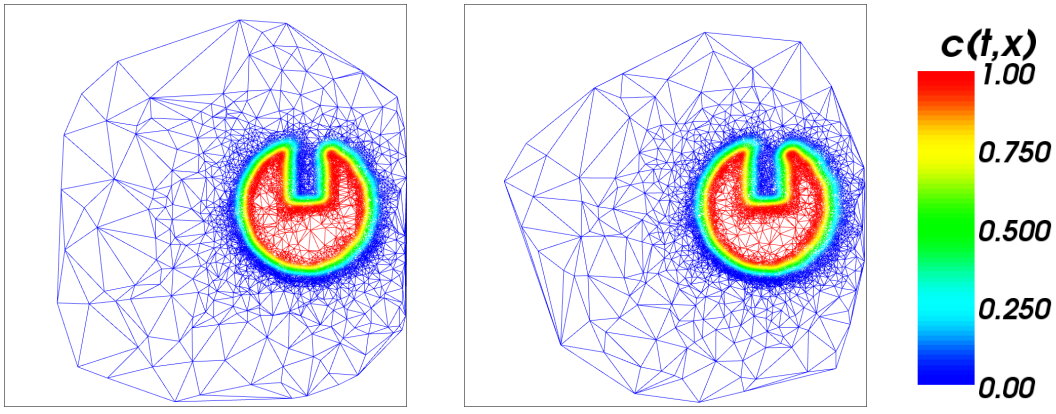


(e) $t = 8.640$ (f) $t = 10.800$

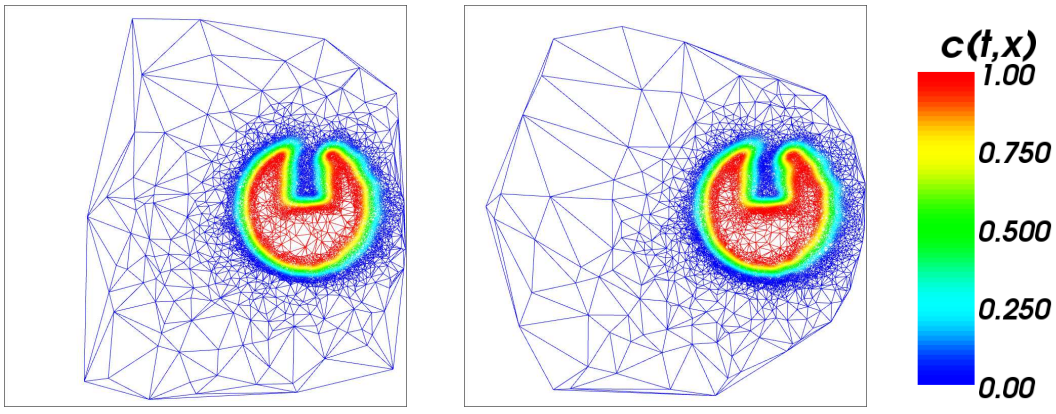
Abb. 74: Delaunay-Triangulierung nach fünf Rotationen. $N = 40.000 grid$



(a) $t = 0$ (b) $t = 2.160$



(c) $t = 4.320$ (d) $t = 6.480$



(e) $t = 8.640$ (f) $t = 10.800$

Abb. 75: Delaunay-Triangulierung nach fünf Rotationen. $N = 40.000 rnd$

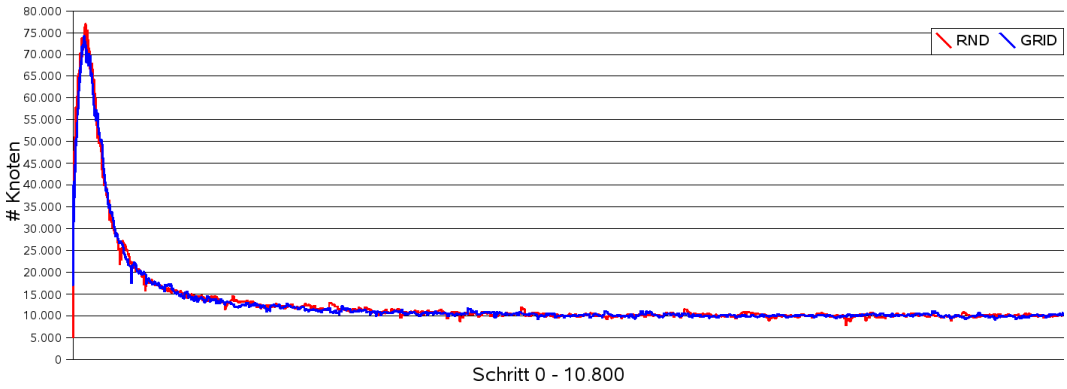
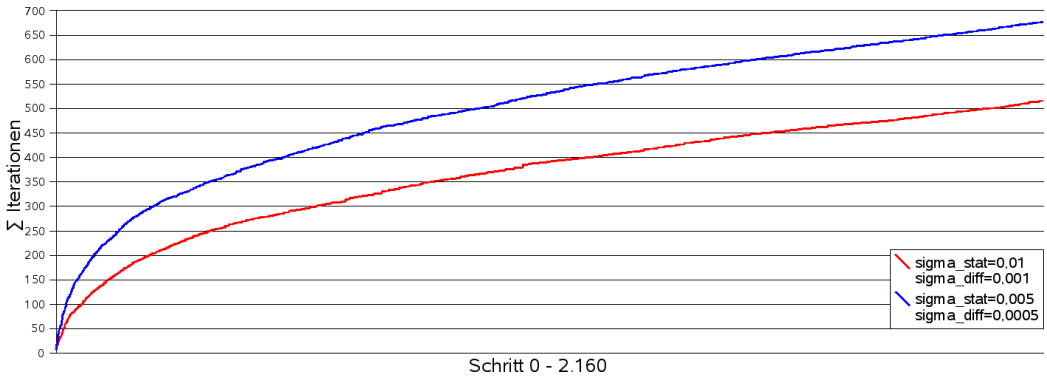


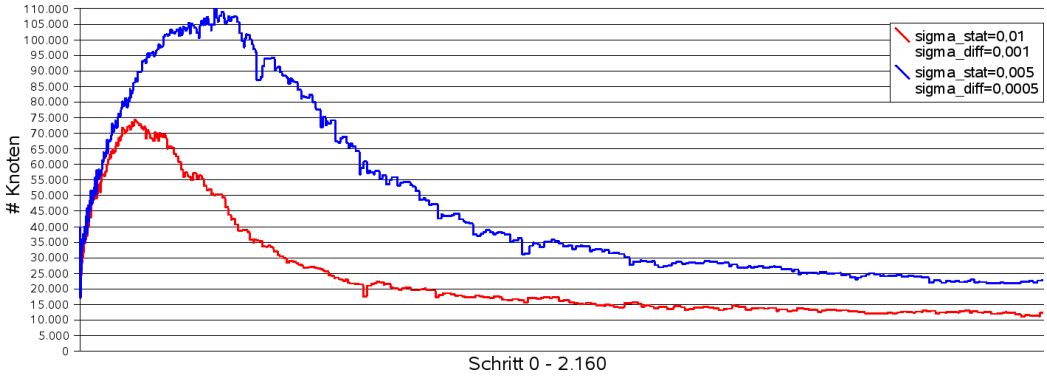
Abb. 76: Entwicklung der Knotenzahl während fünf Rotationen.
$\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$

Parameter Abb. 77 und 78:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$IT_{min} = 0$
	$IT_{max} = 30$
Separationsabstand:	$q_{min} = 0{,}001$
Maximaler Fehler:	$\sigma_{stat} = 0{,}01$
	$\sigma_{diff} = 0{,}001$ (77(c), 78(c))
	$\sigma_{stat} = 0{,}005$
	$\sigma_{diff} = 0{,}0005$ (77(d), 78(d))
Zeitschrittweite:	$\tau = 80$
Zeitschritte:	$t = 2.160$
Zeitintervall:	$\mathcal{I} = [0, 2.160 \cdot \tau]$
Rotation:	$2 \cdot \pi$
Konzentration:	Slotted Disc
Initiale Knotenzahl:	$N = 40.000$ *grid* (77)
	$N = 40.000$ *rnd* (78)

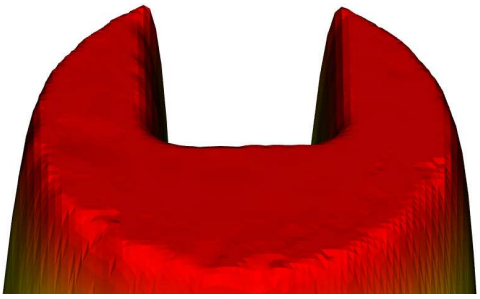
Wir werden daher für die adaptive Advektion der Slotted Disc mit $\sigma_{stat} = 0{,}01$ und $\sigma_{diff} = 0{,}001$ arbeiten. Bei der Verwendung von glatten Ausgangsverteilungen müssen wir diese Werte jedoch anpassen.
Wir betrachten nun die adaptive Advektion für die glatte Ausgangsverteilung der Wendland-Funktion. Hier erhalten wir bei der Adaption der Knotenmenge deutlich geringere maximale Fehler. Damit der Funktionskegel während der



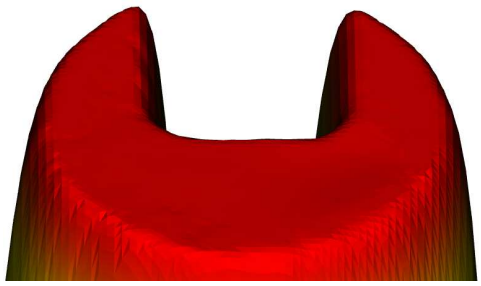
(a) Insgesamt ausgeführte Anzahl von Iterationen



(b) Entwicklung der Knotenzahl

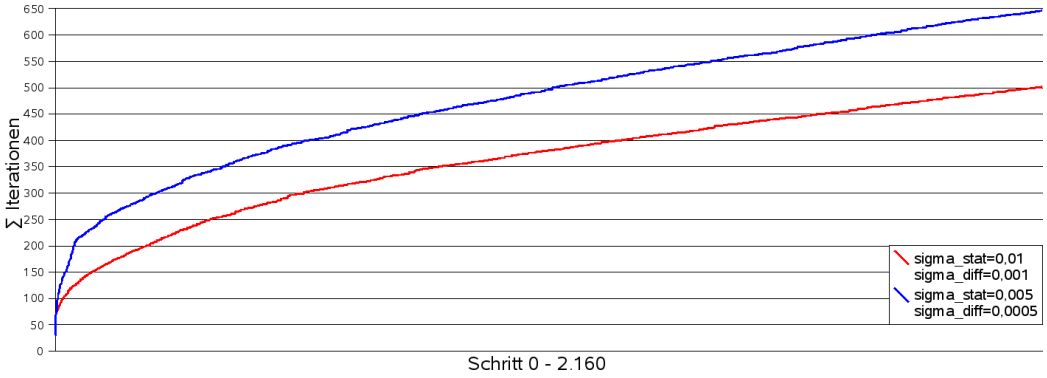


(c) $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$; $t = 2.160$

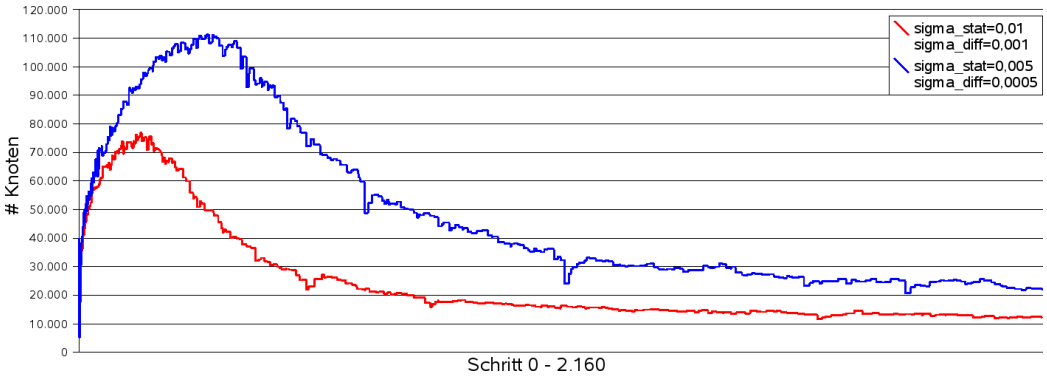


(d) $\sigma_{stat} = 0{,}005$; $\sigma_{diff} = 0{,}0005$; $t = 2.160$

Abb. 77: Eine Rotation bei unterschiedlichen Werten für σ_{stat} und σ_{diff}. $N = 40.000 grid$

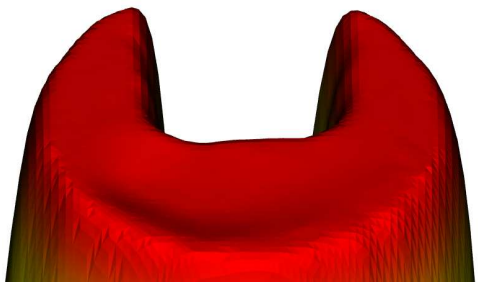


(a) Insgesamt ausgeführte Anzahl von Iterationen



(b) Entwicklung der Knotenzahl

(c) $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$; $t = 2.160$



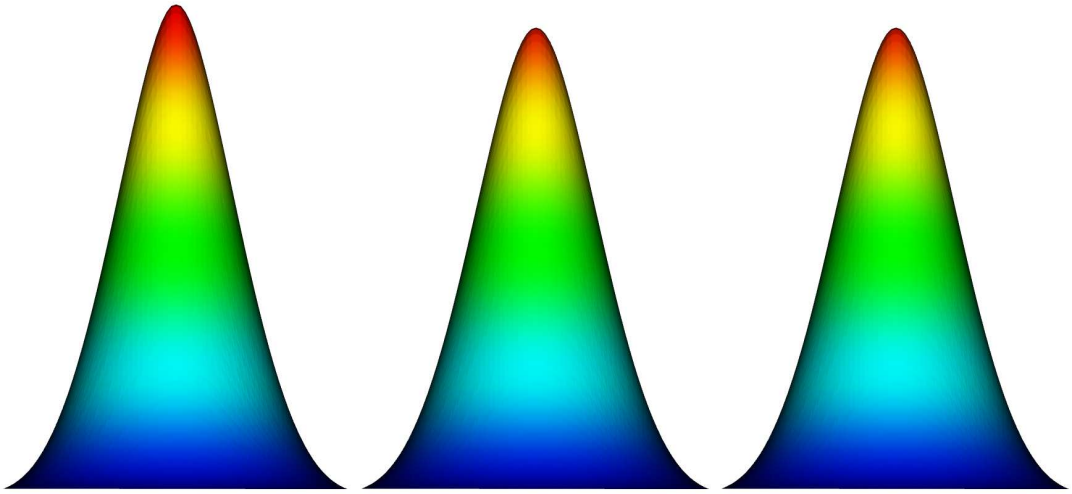
(d) $\sigma_{stat} = 0{,}005$; $\sigma_{diff} = 0{,}0005$; $t = 2.160$

Abb. 78: Eine Rotation bei unterschiedlichen Werten für σ_{stat} und σ_{diff}. $N = 40.000 rnd$

Advektion gut dargestellt werden kann, müssen wir dafür sorgen, dass hinreichend oft eine Adaption ausgeführt wird, um die Knoten mit dem Geschwindigkeitsfeld zu bewegen. Die Abbildungen 79 und 80 zeigen den Kegel der Wendland-Funktion jeweils nach einer halben Rotation. Für $IT_{min} = 1$ findet eine deutlich sichtbare Abflachung statt, so dass wir auch für glatte Ausgangsverteilungen $IT_{min} = 0$ verwenden. Setzen wir $\sigma_{stat} = 0{,}01$ und $\sigma_{diff} = 0{,}001$, so liegt der maximale Fehler bereits vor dem ersten Adaptionsschritt unter σ_{stat}. Es findet keine Adaption statt, da die Knoten das ganze Gebiet Ω überdecken. Der Funktionskegel wird im Falle einer gleichmäßigen Ausgangsverteilung nicht abgeflacht, sondern sogar erhöht. Das Clipping bewirkt hier ein Abschneiden der Kegelspitze (Abb. 79(f)). Bei der Zufallsverteilung findet eine leichte Abflachung statt.
Wir erreichen die Durchführung der Adaption durch Verkleinerung von σ_{stat} und σ_{diff} um den Faktor 10. Hier wird der Funktionskegel sehr gut reproduziert, wie die Abbildungen 79(e) und 80(e) zeigen.

