Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

DIPLOMARBEIT

Optimierung eines Discrete-Particle-Modells (DPM) zur Strömungssimulation von luftunterstützten Sprays

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät: **Maschinenbau und Produktion**
Studiengang: **Maschinenbau**

Autor: Gilberth Sitohang
Mat.-Nr.: 1811262
Erster Prüfer: Prof. Dr.-Ing. Peter Wulf
Zweiter Prüfer : M.Eng. Dipl-Ing. Patrick Diffo

Ausgabedatum: 27.07.2011
Abgabedatum: 29.11.2011

Vorwort

An dieser Stelle möchte ich mich bei Herrn **Prof. Dr.-Ing. Peter Wulf** und **M.Eng. Dipl-Ing. Patrick Diffo** für die tatkräftige Unterstützung und Betreuung meiner Diplomarbeit bedanken.

Zum Abschluss meines Studiums möchte ich **meinen Eltern** danken, die mich nicht nur finanziell, sondern auch moralisch immer unterstützt und mir den Rücken gestärkt haben. Ebenso möchte ich mich bei ***Frau* Sabrina Kayo** herzlich bedanken für die gemeinsame Zeit in Deutschland, von Köln, Merzenich bis Hamburg, Du warst und bist sehr wichtig in meinem Leben. Des Weiteren möchte ich diese Arbeit ganz speziell auch ***Frau* Mervy Sindy Silitonga** widmen, die mich seit meinem Hauptpraktikum in Hamburg Winter 2010 und während der restlichen Prüfungen bis zur Diplomarbeit und meinem Abschluss ständig begleitet und mir Kraft gegeben hat.

Hamburg, November 2011
Gilberth Sitohang

Abbildungsverzeichnis

Tabellenverzeichnis

Formelzeichen

V	Volumen (m³)
ρ	Dichte (kg/m³)
$\vec{u}$	Geschwindigkeitsvektor (m/s)
t	Zeit (s)
$S_{\emptyset}$	Quellterm von ∅ pro Einheitsvolumen
S_m	Massenquelle
A_V	Oberfläche des Volumens (m²)
$\vec{f}$	Ein- und austretende Flüsse
$\vec{n}$	Normalenvektor
a_P	Linearisierte Koeffizient
a_{nb}	Linearisierte Koeffizient für Nachbarzelle
p	Statischer Druck (Pa)
$\bar{\bar{\tau}}$	Spannungstensor (N/m²)
$\vec{g}$	Gravitation (m/s²)
$\vec{F}$	Kraft (N)
u_P	Partikelgeschwindigkeit (m/s)
μ	Dynamische Viskosität (cP , $\mathrm{Pa \cdot s}$)
C_D	Wiederstandbeiwert
α	Halbwinkel des Luftstrahls

Inhaltsverzeichnis

1 Einleitung

1.1 Hintergrund zum Projekt und Schilderung der Problematik

Für die Herstellung ein- oder mehrschichtiger Faserverbundwerkstoffe auf Polyurethanbasis wird seit einigen Jahren das Sprühverfahren angewendet. Das flüssige und reaktive Polyurethan (PUR) wird auf ein Substrat oder in eine Werkzeugform gesprüht. Während des Sprühvorgangs wird das flüssige Polyurethan mit den zusätzlich eingebrachten Fasern vermischt und anschließend zu einem Verbund ausgehärtet. Um die Simulationskosten zu reduzieren wird auf die Simulation des Flüssigkeitszerfalls des PUR in der Spraydüse verzichtet. Die Simulation des Sprühvorgangs wird mit CFD Ansys Fluent modelliert.

Die Sprays sollen folgende Eigenschaften haben:

- Einen definierten Öffnungswinkel des Strahls,
- eine Axialsymmetrie,
- die Luftgeschwindigkeit- und Verteilung in radialer und axialer Sprühstrahl-richtung sowie
- der Tropfen- und Durchmesserverteilung in radialer und axialer Richtung.

Abbildung 1: PUR-CSM Verfahren (© Hennecke GmbH)

1.2 Ziele dieser Diplomarbeit

Ziel dieser Arbeit ist es ein Modell zu entwickeln und mit theoretischen sowie experimentellen Daten im Hinblick auf die Sprayeigenschaften zu untersuchen. Das Sprühluftmodell wird mit Hilfe von Quelltermen aus den spezifisch ausgesuchten Zellen erzeugt. Das Ziel ist dabei ein kegelförmiger Luftstrahl in dem UDF-Programm von Ansys Fluent zu erzeugen.

UDF (User-Defined Function) ist ein vom Benutzer selbst geschriebenes Programm, das in Ansys Fluent hinzugefügt wird. Das UDF-Programm wird benutzt um spezifische Modellparameter wie z.B. die Massenquelle, Impulsquelle, spezifische Randbedingungen, Injektion der Partikel und Materialeigenschaften nach Wunsch des Benutzers in der Simulation zu definieren und verändern. Der Luftstrahl soll außerdem einen symmetrischen Öffnungswinkel (in 2D und 3D) haben. Der Öffnungswinkel lässt sich je nach Anforderungen in der UDF manuell definieren und verändern, ebenso auch die Austrittgeschwindigkeitskomponenten des Luftstrahls aus den Zellen.

Während der Simulation soll sich der Luftstrahl dann in dem Berechnungsgebiet translatorisch mit der vorgegebenen Bewegungsgleichung bewegen. Die Bewegungsgeschwindigkeit des Strahls ist angenähert an die Geschwindigkeit der Roboterdüse bei etwa 0,4 - 0,8 m/s in der Praxis. Neben dem Luftstrahl sind die Partikel mit Hilfe von DPM-Modellen (Discrete-Particle Model) von Ansys Fluent zu simulieren. Die Partikel repräsentieren das flüssige Polyurethan (PUR), das dem Spray zugeführt wird. Die Partikelanzahl und die Verteilung der Partikeldurchmesser sollen der Gleichung von Rosin-Rammler folgen. Die Ergebnisse der Simulation sind anschließend mit der Freistrahltheorie und dem Partikelverteilungsgesetzt nach Rosin-Rammler zu validieren. Für die endgültige Simulation werden der Luftstrahl und die Partikelsimulation anschließend gekoppelt und berechnet. Die Berechnung erfolgt dann im instationären Zustand. Das Modell soll sich dann in dem Berechnungsgebiet nach vorgegebener Bahn bewegen.

1.3 Aufbau der Arbeit

Diese Arbeit wird in zwei Hauptaufgaben eingeteilt: Die Modellierung von Sprühluft und Partikel. Die Sprühluft wird zuerst an verschiedenen Zellkombinationen und unterschiedlichen Zellengrößen untersucht. Aus dieser Zellenkombination entsteht eine künstliche Düse für den Luftstrahl, auch als „virtual Nozzle“ genannt. Es ist zu ermitteln, welche von allen Modellen die besten Sprayeigenschaften besitzt. Die Ergebnisse des Luftstrahls aus verschiedenen Zellkombinationen werden dargestellt, miteinander verglichen und beurteilt. Die Variante, mit den besten Sprayeigenschaften, wird dann als Modell für die Simulation verwendet.

Zur Partikelsimulation wird das DPM-Modell von Ansys Fluent verwendet. Die Simulation läuft zunächst im stationären Zustand. Es ist zu prüfen ob die Partikel symmetrisch verteilt sind und die Form des gesamten Partikelstrahls kegelförmig ist. Danach wird die Partikelsimulation zusammen mit dem Luftstrahlmodell gekoppelt und im instationären Zustand berechnet.

2 Theoretische Grundlagen

In der Strömungsmechanik gibt es verschiedene Vorgehensweisen zur Lösung von strömungstechnischen Problemen. Experimentelle Methoden benötigen oft umfangreiche Versuchsaufbauten, die sehr kosten- und zeitintensiv sein können. Sie werden mehr und mehr durch theoretische Methoden ersetzt [4].

Ziel solcher theoretischen Methoden ist die Lösung der Erhaltungsgleichungen für Masse, Impuls und Energie, die die Bewegung eines strömenden Fluids beschrieben. Analytische Methoden erlauben es nur in wenigen Fällen, meist bei einfachen Geometrien, eine Lösung zu finden. Lösungen sind dann meist möglich zu finden, wenn einzelne Terme der zu lösenden Gleichungen gleich Null sind. Für andere Strömungssituationen können einzelne Terme vernachlässigt werden. Diese Vereinfachung der Gleichungen führt zu einem Fehler und selbst die vereinfachten Gleichungen sind nur in wenigen Fällen analytisch lösbar. Die Untersuchung solcher Strömungen ist wichtig, um die Grundlagen der Strömungsdynamik zu erforschen. Ihre Bedeutung für die Praxis ist allerdings begrenzt [5]. Die nummerische Strömungsmechanik basiert auf den Grundgleichungen der Erhaltungsprinzipien. Sie sind die Massenerhaltung, Impulserhaltung und Energieerhaltung. Weitere Zusatzgleichungen wie z.B. der Ansatz der Turbulenzmodelle $k-\varepsilon$ oder $k-\omega$ für die Simulation können eingebaut werden. Für dieses Luftstrahlmodell wird die Energiegleichung in der Simulation mit Ansys Fluent nicht benötigt. Denn es sind nur

die Massen- und Impulsquelle im UDF-Programm definiert. Daher wird sich hier nur auf die Massenerhaltung und Impulserhaltung konzentriert.

2.1 Numerische Methoden

In dieser Diplomarbeit wird die Berechnungssoftware Ansys Fluent v.13 Linux verwendet. Es basiert auf der Finite-Volumen-Methode. Das Berechnungsgebiet wird in diskrete Zellen, sogenannte „Kontrollvolumen“ aufgeteilt. Die Erhaltungsgleichungen können jetzt für jede Zelle einzeln betrachtet werden. Das Gleichungssystem entsteht aus der Diskretisierung und Linearisierung der Erhaltungsgleichungen. Das Gleichungssystem wird dann vom Solver Ansys Fluent gelöst.

2.1.1 Diskretisierung und Linearisierung

Das Berechnungsgebiet wird mit Hilfe eines Rechengitters diskretisiert. Die Transportgleichung muss für jede Zelle erfüllt sein und werden in integraler Form diskretisiert [4].

Die allgemeine Transportgleichung für eine beliebige Größe ∅ (gesuchte unbekannte Feldgröße) lautet:

$$\frac{\partial(\rho\emptyset)}{\partial t} + \nabla(\rho\emptyset\vec{u}) = \nabla(\Gamma\nabla\emptyset) + S_{\emptyset}$$

Die Transportgleichung kann über ein Kontrollvolumen integriert werden:

$$\int_{\Delta V} \frac{\partial(\rho\emptyset)}{\partial t} dV + \int_{\Delta V} \nabla(\rho\emptyset\vec{u}) dV = \int_{\Delta V} \nabla(\Gamma\nabla\emptyset)\, dV + \int_{\Delta V} S_{\emptyset}\, dV$$

Die Volumenintegrale werden zunächst in Flächenintegralen mit Hilfe des Gaußschen Divergenzsatz $\int_V \nabla\vec{f} dV = \oint_{A_V}(\vec{f}\cdot\vec{n})dA$ überführt. Die Divergenz eines Vektorfelds in einem Volumen entspricht den über die Oberflächen des Volumens ein- und austretenden Flüsse $(\vec{f}\cdot\vec{n})$ des Vektors [1]. Die Diskretisierung der Transportgleichungen findet in algebraischer Form statt. Die Flächenintegrale können näherungsweise gelöst werden. Aus dem Integral wird eine Summe über die

einzelnen Flächen einer Zelle (Index k) gebildet. Für den stationären Zustand lautet die Gleichung:

$$\sum_k \rho \emptyset_k (\vec{u}_k \cdot \vec{n}_k) A_k = \sum_k \Gamma((\nabla \emptyset)_k \cdot \vec{n}_k)) A_k + S_\emptyset \Delta V$$

Die diskreten Werte der skalaren Größe ∅ werden im Zellmittelpunkt gespeichert. Die Werte $\emptyset_k$ werden für den Konvektionsterm auch an den Zellflächen benötigt. Sie müssen aus den vorliegenden Werten an den Zellmittelpunkten interpoliert werden. Dies wird in Ansys Fluent durch verschiedene „Upwind Methoden“ gelöst. Die Größe an einer Zellfläche wird aus den Zellmittelpunktwerten stromaufwärts der betrachteten Zelle ermittelt. Die Transportgleichung hat sowohl die unbekannten Größen ∅ im eigenen Zellmittelpunkt als auch die unbekannten Werte $\emptyset_{nb}$ der Nachbarzelle.

Zu lösen ist noch ein (großes) System linearer Gleichungen für die Zellmittelpunkwerte $\emptyset_P$. Die Gleichung für die Linearisierung lautet:

$$a_P \emptyset_P = \sum_{nb} a_{nb} \emptyset_{nb} + Q_P$$

Dafür werden drei Approximationsarten benutzt:

1. Approximation der Integrale, z.B. mit der Mittelpunktregel
2. Interpolation der Flächenmittelpunktwerte
    a. Interpolation von Gradienten für Diffusionsterme
    b. Upwind und weitere Verfahren für Konvektionsterme
3. Gradientenrekonstruktion in den Zellmittelpunkten [1].

2.2 Massen- und Impulserhaltung

Für alle Strömungssimulationen löst Ansys Fluent die Grundgleichung der Massenerhaltung, Impulserhaltung und Energieerhaltung. Die allgemeine Transportgleichung für die Massenerhaltung ist wie folgt definiert:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = S_m$$

Sie gilt sowohl für die kompressible als auch inkompressible Strömung. S_m ist der Masssource, der zusätzlich in der Kontinuitätsgleichung durch das UDF-Programm

manuell hinzugefügt und in der Transportgleichung berechnet wird. Die Impulserhaltung ist wie folgt definiert:

$$\frac{\partial}{\partial t}(\rho\vec{v}) + \nabla\cdot(\rho\vec{v}\vec{v}) = -\nabla p + \nabla\cdot(\bar{\bar{\tau}}) + \rho\vec{g} + \vec{F}$$

p ist der statische Druck, $\bar{\bar{\tau}}$ ist der Spannungstensor, $\rho\vec{g}$ und $\vec{F}$ sind die Gravitation und äußere angreifende Kräfte (z.B. die Interaktion zwischen zwei Medien bei der Zerstäubung der Flüssigkeit oder der Partikel). Der Term $\vec{F}$ beinhaltet auch das andere Sourceterme z.B. für ein poröses Medium oder UDF-Sourceterme, die vom Benutzer manuell im UDF-Programm eingegeben werden müssen [2]. Die Impulserhaltungsgleichungen sind auch bekannt als Navier-Stokes Gleichungen.

2.3 DPM-Modell (Discrete-Particle-Modell)

Das DPM-Modell ist ein Lagrange diskretes Phasen Modell in Ansys Fluent, das auf der Euler-Lagrange Approximation basiert. Das Strömungsgebiet wird als Kontinuum betrachtet und durch die Navier-Stokes Gleichung gelöst. Die Berechnung der dispersen Phase wird durch die Verfolgung der Bahnkurve von Partikeln bzw. Tröpfchen im berechneten Strömungsgebiet bestimmt. Die disperse Phase kann ein Impulsaustausch, Energieaustausch und eine Masseninteraktion zwischen den Partikeln und dem Fluid sein.

In diesem Modell wurde eine fundamentale Annahme verwendet, dass die disperse Phase einen niedrigen Volumenbruch hat. Es ist zulässig, dass der Massenstrom der Partikel größer als der Massenstrom des Fluids ($\dot{m}_{partikel} \geq \dot{m}_{fluid}$) ist. Die Gleichung für die Bahnkurve der Partikel ist wie folgt definiert:

$$\frac{du_p}{dt} = F_D(u - u_p) + \frac{g_x(\rho_p - \rho)}{\rho_p} + F_x$$

wobei F_x eine zusätzliche Kraft in Bezug auf die Masse eines Partikels ist, und $F_D(u - u_p)$ ist die Auftriebskraft der Masse eines Partikels.

$$F_D = \frac{18\mu}{\rho_p d_p^2}\frac{C_D R_e}{24}$$

R_e ist hier die relative Reynoldszahl und wie folgt definiert:

$$R_e \equiv \frac{\rho d_p\,|u_p - u|}{\mu}$$

Es ist wichtig zu wissen, dass die Gravitation in Ansys Fluent für die Standardeinstellung Null ist. Wenn die Gravitation in der Berechnung berücksichtigt werden sollte, müssen der Betrag und die Richtung für die Gravitation im Eingabefeld Ansys Fluent eingegeben werden [2].

2.3.1 Gleichung für die Bewegung von Partikeln

Die Gleichung für die Bahnkurve von Partikeln wird durch ein Integral der diskreten Zeitschritte bestimmt. Der Weg eines Partikels ist definiert durch:

$$\frac{dx}{dt} = u_p$$

Die Gleichung $\frac{du_p}{dt} = F_D(u - u_p) + \frac{g_x(\rho_p - \rho)}{\rho_p} + F_x$ wird wie folgt zusammengefasst:

$$\frac{du_p}{dt} = \frac{1}{\tau_p}(u - u_p) + a$$

Wobei a die Beschleunigung repräsentiert, die durch äußere Kräfte (außer der Wiederstandkraft) ausgeübt wird. Die Partikelgeschwindigkeit und der Ort an der neuen Position werden durch die folgenden Gleichungen beschrieben:

$$u_p^{n+1} = u^n + e^{-\frac{\Delta t}{\tau_p}}(u_p^n - u^n) - a\tau_p(e^{-\frac{\Delta t}{\tau_p}} - 1)$$

$$x_p^{n+1} = x_p^n + \Delta t(u^n + a\tau_p) + \tau_p\left(1 - e^{-\frac{\Delta t}{\tau_p}}\right)(u_p^n - u^n - a\tau_p)$$

u_p^n und u^n sind die Partikel- und Fluidgeschwindigkeit an der alten Position. Die Partikelgeschwindigkeit an der neuen Position $n+1$ ist wie folgt definiert:

$$u_p^{n+1} = \frac{u_p^n\left(1 - \frac{1}{2}\frac{\Delta t}{\tau_p}\right) + \frac{\Delta t}{\tau_p}\left(u^n + \frac{1}{2}\Delta t u_p^n \cdot \nabla u^n\right) + \Delta t a}{1 + \frac{1}{2}\frac{\Delta t}{\tau_p}}$$

Und die neue Position der Partikel ist:

$$x_p^{n+1} = x_p^n + \frac{1}{2}\Delta t(u_p^n + u_p^{n+1})$$ [2]

2.3.2 Tropfenkollisionsmodelle

In Ansys Fluent kann man die Tropfenkollisionsmodelle für die Partikelsimulation anwenden. Für N Anzahl von Tropfen gibt es $N-1$ mögliche Kollisionspartner. Die Anzahl der Kollisionspaare ist durch $\frac{1}{2}N^2$ definiert. Der Faktor $\frac{1}{2}$ ergibt sich aus der

Annahme, dass die Tropfenkollision zwischen Tropfen A und B gleich B und A ist. Der Algorithmus für die Kollision von Partikeln muss den Faktor $\frac{1}{2}N^2$ des möglichen Zusammenpralls für jeden Zeitschritt berechnen. Weil ein Spray mehrere Millionen Tröpfchen enthalten kann, ist es daher wegen dem hohen Rechneraufwand nicht sinnvoll. Dies führt zu einem Konzept, dem so genannten „Parcels“ Modell. Parcels sind eine repräsentative Anzahl einer großen Tropfenmenge.

Zum Beispiel Bei der Berechnung des Weges einiger Parcels, die jeweils 1000 Tröpfchen beinhalten, ist der Rechneraufwand um den Faktor 10^6 reduziert. Wenn zwei Parcels zusammen prallen, entscheidet der Algorithmus für eine der beiden möglichen Kollisionsarten. Die Tropfen vereinigen sich oder prallen voneinander ab. Welche der beiden Möglichkeiten, ist abhängig von der Weberzahl. Sie ist definiert durch:

$$W_{e_c} = \frac{\rho U_{rel}^2 \overline{D}}{\sigma}$$

U_{rel} ist die relative Geschwindigkeit und $\overline{D}$ ist der arithmetische Mittelwert des Durchmessers zwischen zwei Parcels [2].

2.3.3 Tropfen-Luft-Kopplung

In Ansys Fluent bietet sich die Möglichkeit, die Interaktion zwischen der Luft und den Partikeln für die Simulation zu berücksichtigen. Hierbei werden die diskrete Phase und die disperse Phase im getrennten Solver berechnet. Die Information am Randübergang, zwischen der diskreten Phase und der dispersen Phase, das sog. Fluid-Structure-Interface (FSI), wird für die beide Solver zur Verfügung gestellt. Die Berücksichtigung dieser Information in der Berechnung ist abhängig von den Coupling-Methoden die in der Simulation verwendet wird.

Im One-Way coupling übt der Drück des Fluides auf die Partikel. Die Information über den Druck wird dem Solver für die dispersen Phasen gegeben. Für das Two-Way coupling wird zusätzlich die Position der Partikeln berücksichtigt. Die Information wird dem Solver für die diskrete Phase zur Berechnung weitergegeben. Für diese Simulation wird die Interaktion zwischen dem Luftstrahl und den Partikeln

mit dem Two-way coupling berechnet. Hier werden die Navier-Stokes Gleichungen und die Euler-Lagrange Formulierung in Ansys Fluent gelöst.

2.3.3.1 Two-way coupling

Während der Berechnung der Partikelbahn ermittelt Ansys Fluent die Energie, Masse, und den Impulsaustausch zwischen den Partikeln und dem Luftstrahl. Im Two-way coupling werden die Navier-Stokes- und Euler-Lagrange Gleichungen gelöst bis die Ergebnisse sich nicht mehr verändern. Die Interaktion zwischen dem Strömungsgebiet und den Partikeln ist in **Abbildung 2** dargestellt. Die **Abbildung 3** zeigt das Schema für den Ablauf der Iteration unter Verwendung von Two-way coupling.