`Parameter Abb. 79 und 80:`	
`Adaption & Advektion:`	`RBF-Interpolation`
`RBF:`	`Thin Plate Spline`
`Nachbarschaften:`	$n = 20$
`Ausdünnen:`	$\sigma_{crs} = 0{,}001$
`Verfeinern:`	$\sigma_{ref} = 0{,}1$
`Adaptionsschritte:`	$IT_{min} = 1$
	$IT_{max} = 30$ `(79(b), 79(c))`
	$IT_{min} = 0$
	$IT_{max} = 30$ `(79(e), 79(f))`
`Separationsabstand:`	$q_{min} = 0{,}001$
`Maximaler Fehler:`	$\sigma_{stat} = 0{,}001$
	$\sigma_{diff} = 0{,}0001$ `(79(b), 79(e))`
	$\sigma_{stat} = 0{,}01$
	$\sigma_{diff} = 0{,}001$ `(79(c), 79(f))`
`Zeitschrittweite:`	$\tau = 80$
`Zeitschritte:`	$t = 1.080$
`Zeitintervall:`	$\mathcal{I} = [0, 1.080 \cdot \tau]$
`Rotation:`	π
`Konzentration:`	`Wendland-Funktion`
`Initiale Knotenzahl:`	$N = 40.000$ ***grid*** `(79)`
	$N = 40.000$ ***rnd*** `(80)`

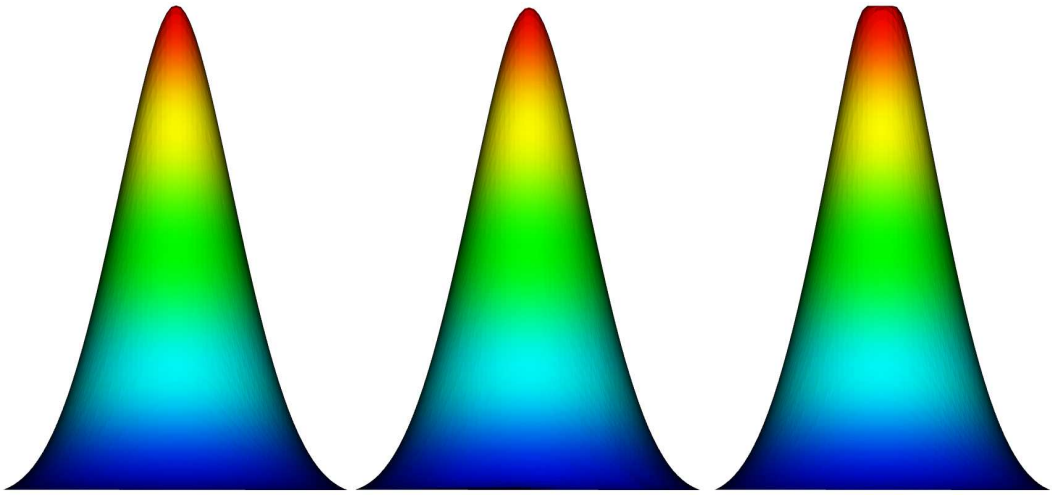
Wir betrachten für $\sigma_{stat} = 0{,}001$ und $\sigma_{diff} = 0{,}0001$ einen Testlauf mit fünf kompletten Rotationen. Es findet eine minimale Abflachung statt. Die Abbil-



(a) $t = 0$

(b) $t = 1.080$; $IT_{min} = 1$; $\sigma_{stat} = 0{,}001$; $\sigma_{diff} = 0{,}0001$

(c) $t = 1.080$; $IT_{min} = 1$; $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$

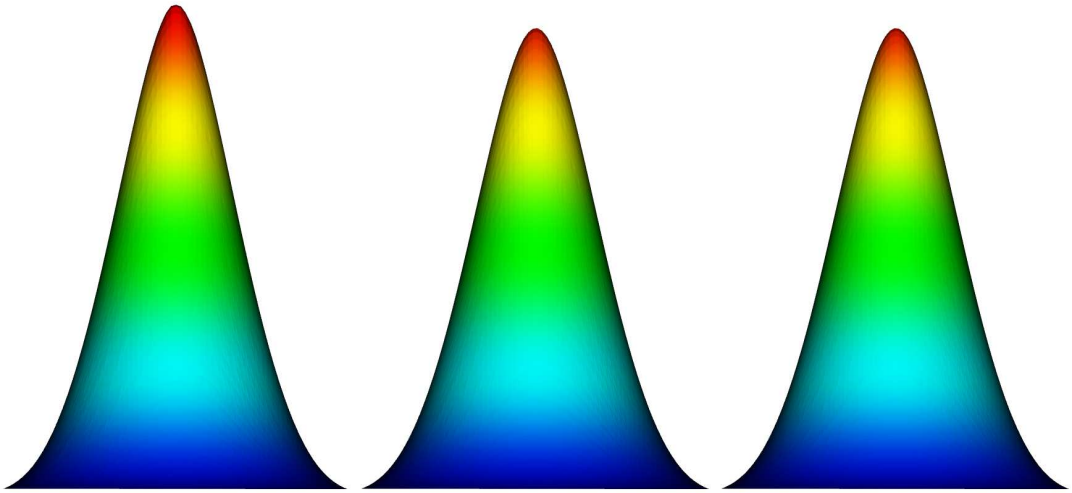


(d) $t = 0$

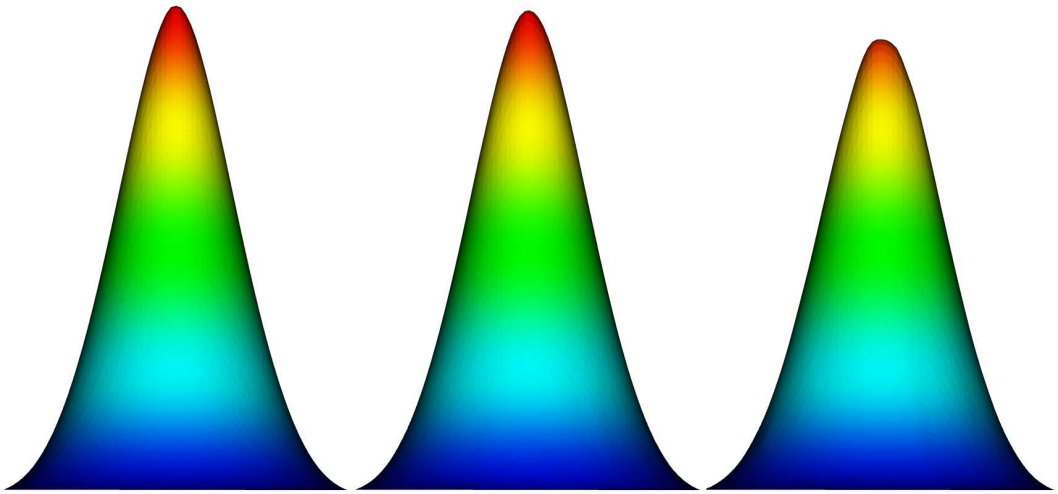
(e) $t = 1.080$; $IT_{min} = 0$; $\sigma_{stat} = 0{,}001$; $\sigma_{diff} = 0{,}0001$

(f) $t = 1.080$; $IT_{min} = 0$; $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$

Abb. 79: Seitenansicht nach einer halben Rotation. $N = 40.000 grid$



(a) $t = 0$ (b) $t = 1.080$; $IT_{min} = 1$; $\sigma_{stat} = 0{,}001$; $\sigma_{diff} = 0{,}0001$ (c) $t = 1.080$; $IT_{min} = 1$; $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$



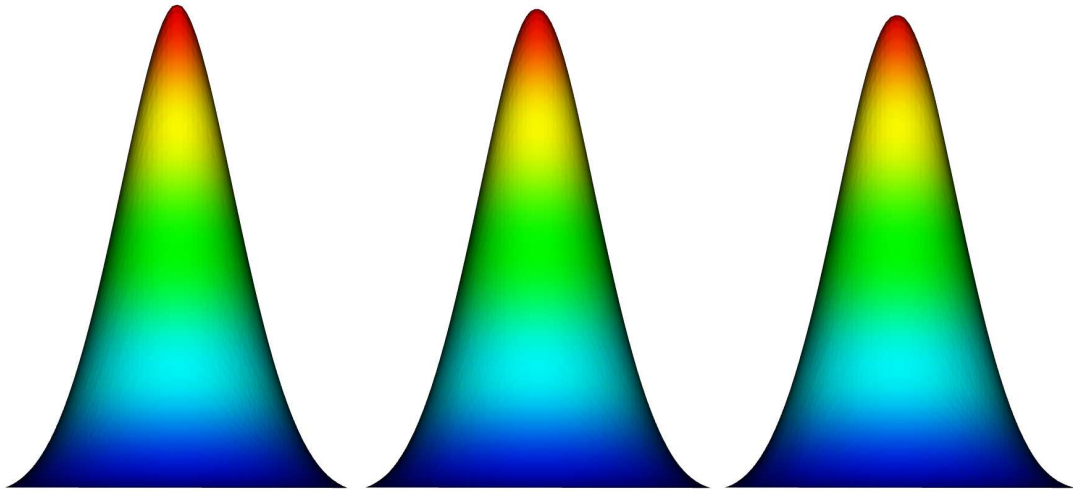
(d) $t = 0$ (e) $t = 1.080$; $IT_{min} = 0$; $\sigma_{stat} = 0{,}001$; $\sigma_{diff} = 0{,}0001$ (f) $t = 1.080$; $IT_{min} = 0$; $\sigma_{stat} = 0{,}01$; $\sigma_{diff} = 0{,}001$

Abb. 80: Seitenansicht nach einer halben Rotation. $N = 40.000 rnd$

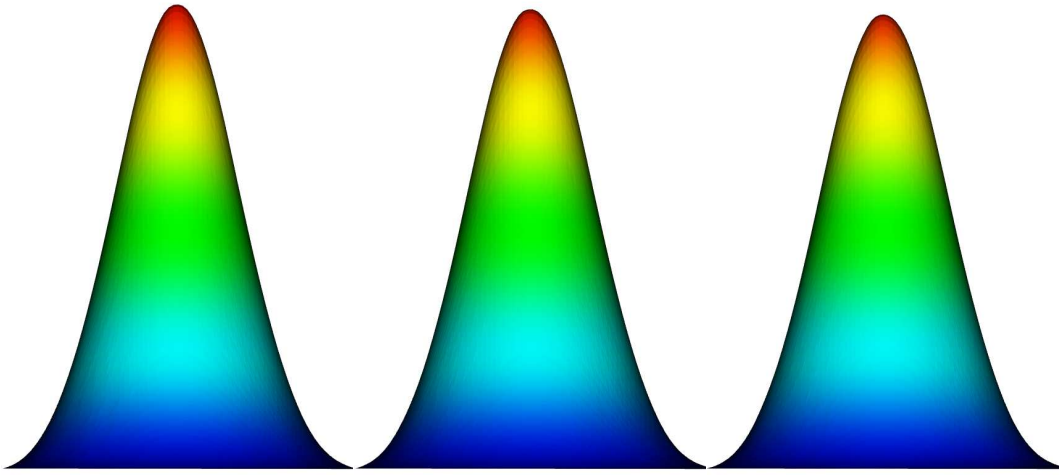
dungen 81 und 82 zeigen den Funktionskegel in der Seitenansicht nach jeweils fünf kompletten Umläufen.

Parameter Abb. 81, 82, 83, 84 und 85:	
Adaption & Advektion:	RBF-Interpolation
RBF:	Thin Plate Spline
Nachbarschaften:	$n = 20$
Ausdünnen:	$\sigma_{crs} = 0{,}001$
Verfeinern:	$\sigma_{ref} = 0{,}1$
Adaptionsschritte:	$IT_{min} = 0$
	$IT_{max} = 30$
Separationsabstand:	$q_{min} = 0{,}001$
Maximaler Fehler:	$\sigma_{stat} = 0{,}001$
	$\sigma_{diff} = 0{,}0001$
Zeitschrittweite:	$\tau = 80$
Zeitschritte:	$t = 10.800$
Zeitintervall:	$\mathcal{I} = [0,\, 10.800 \cdot \tau]$
Rotation:	$10 \cdot \pi$
Konzentration:	Wendland-Funktion
Initiale Knotenzahl:	$N = 40.000$ *grid* (81, 83)
	$N = 40.000$ *rnd* (82, 84)

Bei der Betrachtung der Knotenverteilung fällt auf, dass im in Rotationsrichtung hinteren Bereich des Funktionskegels wesentlich mehr Knoten erzeugt werden als vor dem Kegel. Die Verteilung der Knoten in Ω ist sowohl für die unterschiedlichen Ausgangsverteilungen als auch für die verschiedenen Zeitschritte sehr ähnlich. Die Abbildungen 83 und 84 stellen jeweils die Delaunay-Triangulierung der Knotenmengen dar.
Die Anzahl der Knoten liegt bereits nach wenigen Zeitschritten im Bereich von 15.000. In einigen Zeitschritten kommt es zu einer größeren Reduzierung der Knotenzahl, die sich jedoch stets wieder bis auf 15.000 erhöht (Abb. 85(a)) . Während es bei der Rotation der Slotted Disc in späteren Zeitschritten zu immer weniger Adaptionsschritten kommt, finden bei der Wendland-Funktion zu Beginn sehr wenige Iterationen statt, im späteren Verlauf jedoch immer mehr. Der maximale Fehler bewegt sich dabei sehr unregelmäßig zwischen 0,0003 und 0,001, wo dann jeweils eine Adaption stattfindet. Abbildung 85(c) stellt für die ersten 540 Zeitschritte den maximalen Fehler nach der Adaption dar. Das Verhalten des maximalen Fehlers ist über die gesamte Laufzeit unverändert. Die Abbildung 85(b) zeigt die Gesamtzahl der durchgeführten Iterationen für alle durchgeführten Zeitschritte.

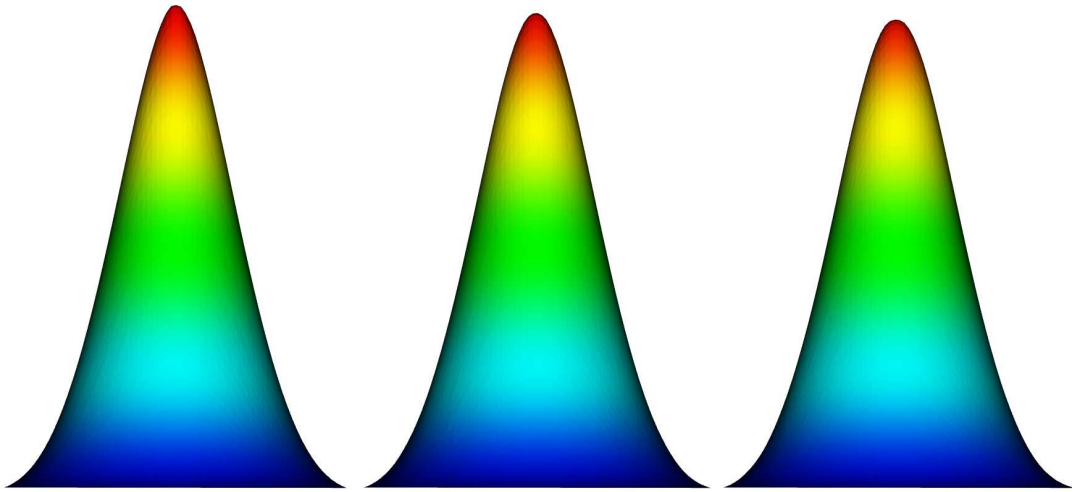


(a) $t = 0$ (b) $t = 2.160$ (c) $t = 4.320$

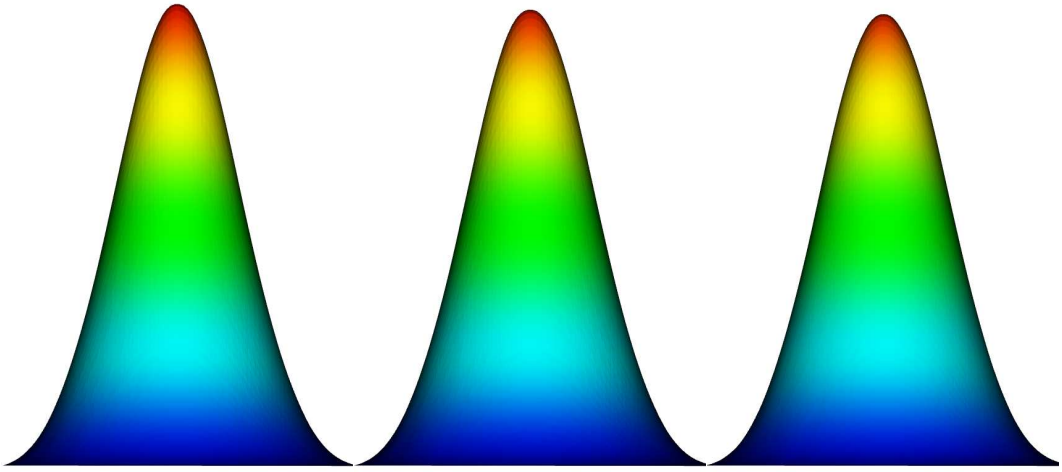


(d) $t = 6.480$ (e) $t = 8.640$ (f) $t = 10.800$

Abb. 81: Seitenansicht nach fünf Rotationen. $N = 40.000 grid$

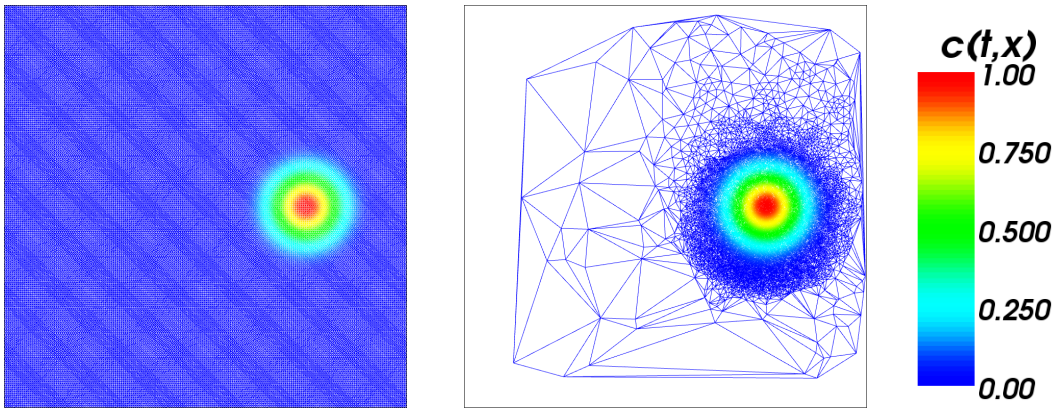


(a) $t = 0$ (b) $t = 2.160$ (c) $t = 4.320$

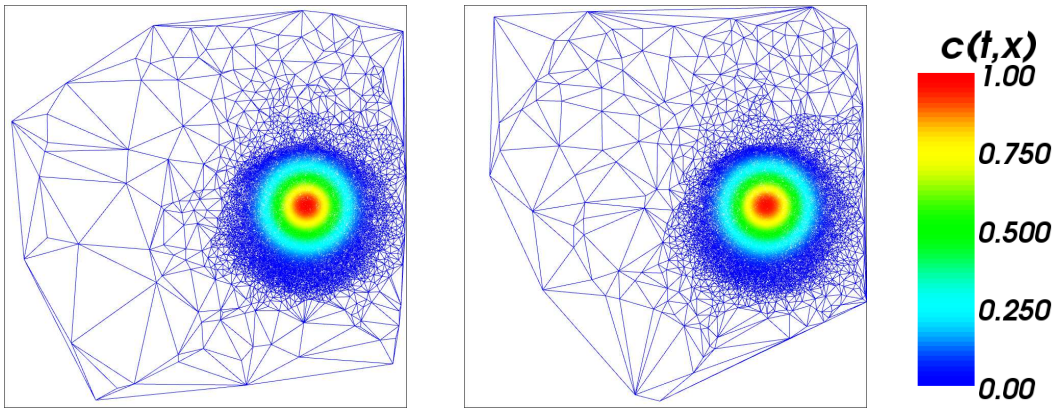


(d) $t = 6.480$ (e) $t = 8.640$ (f) $t = 10.800$

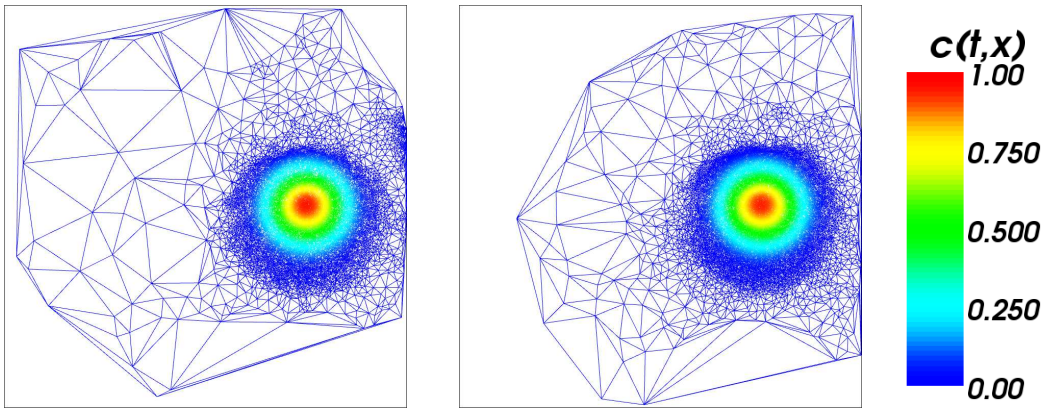
Abb. 82: Seitenansicht nach fünf Rotationen. $N = 40.000 rnd$



(a) $t = 0$ (b) $t = 2.160$

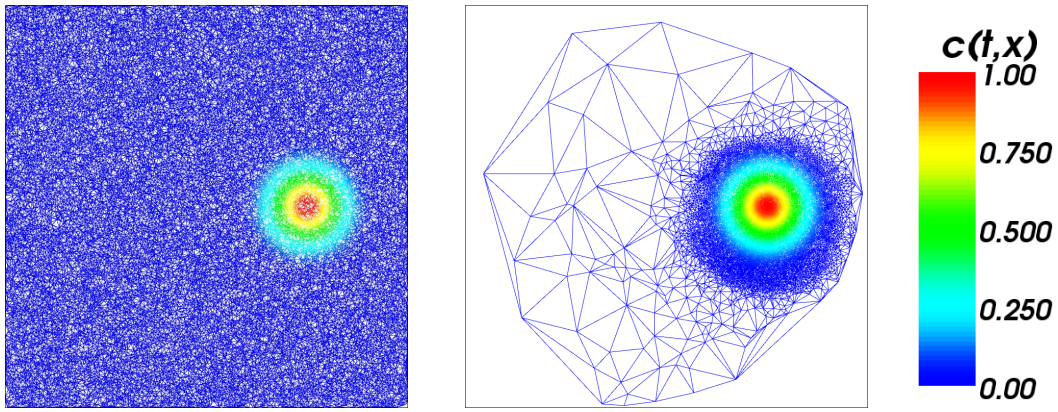


(c) $t = 4.320$ (d) $t = 6.480$

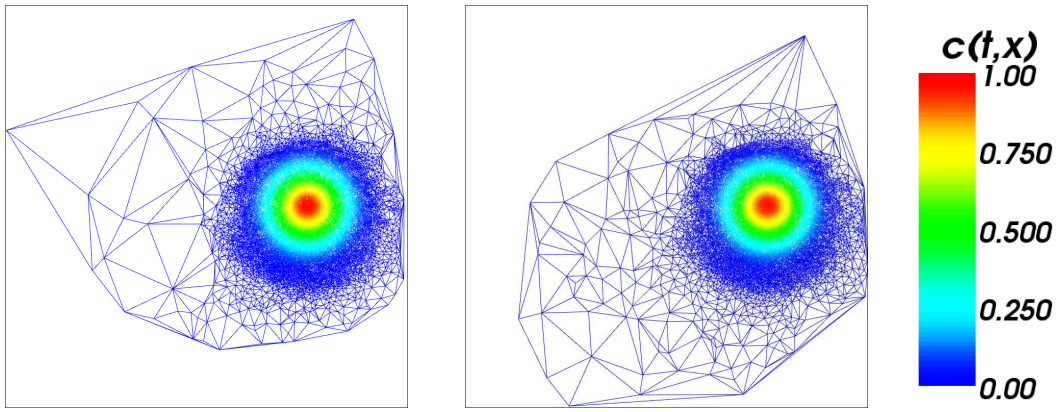


(e) $t = 8.640$ (f) $t = 10.800$

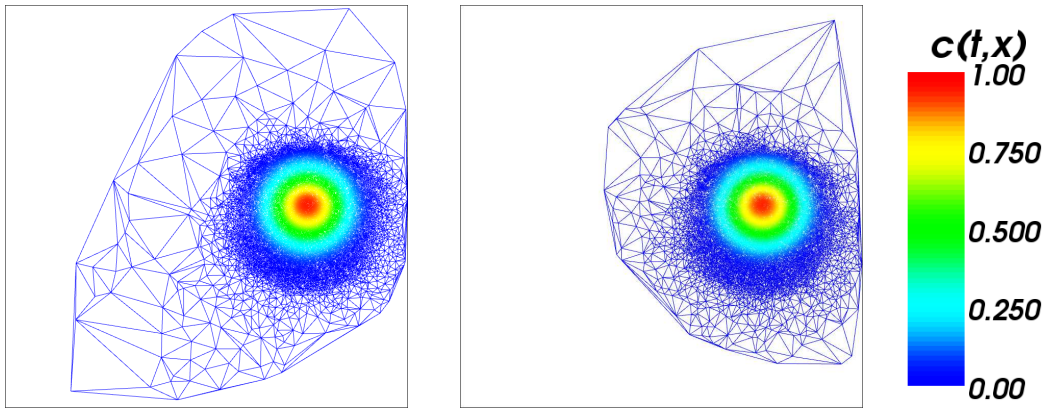
Abb. 83: Delaunay-Triangulierung nach fünf Rotationen. $N = 40.000 grid$



(a) $t = 0$ (b) $t = 2.160$

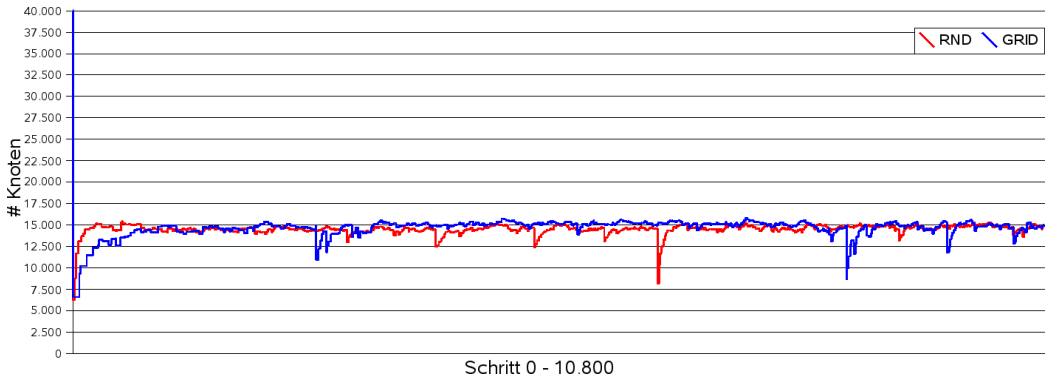


(c) $t = 4.320$ (d) $t = 6.480$

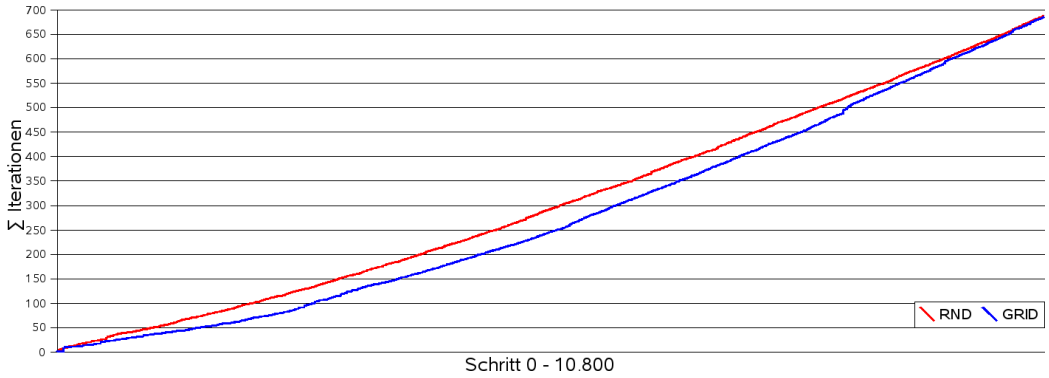


(e) $t = 8.640$ (f) $t = 10.800$

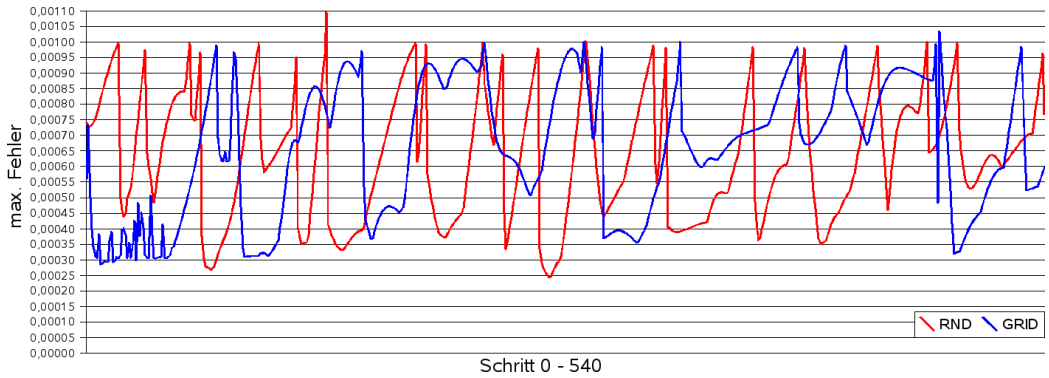
Abb. 84: Delaunay-Triangulierung nach fünf Rotationen. $N = 40.000 rnd$



(a) Entwicklung der Knotenzahl.



(b) Gesamtzahl der Iterationen.



(c) Maximaler Fehler während der ersten 540 Zeitschritte

Abb. 85: Fünf Rotationen der Wendland-Funktion.

In sämtlichen von uns durchgeführten Testreihen haben wir den Polynomgrad $Q = 2$, also lineare Polynome verwendet. Mit anderen Werten für Q konnten wir keine brauchbaren Resultate erzielen.
Um das Verfahren für die Adaption effizienter zu gestalten, haben wir außerdem die Variante getestet, den maximalen Fehler nur jeweils einmal pro Zeitschritt zu berechnen und für alle Iterationen zu verwenden. Dies führte ebenfalls nicht zu einer guten Reproduktion der Konzentration.

4 Fazit & Ausblick

„Nicht etwa, dass bei größerer Verbreitung des Einblickes in die Methode der Mathematik notwendigerweise viel mehr Kluges gesagt würde als heute, aber es würde sicher viel weniger Unkluges gesagt.“
Karl Menger (1902 - 1985)

Nachdem wir die unterschiedlichen Methoden für die Advektion und Adaption detailliert betrachtet haben, werden wir an dieser Stelle die Ergebnisse zusammenfassen.
Die Verwendung der Moving-Least-Squares-Approximation eignete sich sowohl für die Advektion als auch für die Adaption wesentlich weniger gut als die Interpolation mit radialen Basisfunktionen. Bei der Anwendung der reinen Semi-Lagrange-Advektion ohne Adaption der Knotenmenge erzielen wir sehr gute Ergebnisse unter Verwendung der RBFs. Wenn wir eine hinreichend dichte Knotenverteilung in Ω voraussetzen, ist die Advektion mit der RBF-Interpolation sehr unanfällig gegenüber Veränderungen in den Testparametern. Selbst bei ungünstiger Ausgangsverteilung mit sehr stark unstetigen Bereichen und längerer Testdauer ist die Reproduktion noch sehr gut.
Die dynamische Anpassung der Knotenmenge stellte sich uns hingegen als wesentlich problematischer dar. Während unserer Untersuchungen tauchten einige Fragen und Probleme auf, deren Lösungen für zukünftige Betrachtungen interessant sein werden:

- Wir haben für unsere Testreihen ausschließlich stationäre Geschwindigkeitsfelder mit $a(t,x) \equiv a(x)$ verwendet. Dabei mussten wir die Zeitschrittweite τ hinreichend klein wählen, so dass die Konzentration nicht in einen knotenfreien Bereich abgebildet wurde (Abb. 62). Da wir jedoch in der Praxis keine zeitunabhängigen Geschwindigkeitsfelder voraussetzen können, wird es erforderlich sein, τ dynamisch anzupassen.