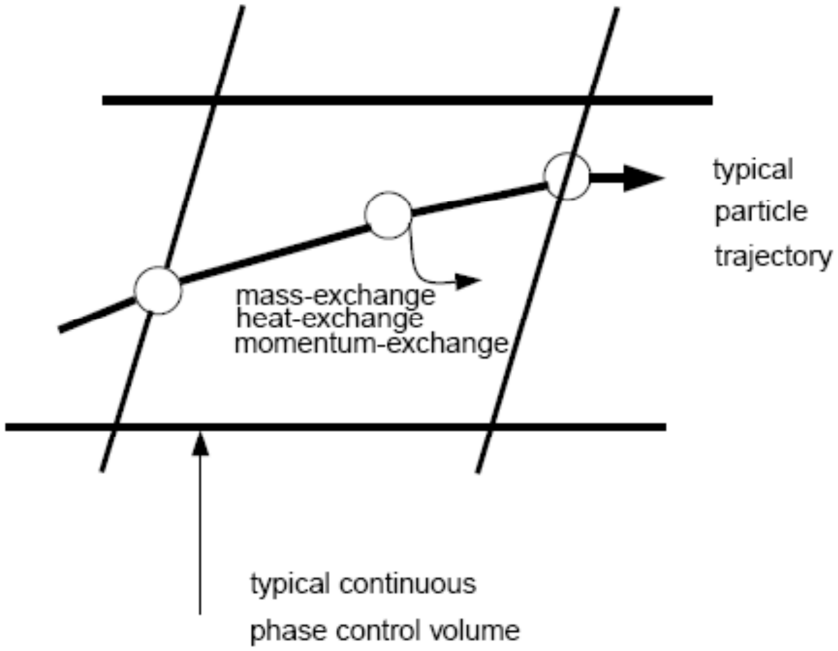


Abbildung 2: Interaktion zwischen der diskreten Phasen und d und diskreten Phase [2]

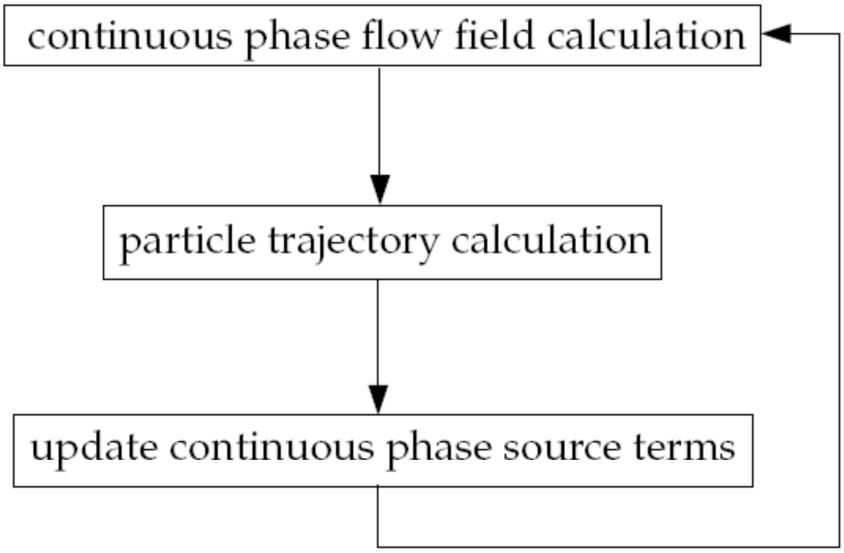


Abbildung 3: Schema coupled diskrete Phase [2]

2.4 Partikelverteilung und Solid Cone Atomizer (nur in 3D)

In dieser Diplomarbeit wird schließlich nur das Injektor-Modell Solid Cone verwendet. Die Partikelverteilung beim Flüssigkeitszerfall während des Sprayvorgangs in der Simulation ist nach der Rosin-Rammler Verteilung festgelegt.

Die gesamte Verteilung wird in entsprechender Anzahl des diskreten Intervalls zugeordnet. Jedes Intervall repräsentiert den durchschnittlichen Durchmesser der Partikel. Nach der Rosin-Rammler Verteilung ist der Massenbruch der Tröpfchen für den Durchmesser größer als d wie folgt definiert:

$$Y_d = e^{-(d/\bar{d})^n}$$

Es sind folgende Einstellungen in Ansys Fluent für Solid Cone Injektion zu aktivieren:

- Position: definiert die Position der Injektor in X, Y, und Z Koordinaten
- Diameter: definiert den Durchmesser der Partikeln in dem Strahl
- Temperature: definiert die Temperatur des Strahls
- Axis: definiert die Symmetrieachse des Strahls in x,y,-und z Koordinaten
- Velocity: definiert die Geschwindigkeit der Partikel
- Cone Angle: definiert den Halbkegelwinkel

- Radius: definiert den inneren Radius, die Partikel werden innerhalb des Radius verteilt.
- Mass flow rate: Massenstrom

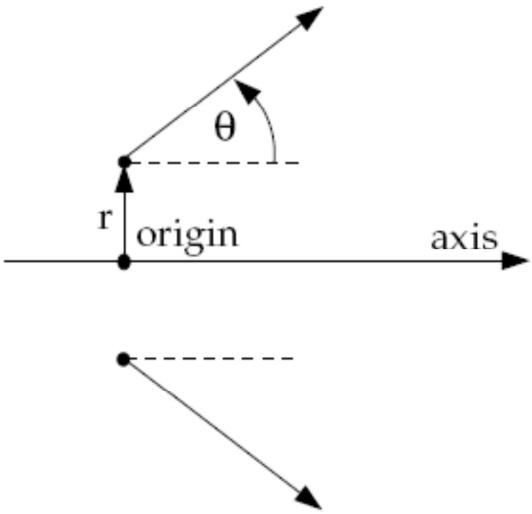


Abbildung 4: Öffnungswinkel eines Sprays [2]

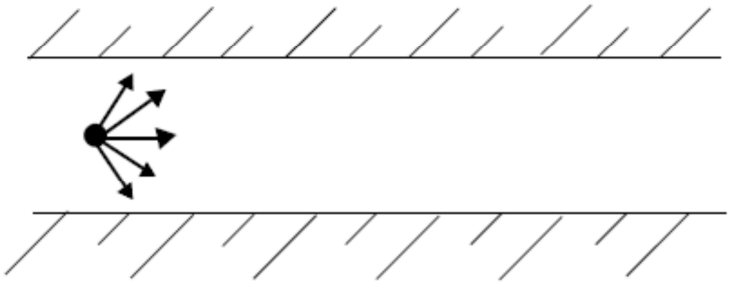


Abbildung 5: Solid Cone (3D) DPM-Modell [2]

Die Partikel sind beim Solid Cone Modell in Ansys Fluent willkürlich verteilt. Für die Standardeinstellung der Rosin-Rammler Verteilung in Ansys Fluent ist der Durchmesserbereich bei einer Injektion etwa 1 bis 200 µm [2].

2.4.1 Randbedingungen für die Partikelsimulation

Für die DPM-Randbedingungen bietet Ansys Fluent verschiedene Auswahlmöglichkeiten. Für diese Partikelsimulation werden „trap“ und „escape“ Randbedingungen benutzt.

- Trap Randbedingung

  Die „trap“ Randbedingung wird nur für den Boden bzw. die Oberfläche des Substrats verwendet. Die Berechnung der Bahnkurve von Partikeln ist beendet, wenn die Partikel den Boden erreichen und sich dort anhaften [2].

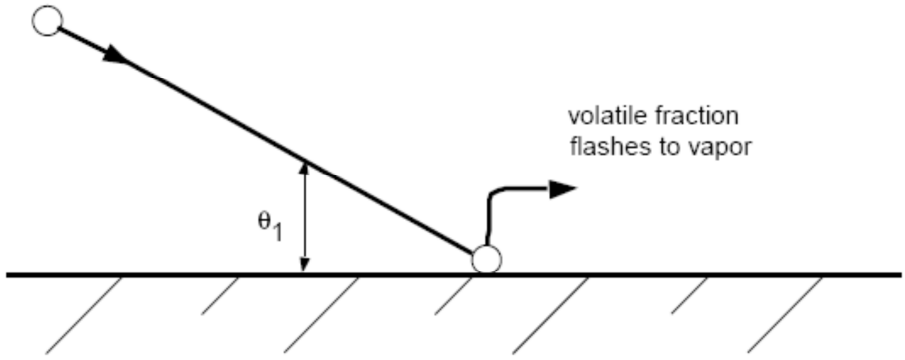


Abbildung 6: Trap Randbedingung [2]

- Escape Randbedingung

  Die Berechnung der Bahnkurve von Partikeln ist beendet, wenn die Partikel durch die Randbedingung „escape“ ausströmen [2].

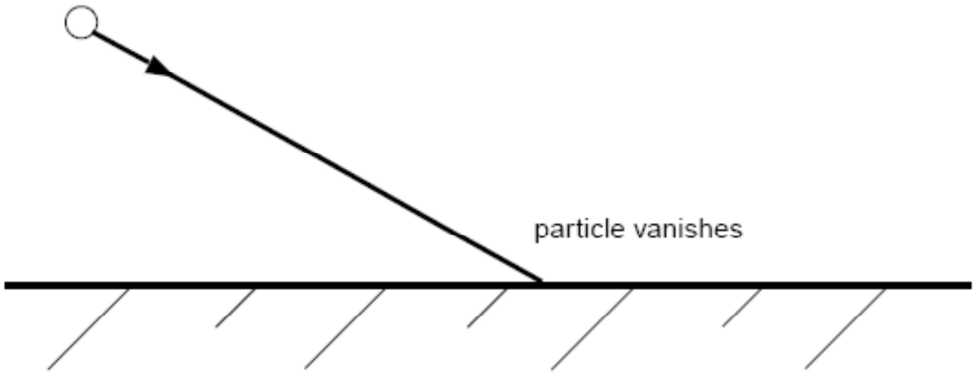


Abbildung 7: Escape Randbedingung [2]

3 User Define Function (UDF) und Sprühluftmodellierung

3.1 User Define Function (UDF)

UDF ist eine Funktion, die man selbst in C-Programmiersprache entwickelt und in den Solver von Ansys Fluent hinzufügen kann. Sie dient zur Erweiterung der Standardfunktionen, die in Ansys Fluent schon vorhanden sind. Zum Beispiel die gewünschten Randbedingungen, Materialeigenschaften, Quellterme für das Strömungsgebiet, Modellparameter wie Partikel, Multiphase Modelle, Initialisieren einer Lösung oder auch Post-Processing zu erweitern. UDF wird mit einem einfachen Texteditor geschrieben und die Datei muss dann mit Extention .c gespeichert werden, z.B. myudf.c [1]. Das Ausführen und das Einlesen vom UDF-Programm werden dann vom Ansys Fluent übernommen. Es muss zunächst vor dem Ausführen von UDF kompiliert werden. Mit Ansys Fluent für Linux Version ist der Compiler direkt vorhanden. Für Windows Version muss zusätzlich ein Compiler installiert werden.

3.1.1 Parallel Simulation

Um die Rechenzeit zu reduzieren wird die Simulation in parallelen Prozess auf mehreren Rechnern durchgeführt. Das Berechnungsgebiet wird in mehrere Gebiete geteilt. Die Einteilung bzw. Partition des einzelnen Gebiets wird dann an den jeweiligen Rechner sogenannte Knoten zugewiesen. Jede Knoten führen dasselbe Programm aus und alle laufen gleichzeitig dann zusammen. Der Hauptcomputer hat

keine Zellinformationen (Zelle, Face, Knoten). Er hat die Aufgabe die Befehle von Cortex(user interface und graphic-related function) zu interpretieren.

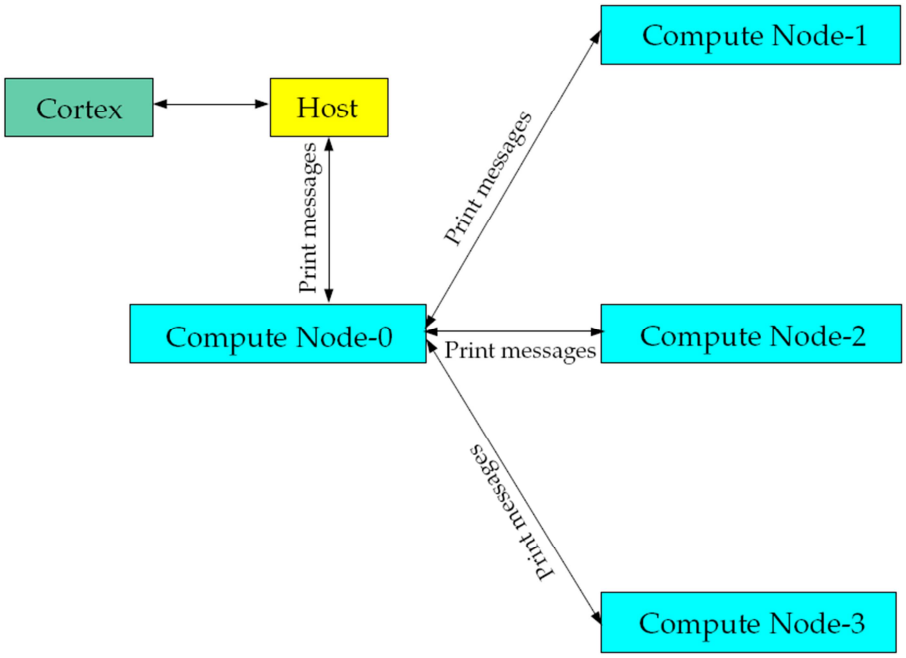


Abbildung 8: Anweisung in Parallel Ansys Fluent [3]

3.2 Sprühluftmodellierung

Es ist definiert, dass die Netze nur aus Hexaederzellen bestehen. Der Grund, warum nur Hexaederzellen für die Modellierung benutzt werden, wird im Unterkapitel 3.2.1 näher erläutert. Statt einer zylindrischen Düse, mitten in der Simulationsumgebung, nutzen wir Quellterme, die in spezifisch ausgesuchten Zellen des Berechnungsgebiets zu platzieren sind. Diese Zellenkombination für die künstliche Düse wird auch als „virtuel Nozzle“ genannt. Die Quellterme sind die Mass- und Momentumsource. Die in der Praxis eingesetzte Düse hat einen Austrittdurchmesser von ca. 7mm. Die Abmessung der künstlichen Düsen wird daher an den tatsächlichen Düsendurchmesser angepasst. Das heißt, zwei Kantenlängen der Zellen entsprechen etwa dem tatsächlichen Düsendurchmesser, siehe **Abbildung 9**. Für die Auslegung der künstlichen Düse sollte die gesamte Zellenanordnung den tatsächlichen Durchmesser 7mm nicht überschreiten.

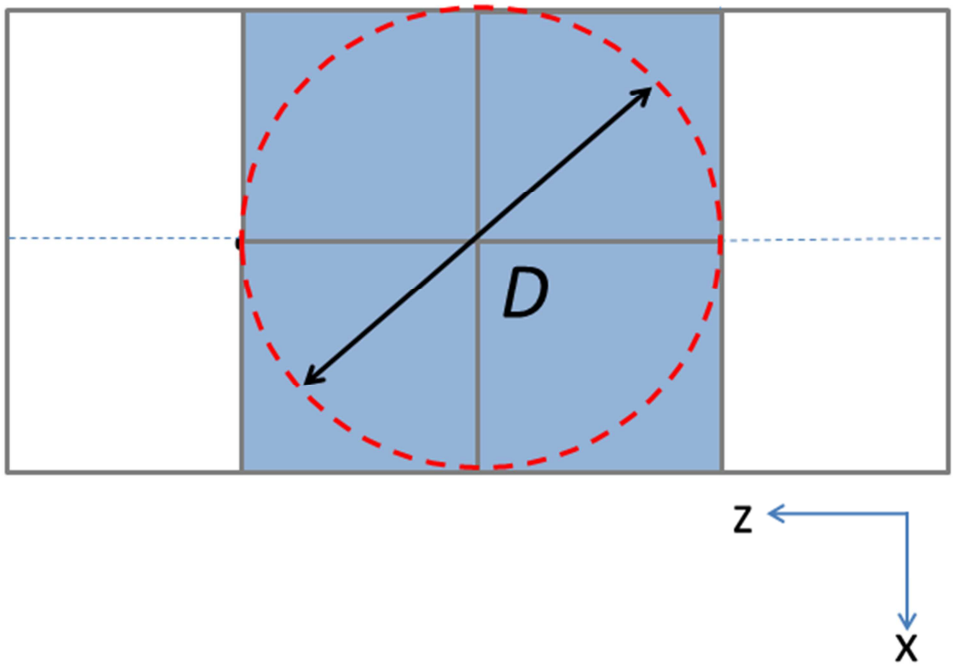


Abbildung 9: Der Durchmesser einer künstlichen Düse

Es sollte möglichst geringe Anzahl an Zellen für die künstliche Düse zu verwenden. Mit zunehmender Anzahl der Zellen entspricht der Durchmesser der künstlichen Düsen nicht dem tatsächlichen Düsendurchmesser. Außerdem ist der zu erwartende Öffnungswinkel des Strahls sehr schwierig zu kontrollieren, da der Luftstrahl aufgrund der steigenden Anzahl an Zellen immer breiter wird.

3.2.1 Vernetzung des Berechnungsgebiets

Für die Modellierung ist ein Begrenzungsraum für den Prozess definiert. Das Substrat, auf dem das PUR mit Faserverstärker aufgesprüht ist, hat die Geometrie einer viereckigen Platte. Der Sprühvorgang erfolgt in der Praxis mit Hilfe eines Roboterarms. Der Luftstrahl tritt aus der Düse heraus und wird mit dem flüssigen Polyurethan und dem Faserverstärker vermischt. Die Mischung wird durch die Roboterbewegung auf dem Substrat gleichmäßig aufgesprüht und kann dort anschließend aushärten.

Das Berechnungsgebiet für die Simulation umfasst die Substratoberfläche bis auf die Höhe des Düsenkopfs, also der Abstand zwischen der Oberfläche und dem Roboter (Düsenaustritt). Für den Gesamtraum des Prozesses ist ein Container mit der

Abmessung 1 x 1 x 0,5 m (Breite x Länge x Höhe) festgelegt. Der Container wird dann mit der Gambit Software vernetzt. Die Randbedingungen für den Container sind wie folgt festgelegt:

- Boden (Substratsoberfläche): Wall
- Die restlichen Wände: pressure Outlet

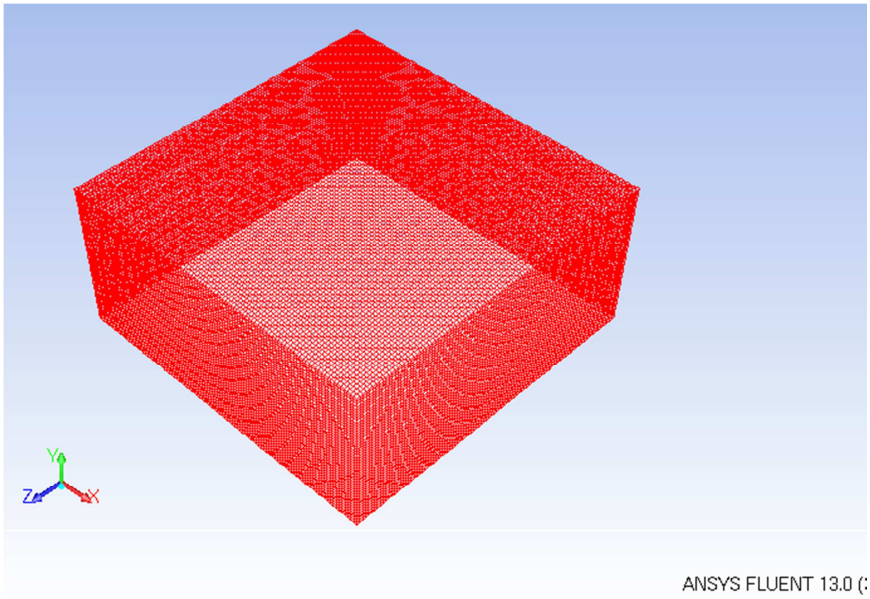


Abbildung 10: Container mit Hexaederzelle

In der **Abbildung 10** ist ein Beispiel eines Containers dargestellt, der die folgenden Abmessungen hat:

- 1 x 1 x 0,5 m (Breite x Länge x Höhe)
- Anzahl der Zellen: 500.000

Die Zellabmessung ist für die Vernetzung nach gleicher Kantenlänge anzustreben um einen symmetrischen Luftstrahl zu erhalten. Der Grund weshalb nur Hexaederzellen für die Modellierung zu benutzen sind, ist der, dass diese ausgesuchten Zellen mit Hilfe ihrer Mittelpunktkoordinaten durch den entwickelten UDF-Algorithmus einfacher zu finden sind. Der Mittelpunktabstand zwischen zwei benachbarten Zellen soll immer näherungsweise konstant sein. Dafür ist eine strukturierte Vernetzung erforderlich. Hier bietet sich die Hexaeder-Vernetzung sehr gut an, denn der Mittelpunktabstand zwischen der Ursprungs- und der Nachbarzelle ist gleich der Kantenlänge einer Zelle.

Bei einer translatorischen Bewegung wechselt sich die Ursprungszelle entlang der Trajektorien und gleichzeitig die Nachbarzelle erneut zu finden. Der Wechsel der Ursprungszelle zu einer neuen Ursprungszelle entlang der Bahnlinie und die Bestimmung der neuen Nachbarzellen bei unstrukturierter Zellen sehr schwierig. Denn die Koordinaten der Zellmittelpunkt bei unstrukturierter Zelle sind beliebig in dem Raum verteilt. Auf diesem Grund ist es einfacher die Zelle aus Hexaeder anstatt unstrukturierter Zellen zu benutzen

3.2.2 Ursprungszelle und Nachbarzellen

Für die künstliche Düse benötigen wir die spezifisch ausgesuchten Zellen und ihre Anordnung. Die erforderlichen Zellen werden mit Hilfe des geschriebenen UDF-Programms gesucht und identifiziert. Die Ursprungszelle ist eine Zelle, die durch die vom Benutzer vorgegebenen Koordinaten des Injektorpunktes ermittelt wird. Der Injektorpunkt liegt deshalb stets in einer Ursprungszelle. Neben der Ursprungszelle gibt es die benachbarten Zellen, die zur Erzeugung der künstlichen Düsen beitragen.

3.2.2.1 Bestimmung der Ursprungszelle

Durch den vorgegebenen Injektorpunkt in dem Berechnungsgebiet ist die Position des Injektors festgelegt. Der Punkt ist durch die Raumkoordinate (X, Y,- und Z) beschrieben. Mit dieser Information sucht der UDF-Algorithmus die Position der Ursprungszelle. Zunächst ermittelt das Programm die Koordinaten des Mittelpunkts (C_CENTROID) von jeder Zelle in dem Berechnungsgebiet. Um die Ursprungszelle finden zu können überprüft das Programm den einzelnen Vektorabstand zwischen dem Zellmittelpunkt und der Referenzposition des Injektors.