- Die Verwendung der Adaption zur Verbesserung der Effizienz ist nur unter bestimmten Gesichtspunkten sinnvoll. Wir müssen eine deutliche Reduktion der Knotenmenge erreichen, da wir einen erheblichen Aufwand für die Verwaltung der Knoten betreiben müssen. Diese beinhaltet für jeden Adaptionsschritt

  die Bestimmung des maximalen Fehlers und der lokalen Fehler,

  die Bestimmung der Delaunay-Triangulierung mit Voronoipunkten,

  die Berechnung der lokalen Interpolante in jedem neuen Knoten und

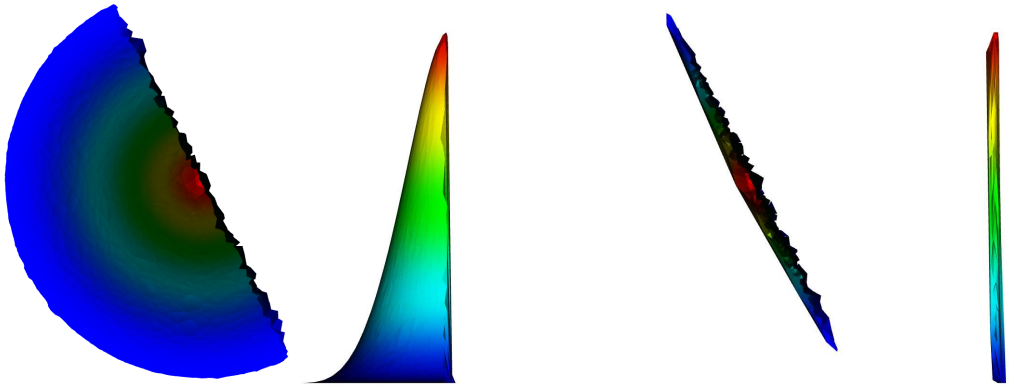
die Bestimmung der Nachbarschaften.

Die Tests haben jedoch gezeigt, dass die Knotenzahl während der Untersuchungen nicht unbedingt unter der Anfangsknotenzahl liegt, sondern teilweise sogar deutlich darüber.

- Um die Knotenzahlen während der Testreihen stabil und überschaubar zu halten, haben wir den Mindest-Separationsabstand q_{min} eingeführt und nach jedem Adaptionsschritt einen Teil der Knoten gelöscht. In vielen der Testreihen stieg durch die Adaption zunächst die Knotenzahl um bis zu 80%. Der Großteil der neu erzeugten Knoten wurde danach direkt wieder entfernt. Dadurch konnten wir insgesamt die Knotenzahl relativ konstant halten. Dies bedeutet jedoch einen immensen Rechenaufwand, der die Effizienz des Verfahrens stark verringert.

- Die von uns verwendeten Anfangskonzentrationen hatten entweder nur scharfe Kanten (Slotted Disc) oder waren komplett glatt (Wendland-Funktion). Dadurch waren die lokalen Fehler bei der Interpolation jeweils in einer vergleichbaren Größenordnung. Dies resultierte in einer guten Reproduktion der Konzentration durch die Adaption. Verwenden wir jedoch eine Konzentration, die sowohl glatte als auch scharfkantige Bereiche enthält, so wird deutlich, dass die Methode zur Adaption hier an ihre Grenzen stößt. Als Beispiel betrachten wir die Wendland-Funktion mit der zusätzlichen Bedingung, dass $c_0(x) \equiv 0$ für $x_2 + 2 \cdot x_1 > 1$ sei. Dies beschreibt einen Schnitt durch den Funktionskegel, wie in Abbildung 86(a) dargestellt. Abbildung 86(b) zeigt die Konzentration nach einem Adaptionsschritt. Der maximale Fehler, der an der Kegelspitze auftritt, ist in diesem Beispiel sehr groß. Da die Knoten auf der glatten Seite des Kegels sehr kleine lokale Fehler aufweisen, werden diese sämtlich ausgedünnt. Es bleibt lediglich die abgeschnittetene Kante des Kegels erhalten. Möglicherweis lässt sich diese Problematik durch die Einführung von unlöschbaren Knoten eingrenzen, so dass stets eine gewisse Überdeckung von Ω gewährleitet wird.

Insgesamt erscheint die Methode zur Adaption der Knotenmenge für die Praxis nicht sehr gut geeignet. Wir benötigen eine recht gute Kenntnis der Anfangsbedingungen, um mit der adaptiven Semi-Lagrange-Advektion brauchbare Ergebnisse zu erzielen. Dies umfasst einerseits Informationen über das Geschwindigkeitsfeld, da wir die Zeitschrittweite τ hinreichend klein wählen müssen. Andererseits benötigen wir Kenntnis über die Konzentration zum Zeitpunkt $t = 0$, um die Parameter für die Simulation einzustellen.
Selbst bei Kenntnis der Ausgangsbedingungen lässt sich ein Auftreten der oben



(a) Ausgangskonzentration

(b) Konzentration nach einem Adaptionsschritt

Abb. 86: Adaption des abgeschnittenen Funktionskegels. $N = 40.000 rnd$

beschriebenen Probleme während der Berechnungen nicht ausschließen. Dies gilt insbesondere für nichttriviale, zeitabhängige Geschwindigkeitsfelder und für Konzentrationen mit sehr großen maximalen Fehlern.

`Parameter Abb. 86:`	
`Adaption:`	`RBF-Interpolation`
`RBF:`	`Thin Plate Spline`
`Nachbarschaften:`	$n = 20$
`Ausdünnen:`	$\sigma_{crs} = 0{,}01$
`Verfeinern:`	$\sigma_{ref} = 0{,}1$
`Adaptionsschritte:`	$t_A = 1$
`Konzentration:`	`Wendland-Funktion (abgeschnitten)`
`Initiale Knotenzahl:`	$N = 40.000$***rnd***

Literatur

[1] J. Behrens, A. Iske: *Grid-Free Adaptive Semi-Lagrangian Advection Using Radial Basis Functions*, Pergamon - Computers and Mathematics with Applications 43, 319 - 327 (2002)

[2] Jörn Behrens, Armin Iske, Stefan Pöhn: *Effective node adaption for grid-free semi-Lagrangian advection*

[3] Jörn Behrens, Armin Iske, Stefan Pöhn: *Adaptive Meshfree Method of Backward Characteristics for Nonlinear Transport Equations*, TU München

[4] Armin Iske: *Adaptive Irregular Sampling in Meshfree Flow Simulation*

[5] Armin Iske, Martin Käser: *Conservative Semi-Lagrangian Advection on Adaptive Unstructured Meshes*

[6] Martin Andreas Käser: *Dissertation - Adaptive Methods for the Numerical Simulation of Transport Processes* 7 - 24, TU München, 2003

[7] Holger Wendland: *Local polynomial reproduction and moving least squares approximation* , Oxford University Press, 2000

[8] Holger Wendland, Christian Rieger: *Approximate Interpolation with Applications to Selecting Smoothing Parameters*, Numerische Mathematik 101, 729-748, 2005

[9] Gerald Warnecke: *Analytische Methoden in der Theorie der Erhaltungsgleichungen*, Teubner, 1999

[10] Paul Houston, David P. Hunt, Jeremy Levesley: *Semi-Lagrangian Radial Basis Function Methods for the Numerical Approximation of the Linear Transport Equation*, IMA Journal of Numerical Analysis, 2005

[11] M. Falcone, R. Ferretti: *Convergence Analysis for a Class of High-Order Semi-Lagrangian Advection Schemes*, SIAM Journal on Numerical Analysis (Volume 35, Issue 3), 909-940, June 1998

[12] Rainer Baule: *Moving least squares approximation mit parameterabhängigen Gewichtsfunktionen*, Diplomarbeit, Georg-August-Universität Göttingen, 2000

[13] David M. Mount, Sunil Arya: *ANN: A Library for Approximate Nearest Neighbor Searching*, *http://www.cs.umd.edu/∼mount/ANN/, Version 1.1.1*

[14] David M. Mount: *ANN Programming Manual*, Institute for Advanced Computer Studies, University of Maryland, 2005

[15] Jonathan Richard Shewchuk: *Triangle, A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator*, Computer Science Division, University of California at Berkeley, *http://www.cs.cmu.edu/∼quake/triangle.html, Version 1.6*

[16] *Ply - Polygon File Format*, *http://astronomy.swin.edu.au/∼pbourke/dataformats/ply/*

[17] Kitware: *Paraview - A Parallel Visualization Application*, *http://www.paraview.org*

[18] Ken Martin, Will Schroeder, Bill Lorensen: *VTK - The Visualization ToolKit*, *http://www.vtk.org*

[19] Holger Wendland: *RBF Surface Creator*, *http://www.num.math.uni-goettingen.de/wendland/*

[20] Christian Menz: *SLM*, Programm zur Durchführung der Semi-Lagrange-Advektion, siehe beiliegender Datenträger.

[21] Christian Menz: *slm_smoothing*, Tool zur optischen Aufbereitung der Berechnungen, siehe beiliegender Datenträger.

[22] Christian Menz: *ply2vtk*, Tool zur Formatkonvertierung, siehe beiliegender Datenträger.

A Variablen & Bezeichner

„Die ganzen Zahlen hat der liebe Gott geschaffen, alles andere ist Menschenwerk.“

Leopold Kronecker (1823 - 1891)

$\mathbb{N}$	Natürliche Zahlen
$\mathbb{R}$	Reelle Zahlen
$T \in \mathbb{N}$	Endzeitpunkt
$\mathcal{I} = [0,T] \subset \mathbb{N}$	Zeitintervall
$t \in \mathcal{I}$	Aktueller Zeitpunkt
$n \in \mathbb{N}$	Aktuelle Gesamtzahl Stützstellen
F	Ortsfunktion der Partikel, orts- und zeitabhängig
F^t	Ortsfunktion der Partikel, ortsabhängig bei festem $t \in \mathcal{I}$
$x := G^x$	Ortsfunktion der Partikel, zeitabhängig bei festem $x \in \Omega$
$\dot{x}(t) := \frac{dx}{dt}(t)$	Ableitung der Ortsfunktion nach der Zeit
$\Omega \subset \mathbb{R}^d$	Betrachtetes Gebiet
$d \geq 0$	Raumdimension des Gebiets Ω
Ω^0	Offenes, zusammenhängendes und beschränktes Kontrollvolumen, $\Omega^t := F(t,\Omega)$
$X := \{x_1, \ldots, x_n\}$, $X \subset \Omega$	Diskrete Knotenmenge
q_X	Separationabstand der Knoten in X
q_{min}	Mindest-Separationabstand. $q_X \geq \frac{q_{min}}{2}$
$x \in X$	Partikel (Knoten, Punkt)
x^-	Upstreampunkt zu x
$\tilde{x}$	Approximation von x^-
$\xi \in \Omega$	Gerade betrachtete Stelle, nicht unbedingt $\xi \in X$
ϕ	Radiale Basisfunktion
r	Support-Radius der RBF-Interpolation
λ_i, μ_i	Koeffizienten der RBF-Interpolation
s	Interpolante
Π_m^d	Linearer Raum der d-variaten Polynome vom Grad kleiner als m
$p \in \Pi_m^d$	Polynom
$B_Q := \{p_1, ..., p_Q\}$	Basis von Π

c	Zeit- und ortsabhängige Dichtefunktion (Konzentration)
$\vec{a}$	Geschwindigkeitsfeld, orts- und zeitabhängig, $\vec{a}(t,x) := \frac{\partial}{\partial t}F(t,x)$
ω_1,ω_2	Skalare zur Festlegung der Geschwindigkeitsfelder
$0 < \tau \leq T$	Zeitschrittweite
η	Signifikanzwert, Fehlerindikator der lokalen Approximation
σ_{crs}, σ_{ref}	Parameter für Verfeinern und Ausdünnen
$t_A \in \mathbb{N}_0$	Aktueller Adaptionsschritt
σ_{stat}, σ_{diff}	Parameter für die stationäre Adaption (max. Fehler)
IT_{min}, IT_{max}	Parameter für die stationäre Adaption (Minimale bzw. maximale Anzahl von Iterationen)
$N \equiv N_0 \in \mathbb{N}$	Anzahl Stützstellen in der Ausgangsverteilung
$N_t \in \mathbb{N}$, $N_{t_A} \in \mathbb{N}$	Anzahl Stützstellen zur Zeit t bzw. nach t_A Iterationen.
$k \in \mathbb{N}$	Anzahl Knoten in jeder der Nachbarschaften
$\mathcal{N} \equiv \mathcal{N}_k(x)$	k-Nachbarschaft von x
$\mathcal{N}^*$	$\mathcal{N}^* := \mathcal{N} \cup \xi$
$\nabla_x = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d})$	Gradient bzgl. x
$V(x)$	Voronoizelle zu x
$V^*(x)$	Voronoipunkte zu x
$T_D(X)$	Delaunay-Triangulierung von X
$L > 0, L \in \mathbb{N}$	Lipschitzkonstante
H	Hilbertraum

B Algorithmen

„Suche das Einfache und misstraue ihm“
Alfred North Whitehead (1861 - 1947)

Wir beschreiben in diesem Abschnitt die für unsere Versuche verwendeten Algorithmen. Wird in einem Algorithmus ein anderer aufgerufen, so verwenden wir die Notation (→ B.xx) mit Verweis auf die Nummer xx des aufgerufenen Algorithmus.

Algorithmus B.1. Adaptive Semi-Lagrange Advektion
Dieser Algorithmus beschreibt die Durchführung eines Zeitschritts τ.

INPUT: $X \equiv X(t)$ und ${c^t}_X$
  WIEDERHOLE {
    Bestimme $\tilde{X}$ durch Adaption (→ B.2)
  } BIS $\tilde{X}$ stationär wird
  Bestimme $\tilde{c}^{t+\tau}_{\tilde{X}}$ durch Advektion (→ B.3)
OUTPUT: $\tilde{X} \equiv \tilde{X}(t+\tau)$ und $\tilde{c}^{t+\tau}_{\tilde{X}}$

Algorithmus B.2. Adaption

INPUT: $X \equiv X(t)$ und ${c^t}_X$
  FÜR ALLE $x \in X${
    Bestimme den lokalen Fehler $\eta(x)$ (→ B.4)
  }
  Bestimme globalen maximalen Fehler $\eta^* := max\{\eta(x)\}$
  FÜR ALLE $x \in X${
    WENN ($\eta(x) > \sigma_{ref} * \eta^*$) DANN
      Verfeinere x(→ B.5)
    SONST WENN ($\eta(x) < \sigma_{crs} * \eta^*$) DANN
      Dünne x aus (→ B.6)
  }
  Erzeuge $\tilde{X}$ durch Löschen und Einfügen von Punkten. (→ B.7)
  Lösche doppelte Punkte (→ B.8)
OUTPUT: $\tilde{X} \equiv \tilde{X}(t)$ und $\tilde{c}^t_{\tilde{X}}$

Algorithmus B.3. Advektion

INPUT: $X \equiv X(t)$, c^t_X

FÜR ALLE $x \in X$ {
Bestimme $\tilde{x}$, die Approx. des Upstreampunktes (→ B.9)
Berechne $s(\tilde{x})$ mit $s \equiv s|_{\mathcal{N}(\tilde{x})}$ (→ B.10(RBF) oder B.11(MLS))
Wende Clipping an mit $\mathcal{N}(\tilde{x}) = X$ an: $s(\tilde{x}) \mapsto \tilde{s}(\tilde{x})$ (→ B.12)
Führe Advektion durch mit $c(t+\tau,x) = \tilde{s}(\tilde{x})$.
}
OUTPUT: $c_X^{t+\tau}$

Algorithmus B.4. Fehlerabschätzung
INPUT: $\xi \in X$, $X \equiv \mathcal{N}(\xi) \subset X(t)$, und ${c^t}_X$.
Berechne s mit $s \equiv s|_X \equiv {c^t}_X$ (→ B.10(RBF) oder B.11(MLS))
Bestimme lokalen Fehler durch $\eta(\xi) := |c(\xi) - s(\xi)|$.
OUTPUT: $\eta(\xi)$

Algorithmus B.5. Verfeinern
Wir werden nur Punkte zur Knotenmenge hinzufügen, die gewissen Kriterien entsprechen, also z.B. in der Boundingbox von X liegen. Die Menge aller Punkte, die diese Kriterien erfüllen, bezeichnen wir als K_ξ.
Das tatsächliche Einfügen geschieht erst nach Aufruf eines confirm() (→ B.7). Zunächst werden die Punkte temporär in einer Liste Ξ gespeichert.
INPUT: $X \equiv X(t)$, $\xi \in X$, ${c^t}_X$, $\Xi \subset \mathbb{R}^3$
Bestimme die Voronoizelle $V(\xi)$ (→ B.14)
FÜR ALLE $v \in V$ {
WENN $v \in K_\xi$ DANN {
Berechne s mit $s \equiv s|_{\mathcal{N}(v)}$ (→ B.10 oder B.11)
Wende Clipping an: $s(v) \mapsto \tilde{s}(v)$ (→ B.12)
Erweitere die Liste neuer Punkte $\tilde{\Xi} := \Xi \cup \tilde{v} = (v_1, v_2, \tilde{s}(v))^\top$
}
}
OUTPUT: $\tilde{\Xi}$

Algorithmus B.6. Ausdünnen
Das Ausdünnen besteht in der Praxis aus zwei Schritten. Zuerst wird jeder Punkt, der auszudünnen ist, in die Liste zu löschender Punkte I eingefügt. Dann wird durch ein confirm() (→ B.7) der Löschvorgang tatsächlich ausgeführt.