Der Vektorabstand ist der Betrag eines Vektors, der aus der Differenz zweier Punkte im Raum entsteht. Vorgegeben in der UDF ist der Referenzpunkt einer Injektorpostition $\overrightarrow{x0}$ (x,y,z). Jedes Mal wenn sich der UDF-Algorithmus in einer Zelle befindet, überprüft das Programm den Vektorabstand durch die Subtraktion der beiden Vektoren zwischen dem Vektor $\vec{x}$ (C_Centroid) der beliebigen Zelle und dem Vektor $\overrightarrow{x0}$ des Injektorpunkts.

$$\boldsymbol{x\_x0} = \vec{x} - \overrightarrow{x0}$$

$$\boldsymbol{abs} = \sqrt{(\boldsymbol{x\_x0})^2}$$

Diese Vektoraddition wird in der UDF durch die Funktion NV_VV(x_x0, =, x, -, x0) ausgeführt. Daraus wird der Betrag $\boldsymbol{abs}$ errechnet. Dieser Betrag wird mit dem anderer Zellen verglichen. Die Ursprungszelle ist dann eindeutig gefunden wenn der Betrag des Vektorabstands am kleinsten ist. Der kleinste Betrag $\boldsymbol{abs}$ überschreibt den alten Betrag und wird in dem Loop stets zwischengespeichert, solange die Ursprungszelle noch nicht gefunden ist.

```
67 thread_loop_c(t,d) /*loops over all cell threads in domain*/
68 {
69
70 begin_c_loop(c,t) /* loops over cells in a cell thread */
71 {
72    C_CENTROID(x,c,t)
73      NV_VV(x_x0, =, x, -, x0);
74      abs = NV_MAG(x_x0);
75
76 /* Finde Ursprungscell  */
77      if(abs < abs_klein)
78           {abs_klein = abs; NV_V(A_C, =, x); a_c = c;
```

Abbildung 11: UDF Bestimmung der Ursprungszelle

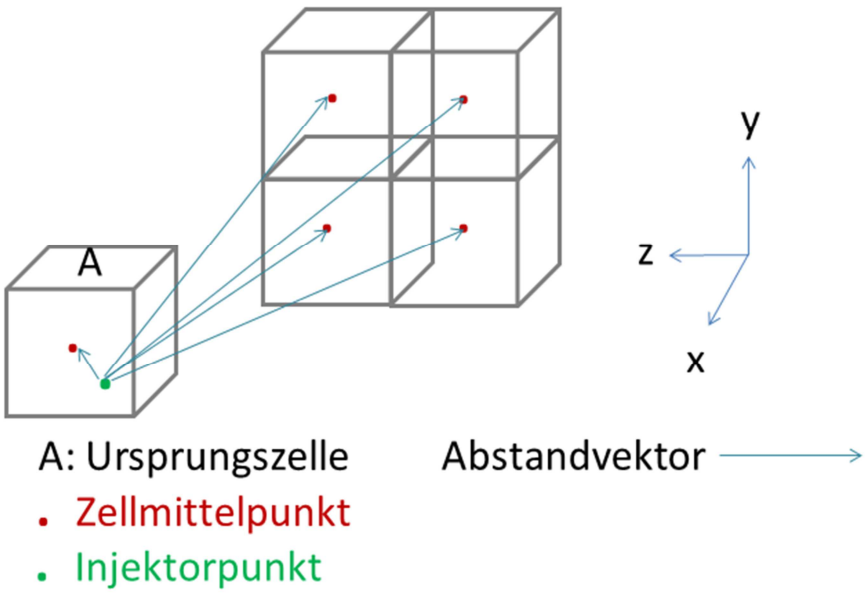


Abbildung 12: Ursprungszelle und Injektorpunkt

3.2.2.2 Bestimmung der Nachbarzelle

Ansys Fluent UDF stellt die Funktion „Adjacent Cell Index“ (F_C0, F_C1) zur Verfügung. Zwei Zellen, die benachbart sind, haben gemeinsam eine angrenzende Face. C0 und C1 sind die Zellindexnummern, die durch das UDF-Macro F_C0 und F_C1 bestimmt werden. F_C0 gibt die Zellindexnummer C0 bzw. F_C1 gibt die Zellindexnummer C1 zurück. Falls eine Face direkt an dem Randbereich liegt, existiert nur C0 (C1 ist undefiniert für eine external Face). Andernfalls wenn eine Face innerhalb der Interior Domain liegt, existieren die Zellindexnummern C0 und C1 [3].

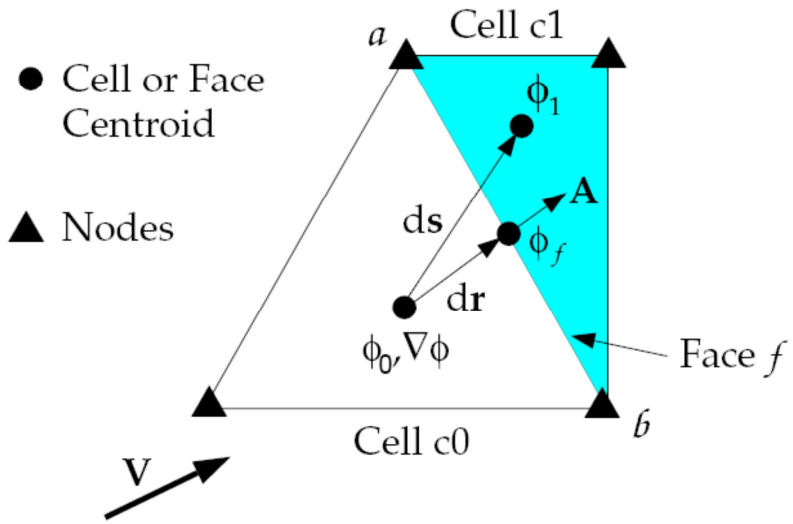


Abbildung 13: Adjacent Cell Index C0 und C1 [3]

$\vec{A}$ ist in Ansys Fluent ein Normalenvektor, der orthogonal an einer Face der Zelle steht, siehe Abbildung 14. Der Normalenvektor zeigt stets die Richtung von der Zelle C0 zur Zelle C1, siehe **Abbildung 13**. Diese Methode wird benutzt um die Nachbarzellen, die die Ursprungszelle umgeben, zu bestimmen. Die Ursprungszelle hat insgesamt sechs Faces, siehe **Abbildung 14**.

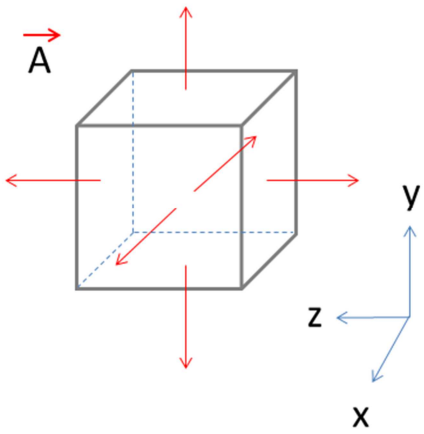


Abbildung 14: Ursprungszelle mit sechs Faces und die Normalenvektoren A

Die Nachbarzellen, die für die künstliche Düse nötig sind, liegen auf derselben Ebene wie die Ursprungszelle. Für die Identifikation der Face muss ein Face-Loop Befehl in der Ursprungszelle ausgeführt werden. Nach dem Face-Loop findet das Programm sechs Face und gleichzeitig die sechs benachbarten Zellen. Die zwei Zellen (in graue Farbe gekennzeichnet) oberhalb und unterhalb der Ursprungszelle in der **Abbildung 15** müssen beim Suchen ausgeschlossen sein. Um diese beiden Zellen beim Suchen auszuschließen, wird das Skalarprodukt angewendet. Für das Skalarprodukt steht ein UDF-Macro „NV_Dot“ von Ansys Fluent zur Verfügung. Es soll nur die Nachbarzelle berücksichtigen, für die das Skalarprodukt null ist.

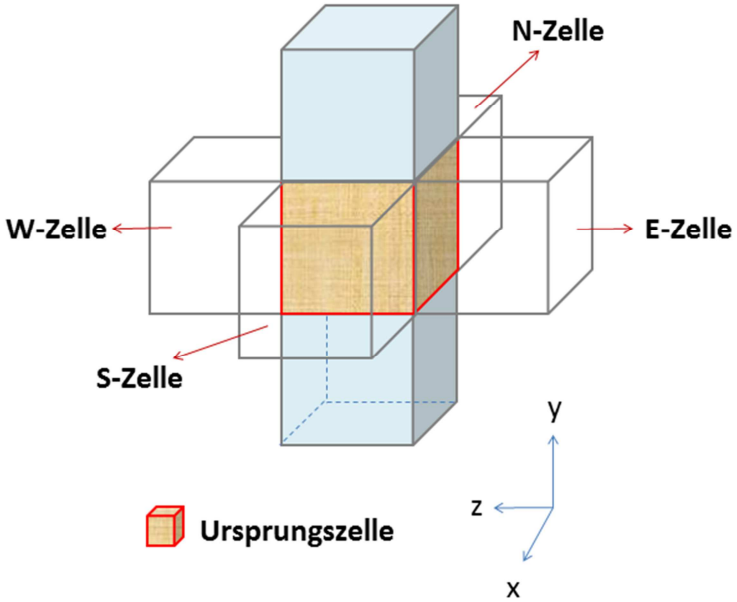


Abbildung 15: Auszuschließende Nachbarzellen (oberhalb und unterhalb der Ursprungszelle)

In UDF wird zunächst ein Einheitsvektor $e = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ definiert. Der Einheitsvektor zeigt in positiver y-Richtung. Er steht orthogonal zu den vier Normalenvektoren der $\vec{A}$ Faces der Seitenflächen und parallel zu den zwei Vektoren der $\vec{A}$ Faces der Ober- und Unterseite des Würfels.

```
86  f = C_FACE(a_c,t,n); tf = C_FACE_THREAD(a_c,t,n); F_CENTROID(face_cent1,f,tf);
87   if(NV_DOT(e, An) == 0.)
88     {  if(c == F_C0(f,tf)) Nachbr_c = F_C1(f,tf);
```

Abbildung 16: UDF-Programm - Lokalisieren der ausgesuchten Nachbarzellen mit Hilfe das Skalarprodukt zweier Vektoren

Das Skalarprodukt zweier Vektoren ist null, wenn die beiden Vektoren rechtwinklig zueinander stehen. Durch die gegebene Bedingung in UDF-Programm (siehe **Abbildung 16**) werden die zwei in **Abbildung 15** dargestellten Nachbarzellen ober- und unterhalb der Ursprungszelle vernachlässigt. Denn das Skalarprodukt der beiden Vektoren ist nicht null.

Die Mittelpunktkoordinate der Ursprungszelle ist jetzt die wichtigste Information für die Bestimmung der Nachbarzellen. Hierin liegt der Vorteil von Hexaederzellen. Da die Ursprungszelle A schon gefunden ist, sind die Positionen der Nachbarzellen um die Ursprungszelle herum leicht zu identifizieren. Die y-Koordinaten der ausgesuchten Nachbarzellen sind auf der gleichen Höhe, siehe **Abbildung 17**. Dadurch können diese Zellen in xz-Ebene (2D-Ebene) betrachtet werden. Dies erleichtert den Algorithmus um die Nachbarzellen zu finden, da sich die Betrachtung des Rechengitters von 3D zu 2D reduziert.

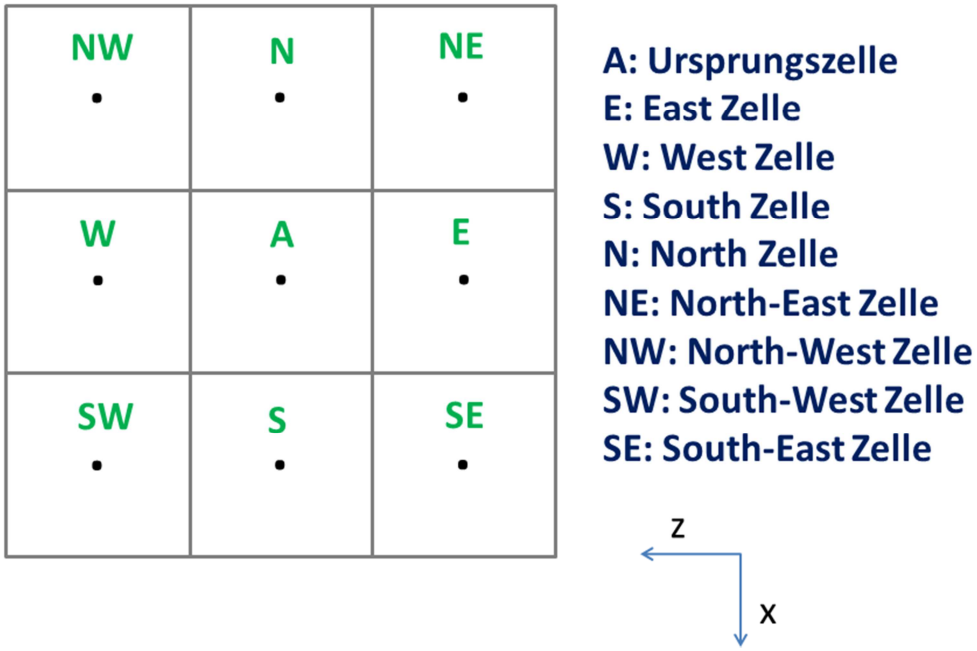


Abbildung 17: Ursprungszelle und Nachbarzellen

Mit Hilfe der in UDF vorhandenen Funktion „Face-Loop“ werden die Facemittelpunkte (F_Centroid) gleichzeitig ermittelt. Es werden zunächst nur die vier Facemittelpunkte der Nachbarzellen (N-, S-, W- und E-Zelle) ermittelt, die an die Ursprungzelle grenzen.

In dem nächsten Schritt werden die Nachbarzellen angesprochen und die richtigen Namen für die jeweiligen Zellen vergeben. Hier überprüft das Programm wieder die vier Normalenvektoren der $\vec{A}$ Faces mit Hilfe des UDF Macros F_C0 und F_C1. Falls der Face-Normalenvektor von der Ursprungszelle zur Nachbarzelle zeigt, dann bekommt die Ursprungszelle den Index C0 bzw. die Nachbarzelle den Index C1. Hier ist der Normalenvektor $\vec{A}$ als positiv definiert. Andernfalls bekommt die Nachbarzelle den Index C0 und die Ursprungszelle den Index C1, wenn der Face-Normalenvektor in die Ursprungszelle hinein zeigt, siehe **Abbildung 18**. Der Normalenvektor $\vec{A}$ ist dann als negativ definiert.

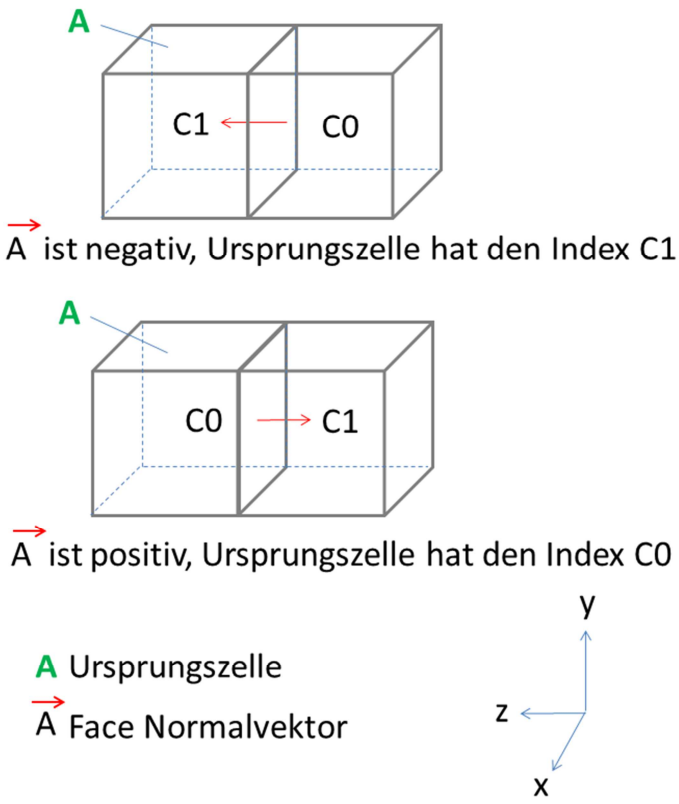


Abbildung 18: Bestimmung des Zellindex C0/C1 durch den Normalenvektor

Die F_C0 und F_C1 Funktionen sind notwendig, weil während des Face-Loops in der Ursprungszelle die Richtung der Face-Normalenvektoren unbekannt ist. Durch die Identifikation des Face-Normalenvektors $\vec{A}$ können die vier Nachbarzellen um die Ursprungszelle herum gefunden werden. Gleichzeitig wird innerhalb des Loops der Zellmittelpunkt durch das UDF Macro C_Centroid ermittelt.

Bis hier ist noch unbekannt, welches eigentlich die N-, W-, S und E-Zellen sind. In dem nächsten Schritt soll die jeweilige Zelle dem richtigen Namen zugeordnet werden. Das Programm soll den Abstand zwischen der Nachbar- und der Ursprungszelle berechnen. Für die Abstandsermittlung der W- und E-Zelle ist nur die z-Koordinate bzw. für die N- und S-Zelle ist die x-Koordinate wichtig. Die Abstandsberechnung wird durch die Subtraktion der Mittelpunktkoordinate zwischen den einzelnen Nachbarzellen und der Ursprungszelle realisiert, siehe **Abbildung 19**. Der Abstand hat die Länge, die etwa gleich der Kantenlänge einer Zelle ist.

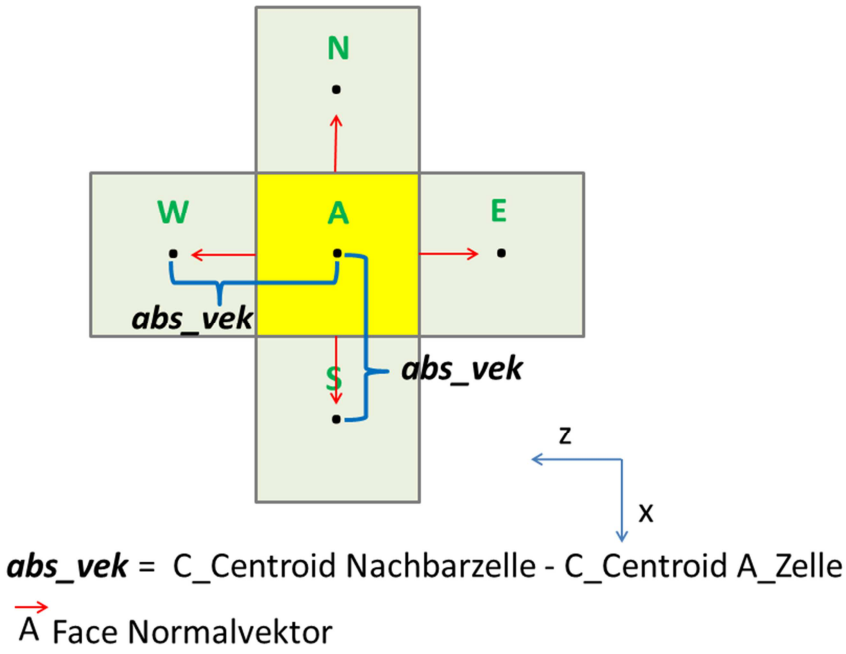


Abbildung 19: abs_vek und Face Normalenvektor für Bestimmung der Nachbarzelle

Für die vier Nachbarzellen sind folgende Vektoradditionen festgelegt:

$$\boldsymbol{abs\_vek} = \boldsymbol{S\_C} - \boldsymbol{A\_C}$$
$$\boldsymbol{abs\_vek} = \boldsymbol{N\_C} - \boldsymbol{A\_C}$$
$$\boldsymbol{abs\_vek} = \boldsymbol{E\_C} - \boldsymbol{A\_C}$$
$$\boldsymbol{abs\_vek} = \boldsymbol{W\_C} - \boldsymbol{A\_C}$$

Für S- und N-Zelle wird nur die x-Komponente der $\boldsymbol{abs\_vek}$ Vektoren geprüft bzw. für W- und E-Zelle nur die z-Komponente.

Als Beispiel betrachten wir zwei Zellen, A_C und S_C. Hier haben wir zwei Mittelpunktkoordinaten, die durch das UDF Macro C_Centroid ermittelt wurden.

$$\boldsymbol{A\_C} = \begin{pmatrix} 0{,}503 \\ 1{,}004 \\ 0{,}757 \end{pmatrix}, \boldsymbol{S\_C} = \begin{pmatrix} 0{,}516 \\ 1{,}004 \\ 0{,}757 \end{pmatrix}$$

Das Programm führt die Vektoraddition $\boldsymbol{abs\_vek} = \boldsymbol{S\_C} - \boldsymbol{A\_C}$ aus. Wir bekommen:

$$\boldsymbol{abs\_vek} = \begin{pmatrix} 0{,}013 \\ 0 \\ 0 \end{pmatrix}$$

Da die Zellen alle auf der gleichen Ebene liegen, ist die y-Komponente von ***abs_vek*** null. Ebenso läuft der Wert der z-Komponente von ***abs_vek*** näherungsweise gegen null, da eine gleichmäßige Teilung bei der Vernetzung und die Verwendung von Hexaedernetze mit gleicher Kantenlänge vorliegt. Nur die x-Komponente von ***abs_vek*** ist wichtig für die Bestimmung der S-Nachbarzelle.

Die x-Komponente von ***abs_vek*** muss größer als der Vergleichswert sein. Dieser Vergleichswert ist wichtig um dem Programm mitzuteilen, dass die gesuchte Nachbarzelle in der positiven oder negativen Richtung der globalen kartesischen Achse liegt. In dieser Simulation ist der Vergleichswert 10^{-3} gewählt, da die Kantenlänge der Zelle für diesen Versuch 0,0133 m beträgt. Aus der Größe des Containers und der Anzahl der Zellen in der Domain lässt sich die Kantenlänge der Zelle leicht ermitteln. In diesem Beispiel ist ***abs_vek*[x] =** 0,013, positiv und 0,013 > 10^{-3}. Das heißt die S-Nachbarzelle ist eindeutig durch zwei Bedingungen festgelegt, und zwar:

1. Durch den Normalenvektor $\vec{A}$, der senkrecht zwischen der Face der Ursprungs- und Nachbarzelle steht und
2. das die Vektorkomponente aus ***abs_vek*[x]** größer ist als der Vergleichswert.

An dieser Stelle muss das Vorzeichen für den Vergleichswert beachtet werden. Für die Bestimmung der vier Nachbarzellen sind die Bedingungen für ***abs_vek*** wie folgt definiert:

- abs_vek [z] > 10^{-3} → W-Zelle
- abs_vek [z] < -10^{-3} → E-Zelle
- abs_vek [x] > 10^{-3} → S-Zelle
- abs_vek [x] < -10^{-3} → N-Zelle

Die Nachbarzelle wird dann mit dem richtigen Namen bezeichnet. Bis hier sind die vier Nachbarzellen (W, E, S und N) um die Ursprungszelle herum identifiziert. Es sind noch vier zusätzlichen Nachbarzellen (NE, NW, SE und SW) zu finden, siehe **Abbildung 20**. Um diese vier weiteren Zellen zu finden wird eine ähnliche Vorgehensweise verwendet.

In diesem Versuch werden die gefundene S- und N-Zelle als Referenzzelle benutzt um die benachbarten Zellen (NE, NW, SE und SW) zu finden. In der jeweiligen Referenzzelle wird ein Face Loop-Befehl ausgeführt. Durch den Face-Loop werden die Faces und ihre Mittelpunktkoordinate ermittelt. Der Normalenvektor $\vec{A}$ an der Face bestimmt die angrenzende Nachbarzelle. Jetzt muss nur noch die positive oder negative Richtung der gesuchten Nachbarzellen in der kartesischen Achse mit einem Vergleichswert verglichen werden. Um die Richtung der Zellen zu bestimmen wird der Mittelpunkt der Face mit dem Mittelpunkt der S-Zelle (die z-Komponente) gemessen. Wenn die Richtung positiv ist, dann ist die angrenzende Nachbarzelle SW bzw. NW. Andernfalls ist die angrenzende Nachbarzelle SE bzw. NE für den Fall das die Richtung negativ ist, siehe **Abbildung 20**.

Der Algorithmus wird direkt nach dem Finden der N-, S-, W-, und E-Nachbarzelle ausgeführt. Die Bedingungen für die positive und negative Richtung sind wie folgt aufgestellt:

- S_C [z] – f_Centroid_S [z] > -10^{-3} → SW-Zelle
- S_C [z] – f_Centroid_S [z] > 10^{-3} → SE-Zelle
- N_C [z] – f_Centroid_N [z] > 10^{-3} → NE-Zelle
- N_C [z] – f_Centroid_N [z] > -10^{-3} → NW-Zelle

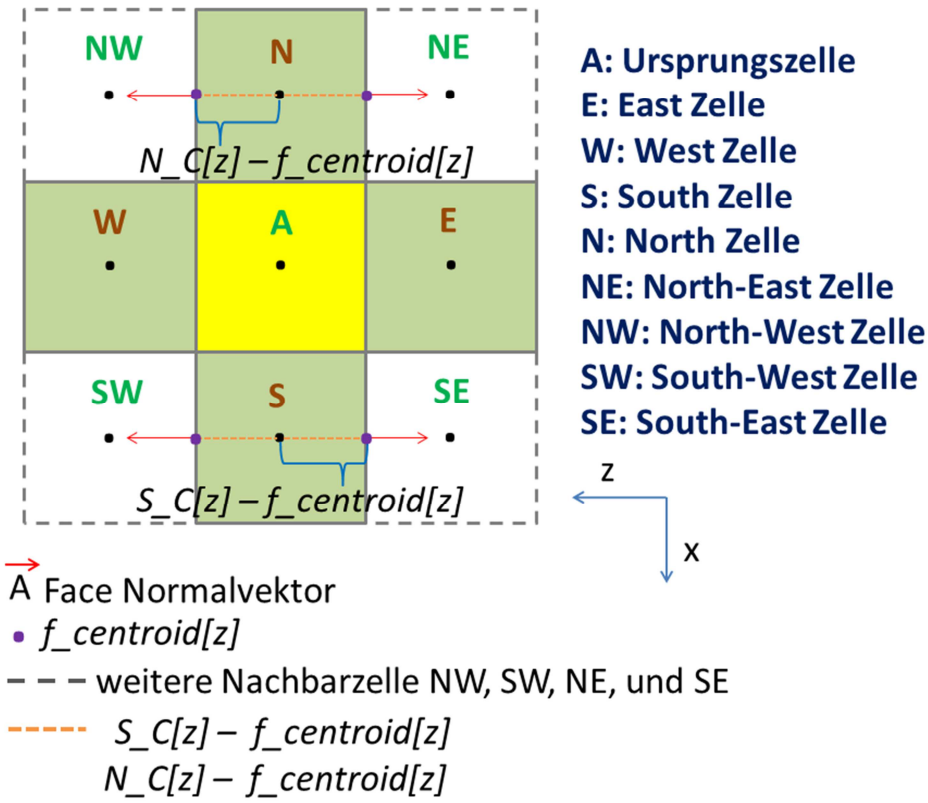


Abbildung 20: vier weiteren Nachbarzelle NW, SW, NE, und SE

Insgesamt sind acht Nachbarzellen zu lokalisieren und ihnen der dazugehörige Zellenname zuzuordnen. Der Grund warum wir insgesamt acht Nachbarzellen benötigen, ist damit zu begründen, dass die Position des Injektors in der Ursprungszelle für den allgemeinen Fall vorbereitet ist. Später wird sich der Injektorpunkt in der transienten Berechnung entlang der vorgegebenen Trajektorien bewegen. Dadurch wechselt die Ursprungszelle auch in der Domain in Abhängigkeit von der Position des Injektorpunkts.

Um die richtigen Zellen anzusprechen während der instationären Berechnung wird die Position des Injektorpunkts in der Ursprungszelle in vier Bereiche aufgeteilt, die später für die Aktivierung der drei Nachbarzellen für „virtual Nozzle“ notwendig sind. In der Abbildung 21 sehen wir ein Beispiel der Aktivierung der vier Zellen. Die Aktivierung der Zellen wird durch die Position des Injektorpunkts in der Ursprungszelle A bestimmt. Die Position des Injektors ist in dem 4. Bereich der Ursprungszelle. Dadurch werden die E-, S-, und SE-Nachbarzellen aktiviert. Damit ist die Aktivierung der vier Zellen wie folgt festgelegt:

- Injektorpunkt im 1. Bereich der A-Zelle: W, SW, und S aktiviert

- Injektorpunkt im 2. Bereich der A-Zelle: W, NW, und N aktiviert
- Injektorpunkt im 3. Bereich der A-Zelle: N, NE, und E aktiviert
- Injektorpunkt im 4. Bereich der A-Zelle: E, SE, und S aktiviert

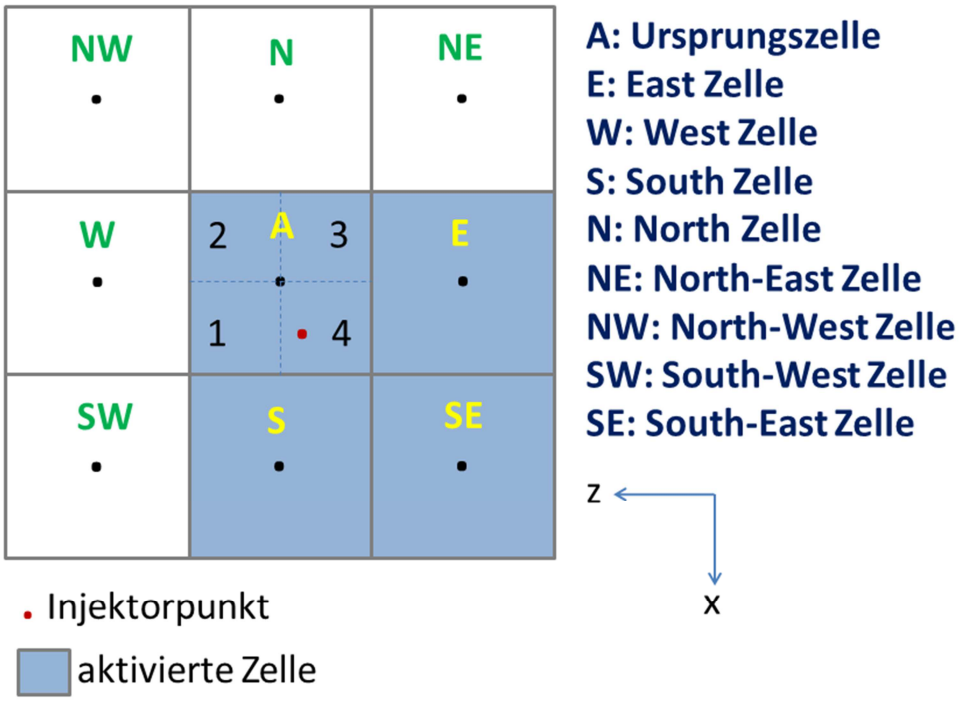


Abbildung 21: Beispiel eine Aktivierung der Zellen durch die Position des Injektorpunktes

3.2.3 Aktivierung der Ursprungszelle und Nachbarzellen für die instationären Simulation

Für die Bewegung des Luftstrahls muss die Anordnung der ausgesuchten Zellen weiterhin unverändert bleiben. Da das Berechnungsgebiet mit der strukturierten Vernetzung Hexaeder diskretisiert wurde, werden die ausgesuchten Zellen entlang der Trajektorien aktiviert. Hier wird der Injektorpunkt als Referenz für die Aktivierung der ausgesuchten Zellen benutzt. Es wird vereinbart, dass der Injektorpunkt sich immer in einer Ursprungszelle befindet. Somit ist die Lage der Ursprungszelle während der Bewegung stets definiert.

Weiterhin müssen nur noch die erforderlichen Nachbarzellen aktiviert werden. Um eine einwandfreie Zellaktivierung mit Hilfe des UDF-Programms durchführen zu

können, müssen die jeweiligen Zellen mit den richtigen Zellnamen versehen werden. Mit den entsprechenden Namen werden die Zellen angesprochen und identifiziert.
Für die Nachvollziehbarkeit wird die Betrachtung der Ursprungszelle mit dem Referenzpunkt für die künstliche Düse aus der xz-Ebene (Draufsicht) in der **Abbildung 22** dargestellt. Die Ursprungszelle wird ausgehend vom Zellmittelpunkt in vier Bereiche eingeteilt. Die zwei Bereiche die mit Nr. 2 und 3 gekennzeichnet sind, befinden sich in der oberen Reihe und die mit Nr. 1 und 4 in der unteren Reihe. Diese Einteilung wird für die Aktivierung der angrenzenden Nachbarzellen benutzt.

In dem unten dargestellten Beispiel befindet sich ein Injektorpunkt im Bereich 4 der Ursprungszelle (siehe **Abbildung 22**). Die Bewegung des Injektorpunkts findet auf einer Gerade zwischen den Bereichen 1 und 4 der Ursprungszelle statt und über deren Grenzen hinaus. Solange der Injektorpunkt sich in dem Bereich 4 befindet, werden die Nachbarzellen E-, S- und SE-Zellen aktiviert. Beim Übergang der Grenze zum Bereich 1 werden die S-, W- und SW-Nachbarzellen aktiviert. Analog gilt es auch für den Bereich 2 und 3.

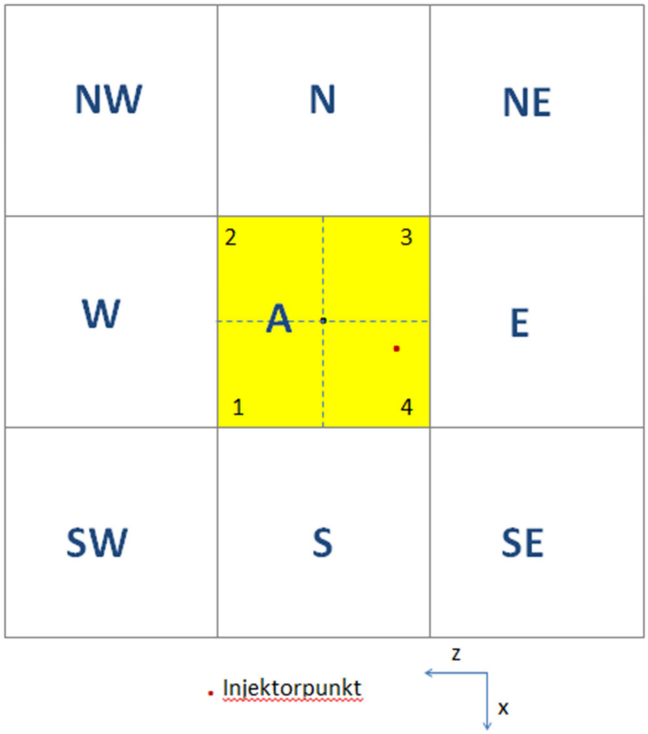


Abbildung 22: Ursprungszelle mit vier Bereich Einteilung

Damit die Aktivierung der Nachbarzellen in Abhängigkeit des Injektorpunkts einwandfrei läuft, werden vier Fälle für den Grenzübergang unterschieden. Beim Grenzübergang des Injektorpunkts in der Ursprungszelle sorgt das UDF-Programm dafür dass die richtigen Nachbarzellen aktiviert worden sind. Die vier Fälle sind in **Abbildung 23**, 22, 23, 24 dargestellt.

Zunächst wird die Position des Injektorpunkts in der Ursprungszelle lokalisiert. Die Position wird durch das UDF-Programm überprüft ob der Injektorpunkt sich in obere Reihe (Bereich 2 oder 3) oder untere Reihe (Bereich 1 oder 4) der Ursprungszelle befindet (siehe **Abbildung 22**). Wenn der Injektorpunkt sich in unterer Reihe der Ursprungszelle (Bereich 1 oder 4) befindet, kommen die Fälle I und II nur in Betracht. Durch die Bewegung des Injektorpunkts gerade entlang der Trajektorien schaltet das UDF-Programm entweder den Fall I oder II ein.

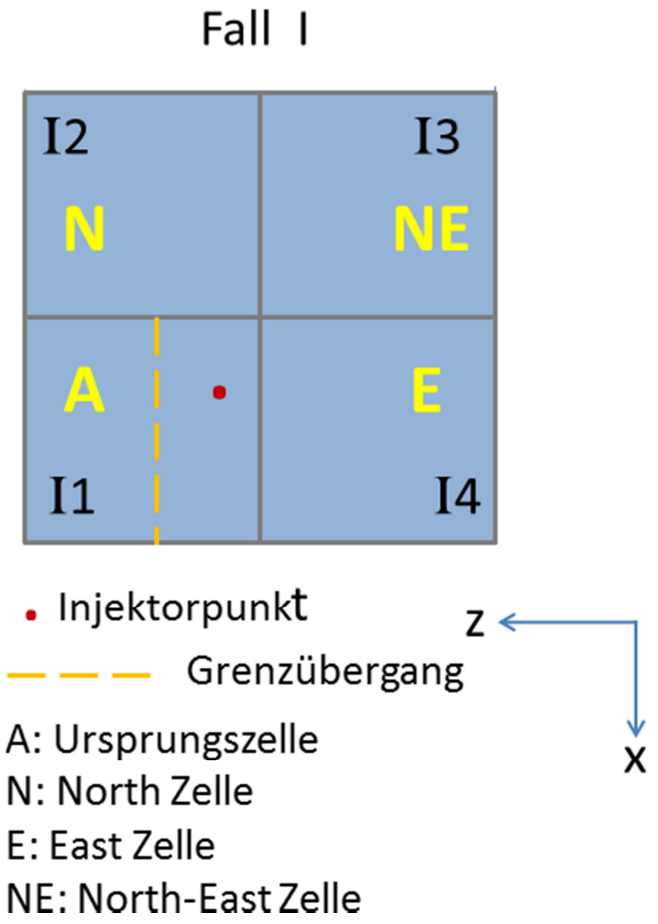


Abbildung 23: Fall I - Aktivierung der Nachbarzelle durch Bewegung des Injektorpunkts

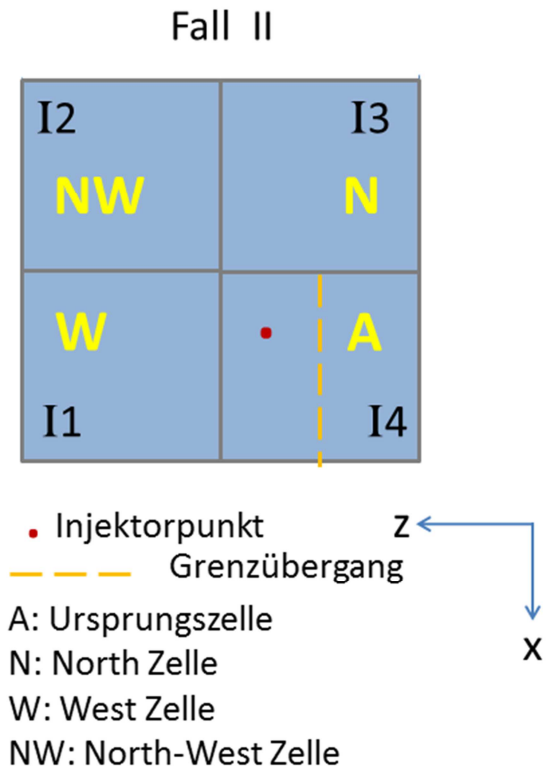


Abbildung 24: Fall II - Aktivierung der Nachbarzelle durch Bewegung des Injektorpunkts

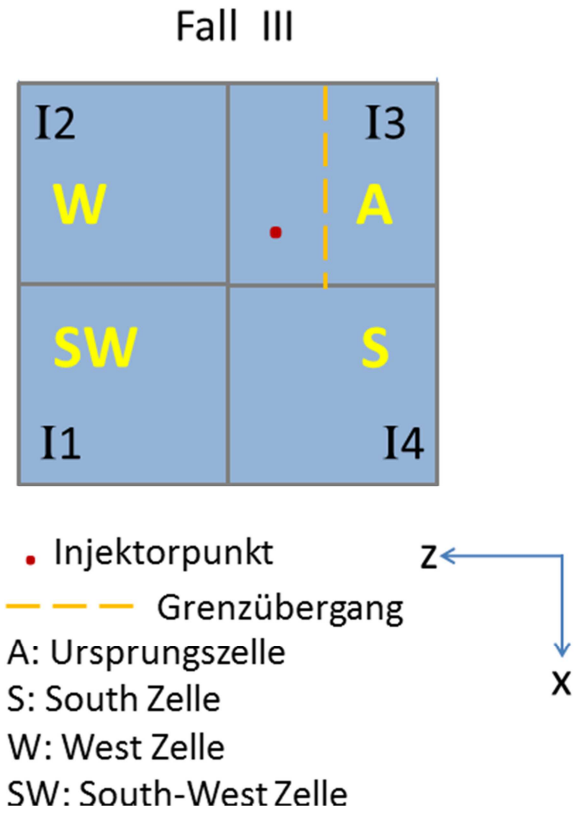


Abbildung 25: Fall III - Aktivierung der Nachbarzelle durch Bewegung des Injektorpunkts

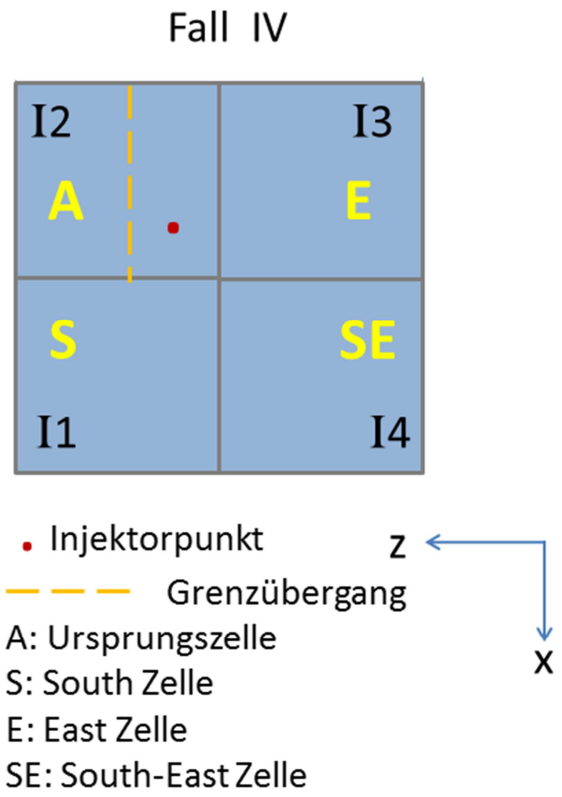


Abbildung 26: Fall IV - Aktivierung der Nachbarzelle durch Bewegung des Injektorpunkts

Hier ist ein Beispiel für die Bewegung des Injektorpunkts mit dem Wechselvorgang von einer alter Zelle zur neuen Zelle zwischen Fall I und II in **Abbildung 27** dargestellt. Wir verfolgen die Bewegung des Injektorpunkts in der Zellen. Es ist festgelegt wo der Injektorpunkt ist, ist dort auch die Ursprungszelle. Die Position Nr.1 zeigt dass das UDF-Programm den Fall II eingeschaltet hat. Sobald der Injektor in eine neue Zelle (siehe die Lage des Injektorpunkts Nr.2, schraffierter Bereich in **Abbildung 27**) rein geht, wird dort die Ursprungszelle durch den UDF-Algorithmus definiert. Nach der Unterscheidung der vier Fälle muss es hier den Fall I eingeschaltet werden. Der Injektorpunkt bewegt sich weiter und verlässt den schraffierten Bereich, der mit der Nr.2 gekennzeichnet ist. Sobald der Injektor den Grenzübergang (mit gelber Strichlinie gekennzeichnet, siehe **Abbildung 27**) zum Bereich der mit Nr.3 gekennzeichnet ist, durchquert, werden zusätzlich zwei neuen Zellen aktiviert und der Fall wechselt sich zu Fall II.

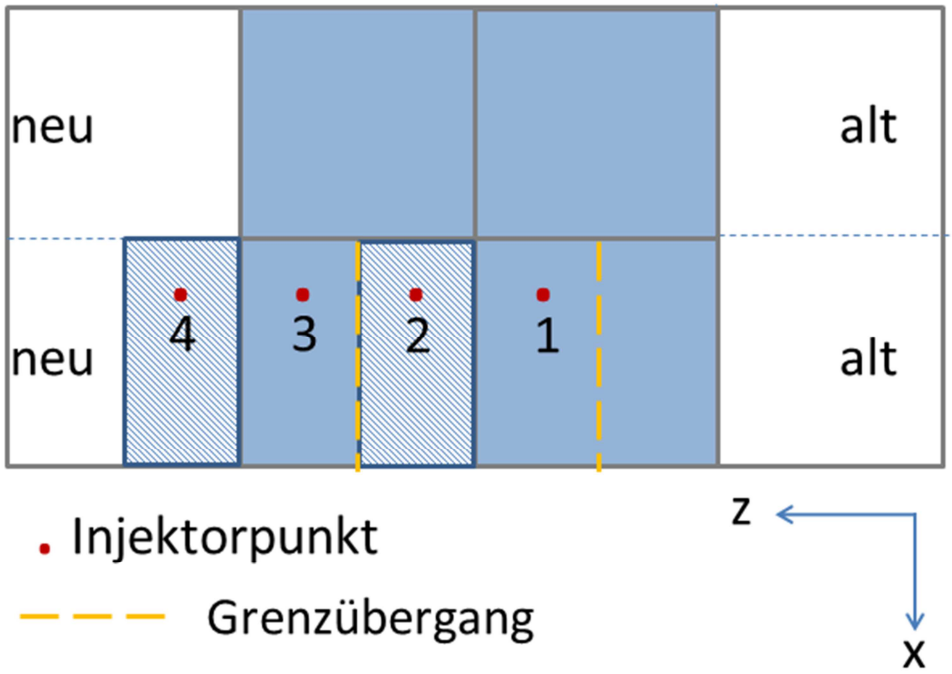


Abbildung 27: Beispiel eine Bewegung des Injektorpunkts und der Aktivierungsvorgang der Nachbarzellen

4 Luftstrahlmodelle (stationäre Simulation)

Um einen symmetrischen und kegeligen Luftstrahl zu modellieren werden verschiedene „virtual Nozzle“ Modelle aus den Zellenkombinationen entwickelt. Die Zellenkombinationen für den Luftstrahl werden in verschiedenen Varianten getestet. Die Testsimulation läuft im stationären Zustand bei ca. 300 Iterationen bis der Luftstrahl vollständig ausgebildet ist. Die Ergebnisse aller getesteten Varianten werden verglichen und anhand der Sprayeigenschaften beurteilt. Der Kegelwinkel wird für die Simulation mit 30° angenommen. Um den gewünschten Öffnungswinkel zu erreichen werden die Geschwindigkeitskomponenten in der Zelle eingestellt. Die folgenden Anforderungen sind für die Simulation in UDF eingestellt:

- Öffnungswinkel des Strahls: 30° (für diese Simulation)
- Luftgeschwindigkeit in die jeweiligen x-, y-, und z-Richtungen: $\begin{pmatrix} 10{,}7 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: unterschiedliche Einstellungen von Modell zu Modell
- Massenstrom: 0,006125 kg/s

4.1 Untersuchung des kegelförmigen Luftstrahls

Das Ziel ist ein Luftstrahl, der die Sprayeigenschaften erfüllt. Der Luftstrahl soll sowohl axialsymmetrisch wie auch kegelförmig sein und den gewünschten Öffnungswinkel besitzen. Um diesen Luftstrahl zu erzeugen, ist die Untersuchung an den künstlichen Düsen mit Hilfe verschiedener Zellkombinationen erforderlich. Für

die Untersuchung werden drei verschiedene Container mit Hexaederzellen benutzt. Die Abmessungen der Container sind in **Tabelle 1** dargestellt.

			Container 1	Container 2	Container 3
Abmessung	m	**X**	1	1	2
		Y	0,5	1	0,5
		Z	1	1	2
Teilung		**X**	100	70	150
		Y	50	70	38
		Z	100	70	150
Kantenlänge	m	**X**	0,01	0,014285714	0,013333333
		Y	0,01	0,014285714	0,013157895
		Z	0,01	0,014285714	0,013333333
Fläche der Face	m^2	**XY**	0,0001	0,000204082	0,000175439
		YZ	0,0001	0,000204082	0,000175439
		XZ	0,0001	0,000204082	0,000177778
Volume einer Zelle	m^3		0,000001	2,91545E-06	2,33918E-06
Anzahl der gesamten Zelle			500000	343000	855000

Tabelle 1: Container für die Simulation

Um das Verhalten des Luftstrahls zu analysieren, werden die Ergebnisse gegenübergestellt und verglichen. Aus den Ergebnissen wird nur ein Modell für die endgültige Simulation ausgewählt. Es soll die besten Eigenschaften des Luftstrahls besitzen. Alle Ergebnisse werden in Anhang dokumentiert.