INPUT: $i \in \mathbb{N}$, $I \subset \mathbb{N}_0$
  Füge i in die Liste zu löschender Punkte ein: $\tilde{I} := I \cup i$
OUTPUT: $\tilde{I}$

Algorithmus B.7. Änderungen durchführen (confirm)
INPUT: $X \equiv X(t)$, $I \subset \mathbb{N}_0$, $\Xi \subset \mathbb{R}^3$
  FÜR ALLE $i \in I$
    Lösche markierte Punkte $x_i \in X$ durch $\tilde{X} := X \backslash x_i$
  FÜR ALLE $\xi = (\xi_1,\xi_2,\xi_3)^\top \in \Xi \{$
    Füge Punkt ein durch $\tilde{X} := \tilde{X} \cup (\xi_1,\xi_2)^\top$
    Setze Funktionswert durch $\tilde{c}^t_{\tilde{X}}(\xi) := \xi_3$
  $\}$
  Aktualisiere die Gesamtstruktur der Daten ($\rightarrow$ B.16)
OUTPUT: $\tilde{X} \equiv \tilde{X}(t)$ und $\tilde{c}^t_{\tilde{X}}$

Algorithmus B.8. Duplikate löschen
Sobald der Mindest-Separationsabstand q_{min} unterschritten wird, löschen wir stets den später eingefügten Knoten. Dies müssen wir beim Indizieren der Knoten berücksichtigen.
INPUT: $X \equiv X(t)$
  FÜR ALLE $\xi \in X$
    Bilde $\tilde{X} := X \setminus \{x \in X : \|\xi - x\|_2 < q_{min}\}$
OUTPUT: $\tilde{X} \equiv \tilde{X}(t)$

Algorithmus B.9. Approximation des Upstreampunktes
INPUT: $x \in \mathbb{R}^2$, $a(t,\cdot)$, $k \in \mathbb{N}$
  Berechne α_k mit der Rekursionsformel
    $\alpha_{n+1} = \tau \cdot a(t + \frac{\tau}{2}, x - \frac{\alpha_n}{2}), \alpha_0 = 0$
  Setze $\tilde{x} := x - \alpha_k$
OUTPUT: $\tilde{x}$

Algorithmus B.10. Interpolation mit radialen Basisfunktionen
Wir verwenden bei der Interpolation 2-variate Monome $p_1, \ldots, p_q$ vom Grad kleiner als m als Basis von Π^d_m, wie in Kapitel 2.3 auf Seite 13 beschrieben.

Wir verwenden außerdem eine vorher festgelegte radiale Basisfunktion ϕ sowie einen Parameter $\rho > 0$ zur Stabilisierung des linearen Gleichungssystems. Weiterhin verwenden wir die Variablen $A \in \mathbb{R}^{(k+q)\times(k+q)}$, und b,$u \in \mathbb{R}^{(k+q)}$.

INPUT: $\xi \in X$, $\{x_1,\ldots,x_k\} = \mathcal{N}(\xi) \subset X(t)$ und $c^t_{\mathcal{N}}$
  FÜR ALLE i von 1 bis k {
    FÜR ALLE j von 1 bis k
      Setze den Eintrag $A_{(i,j)} := \phi(\|x_i - x_j\|_2)$
    FÜR ALLE j von 1 bis q
      Setze den Eintrag $A_{(j+k,i)} = A_{(i,j+k)} := p_j(x_i)$
    Setze den Eintrag $b_i := c(t,x_i)$
    Addiere Stabilisierungsparameter: $A_{(i,i)} += \rho$
  }
  FÜR ALLE i von 1 bis q {
    Setze den Eintrag $b_{k+i} := 0$
    FÜR ALLE j von 1 bis q
      Setze den Eintrag $A_{(k+i,k+j)} := 0$
  }
  Löse das lineare Gleichungssystem $A \cdot u = b$ nach u
  FÜR ALLE i von 1 bis q
    Summiere $P := P + u_{i+k} \cdot p_i(\xi)$
  FÜR ALLE i von 1 bis k
    Summiere $S := s + u_i \cdot \phi(\|\xi - x_i\|_2)$
  Berechne $s(\xi) := S + P$
OUTPUT: $s(\xi)$

Algorithmus B.11. Moving Least Squares Approximation
Wir verwenden bei der Approximation 2-variate Monome, die eine Basis $B = \{p_1, \ldots, p_q\}$ von Π_m^2 bilden.
Es ist $x^m = (x_1,x_2)^m := x_1^{m_1} x_2^{m_2}$ mit $m_1 + m_2 \leq m$.
Wir verwenden außerdem eine vorher festgelegte radiale Basisfunktion ϕ mit Supportradius $r = 1$ als Gewichtsfunktion.
Weiterhin verwenden wir die Variablen $A \in \mathbb{R}^{q \times q}$, $b,u \in \mathbb{R}^q$ und $a^* \in \mathbb{R}^k$.

INPUT: $\xi \in X$, $\{x_1, \ldots, x_k\} = \mathcal{N}(\xi) \subset X(t)$ und $c_{\mathcal{N}}^t$
  Bestimme $\delta := max_{x \in \mathcal{N}} \{\|\xi - x\|_2\}$
  FÜR ALLE i von 1 bis q {
    FÜR ALLE j von 1 bis q {
      FÜR ALLE n von 1 bis k {
        Skaliere $\hat{x}_n := \frac{x_n - \xi}{\delta}$
        Summiere $S := S + \phi(\hat{x}) \cdot p_j(\hat{x}) \cdot p_i(\hat{x})$
      }
      Setze den Eintrag $A_{(i,j)} := S$
    }
    Setze $b_i := 0$
  }
  Setze $b_1 := 1$
  Löse das lineare Gleichungssystem $A \cdot u = b$ nach u
  FÜR i von 1 bis k {
    FÜR j von 1 bis q
      Summiere $S := S + u_j \cdot p_j(\hat{x}_i)$
    Setze $a_i^* := S \cdot \phi(\hat{x}_i)$
    Summiere $s(\xi) := s(\xi) + a_i^* \cdot c(t,x_i)$
  }
OUTPUT: $s(\xi)$

Algorithmus B.12. Clipping

INPUT: $\xi \in \mathbb{R}^2$, $\mathcal{N}(\xi) \subseteq X(t)$, s
  Bestimme $M := max\{c(x) : x \in \mathcal{N}\}$
  Bestimme $m := min\{c(x) : x \in \mathcal{N}\}$
  WENN $s(\xi) > M$
    Setze $\tilde{s}(\xi) := M$
  SONST WENN $s(\xi) < m$
    Setze $\tilde{s}(\xi) := m$
  SONST
    Setze $\tilde{s}(\xi) := s(\xi)$
OUTPUT: $\tilde{s}(\xi)$

Algorithmus B.13. Delaunay-Triangulierung

Für die Erzeugung der Delaunay-Triangulierung verwenden wir die *triangle-Bibliothek* [15]

INPUT: $X \equiv X(t)$
  Erzeuge die $\triangle$-Indexmenge $T_D(X)$ durch Triangulierung von X
    ($\rightarrow$ triangulateio)
OUTPUT: $T_D(X) \subset \mathbb{N}_0^3$

Algorithmus B.14. Bestimmung der Voronoi-Zelle

Die Voronoizelle ist der zur Delaunay-Triangulierung duale Graph. Daher verwenden wir die vorliegenden Triangulierungsdaten zur Berechnung der Voronoizelle. T_ξ sei die Menge aller Dreiecke in T, die ξ enthalten.

INPUT: $\xi \in X \equiv X(t)$, $T_\xi \subset \mathbb{N}_0^3$
  FÜR ALLE $v \in T_\xi$ {
    Bestimme den Schnittpunkt M der Mittelsenkrechten von v
    Füge M zu $V(\xi)$ hinzu
  }
OUTPUT: $V(\xi)$

Algorithmus B.15. Bestimmung der Nachbarschaften

Die Nachbarschaften werden wir mit der *ANN-Bibliothek* ([13], [14]) bestimmen, die einen KD-Tree $KD(X)$ verwendet.

INPUT: $X \equiv X(t)$, $\xi \in X$, $k \in \mathbb{N}$ und $KD(X)$.
  Bestimme die Nachbarschaft $\mathcal{N}_k(\xi) \subset X$ ($\rightarrow$ ANNksearch).
OUTPUT: $\mathcal{N}_k(\xi)$

Algorithmus B.16. Aktualisierung der Strukturen

INPUT: $X \equiv X(t)$
  Bestimme die Delaunay-Triangulierung $T_D(X)$ ($\rightarrow$ B.13)
  Baue neuen KD-Tree $KD(X)$ auf
  FÜR alle $\xi \in X$
    Bestimme die Nachbarschaft $\mathcal{N}_k(\xi)$ ($\rightarrow$ B.15)
  FÜR ALLE $t \in T_D(X)$, $t \hat{=} (x,y,z)^\top \in X^3$
    Aktualisiere T_x, T_y, T_z mit t
OUTPUT: $T_D(X) \subset \mathbb{N}_0^3$, $\mathcal{N}_k(\xi), T_\xi \quad \forall\, \xi \in X$

Algorithmus B.17. Bestimmung des KD-Tree

Die Bestimmung des KD-Tree wird rekursiv mit der Median-Methode (Kapitel 2.2) durchgeführt. Wir teilen X solange, bis sich in jedem Teil nur noch genau ein Punkt befindet.
Ein Blatt werden wir notieren, indem wir einen Punkt aus X als Unterbaum einfügen. Für Verzweigungen in den linken bzw. rechten Unterbaum schreiben wir KD_{links} bzw. KD_{rechts}.

INPUT: $X \equiv X(t)$
  WENN $|X| = 1$ DANN
    $KD(X) := x_1 \in X$
  SONST $\{$
    Teile X in X_1 und X_2
    $KD(X)_{\texttt{links}} := KD(X_1)$, $KD(X)_{\texttt{rechts}} := KD(X_2)$
  $\}$
OUTPUT: $KD(X)$

C Programm

„Dass die niedrigste aller Tätigkeiten die arithmetische ist, wird dadurch belegt, dass sie die einzige ist, die auch durch eine Maschine ausgeführt werden kann. Nun läuft aber alle Analysis finitorum et infinitorum im Grunde doch auf Rechnerei zurück. Danach bemesse man den ‚mathematischen Tiefsinn'.“

Arthur Schopenhauer (1788 - 1860)

Für sämtliche in dieser Arbeit durchgeführten Testreihen wurde das Programm `slm` verwendet ([20]). Das Pogramm wurde vom Autor dieser Arbeit entwickelt und steht quelloffen auf dem beiliegenden Datenträger zur Verfügung.
Die verwendete Programmiersprache ist `C++`.

C.1 Klassenstruktur

Abb. 87 zeigt die verwendeten Klassen und Bibliotheken in der Übersicht als Klassendiagramm.
Wir werden im folgenden Abschnit näher auf die einzelnen Klassen sowie die enthaltenen Features eingehen.

C.2 Klassen

ANN

Wir benötigen fur unsere Betrachtungen eine effiziente Methode zur Bestimmung der Nachbarschaft eines Punktes. Wir verwenden für die Berechnung der nächsten Nachbarn die so genannte *ANN Bibliothek* [13] in der Version 1.1.1 vom 04.08.2006. Die Bibliothek stellt uns die Strukturen `ANNpoint` und `ANNpointArray` zur Verfügung, die wir für die Verwaltung der Punkte nutzen:

```
typedef double ANNcoord;
typedef ANNcoord *ANNpoint;
typedef ANNpoint *ANNpointArray;
```

`class ANNkd_tree`
Die ANN-Bibliothek stellt uns außerdem die Klasse `ANNkd_tree` zur Verfügung, die wir zur Bestimmung der Nachbarschaften einsetzen.

Autoren: D. M. Mount, S. Arya

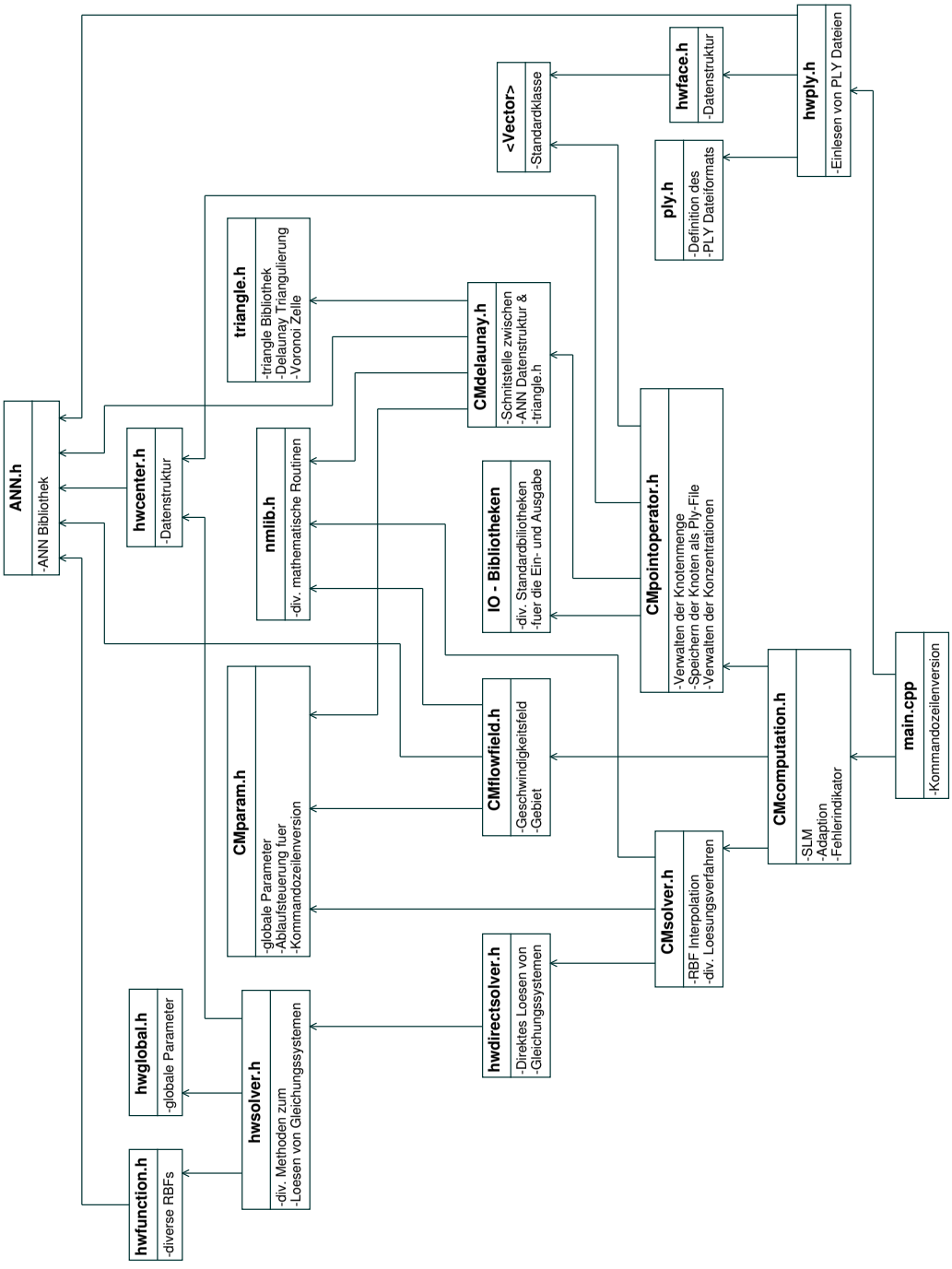


Abb. 87: Klassenstruktur mit Inkludierungspfeilen

ply

Enthält die Definition des *ply*-Dateiformats zur Beschreibung eines Polygonobjekts. (Siehe auch [16].)

Autor: Greg Turk

<vector>

Vector-Standardbibliothek. Stellt den Containertyp *vector* zur Verfügung.

hwface

Die Klasse `hwface` dient zur Speicherung von Polygoninformationen. Jede gespeicherte Fläche enthält Informationen zu der Anzahl der Ecken sowie die Indizes der Eckpunkte.

`hwface(int n)`
Konstruktor. Erzeugt eine Fläche mit n Ecken.

`~hwface`
Destruktor.

`int nverts()`
Liefert die Anzahl der Ecken zurück.