Die Luftmasse strömt aus dem Zellenmittelpunkt als Massenquelle heraus. Der Massenstrom wird in der UDF definiert. Die Luft strömt in alle Richtungen durch alle sechs Begrenzungsflächen der Zellen heraus. Die Impulsquelle zwingt die Luftmasse durch bestimmte Flächen der Zelle auszuströmen. In einer Zelle können maximal nur drei Impulsquellen definiert werden. Die Richtung des Luftstrahls in einer Zelle ergibt sich aus dem resultierenden Vektor der Geschwindigkeitskomponenten. Zunächst wird der Öffnungswinkel des Strahls in Abhängigkeit der Impulsquelle in 2D-Ebene untersucht.

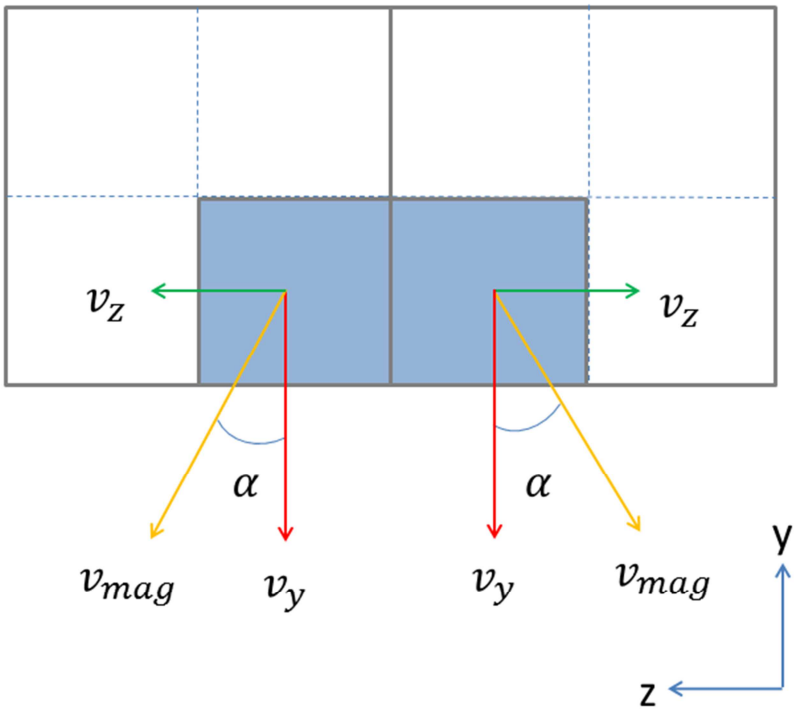


Abbildung 28: Der Halbwinkel eines Luftstrahls und die Geschwindigkeitskomponenten in YZ-Ebene

Mit dem vorgegebenem Öffnungswinkel 30° bzw. Halbwinkel 15° und der festgelegten Geschwindigkeit in y-Richtung wird die Geschwindigkeitskomponente v_z berechnet. Hier werden zwei Parameter die Geschwindigkeit v_y und der Halbwinkel α in dem UDF-Programm festgelegt. Die Geschwindigkeit und der Halbwinkel können beliebig auf Wunsch des Benutzers in dem UDF-Programm eingestellt werden. Die Geschwindigkeiten v_x und v_z werden durch UDF-Programm wie folgt berechnet:

$$v_x = v_y * tan\alpha$$

$$v_z = v_y * tan\alpha$$

Das Programm erhöht den Momentum solange, bis die Geschwindigkeiten v_x und v_z in der ausgesuchten Zelle sich einstellen. Damit der Momentum sich bis zum Erreichen der gewünschten Geschwindigkeiten weiter erhöht, wird ein Regler zusätzlich in dem Programm eingebaut. Der Momentum bekommt einen beliebigen Anfangswert in dem UDF-Programm. Er ist durch den Benutzer definiert. Dieser Wert wird dann in der Gleichung eingesetzt. Außerdem bekommt die Gleichung einen konstanten Faktor Lambda. Dieser Wert ist auch beliebig durch den Benutzer festgelegt. Der Wert lambda sagt aus, wie schnell oder langsam die zu erreichende

Geschwindigkeit konvergieren sein sollte. Der Auszug des UDF-Programms für den Momentum in x-Richtung ist in Abbildung 29 dargestellt.

```
160  /* Momentum in x-Richtung) */
161
162  real ccI1= 0; real I1_vx = 0.;
163  real ccI2= 0; real I2_vx = 0.;
164  real ccI3= 0; real I3_vx = 0.;
165  real ccI4= 0; real I4_vx = 0.;
166
167  DEFINE_SOURCE(xmom_source,c,t,dS,eqn)
168  {
169  real source = 0;
170  real lambda = 2000;
171  real ziel_x = ziel_y * tan(spray_half_angle/180*3.14159);
172
173  if (c == I1)
174      {source = ccI1+lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eq
175  else if (c == I2)
176      {source = ccI2-lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eq
177  else if (c == I3)
178      {source = ccI3-lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eq
179  else if (c == I4)
180      {source = ccI4+lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eq
181
```

Abbildung 29: Beispiel UDF-Programm für die Regelung des Momentums in x-Richtung mit der Zielgeschwindigkeit v_x

Hier als Beispiel betrachten wir eine Gleichung für den Momentum in x-Richtung in einer Zelle Nr.1 mit der eingebauten Regelung. Die Gleichung ist wie folgt definiert:

source = ccI1+lambda(ziel_x-fabs(C_U(c,t)))*

CCI1 ist der Momentum für die Zelle Nr.1 und zunächst als Null initialisiert. CCl1 wird in den nächsten Loop immer mit einem neuen Wert aus dem linken Term „source“ zugewiesen und überschrieben. „Ziel_x“ ist die zu erreichende Geschwindigkeit. Mit Hilfe eines UDF-Macro „fabs(C_U(c,t))“ kann die vorhandene Geschwindigkeit v_x in der Zelle ermittelt werden. Die Differenz aus der zu erreichenden Geschwindigkeit und der vorhandenen Geschwindigkeit wird mit dem Faktor Lamda multipliziert. Danach wird es dem vorhandenen Momentum ccl1 addiert. Der Vorgang wird in dem Loop durchlaufen bis die zu erreichenden Geschwindigkeiten in der Zelle sich stabil eingestellt haben. Die Ergebnisse mit der Regelung kann man in **Abbildung 30** für die Geschwindigkeit 10,7 m/s und **Abbildung 31** für die Geschwindigkeit 40 m/s.

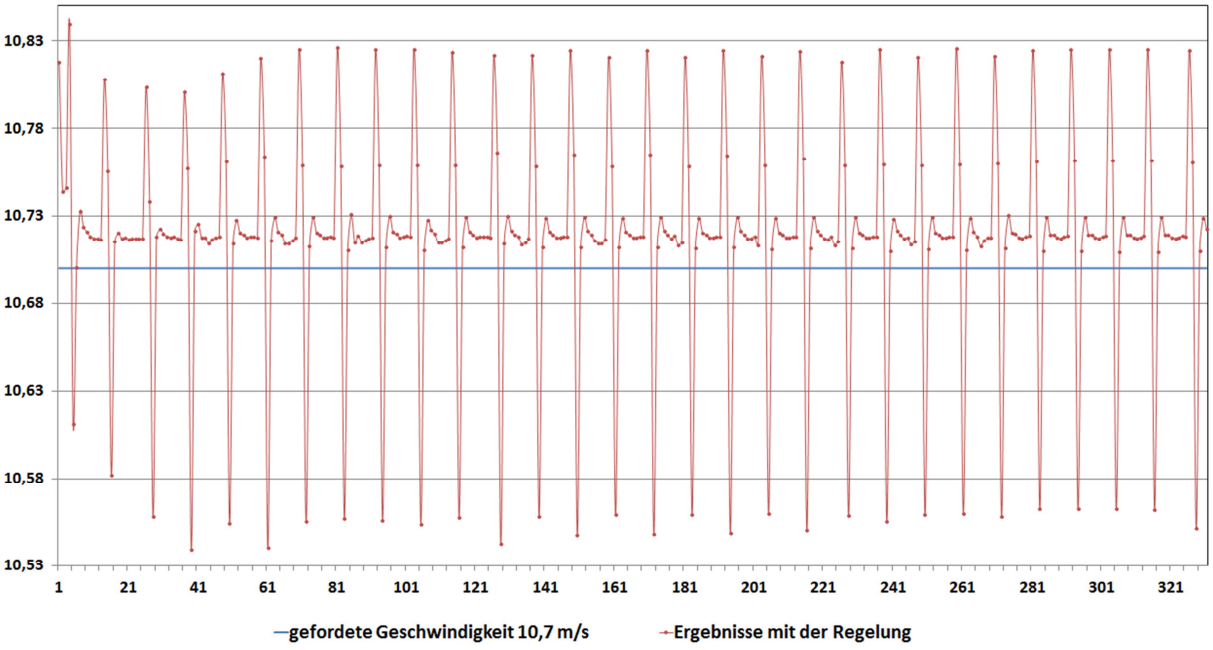


Abbildung 30: Regelung für den Momentum bei geforderter Geschwindigkeit 10,7 m/s

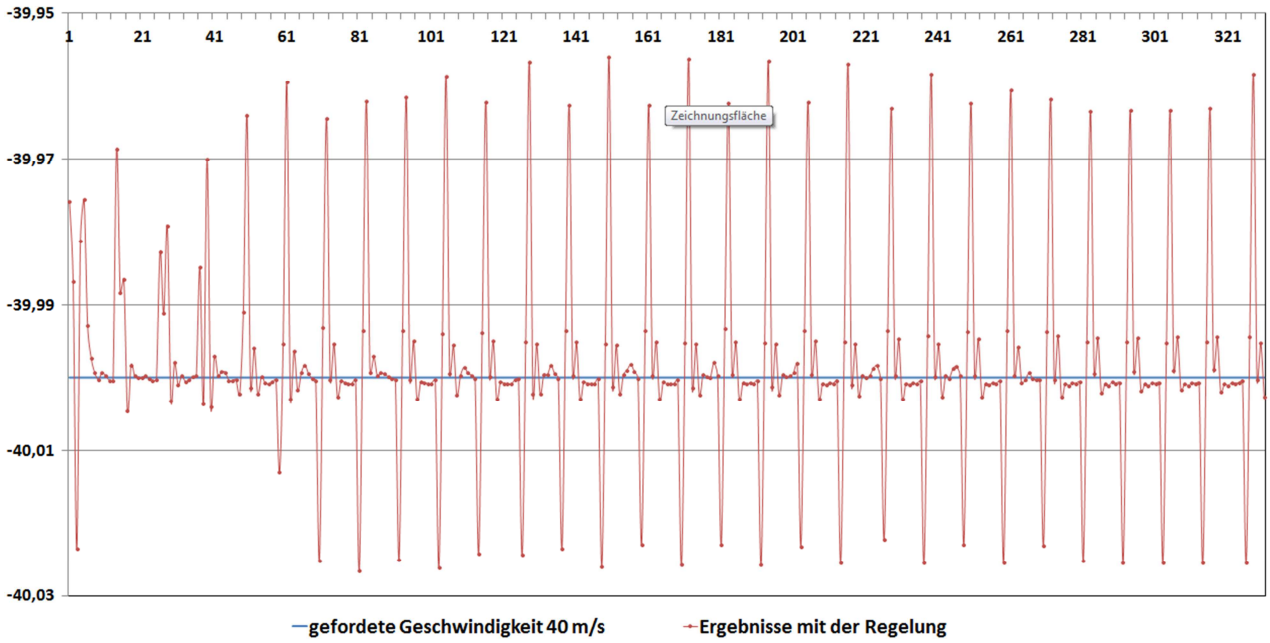


Abbildung 31: Regelung für den Momentum bei geforderter Geschwindigkeit 40 m/s

Um herauszufinden welche maximale Geschwindigkeiten in x- und y-Komponenten aus der Zelle in die jeweilige Richtung sich einstellen, wird zunächst bei einer Testsimulation nur die Massenquelle verwendet. Diese Information wird für die Einschätzung der Geschwindigkeit v_y für das UDF-Programm benötigt. Die Rechnung läuft in einem stationären Zustand bei ca. 100 Iterationen. Aus den Ergebnissen wird dann geschätzt, welche Impulsmomente in die Sourceterme in der

UDF erforderlich sind, um die gewünschten Geschwindigkeiten und den Öffnungswinkel des Strahls zu erreichen. Der Die erforderlichen Impulsquellen werden in UDF definiert. Die Anzahl der Zellen für die künstliche Düse wird bis maximal vier Zellen vereinbart. Der Grund warum wir nur maximal vier Zellen für die künstliche Düse verwenden wird im Kapitel 4.4.6 Sondermodelle erklärt.

Die Geschwindigkeitskomponenten werden zunächst für die Simulation der gesamten Modelle 40 m/s in die y-Richtung angenommen. Der Halbwinkel beträgt ca. 15° bzw. der gesamte Kegelwinkel 30°. Diese Ang aben werden in dem UDF-Programm festgelegt. Die Zielgeschwindigkeit in x-Richtung wird durch UDF-Programm berechnet. Sie beträgt 10,7 m/s.

4.2 Zwei-Zellen-Modelle

Die Zwei-Zellen-Modelle bestehen jeweils aus der Kombination einer Ursprungszelle und einer Nachbarzelle. Die Methode um die beiden Zellen in dem Berechnungsgebiet zu identifizieren ist bereits im Kapitel 3 ausführlich erklärt worden. In den zwei Zellenmodellen werden verschiedene Richtungen des Momentums ausprobiert. Für diese Modelle gibt es insgesamt zwei Varianten, und zwar Variante 0 und 5. Wobei die Variante 5 noch in Variante 5a und 5b unterteilt wird.

4.2.1 Variante 0 – Zwei-Zellen-Modell

In dieser Variante werden die Momentum nur in y-Richtung gegeben. Aus den Zellmittelpunkten strömt die Luft heraus, siehe Abbildung 32. Es sind folgende Einstellungen in UDF vordefiniert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: 40m/s
- Momentum: -3000
- Öffnungswinkel: nicht angegeben, da die Quelle nur in y-Richtung vorhanden sind.

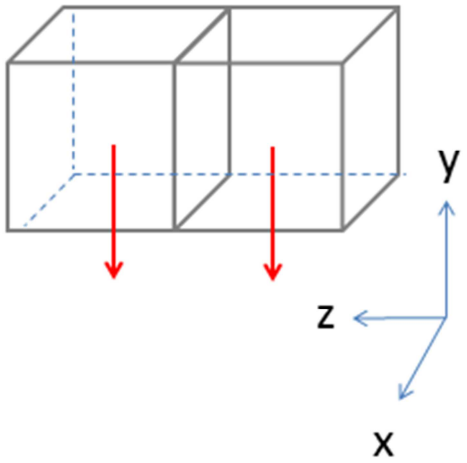


Abbildung 32: Zwei-Zellen-Modell Variante 0

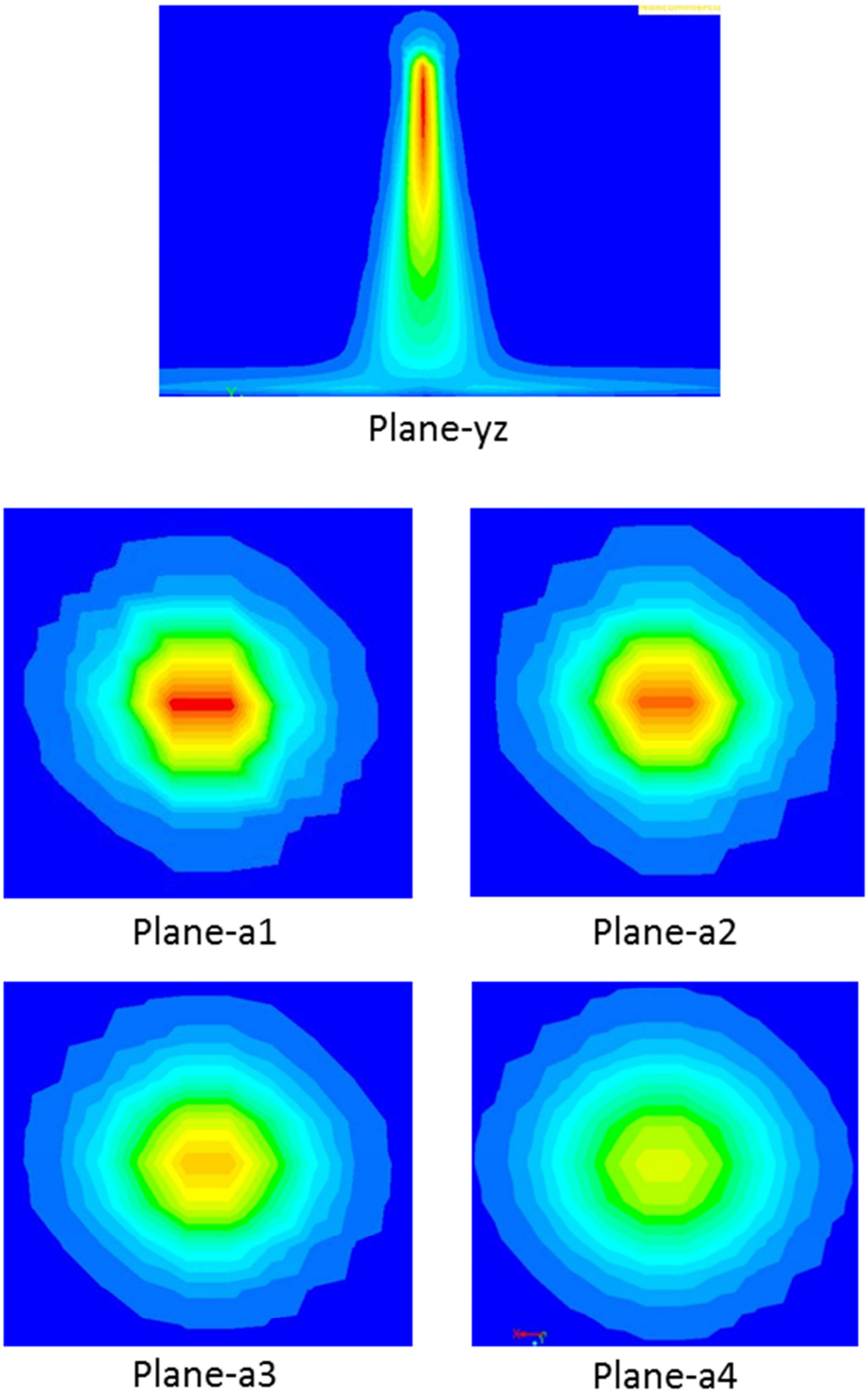


Abbildung 33: Ergebnisse Variante 0 (stationär)

4.2.2 Variante 5a – Zwei-Zellen-Modell

Die Variante 5a und 5b unterscheiden sich nur in der Richtung der in UDF gegebenen Momentum. Sowohl in Variante 5a als auch in Variante 5b werden zwei Momentum in der jeweiligen Zelle eingefügt. Es sind folgende Einstellungen für Variante 5a in UDF definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit : $\begin{pmatrix} 0 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 0 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

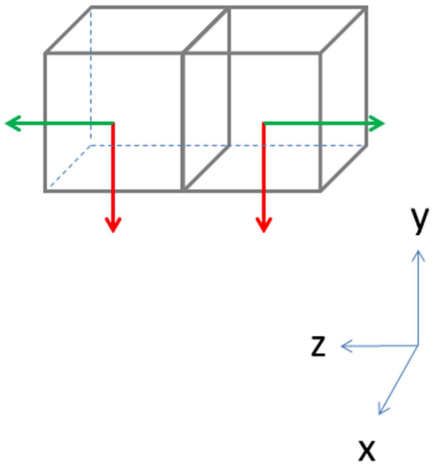
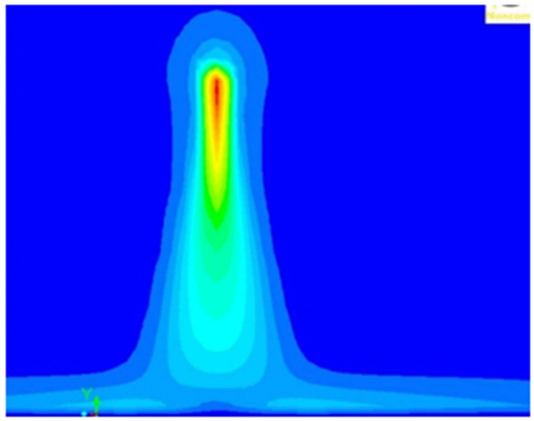
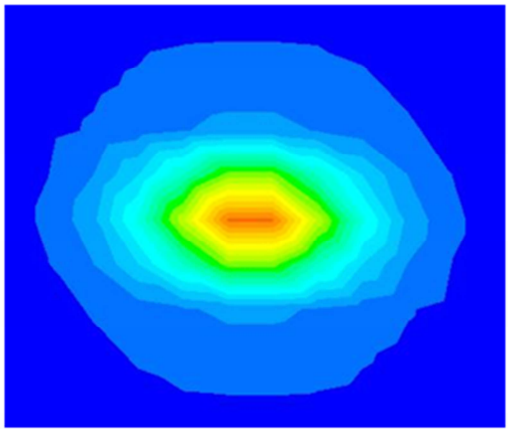


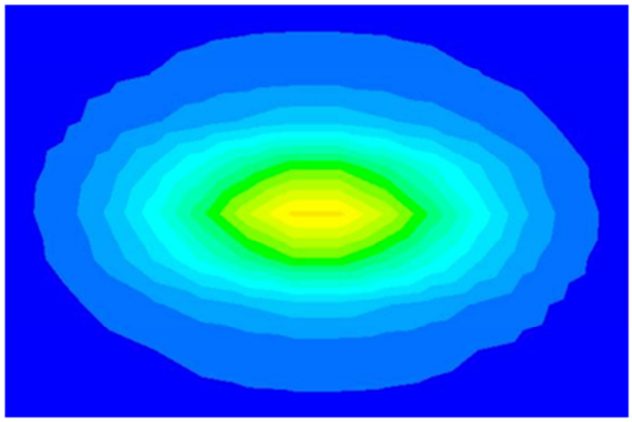
Abbildung 34: Zwei-Zellen-Modell - Variante 5a



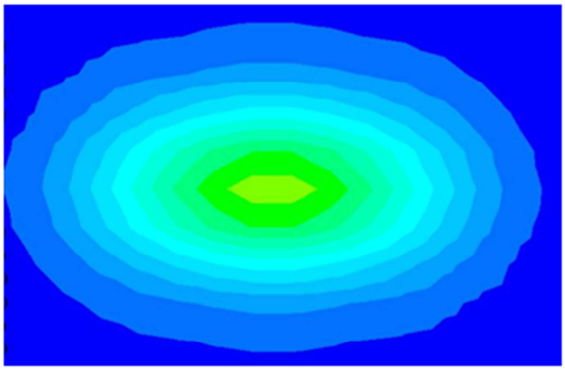
Plane-yz



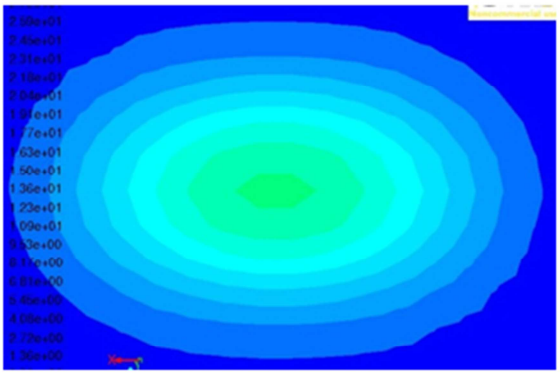
Plane-a1



Plane-a2



Plane-a3



Plane-a4

Abbildung 35: Ergebnisse Variante 5a (stationär)

Aus den Ergebnissen kann man den Luftstrahl für die Variante 5a erkennen. Der Strahl ist zwar kegelförmig ausgebildet, aber von den Schnittebenen a1-a4 her ist deutlich zu erkennen, dass der Strahl nicht rund ist. Der Strahl besitzt eine elliptische Form.