`int vert(int i)`
Gibt Index der `i`-ten Ecke zurück.

`void addVert(int i)`
Setzt Index der `i`-ten Ecke.

Autor: H. Wendland

hwply

Stellt die Datenstrukturen `Vertex` und `Face` zur Verwaltung von Punkt-, und Flächendaten zur Verfügung sowie eine Funktion zum Einlesen von ply-Files.

```
typedef struct Vertex {
```

```
  float x, y, z;
  float nx, ny, nz;
  void *other_props;
}

typedef struct Faces {
  unsigned char nverts;
  int *verts;
  void *other_props;
}
```

Datenstrukturen zur Verwaltung von Punkten und Flächen.

`bool hwreadAPly(const char* name, ANNpoint &lower, ANNpoint &upper, ANNpointArray &data, int &n, Face** &faces, int &nfaces)`
Funktion zum Einlesen des ply-Files `name`. Die Punktdaten der `n` Punkte werden im Array `data` gespeichert, die `nfaces` Flächen in `faces`. `upper` und `lower` beschreiben die *Bounding Box*. Alle Punkte liegen im von `lower` und `upper` aufgespannten Quader. Der Funktionswert von `hwreadAPly` gibt an, ob das Einlesen erfolgreich war.

`void hwdeallocFaces(Face** f,int n)`
Freigeben von Speicherplatz.

Autor: H. Wendland

nmlib

Die Klasse stellt einige Funktionen zum Lösen mathematischer Gleichungen zur Verfügung. Wir verwenden zum Lösen linearer Gleichungssysteme der Form $Ax = b$ mit $A \in \mathbb{R}^{n \times n}$ und x,$b \in \mathbb{R}^n$ die Funktion

`bool LSSolve(const int n, Matrix A, Vector b).`
Der Funktionswert gibt dabei an, ob das Gleichungssystem glöst werden konnte. Der Vektor `b` enthält nach der Berechnung den Lösungsvektor x.

Autor: H. Wendland

triangle

Eine sehr effiziente Möglichkeit der Triangulierung implemetiert Shewchuk in [15]. Wir greifen auf die folgende Funktion in der Version 1.6 zurück:

`void triangulate(char *triswitches, struct triangulateio *in, struct triangulateio *out, struct triangulateio *vorout)`
Der Datentyp `struct triangulateio` enthält eine Vielzahl von Attributen. Uns interessiert an dieser Stelle nur, dass aus einer Punktliste `in` eine Delaunay-Dreicksliste `out` und eine Liste mit Voronoipunkten `vorout` erzeugt werden. `triswitches` teilt der Funktion eine Reihe von Optionen mit.

Autor: J. R. Shewchuk

<IO-Bibliotheken>

Wir verwenden für die Ein- und Ausgabe eine Reihe von Standardbibliotheken, die wir unter `IO-Bibliotheken` zusammenfassen:
`<string>`, `<stdio.h>`, `<stdlib.h>`, `<iostream>`, `<fstream>`.

CMdelaunay

Die Klasse `CMdelaunay` stellt Routinen zur Bestimmung von Delaunay-Triangulierungen und Voronoi-Zellen zur Verfügung.

`CMdelaunay()`
`CMdelaunay(bool voronoi)`
Konstruktor. Der Parameter `voronoi` gibt an, ob auch das Voronoi-Diagramm berechnt werden soll.

`~CMdelaunay()`
Destruktor. Allokierter Speicher wird freigegeben.

`int** getTriangles(int nop, const ANNpointArray &points, int *NOT, int *NOC)`
Liefert die Delaunay-Triangulierung der Knotenmenge `points` zurück. Das Ergebnisarray enthält in der `i`-ten Zeile genau `NOC[i]` Einträge, die den Indizes der Knoten entsprechen, die zum `i`-ten Dreieck gehören. Es existieren `NOT` Dreiecke. Unter regulären Bedingungen, d.h. einer nicht entarteten Knotenmenge `points` gilt stets `NOC[i]` $= 3$.

`ANNpointArray getVoronoi(int nop, const ANNpointArray &points, int *NOV)`
Liefert ein Array mit den zur Knotenmenge `points` gehörigen Voronoipunkten zurück. `nop` gibt die Anzahl der Punkte in der Eingabemenge `points` an. Die Berechnung wird sinnvollerweise nur auf den ersten beiden Raumdimensionen ausgeführt. Die zurückgelieferte Knotenmenge enthält `NOV` Voronoi-Punkte.

`ANNpoint center(const ANNpoint X, const ANNpoint Y, const ANNpoint Z, int dim=2)`
`ANNpoint center(const ANNpoint X, const ANNpoint Y, int dim=2)`
Funktionen zur Berechnung des Mittelpunktes von `X,Y[,Z]`. Bei zwei Punkten wird der Mittelpunkt zwischen `X` und `Y` berechnet, bei drei Punkten der Mittelpunkt des Umkreises des Dreiecks $\triangle$(`X,Y,Z`)

Autor: Chr. Menz

hwcenter

Die Klasse *hwcenter* bietet einige wichtige Routinen zur Verwaltung von Punktmengen und der Suche nach Nachbarschaften mittels der *ANN-Bibliothek* [13].

`hwcenter(int n, int dim, ANNpoint min, ANNpoint max, ANNpointArray data, bool createKDTree)`
Konstruktor. Eine Anzahl `n` von Punkten der Dimension `dim` wird im Array `data` übergeben. Die Boundingbox, die alle Punkte enthält, wird durch `min` und `max` begrenzt. `createKDTree` liefert die Anweisung, ob ein KD-Tree erzeugt werden soll, um die Bestimmung von Nachbarschaften zu ermöglichen.

`~hwcenter()`
Destruktor. Allokierter Speicher wird freigegeben. Auch das dem Konstruktor übergebene Array `data` wird freigegeben.

`ANNpoint point(int i)`
Liefert den Zeiger auf den Punkt mit Index `i`.

`void set_point(int i, ANNpoint X)`
Setzt die Koordinaten des Punktes mit Index `i` auf die Werte von `X`.

`int space_dimension()`

`int number_of_points()`
`ANNpoint max_of_BB()`
`ANNpoint min_of_BB()`
Diese Funktionen liefern Informationen über den aktuellen Zustand der Daten wie Anzahl und Dimension der Punkte sowie über die Boundingbox.

`void annkSearch(ANNidxArray ANNpoint X, int k, ANNidxArray nn_idx, ANNdistArray dd, double eps = 0.0)`
(Approximatives) Ermitteln der `k` nächsten Nachbarn von `X`. Die Indizes werden als `nn_idx` zurückgegeben, `dd[i]` enthält jeweils das Quadrat des euklidischen Abstands zwischen dem Punkt mit Index `nn_idx[i]` und `X`. `eps` stellt eine Fehlertolereanz dar, die wir maximal akzeptieren. Wir verwenden `eps` $= 0.0$

`bool has_tree()`
`void build_kd_tree()`
`void delete_kd_tree()`
Funktionen zum Erzeugen bzw. Löschen des KD-Tree sowie zur Abfrage, ob ein KD-Tree vorhanden ist.

`void print()`
Ausgabe aller Punkt nach stdout.

`double separation_distance()`
Liefert den Separationsabstand $q_{\texttt{data}}$.

Autor: H. Wendland

CMpointOperator

Die Klasse *CMpointOperator* dient zur Verwaltung von Punktmengen, dem einfachen Löschen und Hinzufügen von Punkten. Zudem werden Funktionen zur Durchführung einer Triangulierung sowie zum Ermitteln von Voronoi-Zellen und Nachbarschaften von Punkten bereitgestellt.

`CMpointOperator(int number, int d, ANNpointArray points, double *values, int number_of_neighbors)`
Konstruktor. Es werden `number` `d`-dimensinale Knoten `points` mit zugehörigen Funktionswerten `values` übergeben. `number_of_neighbors` gibt die Anzahl der Punkte pro Nachbarschaft an.

`~CMpointOperator()`
Destruktor. Löschen sämtlicher gespeicherter Daten und freigeben des Speichers.

```
int number_of_points()
int number_of_triangles()
int space_dimension()
```
Zugriff auf die aktuellen Werte für die Anzahl der gespeicherten Punkte, die Anzahl der Dreicke in der Triangulierung der Punkte sowie die Raumdimension.

`ANNpoint point(int i)`
Liefert den Zeiger auf einen einzelnen Knoten mit Index `i`. Der Speicherbereich darf nicht durch die rufende Funktion freigegeben werden.

`ANNpointArray points()`
Liefert ein neues Array mit allen Knoten zurück.

```
double value(int i)
double *values()
```
Liefert den Funktionswert des Knotens mit Index `i` zurück bzw. ein neues Array mit allen Funktionswerten.

`void add_point(const ANNpoint P, const double value)`
Fügt einen Punkt der Liste von Punkten hinzu, die bei Aufruf von `confirm()` der Punktmenge hinzugefügt werden.

`void delete_point(int i)`
Markiert den Punkt mit Index `i` zum Löschen. Der Punkt wird erst durch den Aufruf von `confirm()` tatsächlich gelöscht.

`void set_point(int i, ANNpoint P, double value)`
Verschiebt den Knoten mit Index `i` auf die Koordinaten von `P` sowie den Funktionswert auf `value`. Die Änderung wird mit dem Aufruf von `confirm()` wirksam. Der Index des Punktes ist dabei nicht invariant.

`void confirm()`
Führt alle bisher vorgemerkten Änderungen an der Punktmenge durch. Dabei werden die Indizes der Punkte verändert.

`void set_values(double* newValues)`
Setzt alle Funktionswerte auf die Werte aus `newValues`. Diese Änderung wird sofort durchgeführt. `newValues` wird gelöscht und auf `NULL` gesetzt.

`int** triangles(int* number_of_triangles)`
Liefert die Delaunay-Triangulierung der Knotenmenge als neues Array zurück. Die `number_of_triangles` Zeilen enthalten jeweils drei Indizes von Punkten, die zusammen ein Dreieck bilden. Die Triangulierung wird unabhängig von den Funktionswerten durchgeführt.

`ANNpointArray voronoi(int i, int* number)`
Liefert ein neues Array mit den `number` Vornoipunkten zum Knoten mit Index `i`. Die Ermittlung der Voronoipunkte wird unabhängig von den Funktionswerten durchgeführt. Die Voronoipunkte haben ebenfalls keine Funktionswerte.

`int number_of_neighbors()`
`void set_number_of_neighbors(int val)`
Zugriffsfunktionen für die eingestellte Anzahl von Nachbarn in jeder Nachbarschaft.

`int** neighbors()`
Liefert die kompletten Nachbarschaften aller Knoten. In der `i`-ten Zeile stehen die `number_of_neighbors` Indizes der nächsten Nachbarn von Punkt `i`. Dabei erfolgt eine Sortierung aufsteigend nach dem euklidischen Abstand(Funktionswerte werden ignoriert).

`int* neighbors(int i)`
`int* neighbors(const ANNpoint X)`
`int* neighbors_special(int i)`
`int* neighbors_special(const ANNpoint X)`
Liefert die `number_of_neighbors` Nachbarn von Punkt mit Index `i` bzw. Punkt `X` zurück. In der normalen Variante werden die als zu löschen markierten Punkte mit einbezogen. Die „special“-Version liefert ausschließlich Punkte, die noch nicht zum Löschen markiert sind. Für die Rückgabewerte wird jeweils der benötigte Speicherplatz neu allokiert.

`int* connectedNeighbors(int i, int* number)`
Liefert ein neues Array mit den `number` Knoten, die über die Triangulierung mit Knoten `i` verbunden sind.

`void save_ply(const string filename, const bool with_triangles, const bool with_voronoi = false)`
Speichert die aktuelle Punktmenge mit Funktionswerten in das ply-File `filename`, wahlweise mit Triangulierungsdaten und Voronoipunkten.

`void remove_doubles(int* new_points = NULL)`
Entfernt alle Punkte aus der Punktmenge, deren Abstand kleiner als q_{min} (siehe C.3) ist. Führt danach ein `confirm()` aus.

`double distance(const ANNpoint X, const ANNpoint Y)`
Liefert den euklidischen Abstand (2-dimensional) zwischen `X` und `Y`.

Autor: Chr. Menz

hwglobal.h

Enthält globale Variablen. Die für uns interessanten sind

`MAX(x,y)`
`MIN(x,y)`
Funktionen zur Bestimmung des Maximums bzw. Minimums von `x` und `y`.

`DIST3(x,y,z)`
Länge des Vektors $(x,y,z)^T$.

`PI`
Wert von Π.

`enum HWbasisFunctionType {HWWENDLAND1, HWWENDLAND2, HWWENDLAND3, HWGAUSSIAN, HWMULTIQUADRIC, HWINVERSEMULTIQUADRIC, HWTHINPLATE, HWCUBIC, HWVOLUME}`

`enum HWapproximationType {HWnone, HWrbfDirect, HWrbfPum, HWmls}`

Autor: H. Wendland

hwfunction

Die virtuelle Klasse *hwfunction* definiert die Grundstruktur für die verwendeten Funktionen. Jede abgeleitete Klasse hat mindestens die folgenden Features.

`double at(const ANNpoint &X)`
Liefert den Funktionswert an der Stelle `X`.

`double derivative(const ANNpoint &X, int ord, int j)`
Liefert den Wert der `ord`-ten Ableitung in der `j`-ten Komponente.

`virtual ~hwfunction()`
Destruktor.

Autor: H. Wendland

hwwendland : public hwfunction

Implemetierung der Wendland-Funktionen, basierend auf `hwfunction`.

`hwwendland(int dim=2, int k=1)`
Konstruktor. Es werden die Dimension `dim` und der Parameter `k` für die Verwendung der Wendland-Funktion übergeben.
Die Funktionsgleichungen für die Wendland-Funktionen ergeben sich für die Dimensionen 2 und 3 als

$\Phi_{k=0} = (1-r)^2$ (Wendland-1)
$\Phi_{k=1} = (1-r)^4(4r+1)$ (Wendland-2)
$\Phi_{k=2} = (1-r)^6(35r^2+18r+3)$ (Wendland-3)

`void setSupportRadius(double c)`
`double supportRadius()`
Zugriffsfunktionen für den Support-Radius der Wendland-Funktion. Dabei wird der Radius r skaliert durch $\frac{r}{c}$.

Autor: H. Wendland

hwthinplate : public hwfunction

Implementierung des Thin Plate Spline, basierend auf `hwfunction`.

`hwthinPlate(int dim=2, int k=1)`
Konstruktor. übergeben werden die Dimension `dim` und der Index `k`. Die Funktion liefert $r^k \log(r)$.

Autor: H. Wendland

hwpolynomial : public hwfunction

Implementierung der Monome, basierend auf `hwfunction`.

`hwpolynomial(int dim=2, int index=0)`
Konstruktor. Neben der Dimension `dim` wird der `index` übergeben.

`void set_index(int index)`
Festlegen des Index. Die Zuordnung von `index` zu den verwendeten Monomen entspricht
$\{x^{index}\}_{d=1}$,
$\{x,y,x^2,xy,y^2,x^3,x^2y,xy^2,y^3\}_{d=2}$,
$\{x,y,z,x^2,xy,xz,y^2,yz,z^2,x^3,x^2y,x^2z,xy^2,xyz,xz^2,y^3,y^2z,yz^2,z^3\}_{d=3}$.

Autor: H. Wendland

CMmls_phi : public hwfunction

Implementierung der Gewichtsfunktion für die MLS-Approximation. Die Funktion hat die Form $f(x) = \|x\|_2^{exponent}$.

`CMmls_phi(int dim=2, double support_radius=1.0,`
`bool singular=true, double exponent=1.0)`
Konstruktor. Legt die Dimension `dim`, den Support-Radius `r` und den Exponenten `exponent` fest. `singular` gibt an, ob bei $x = 0$ eine Singularität eingefügt werden soll.

`double at(const ANNpoint &X)`
Liefert den Wert der Gewichtsfunktion an der Stelle `X` zurück.

`inline double derivative(const ANNpoint &X, int ord, int j)`
Dummy. Liefert stets 0 zurück.

Autor: Chr. Menz

hwsolver : public hwfunction

Die virtuelle Klasse *hwsolver* ist von der Klasse *hwfunction* abgeleitet und liefert die Standardfunktionen für die Interpolation von Funktionen.

`hwsolver(hwfunction *p, hwcenter *c, double *f = NULL,`
`int dim = 2)`
Konstruktor. Es werden `dim`-dimensionale Stützstellen `c`, die Interpolationsfunktion `p` und die Funktionswerte `f` in den Stützstellen übergeben.

`bool keepData`
`~hwsolver()`
Destruktor. Löscht die Stützstellen, die Funktion und die Funktionswerte aus dem Speicher. Ist `keepData` gesetzt, so wird lediglich das hwsolver-Objekt gelöscht, die Daten bleiben erhalten.