4.2.3 Variante 5b – Zwei-Zellen-Modell

Die Variante 5b hat die gleiche Zellanordnung wie Variante 5a, nur die Momentum in z-Richtung sind umgekehrt. Aus dem gegebenen Momentum lässt sich die Richtung der resultierenden Vektoren andeuten, siehe Abbildung 36. In dieser Variante kreuzen sich die resultierenden Geschwindigkeitsvektoren. Folgende Einstellungen werden für Variante 5b in UDF vordefiniert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: $\begin{pmatrix} 0 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 0 \\ -3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

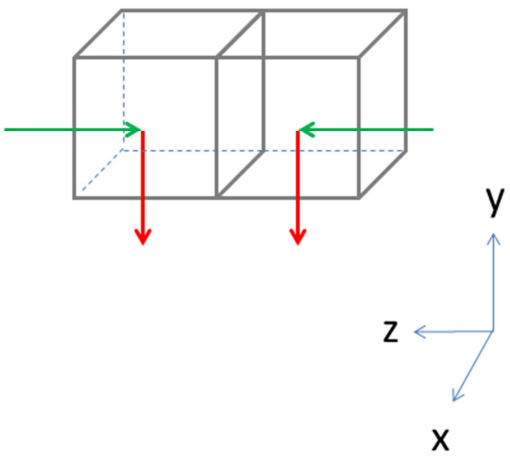


Abbildung 36: Zwei-Zellen-Modell - Variante 5b

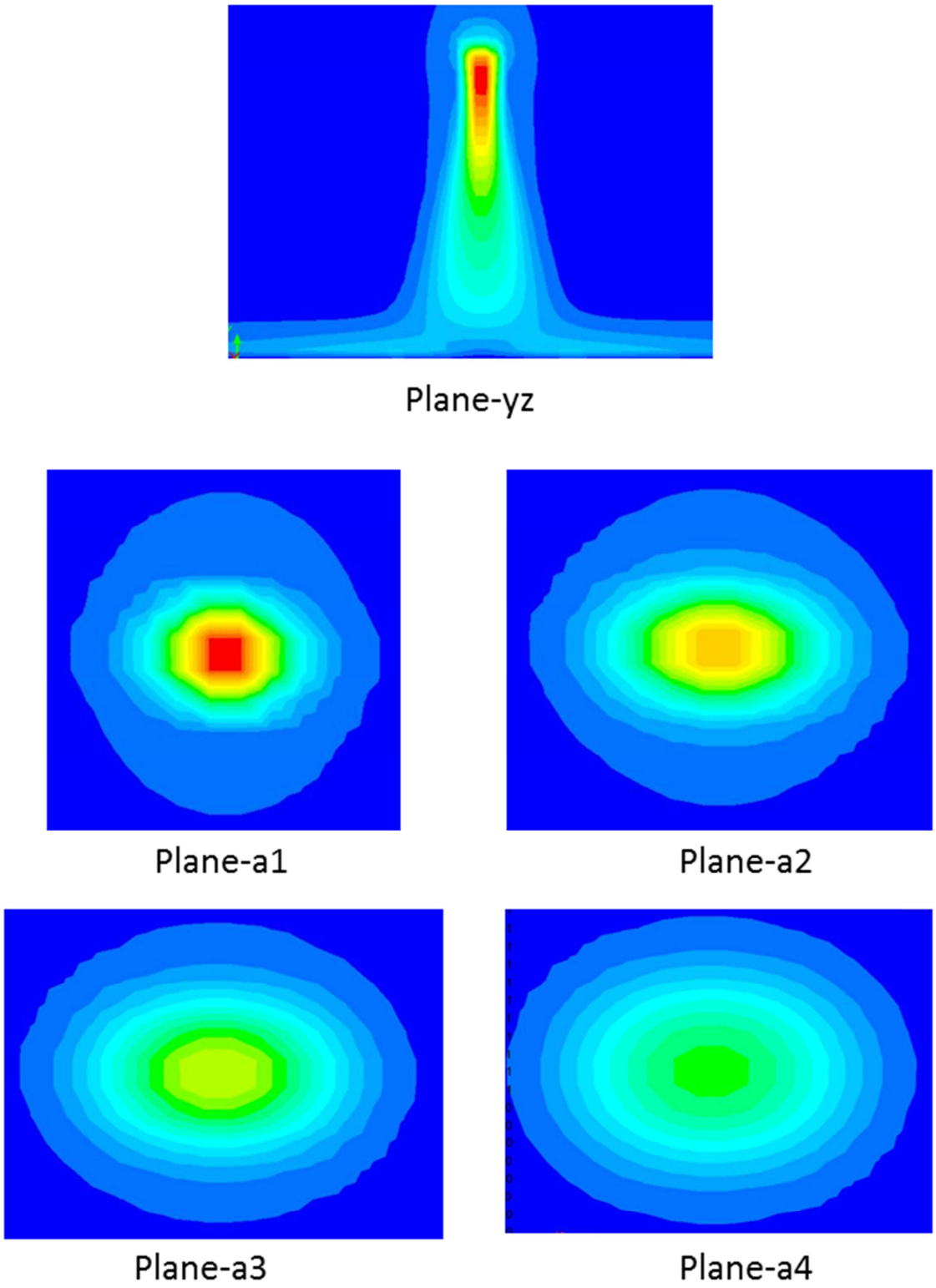


Abbildung 37: Ergebnisse Variante 5b (stationär)

4.3 Drei-Zellen-Modelle

4.3.1 Variante 4 – Drei-Zellen-Modell

Das Drei-Zelle-Modell ist die Erweiterung von den Zwei-Zellen-Modellen. Das Modell wird als Variante 4 genannt. Es sind folgende Einstellungen für Variante 4 in UDF definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit : $\begin{pmatrix} 0 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 0 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

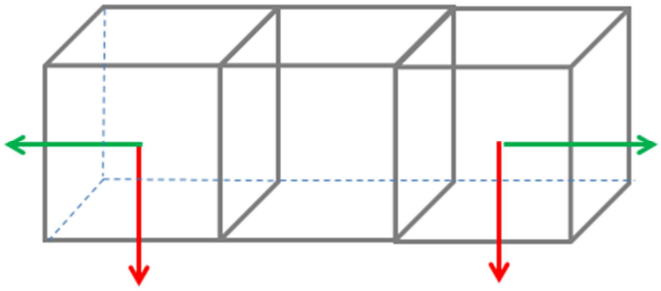
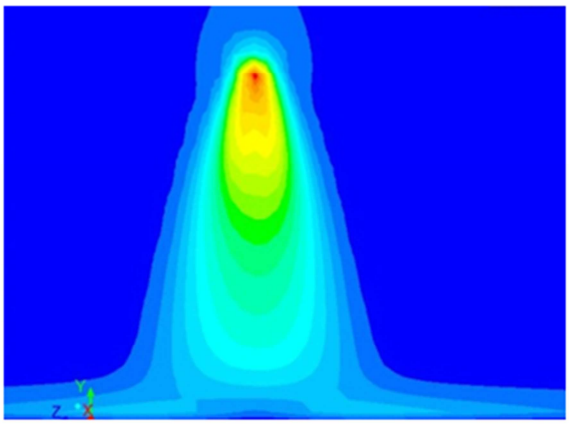
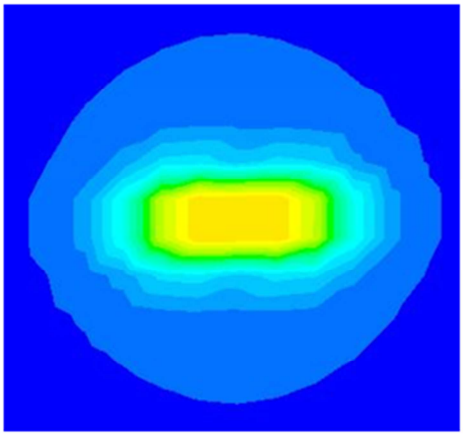


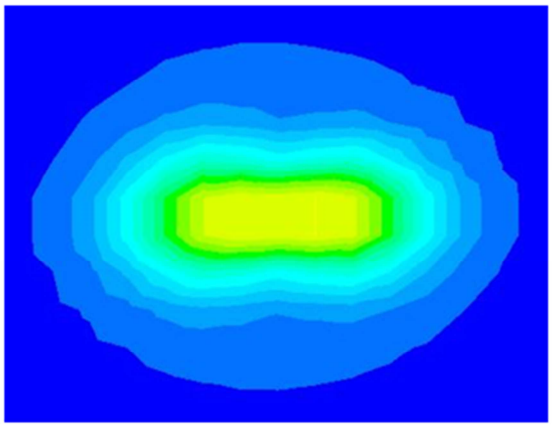
Abbildung 38: Drei-Zellen-Modelle – Variante 4



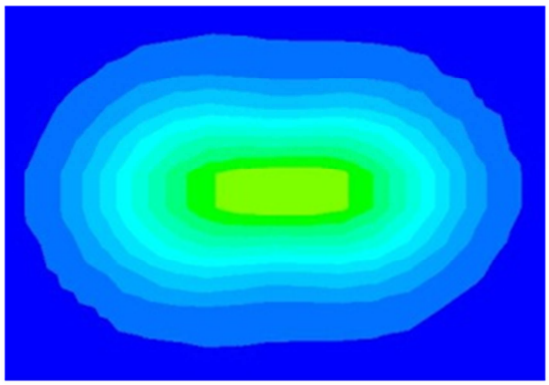
Plane-yz



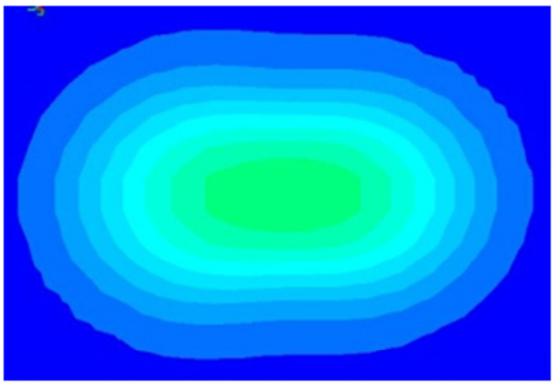
Plane-a1



Plane-a2



Plane-a3



Plane-a4

Abbildung 39: Ergebnisse Variante 4 (stationär)

4.4 Vier-Zellen-Modelle

Mit den Vier-Zellen-Modellen werden verschiedene Momentum-Einstellungen ausprobiert. In der Variante 1a und 1b werden jeweils zwei Momentum in eine Zelle gegeben. Für die Varianten 2a bis 2c bzw. Vier-Zellen-Modelle werden den jeweiligen Zellen drei Impulsquellen eingefügt.

4.4.1 Variante 1a – Vier-Zellen-Modell

Die Variante 1a besteht aus zweier Zellen, die in zwei Ebenen übereinander gestapelt sind. Aus den zwei oberen Zellen werden die Momentum in y-Richtung und an der linken Zelle in negative x-Richtung bzw. rechten Zelle in positive x-Richtung hinzugefügt. In den beiden unteren Zellen werden die Momentum in y-Richtung und die linke Zelle in positive z-Richtung bzw. rechten Zelle in negative z-Richtung hinzugefügt. Das Modell ist in Abbildung 40 dargestellt. Die Einstellungen für Variante 1a sind wie folgt definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: $\begin{pmatrix} 0 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum für die beiden oberen Zellen: $\begin{pmatrix} 3000 \\ 3000 \\ 0 \end{pmatrix}$
- Momentum für die beiden unteren Zellen: $\begin{pmatrix} 0 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

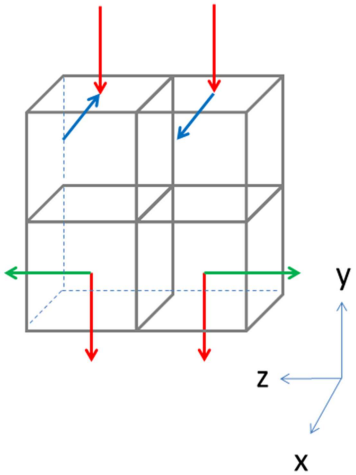


Abbildung 40: Vier Zellenmodell - Variante 1a

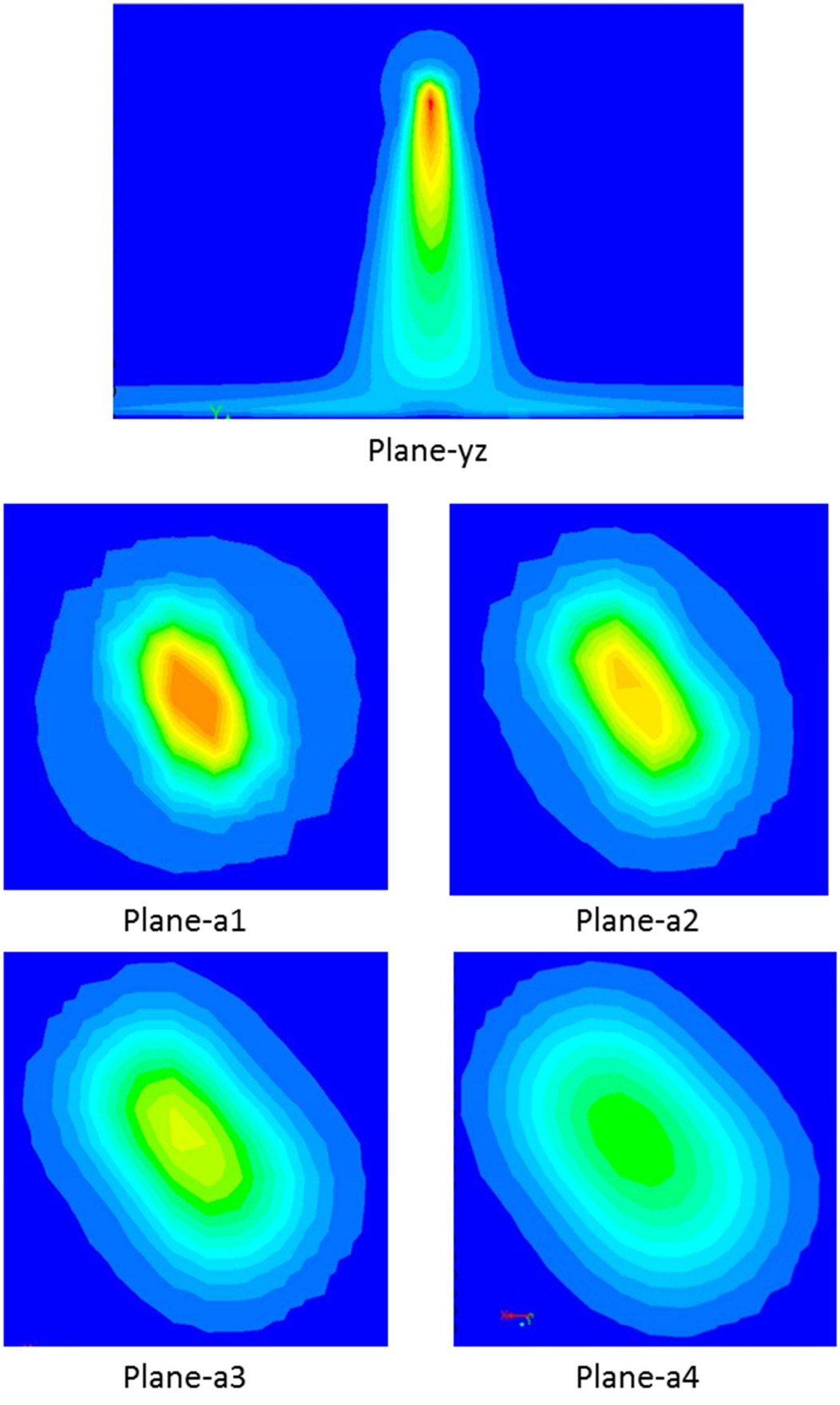


Abbildung 41: Ergebnisse Variante 1a (stationär)

In der Abbildung 41 kann man durch die Schnittebene sehen, dass der Luftstrahl eine elliptische Form hat und etwa 45-60° von der Z -Achse abweicht.

4.4.2 Variante 1b – Vier-Zellen-Modell

Die Zellenanordnung für diese Variante unterscheidet sich nicht von Variante 1a. Nur die Momentum in unteren Zellen und oberen Zellen, die in Variante 1a sind, werden vertauscht, siehe Abbildung 42.

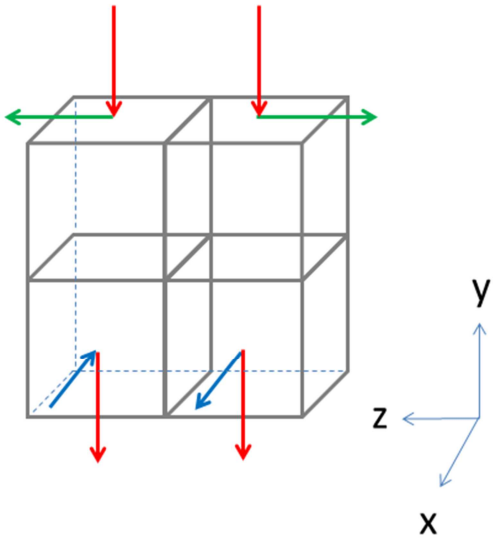


Abbildung 42: Variante 1b – Vier-Zellen-Modell

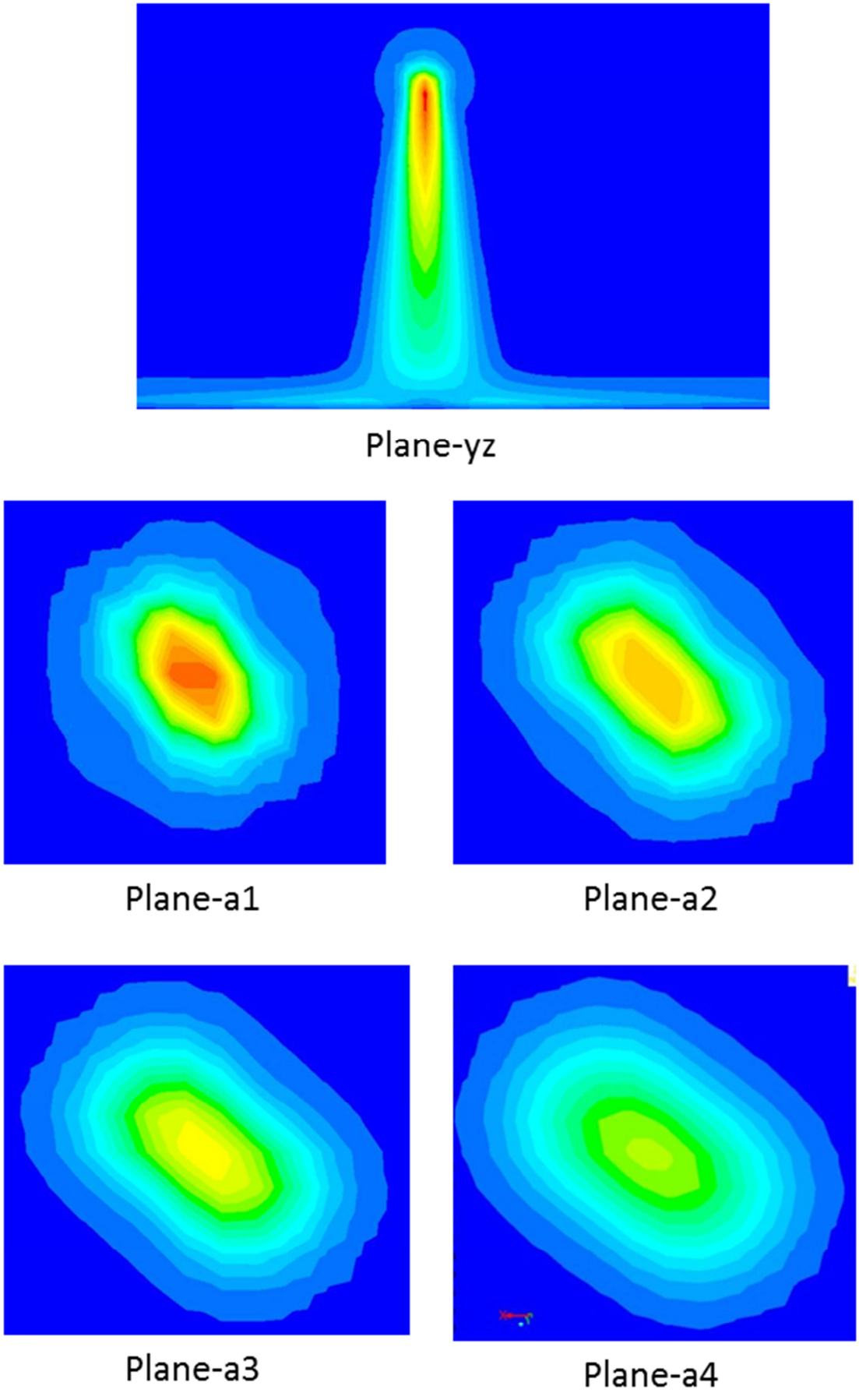


Abbildung 43: Ergebnisse Variante 1b (stationär)

4.4.3 Variante 2a – Vier-Zellen-Modell

Die Einstellungen für das Vier-Zellen-Modell Variante 2a sind wie folgt definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: $\begin{pmatrix} 10{,}7 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 3000 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

Die Richtung für die einzelnen Komponenten in den jeweiligen Zellen wurde mit entsprechendem Vorzeichen in dem UDF-Programm vorgegeben.

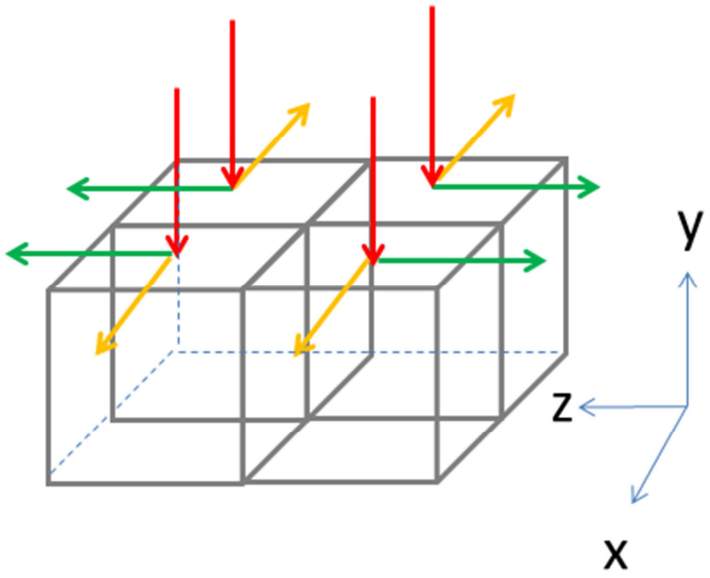
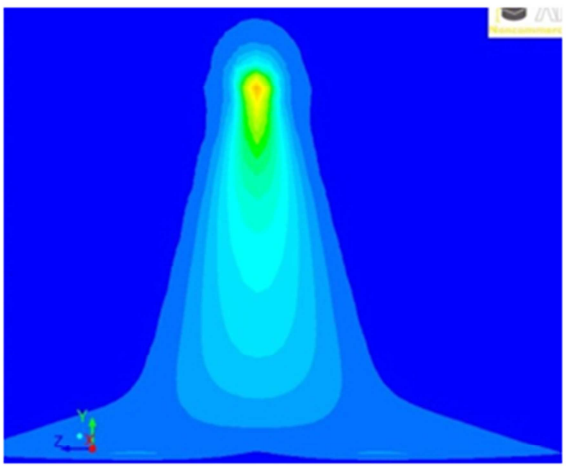
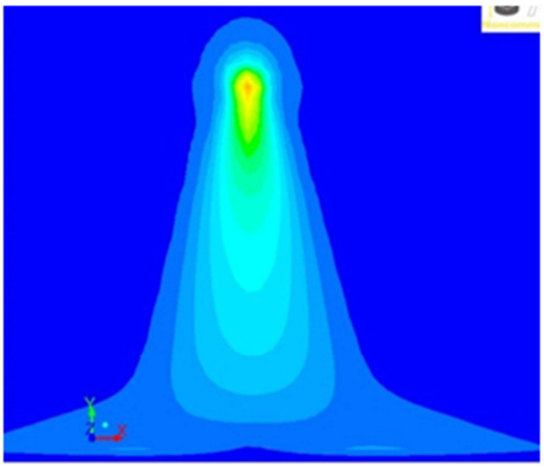


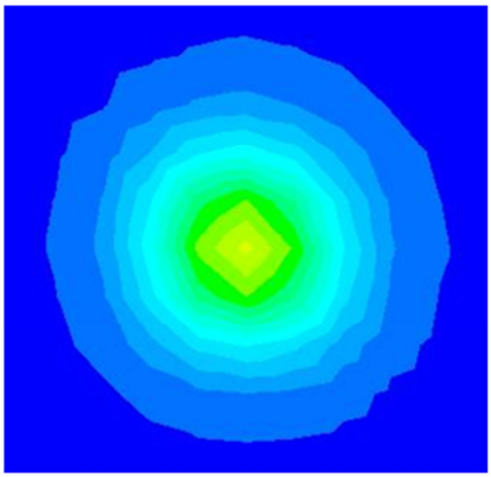
Abbildung 44: Variante 2a – Vier-Zellen-Modell



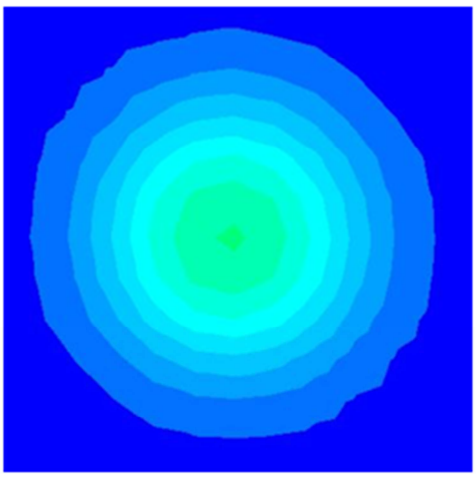
Plane-yz



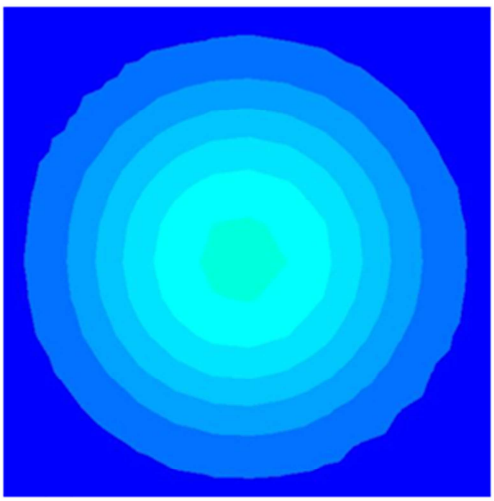
Plane-yx



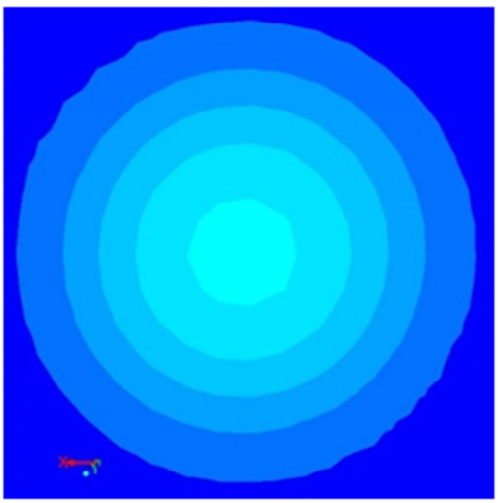
Plane-a1



Plane-a2



Plane-a3



Plane-a4

Abbildung 45: Ergebnisse Variante 2a (stationär)

4.4.4 Variante 2b – Vier-Zellen-Modell

Die Einstellungen für das Vier-Zellen-Modell Variante 2b sind wie folgt definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: $\begin{pmatrix} 10{,}7 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 3000 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

Die Richtung für die einzelnen Komponenten in den jeweiligen Zellen wurde mit entsprechendem Vorzeichen in dem UDF-Programm vorgegeben.

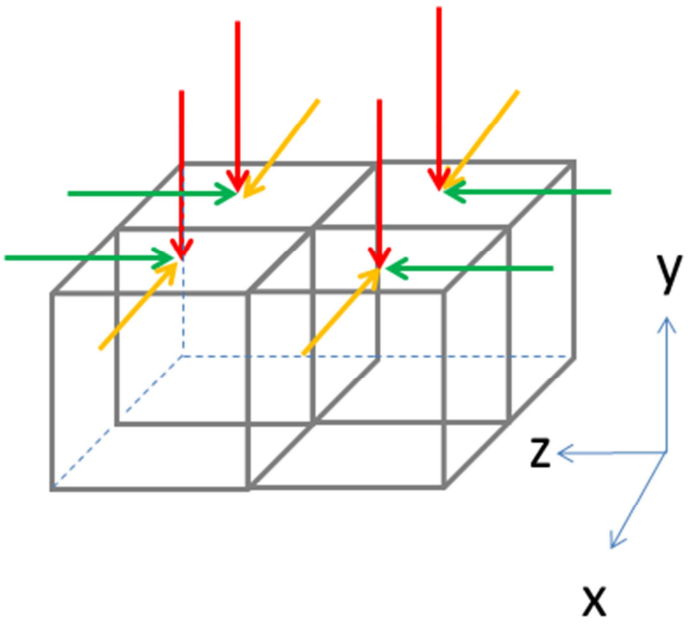


Abbildung 46: Variante 2b – Vier-Zellen-Modell

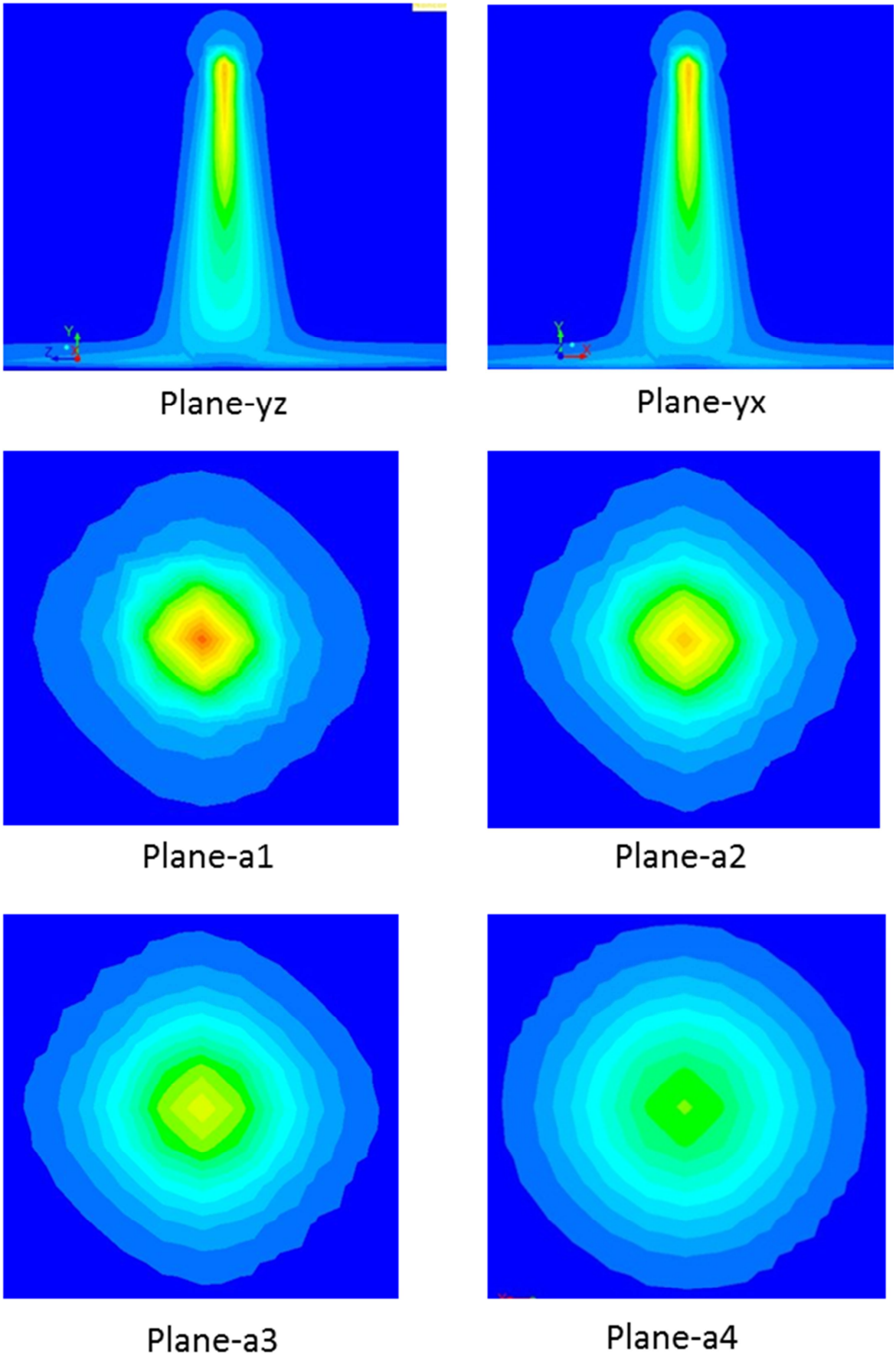


Abbildung 47: Ergebnisse Variante 2b (stationär)

4.4.5 Variante 2c – Vier-Zellen-Modell

Die Einstellungen für das Vier-Zellen-Modell Variante 2c sind wie folgt definiert:

- Masssource: 0,006125 Kg/s
- Geforderte Luftgeschwindigkeit: $\begin{pmatrix} 10{,}7 \\ 40 \\ 10{,}7 \end{pmatrix}$ m/s
- Momentum: $\begin{pmatrix} 3000 \\ 3000 \\ 3000 \end{pmatrix}$
- Öffnungswinkel: 30°

Die Richtung für die einzelnen Komponenten in den jeweiligen Zellen wurde mit entsprechendem Vorzeichen in dem UDF-Programm vorgegeben.

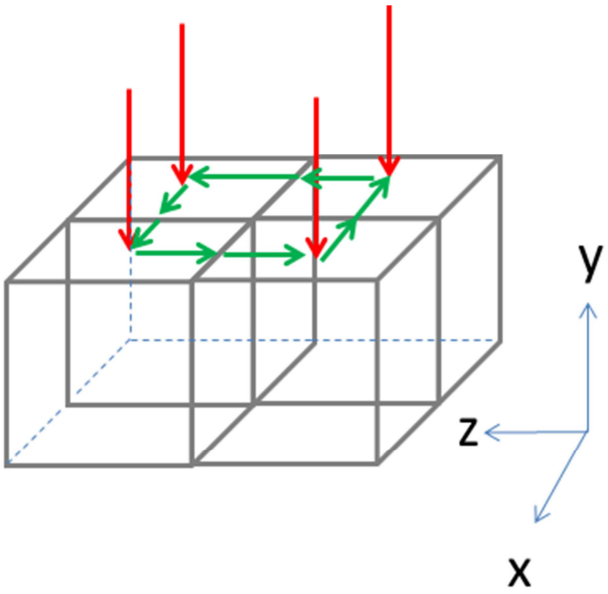
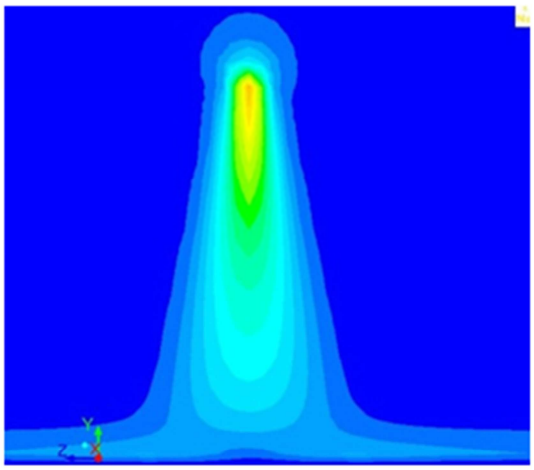
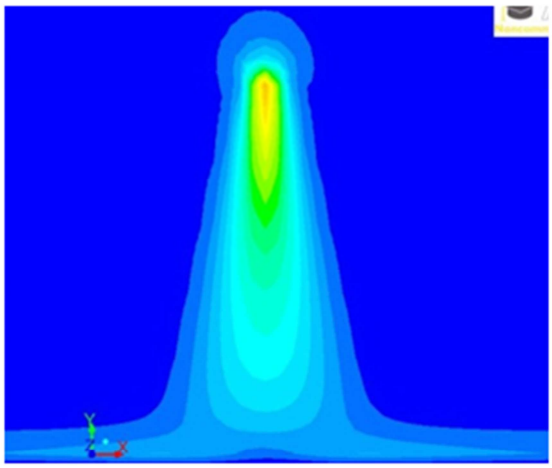


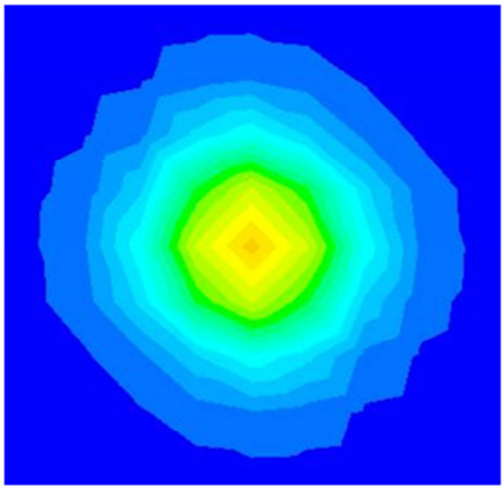
Abbildung 48: Variante 2c – Vier-Zellen-Modell



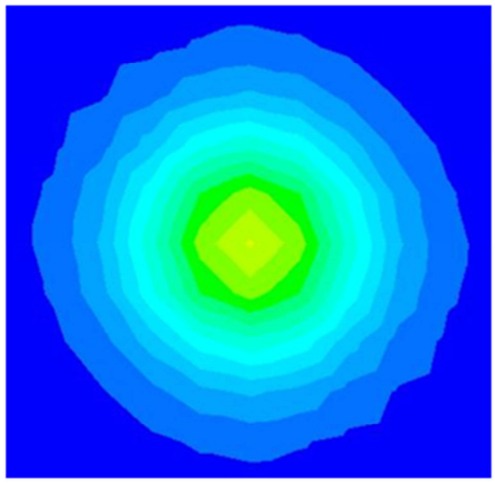
Plane-yz



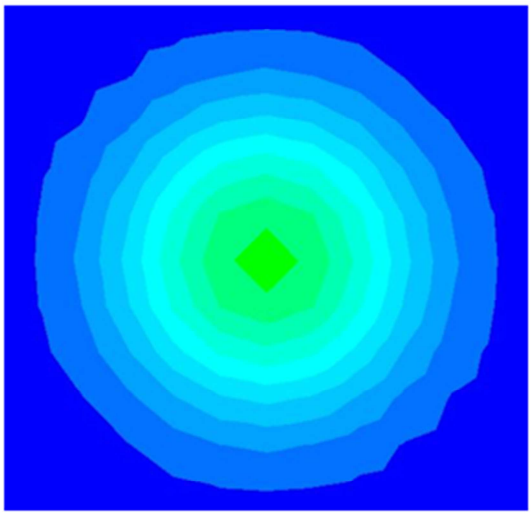
Plane-yx



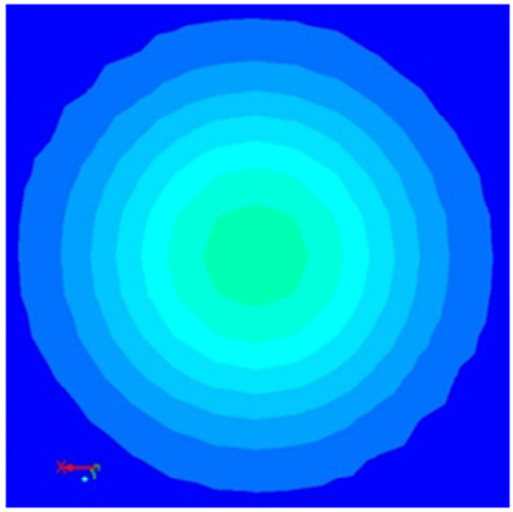
Plane-a1



Plane-a2



Plane-a3



Plane-a4

Abbildung 49: Ergebnisse Variante 2c (stationär)

4.4.6 Sondermodell

Zu diesem Modell gehört die Variante 3. Die Anordnung der Zelle kann man in **Abbildung 50** sehen. Die Variante 3 wurde für die Simulation nicht getestet.

Der Grund dafür ist, dass mit fünf Zellen der repräsentative Durchmesser schon viel größer ist als der tatsächliche Düsendurchmesser. Außerdem hat die Anordnung der Zelle eine Lücke an der Seite. Dies könnte dazu führen dass der erzeugte Luftstrahl nicht kreisförmig ist. Um diese Lücke zu schließen sind eventuell weitere Zellen erforderlich. Allerdings sind mehr als fünf Zellen für die künstlichen Düsen nicht sinnvoll. Der Rechneraufwand nimmt dabei auch zu. Mit vier Zellen stößt man schon an die Grenze der möglichen Zellenkombinationen.

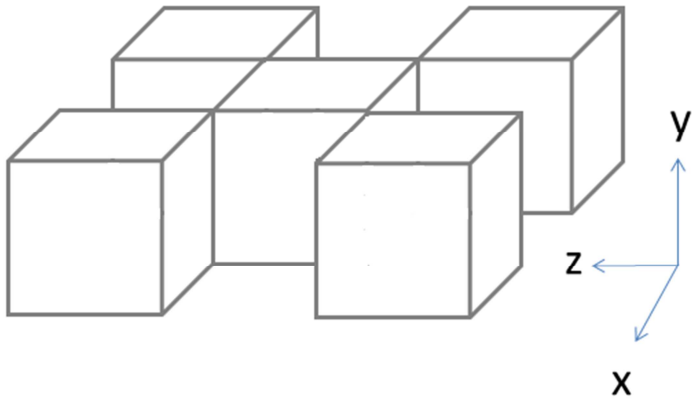


Abbildung 50: Fünf-Zellen-Modell – Sondermodell

4.5 Der ausgewählte Luftstrahl

Aus den verschiedenen Varianten werden die Ergebnisse in DIN-A3 Blatt (siehe Anhang) dargestellt und gegenübergestellt. Für Die Beurteilung der Sprayeigenschaften werden verschiedene Schnittebenen erzeugt. Die Variante 2a bis 2c haben die besten Sprayeigenschaften. Hier wird die Variante 2a in Kapitel 4.4.3 für das beste Ergebnis ausgewählt. Plane YZ und YX sind die symmetrische Schnittebene, die genau durch den Zellmittelpunkt gelegt. Somit für die instationären Simulationen wird nur die Variante 2a als Standardmodell verwendet.

4.6 Partikelsimulation

Die Partikelsimulation wird in einem stationären Zustand simuliert. Hier wird keine Luft in der Simulation verwendet. In Ansys Fluent wird die Injektorposition eingestellt und die Anzahl der Partikelmenge eingegeben. Die Ergebnisse kann man in Abbildung 51 sehen. In Bilder sind ja transient Modus in Ansys Fluent eingestellt. Doch in dem UDF-Programm wird die Bewegungsgleichung für die Simulation deaktiviert.

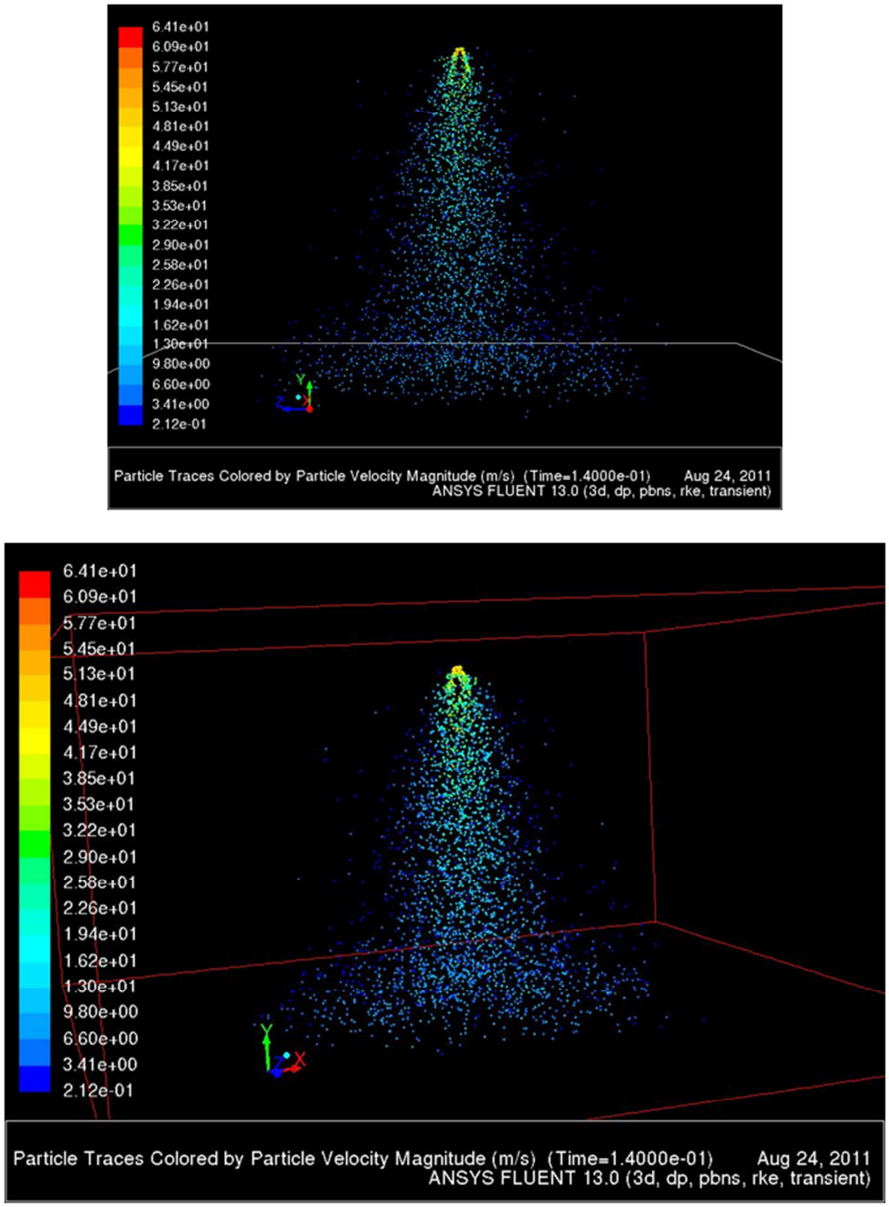


Abbildung 51: Partikelsimulation ohne Luftstrahl (stationär)

5 Partikel- und Sprühluftsimulation

In diesem Kapitel werden die Ergebnisse der instationären Simulation für den Luftstrahl und Partikel vorgelegt. Die Simulationen werden parallel ausgeführt.