`void setRightHandSide(double *f)`
`void setCenters(hwcenter *c)`
`void setBasisFunction(hwfunction *p)`
Funktionen zum Setzen der Knoten, des Funktionswertes und der Funktion für die Interpolation.

`virtual void computeSolution()=0`
Virtuelle Funktion zum Durchführen der Interpolation.

Autor: H. Wendland

hwdirectsolver : public hwsolver

`hwdirectsolver(hwfunction *p, hwcenter *c, double *f = NULL,`
`int dim = 2, int method = 0)`
Konstruktor. `p, c, f` und `dim` werden direkt an den Konstruktor von hwsolver übergeben. `method` legt fest, welches Verfahren für die Interpolation verwendet wird (0=ESV, 1=CG).

`~hwdirectsolver()`
Destruktor.

`void computeSolution()`
Durchführen der Interpolation.

`double at(const ANNpoint&X)`
Liefert den Wert der Interpolante an der Stelle `X`.

`double derivative(const ANNpoint &X, int i, int j)`
Liefert den Wert der `i`-ten Ableitung in der `j`-ten Komponente der Interpolante an der Stelle `X`.

`void set_method(int val)`
Festlegen der Interpolationsmethode.

Autor: H. Wendland

CMsolver : public hwsolver

Die Klasse *CMsolver* leitet sich von der Klasse *hwsolver* ab. Sie beherrscht die RBF-Interpolation, die MLS-Approximation sowie das Clipping während der Berechnungen.

`CMsolver(hwfunction *p, hwcenter* c, double *f, int d)`
Konstruktor. Die Parameter werden an den Konstruktor der Klasse *hwsolver* weitergegeben.

`~CMsolver()`
Destruktor.

`void set_trunc(bool val)`
`void set_NONP(int val)`
`void set_solver(CMsolverType val)`
`void set_epsilon(double val)`
Funktionen zum Setzen der Solver-Parameter:

`trunc`	Clipping an/aus
`NONP`	Anzahl der nächsten Nachbarn
`solver`	`RBFdirect` oder `MLS`, siehe C.3
`epsilon`	Für eine bessere Performance werden Unterschiede in den Funktionswerten $<$ `epsilon` ignoriert.

`double at(const ANNpoint &X)`
Interpoliert (RBF) oder approximiert (MLS) auf den nächsten Nachbarn von

X und liefert den Werte der Interpolante bzw. Approximante an der Stelle `X` zurück. Der Wert wird ggfs. durch Clipping angepasst.

`void set_polynomial_degree (int val)`
Setzt der maximalen Grad der bei der Interpolation/Approximation verwendeten Polynome.

`void computeSolution()`
Da wir jeden Wert direkt ermitteln in `at()`, ist diese Funktion lediglich als Dummy implementiert.

`double derivative(const ANNpoint &X, int ord, int j)`
Diese Funktion wird von *hwfunction* gefordert, jedoch hier nicht benötigt. Der Funktionswert ist stets `0`.

`void set_display_truncation(bool val)`
Anzeigen eventuell durchgeführter Clippings in stdout.

Autor: Chr. Menz

CMflowfield

Virtuelle Klasse. Alle abgeleiteten Klassen haben wenigstens die folgenden Features:

`virtual ~CMflowfield()`
Destruktor.

`void set_time(const double val)`
`void set_tau(const double val)`
`void set_omega(const double val)`
Funktionen zum Setzen der aktuellen Zeit t, der Zeitschrittweite τ und der Stärke des Geschwindigkeitsfeldes ω.

`Vector a(const double t, const ANNpoint &X)`
Gibt den Vektor der einwirkenden Kraft im Punkt X zur Zeit t zurück.

`ANNpointArray vertices(int *n)`
Liefert die `n` Ecken des Gebiets, auf dem das Geschwindigkeitsfeld definiert ist, als Array zurück.

`bool isInside(const ANNpoint &X)`
Gibt an, ob der Punkt X sich innerhalb oder auf der Grenze des Gebiets befindet.

`bool onBorder(const ANNpoint &X)`
Gibt an, ob X auf der Grenze des Gebiets liegt.

`double borderValue(const ANNpoint &X)`
Liefert den vordefinierten Funktionswert für X, falls X auf der Grenze liegt, ansonsten 0.

`ANNpoint upstreamPoint(const ANNpoint &X, bool *onBorder)`
Ermittelt die Stelle X^-, an der sich X zur Zeit $t-\tau$ befand. Hier wird je nach Geschwindigkeitsfeld ggfs. auch $\tilde{X}$ bestimmt. Falls der Upstreampunkt außerhalb des Gebiets liegt, wird automatisch nur bis zur Gebietsgrenze gerechnet. X^- liegt dann auf der Grenze, `onBorder` wird gesetzt.

`ANNpoint downstreamPoint(const ANNpoint &X, bool *onBorder)`
Ermittelt die Stelle X^+, an der sich X zur Zeit $t+\tau$ befinden wird. Je nach Geschwindigkeitsfeld kann auch approximiert werden. Falls der Upstreampunkt außerhalb des Gebiets liegt, wird automatisch nur bis zur Gebietsgrenze gerechnet. X^+ liegt dann auf der Grenze, `onBorder` wird gesetzt.

Autor: Chr. Menz

CMflow_square_tangential : public CMflowfield

Zeitunabhängiges Geschwindigkeitsfeld mit quadratischer oder rechteckiger Grundfläche. Es wird eine Drehung um ein Schwerkraftzentrum vollführt. Der Richtungsvektor der einwirkenden Kraft im Punkt $X := (x,y)$ ist bei Drehung um den Ursprung $a(t,X) = \omega(-y,x)^T$ mit einem Skalar ω, der die Drehrichtung und die Stärke des Feldes festlegt.

`CM_flow_square_tangential(const ANNpoint &minBB,`
`const ANNpoint &maxBB, const ANNpoint &centerP = NULL)`
Konstruktor. Die Punkte `minBB` und `maxBB` geben die gegenüberliegenden Ecken des rechteckigen Gebiets vor, in dem das Geschwindigkeitsfeld definiert ist. `centerP` gibt das Drehzentrum an. Ist `centerP = NULL`, so wird automatisch das geometrische Zentrum des Gebiets zum Drehzentrum.

Autor: Chr. Menz

CMflow_square_linear : public CMflowfield

Zeitunabhängiges Geschwindigkeitsfeld mit quadratischer oder rechteckiger Grundfläche. Die einwirkende Kraft a beschreibt auf dem gesamten Gebiet eine homogene Kraft in dieselbe Richtung. Ist der Richtungsvektor der Kraft vorgegeben als $(x,y)^T$, so ist $a(t,X) = \omega(x,y)^T$ für alle Punkte im Gebiet.

`CM_flow_square_tangential(const ANNpoint &minBB,`
`const ANNpoint &maxBB, const ANNpoint &directionP)`
Konstruktor. `minBB` und `maxBB` definieren das Gebiet, `directionP` gibt die Richtung des der einwirkenden Kraft vor.

Autor: Chr. Menz

CMcomputation

Die Klasse `CMcomputation` stellt Routinen zur Durchführung der semi-Lagrange-Advektion zur Verfügung.

`CMcomputation(hwcenter *sc = NULL)`
Konstruktor. Als Parameter wird die initiale Knotenmenge übergeben. Die Daten werden dreidimensional erwartet. Die ersten beiden Koordinaten werden als Ortsinformation eines Knotens betrachtet, die dritte als Funktionswert in diesem Knoten. Die Daten aus `sc` werden umkopiert, der Speicherinhalt bleibt unverändert.

`~CMcomputation()`
Destruktor. Sämtlicher allokierter Speicher wird wieder freigegeben.

`void set_flowfield(CMflowFieldType val)`
Festlegen des Geschwindigkeitsfeldes. (Typen siehe C.3)

`void set_loadedStep(int val)`
Setzen der initialen Schrittnummer. Dient dem automatischen Speichern mit Schrittnummer im Dateinamen.

`void set_saveFile(const string name)`

Festlegen des Dateinamens bei Speicherungen.

`void set_solver(CMsolverType adaptionSolver, CMsolverType timestepSolver)`
Setzen der Methoden, die für die Adaption bzw. den Zeitschritt verwendet werden. (Typen siehe C.3)

`void set_function(CMfunctionType adaptionFunction, CMfunctionType timestepFunction)`
Setzen der Funktion für Adaption bzw. Zeitschritt. Falls das MLS-Verfahren verwendet wird, wird die festgelegte Funktion als Gewichtsfunktion des MLS benutzt. (Typen siehe C.3)

`void set_truncation(bool adaptionTruncation, bool timestepTruncation)`
Ein- oder Ausschalten des Clippings. Es wird unterschieden zwischen den Parametern `adaption`, welcher das lokale Clipping für die Adaption steuert, und `timestep`, der das globale Clipping während eines Zeitschritts festlegt.

`void set_NONP(int adaptionNONP, int timestepNONP)`
Setzt die Anzahl der Knoten in der Nachbarschaft, jeweils ein Wert für die Adaption und für den Zeitschritt.

`void set_epsilon(double val)`
Setzen einer Variable `epsilon`. Diese dient dazu, geringe Unterschiede in Funktionswerten zu ignorieren, so dass in der optischen Darstellung glattere Flächen entstehen.

`void set_numerOfSteps(int val)`
Festlegen der Anzahl der durchzuführenden Zeitschritte beim Aufruf von `compute()`

`void set_2DstartData(hwcenter *sc)`
Setzen der initialen Knotenverteilung und Funktionswerte. `sc` muss die Punktdaten dreidimensional enthalten, so dass die ersten zwei Koordinaten als Ortsangaben, die dritte Koordinate als Funktionswert interpretiert werden können.

`void compute()`
Startet die Berechnung mit den gesetzen Parametern.

`int calculatedSteps()`
Liefert die Anzahl der insgesamt berechneten Zeitschritte zurück.

`ANNpoint point(int n)`
Liefert einen Zeiger auf den Knoten mit Index `n` zurück. Der Speicherbereich darf nicht von der rufenden Funktion freigegeben werden. Auf ein Umkopieren der Werte wurde aus Performancegründen verzichtet.

`double value(int n)`
Liefert den Funktionswert des Knotens mit Index `n` zurück.

`hwcenter* displayCenters()`
Der aktuelle Stand der Berechnung wird als neues hwcenter-Objekt zurückgeliefert. Die Punktdaten sind dreidimensional mit zwei Ortskoordinaten und einem Funktionswert. Außerdem wird eine Triangulierung zur besseren optischen Darstellung mit übergeben.

`void displayTriangles()`
`void displayNeighbors(int i)`
`void displayConnectedNeighbors(int i)`
`void displayVoronoi(int i)`
Funktionen zur Ausgabe von Detailinformationen nach stdout. Diese in erster Linie zu Debugging-Zwecken implementierten Funktionen stellen die aktuelle Triangulierung aller Knoten bzw. zu einem Knoten mit Index `i` die nächsten Nachbarn, die über die Triangulierung mit dem Knoten verbundenen Nachbarn oder die Ecken der Voronoizelle dar.

`void save_ply(const string filename, const bool with_triangles)`
Der aktuelle Stand der Berechnungen wird in die Datei `filename` geschrieben. Die Triangulierung wird je nach `with_triangles` mit gespeichert.

`void auto_save_ply(int number)`
Funktion, um in jedem Schritt die aktuelle Knotenmenge mit Funktionswerten in ein Ply-File zu speichern. Der Dateiname wird automatisch mit Parameterinformationen sowie der aktuellen Schrittnummer `number` erzeugt. Die Triangulierung wird automatisch mitgespeichert.

`void save_knotset()`
Es wird nur die Knotenmenge gespeichert, alle Funktionswerte werden auf `0` gesetzt.

`double distance(const ANNpoint X, const ANNpoint Y,`
`int dim = 2)`
Berechnet den euklidischen Abstand zwischen `X` und `Y` in `dim` Dimensionen.

`hwfunction* gethwfunction(CMfunctionType val)`
Liefert den Zeiger auf ein neues Objekt des übergebenen Funktionstyps zuück. (Typen siehe C.3)

Autor: Chr. Menz

main.cpp

Hauptmodul. Die Funktion `main()` übernimmt das Auswerten der übergebenen Kommandozeilenparameter sowie den Aufruf der eigentlichen Rechenroutinen.

`main(int argc, char** argv)`
Die Funktion `main()` akzeptiert als Kommandozeilenparameter den Dateinamen des einzulesenden ply-Files sowie eine initiale Schrittnummer. Wird kein Parameter angegeben, so werden die in *CMparam.h* (Siehe Kapitel C.3) angegebenen Werte verwendet. Die Angabe der Schrittnummer dient zum Einfügen der Schrittnummer in den Dateinamen beim automatischen Speichern.

Autor: Chr. Menz

C.3 Parameter

Die Datei *CMparam.h* ist die zentrale Steuerdatei für den Programmablauf. Sämtliche globalen Parameter werden hier gesetzt Außerdem enthält die Datei Enum-Deklarationen und einfache Funktionen. Es folgt eine Auflistung aller implementierten Parameter mit einer Beschreibung ihrer Funktion. Sofern ein standardmäßig verwendeter Wert existiert, werden wir diesen in der rechten Spalte angeben.

Enum-Deklarationen
`enum CMsolverType {RBFdirect, MLS}`
`enum CMfunctionType {gauss, wendland, thinPlate, cm_mls_phi}`
`enum CMflowFieldType {square_tangential, square_linear}`

Funktionen

```
#define MAX(x,y) ((x) > (y) ? (x) : (y))
#define MIN(x,y) ((x) < (y) ? (x) : (y))
#define ABS(x) MAX(x,(-x))
#define DIST2(x,y) sqrt((x)*(x)+(y)*(y))
#define DIST3(x,y,z) sqrt((x)*(x)+(y)*(y)+(z)*(z))
```

Parameter

Parameter für die Advektion		
NUMBER_OF_STEPS (integer)	Anzahl der insgesamt durchzuführenden Zeitschritte.	
DO_TIMESTEP (boolean)	Legt fest, ob die Advektion in jedem Schritt durchgeführt wird.	
TS_SOLVER (CMsolverType)	Definiert den Solver-Typ.	
TS_FUNCTION (CMfunctionType)	Definiert die Funktion für die Interpolation bzw. Approximation.	
TS_TRUNC_ENABLED (boolean)	Aktiviert oder deaktiviert das Clipping.	`true`
TS_NONP (integer)	Setzt die Anzahl der Punkte in den Nachbarschaften.	
TS_POLY_DEG (integer)	Polynomgrad TPD:= $Q-1$. TPD= 0: Konstanten, TPD= 1: Lineare Polynome.	`1`
REFINE_WITH_ TIMESTEP_SOLVER (boolean)	Legt fest, ob die Solver-Einstellungen des TS-Solvers auch für die Verfeinerung gelten sollen. Ansonsten (`false`) gelten die Einstellungen des Adaptions-Solvers.	`false`

Parameter für die Adaption		
DO_ADAPTION (boolean)	Legt fest, ob Adaptionen durchgeführt werden.	
ADAPT_SOLVER (CMsolverType)	Definiert den Solver-Typ.	
ADAPT_FUNCTION (CMfunctionType)	Definiert die Funktion für die Interpolation bzw. Approximation.	
ADAPT_TRUNC_ENABLED (boolean)	Aktiviert oder deaktiviert das Clipping.	`true`
ADAPT_NONP (integer)	Setzt die Anzahl der Punkte in den Nachbarschaften.	
SIGMA_REF (double)	Verfeinerungsparameter σ_{ref}.	
SIGMA_CRS (double)	Ausdünnparameter σ_{crs}.	

SIGMA_REF_STEPS (integer) SIGMA_CRS_STEPS (integer)	Anzahl der Schritte beim automatischen Anpassen von σ_{ref} und σ_{crs}. Dient zur Automatisierung von Testläufen mit unterschiedlichen Parametern. Nach Abschluß einer Testreihe wird mit veränderten Parametern neu gestartet. Es werden insgesamt SIGMA_REF_STEPS · SIGMA_CRS_STEPS Testläufe ausgeführt.	1
SIGMA_REF_STEPSIZE (double) SIGMA_CRS_STEPSIZE (double)	Wert, um den σ_{ref} bzw. σ_{crs} bei jedem Schritt verändert wird.	0.0
ITERATIONS (integer)	Legt die maximale Anzahl von Adaptionen pro Zeitschritt fest (IT_{max}).	
MIN_ITERATIONS (integer)	Legt die minimale Anzahl von Adaptionen pro Zeitschritt fest (IT_{min}).	
MAX_ERROR_TO_STOP (double)	Unterschreitet der maximale Fehler bei einer Adaption diesen Wert, so gilt die Adaption als stationär (σ_{stat})	
ERROR_STATIONARY (double)	Ist die Differenz zweier maximaler Fehler aus zwei nachfolgenden Adaptionsschritten geringer als dieser Wert, so gilt die Adaption als statinär (σ_{diff}).	
PERCENTAGE_ STATIONARY (double)	Ist der Anteil gelöschter und eingefügter Punkte bei einer Adaption im Verhältnis zur Gesamtpunktzahl unter diesem Wert, so gilt die Adaption als statinär.	0.0
REFINE_MIN (integer)	Werden weniger Punkte als REFINE_MIN verfeinert, so gilt die Adaption als statinär.	0
MAX_TO_DELETE (integer)	Maximale Anzahl von Punkten, die in einem Adaptionsschritt gelöscht werden dürfen.	10^7
USE_GLOBAL_MAX_ERROR (boolean)	true: Der maximale Fehler des ersten Adaptionsschrittes wird verwendet. false: Für jeden Adaptionsschritt wird der maximale Fehler separat bestimmt.	false
INTERPOLATE_WITH_ DELETED_POINTS (boolean)	true: Die gelöschten Punkte werden bis zum Ende des Adaptionsschrittes weiterhin für die Interpolation / Approximation verwendet.	true
VORONOI_WITH_ DELETED_POINTS (boolean)	true: Die gelöschten Punkte gehen bis zum Ende des Adaptionsschrittes in die Ermittlung der Voronoizelle ein.	true

MAX_VORO_DIST (double)	Legt die Entfernung fest, die ein einzufügender Voronoipunkt maximal zum verfeinerten Punkt haben darf. Weiter entfernte Voronoipunkte werden nicht eingefügt.	DBL_MAX
ADAPT_WITH_POINT (boolean)	Nachbarschaften werden immer mit dem Punkt selber gebildet. Dies dient zum Debuggen. Der lokale Fehler wird dadurch minimiert.	false
ADAPT_POLY_DEG (integer)	Polynomgrad APD:= $Q-1$. APD= 0: Konstanten, APD= 1: Lineare Polynome.	1
ADD_BOUNDARY (boolean)	Legt fest, ob nach jedem Adaptionsschritt die Ecken von Ω zur Knotenmenge hinzugefügt werden sollen.	false

Duplikate löschen		
DO_REMOVE_DOUBLES (boolean)	Legt fest, ob nach jedem Adaptionsschritt doppelte Punkte entfernt werden sollen.	true
DO_REMOVE_DOUBLES_BEFORE (boolean)	Prüft bereits vor dem Einfügen neuer Punkte auf Duplikate. Bei zu geringem Abstand zu einem vorhandenen Knoten wird nicht eigefügt. Problematisch, da möglicherweise der bereits vorhandene Knoten gelöscht wird.	false
MIN_SEP_DST (double)	Legt den Mindestabstand (euklidisch) zwischen verschiedenen Punkten fest. Liegen zwei Punkte dichter zusammen, gelten sie als derselbe Punkt. Dies verhindert das Entarten der Knotenmenge.	

Solver- & Funktions-Parameter		
EPSILON (double)	Liegen zwei Funktionswerte weniger als EPSILON auseinander, werden sie als gleich betrachtet. Erhöht die Performance und verringert Oszillationen.	0.0
RHO (double)	Dieser Wert wird bei der RBF Interpolation auf die Elemente der Hauptdiagonalen der Matrix addiert, um das LGS zu stabilisieren.	10^{-5}
RBF_SUPPORT_RADIUS (double)	Support Radius der gewählten RBF.	1

MLS_USE_FIXED_DELTA (boolean)	Legt fest, ob der Radius beim MLS einen festen Wert MLS_SUPPORT_DELTA hat oder dynamisch in Abhängigkeit von der ermittelten Nachbarschaft gebildet wird.	false
MLS_SUPPORT_DELTA (double)	Setzt den fixen Radius für das MLS-Verfahren, wenn MLS_USE_FIXED_DELTA gesetzt ist.	0.1
MLS_NONP_MAX (integer)	Legt die Anzahl von Knoten fest, die maximal bei fixem delta verwendet wird.	100
MLS_SINGULAR (boolean)	Legt fest, ob eine Singularität in die Gewichtsfunktion eingefügt werden soll.	true
MLS_EXPONENT (integer)	Legt den Exponenten der Gewichtsfunktion bei der MLS-Approximation fest.	-2
TPS_K (integer)	Legt den Index des Thin Plate Spline fest.	1
WEND_K (integer)	Legt den Index der Wendland-Funktion fest.	1

Geschwindigkeitsfeld		
FLOW_FIELD (CMflowField)	Bestimmt den Typ des Geschwindigkeitsfeldes.	
VELOCITY_CENTER_X (double) VELOCITY_CENTER_Y (double)	Bei tangentialem Geschwindigkeitsfeld die Koordinaten den Mittelpunkts, bei linearem GF die Richtung.	$X = 0$ $Y = 0$
OMEGA (double)	Bei tangentialem GF die Winkelgeschwindigkeit, bei linearem GF die Länge des Geschwindigkeitvektors.	$3636 \cdot 10^{-8}$
TAU (double)	Zeitschrittweite τ	
ITERATION_DEPTH (integer)	Anzahl N_{it} der Iterationen für die Bestimmung des Upstreampunktes	5

Dateien laden & speichern		
DEFAULT_PLY_FILE PLY_FILE_FOLDER (string)	Dateiname des zu ladenden Plyfiles. Wenn DPF Pfadangaben enthält, muss PFF leer sein! Wird PFF gesetzt, so darf der Dateiname nur ohne Pfadinformationen gesetzt werden. Das Plyfile wird dann aus dem Ordner des Programms gelesen.	
INITIAL_STEP_NUMBER (integer)	Schrittnummer des geladenen Plyfiles. Dient der Generierung von Dateinamen bei automatischen Speichern.	0

TS_FUNCTION_NAME ADAPT_FUNCTION_NAME (string)	Bezeichner der Adaptions- bzw. Zeitschrittfunktion. Werden für die Generierung des Dateinamens beim automatischen Speichern verwendet.	
SAVE_EVERY_STEP (boolean)	Speichert nach jedem Zeitschritt automatisch ein Plyfile.	true
EVERY_NTH_STEP (integer)	Speichert nur jeden ENS-ten Schritt ab.	
SAVE_TRIANGLES (boolean)	Speichert Triangulierungsinformationen in jedes Plyfile.	true
SAVE_VORONOI (boolean)	Speichert alle Voronoipunkte mit in jedes Plyfile. Die Voronoipunkte werden nicht mit in die Triangulierung einbezogen.	false
APPEND_STEP_NUMBER (boolean)	Hängt an den Dateinamen der automatisch gespeicherten Plyfiles die Schrittnummer an. Verhindert das Überschreiben der Dateien.	true
SAVE_ERROR_ EVERY_STEP (boolean)	Speichert nach jedem Zeitschritt ein Plyfile mit der aktuellen Knotenmenge und dem aktuellen Fehler in jedem Knoten als Funktionswert. Dateiname wird automatisch generiert.	false
SAVE_KNOTSET_ EVERY_STEP (boolean)	Speichert nach jedem Zeitschritt die Knotenmenge. Die Funktionswerte werden auf 0 gesetzt.	false

Programm-Ausgaben		
DISPLAY_PARAMETER_ SETTINGS (boolean)	Zeigt alle eingestellten Parameterwerte vor der Berechnung einmal an.	true
DISPLAY_REFINEMENT_ TRUNCATION (boolean)	Zeigt Informationen zum Clipping bei der Adaption an.	false
DISPLAY_SAVEFILE_ INFO (boolean)	Zeigt Informationen über die gelesenen und geschriebenen Dateien.	false
DISPLAY_STATUS (boolean)	Zeigt während der Berechnungen den aktuellen Status.	true
DISPLAY_INFO (boolean)	Zeigt Informationen über die Ergebnisse der aktuellen Berechnungen.	true
DISPLAY_STATIONARY (boolean)	Zeigt eine Adaptionsstatistik an, sobald die Adaption stationär wird.	true
DISPLAY_ERROR (boolean)	Zeigt bei jedem Adaptionsschritt Informationen zur Fehlerabschätzung an.	true

DO_ERR_STAT (boolean)	Führt eine erweiterte Fehlerbetrachtung und -statistik durch.	false
DISPLAY_ERROR_STAT (boolean)	Ausgabe der Fehlerstatistik, wenn DO_ERR_STAT gesetzt ist.	false
ES_STEPS_TO_OBSERVE (integer)	Gibt an, wie viele Zeitschritte maximal erfasst werden sollen (Speichermanagement).	10^4
DISPLAY_MEMORY_ ALLOCATION (boolean)	Liefert eine Ausgabe, sobald Speicher allokiert wird.	false

Globale Konstanten		
NOP_TO_EXIT (integer)	Wird diese Anzahl an Knoten überschritten, wird das Programm sofort beendet. Verhindert das Vollschreiben der Festplatte bei langen Testreihen.	10^6
DBL_EPS (double)	Wert für den kleinsten unterscheidbaren Betrag.	10^{-8}
NUMBER_OF_DOUBLES (integer)	Angenommene maximale Anzahl von Knoten, die in einer MIN_SEP_DIST-Kugel liegen. Wird für Speichermanagement benötigt.	10
MAX_MEMBERSHIPS (integer)	Angenommene maximale Anzahl von Dreicken, zu denen ein Knoten in der Triangulierung gehört. Wird für Speichermanagement benötigt.	50

D Einbindung & Darstellung

„Die Wahrheit hat tausend Hindernisse zu überwinden, um unbeschädigt zu Papier zu kommen und von Papier wieder zu Kopf.“
Georg Christoph Lichtenberg (1742 - 1799)

Sämtliche von uns durchgeführten Berechnungen lieferten die Daten im *Polygon File Format (ply)*. Dieses Dateiformat dient der einfachen Darstellung von Punktmengen und Flächen, wie in D.3 beschrieben. Für die grafische Darstellung der erzeugten *ply*-Dateien wurde der *RBF Surface Creator* von H. Wendland verwendet. Dabei diente das unten beschriebene Tool *slm_smoothing* ([21]) zur Verbesserung der optischen Darstellung der Interpolante bzw. Approximante.
Um die Datenmengen für diese Arbeit abzubilden, wurde das Programm *Paraview* in der Version 2.6.0 ([17]) verwendet. Dazu wurden die abzubildenden *ply*-Dateien in das von Paraview verwendete *vtk*-Format konvertiert, auf das wir noch zurückkommen werden. Für die Konvertierung wurde das Kommandozeilentool *ply2vtk* ([22]) verwendet.

D.1 slm_smoothing

Um die Interpolante bzw. Approximante dreidimensional darzustellen, wird ein regelmäiges Gitter aus Dreiecken verwendet. Dazu wird für 40.000 gleichmäßig über die Bounding Box verteilte Knoten zunächst die Delaunay-Triangulierung bestimmt. Danach wird für jeden Knoten der Wert der Interpolante bzw. Approximante berechnet. Die Basisfunktion sowie die Größe der Nachbarschaften entsprechen dabei jeweils den Parametern der zugehörigen Testreihe.
Die Eingabe

```
smooth meinfile.ply
```

bewirkt die Konvertierung der Datei
`meinfile.ply` nach `meinfile_smooth.ply`.

Die für diese Arbeit verwendete Programmversion befindet sich quelloffen auf dem beiliegenden Datenträger.

Autor: Chr. Menz

D.2 ply2vtk

Das Tool *ply2vtk* dient der Konvertierung des ply-Dateiformats in das Format vtk. Die Konvertierung wird über die Kommandozeile gestartet, wobei die zu konvertierende Datei als Parameter übergeben wird.
Dabei ist zu beachten, dass die Dateiendung *ply* nicht mit angegeben wird.
Der Befehl

```
ply2vtk meinfile
```

bewirkt die Konvertierung der Datei
`meinfile.ply` nach `meinfile.vtk`.
Dateikommentare werden nicht mit übertragen.
Die für diese Arbeit verwendete Programmversion befindet sich quelloffen auf dem beiliegenden Datenträger.

Autor: Chr. Menz

D.3 Dateiformat *ply*

Eine genaue Beschreibung des ply-Dateiformats wird in [16] gegeben.
Für die in dieser Arbeit durchgeführten Betrachtungen wurde folgender Dateiaufbau verwendet:

Definition der Datei als *ply*:

```
ply
```

Definition des Zahlenformats:

```
format ascii 1.0
```

Kommentare:

```
comment Meine Ply-Datei meinfile.ply
comment Das ist eine Pyramide!
```

Festlegen der Anzahl Punkte:

```
element vertex 8
```

Definitionen, welche Werte pro Punkt angegeben werden:

```
property float32 x
property float32 y
property float32 z
property float32 confidence
property float32 intensity
```

Anzahl der Flächen angeben und welche Werte pro Fläche angegeben werden:

```
element face 6
property list uint8 int32 vertex_indices
```

Ende der Definitionen:
```
end_header
```
Punktliste:
```
1.00000 1.00000 0.0000 1.0000 0.50000
1.00000 0.00000 0.0000 1.0000 0.50000
0.00000 0.00000 0.0000 1.0000 0.50000
0.00000 1.00000 0.0000 1.0000 0.50000
0.50000 0.50000 1.0000 1.0000 0.50000
```
Flächenliste. Die erste Zahl gibt die Anzahl der Ecken an, danach folgen die Indizes der Ecken:
```
3 1 0 2
3 2 0 3
3 1 4 0
3 1 2 4
3 2 3 4
3 3 0 4
```

D.4 Dateiformat *vtk*

Die genaue Definition des vtk-Formats wird in [18] gegeben.
Für die Visualisierung der Dateien zur Druckausgabe wurde der folgende Dateiaufbau verwendet:

Angabe des Dateiformats mit Version:
```
# vtk DataFile Version 2.0
```
Eine Kommentarzeile, maximal 255 Zeichen:
```
meinfile.vtk, das ist eine Pyramide
```
Angabe, ob die Daten als ASCII oder binär vorliegen:
```
ASCII
```
Festlegen des Datentyps *Polygondaten*:
```
DATASET POLYDATA
```
Anzahl der Eckpunkte angeben und Dateiformat. Die Punkte werden immer dreidimensional erwartet:
```
POINTS 5 float
```
Punktliste:
```
1.00000 1.00000 0.0000
1.00000 0.00000 0.0000
0.00000 0.00000 0.0000
0.00000 1.00000 0.0000
0.50000 0.50000 1.0000
```

Definition der Anzahl (6) der Flächen und Gesamtzahl an Werten (24).
Dabei steht als erste Zahl die Anzahl der folgenden Ecken, danach die Indizes der Ecken:

```
POLYGONS 6 24
3 1 0 2
3 2 0 3
3 1 4 0
3 1 2 4
3 2 3 4
3 3 0 4
```

Zusätzliche optionale Werte zu den einzelnen Punkten.
Wir verwenden die Datenreihe *Farbwerte* für die Farbgebung, die Reihe hat *float*-Zahlenformat und enthält je einen Wert pro Punkt:

```
POINT_DATA 6
SCALARS Farbwerte float 1
```

Verwendung der standardmäßigen Farbtafel des darstellenden Programms:

```
LOOKUP_TABLE default
0.00000
0.00000
0.00000
0.00000
0.00000
1.00000
```

Datenträger

Auf dem dieser Arbeit beiliegenden Datenträger befinden sich folgende Inhalte:

- Eine Kopie dieser Arbeit im PDF-Format
  `/documents/diplom-menz_christian.ps`
- Eine Kopie dieser Arbeit im PS-Format
  `/documents/diplom-menz_christian.pdf`
- ([13]) ANN-Bibliothek in der Version 1.1.1
  `/program/ann_1.1.1/`
- ([14]) ANN-Manual im PDF-Format
  `/program/ann_1.1.1/doc/ANNmanual.pdf`
- ([20]) Die Quellcodes des Programms *slm*
  `/program/slm/`
- ([21]) Die Quellcodes des Programms *slm_smoothing*
  `/program/slm_smoothing/`
- ([22]) Die Quellcodes des Programms *ply2vtk*
  `/program/ply2vtk/`
- Anleitung zum Kompilieren der Programme
  `/program/documents/README`

Die auf dem Datenträger befindlichen Quellcodes sowie die vorliegende Arbeit sind im Internet unter folgenden Adressen zum Download bereitgestellt:

- Enthaltender Ordner:
  `ftp://ftp.num.math.uni-goettingen.de/pub/dipl/`
- Diplomarbeit im PDF-Format: `diplom-menz_christian.ps.gz`
- Diplomarbeit im PS-Format: `diplom-menz_christian.pdf.zip`
- Quellcodes: `diplom-menz_christian_progsrc.zip`

Kontakt

Christian Menz
`menz@math.uni-goettingen.de`