5.1 Sprühluftsimulation (instationär)

Das Vier-Zellen-Modell bzw. die Variante 2a wird für die instationäre Simulation benutzt. Die Ergebnisse des Luftstrahls sind in **Abbildung 52** und **Abbildung 53** dargestellt. In der sieht man dass der Luftstrahl sich noch bildet. Bei t = 0,125s hat er sich vollständig gebildet.

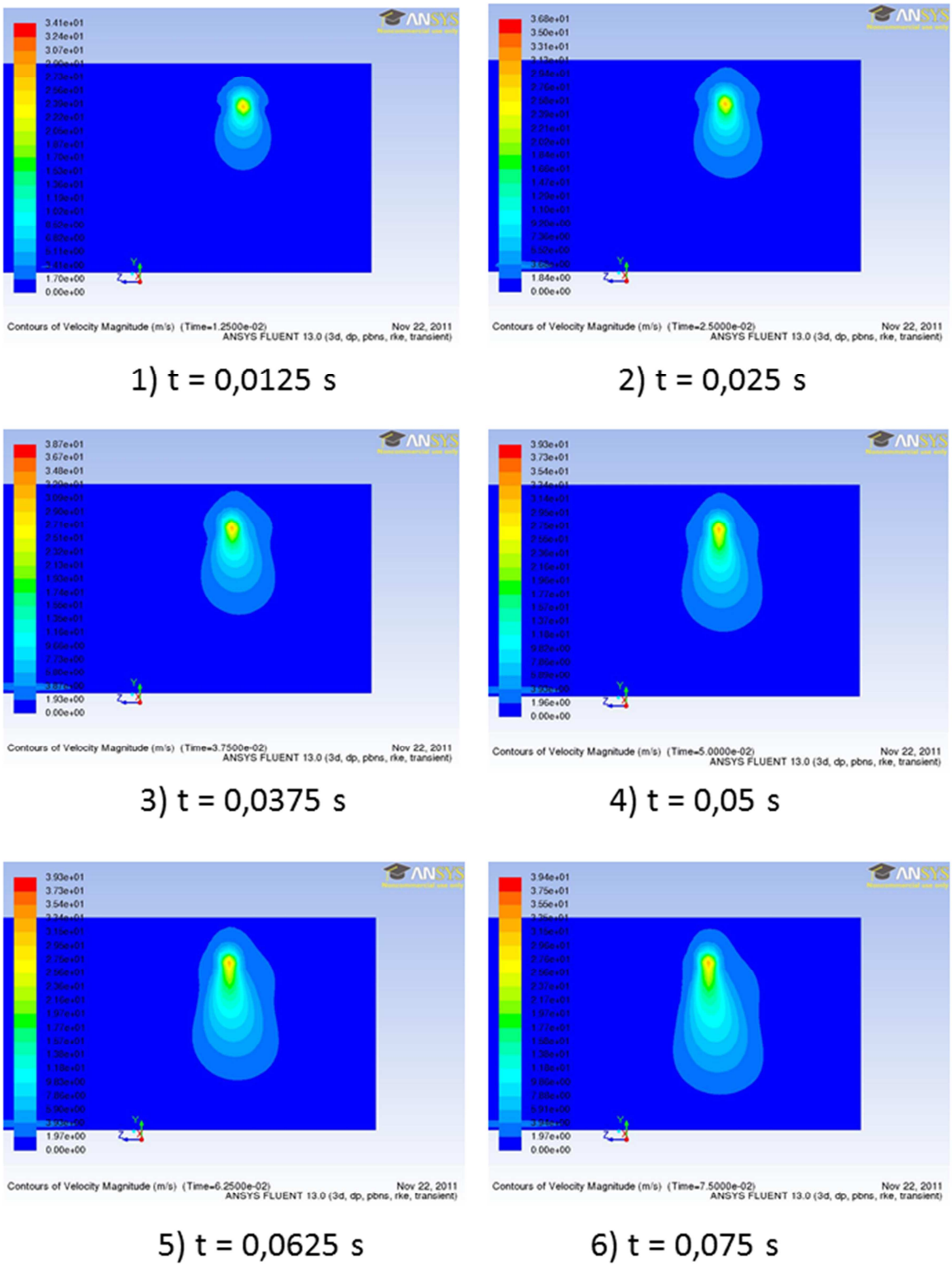
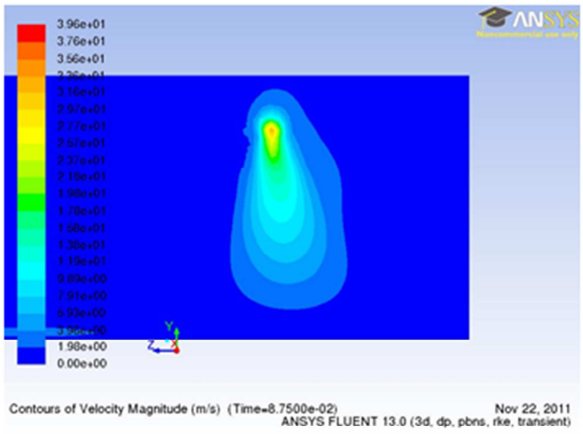
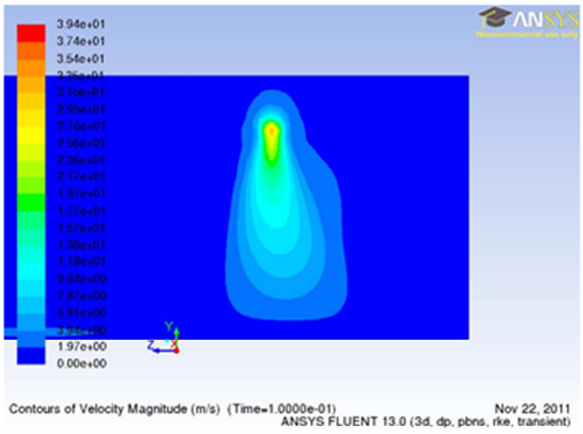


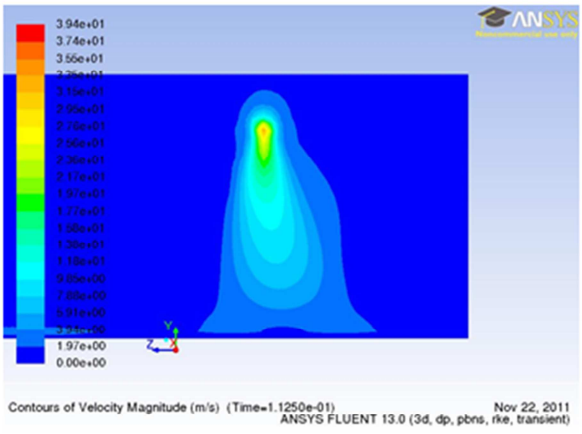
Abbildung 52: Luftstrahl instationäre Simulation (a)



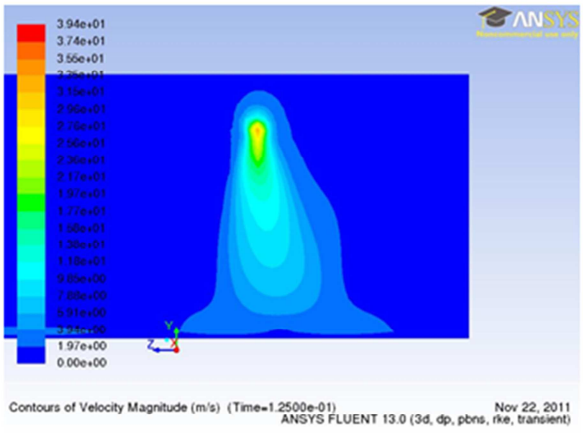
7) t = 0,0875 s



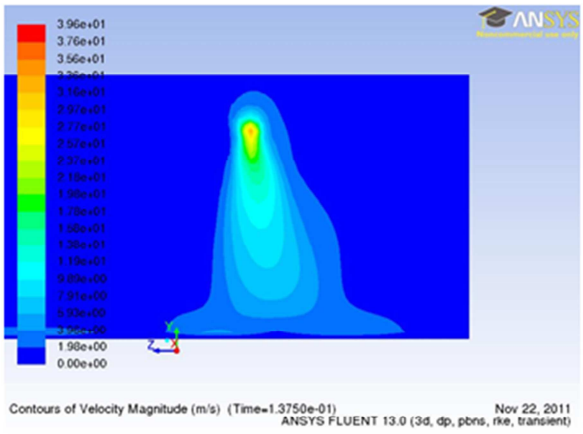
8) t = 0,1 s



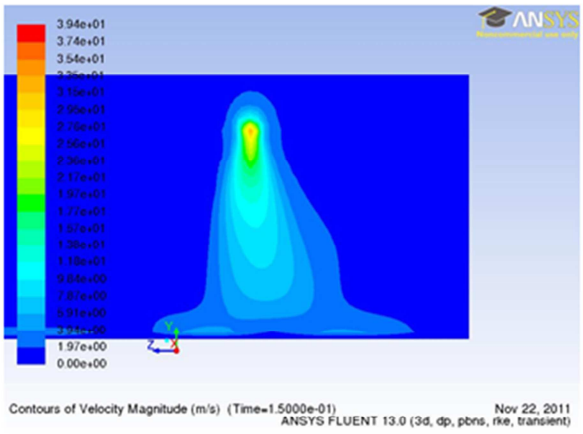
9) t = 0,1125 s



10) t = 0,125 s



11) t = 0,1375 s



12) t = 0,15 s

Abbildung 53: Luftstrahl instationäre Simulation (b)

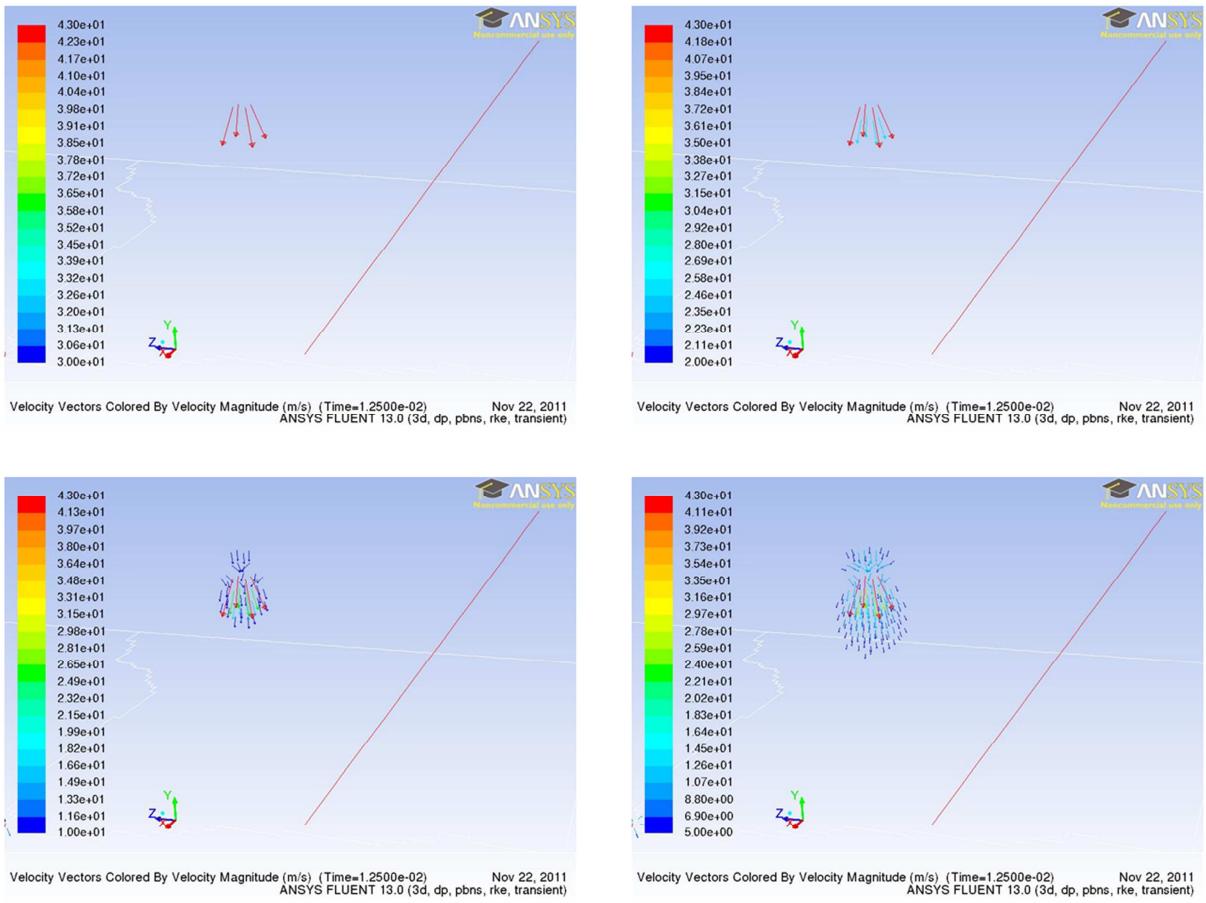
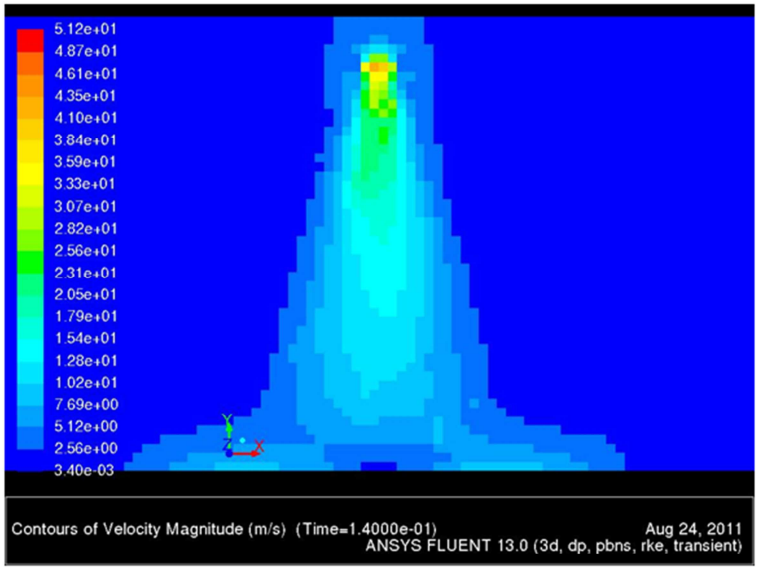


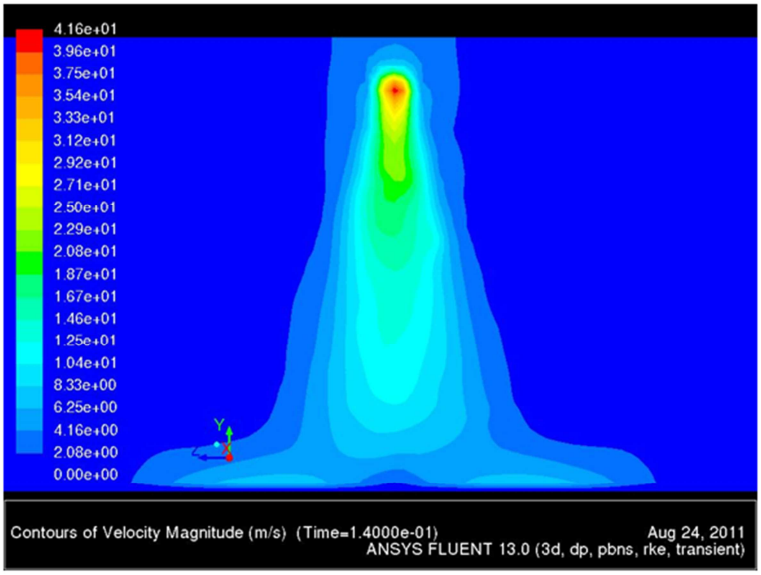
Abbildung 54: Aufbau des Luftstrahls instationäre Simulation Vektorplot

In der Abbildung 54 kann man die Geschwindigkeitsvektoren aus den Zellen genau sehen. Es sind die resultierenden Vektoren aus drei Geschwindigkeitskomponenten. Zunächst gehen die vier Geschwindigkeitsvektoren aus den ausgesuchten Zellen heraus. Während der Simulation bilden sich immer mehr Vektoren oberhalb der ausgesuchten Zelle aus. Denn die Luft wird von oben aus den anderen Zellen gesaugt.

Mit der funktionierenden Partikelsimulation soll der Luftstrahl zusammen in einer Simulation getestet werden. Hier wird die Partikel-Luft-Kopplung aktiviert. In der **Abbildung 55** kann man sehen dass die Form des Luftstrahls etwas verzerrt ist. Der Grund dafür ist das, dass die Partikel-und Luftinteraktion aktiv ist. Man kann nochmal die symmetrischen Eigenschaften der Simulation in den vier Schnittebenen in **Abbildung 56** sehen.



Cell-Base Plot t = 0,14 s



Contour Plot t = 0,14 s

Abbildung 55: Luftstrahl mit Partikeln (instationär) Contour Plot yz-Ebene

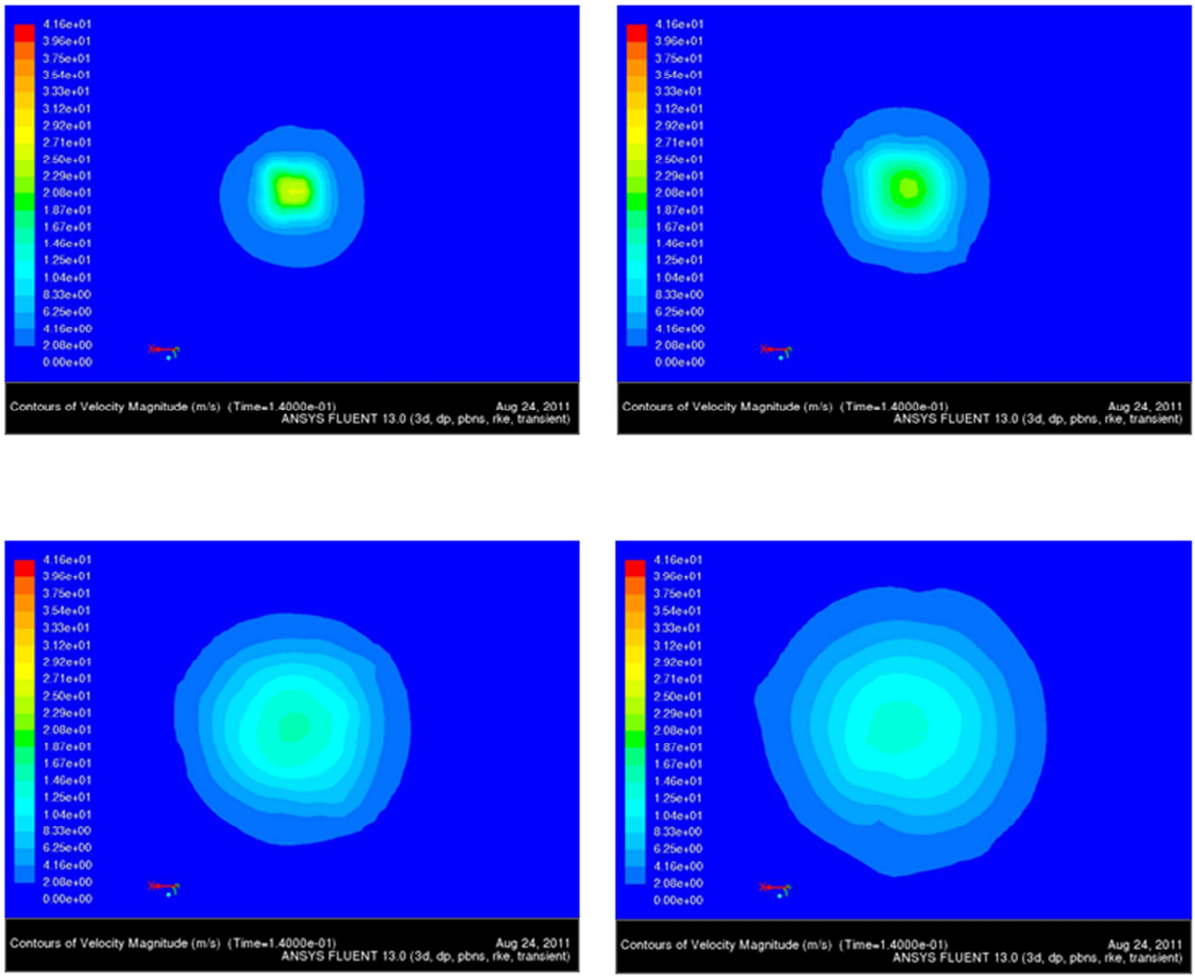


Abbildung 56: Luftstrahl mit Partikeln (instationär) Schnitt Plane a1-a4

6 Zusammenfassung und Ausblick

Ziel dieser Diplomarbeit waren ein kegelförmiger Luftstrahl und die Partikelsimulation mit Hilfe eines DPM-Modells in Ansys Fluent zu entwickeln. Der Luftstrahl wurde mit Hilfe mehrerer Impuls- und Massenquellen erzeugt. Durch die beliebige Vorgabe einer Injektorposition in den Raumkoordinaten sollte das Programm im Berechnungsgebiet eine künstliche Düse erzeugen. Mit Hilfe des UDF-Programms wurden die spezifisch ausgesuchten Zellen für die künstliche Düse festgelegt. Um eine passende Düse für das Spray auswählen zu können wurden verschiedene Varianten der künstlichen Düse entwickelt und getestet. Aus diesen Zellen bildete sich der Luftstrahl. Die Luftstrahl- und Partikelsimulation soll kegelförmig und symmetrisch sein.

Die Luftstrahl- und Partikelsimulation im stationären Zustand wurden getrennt durchgeführt. Aus den Ergebnissen der verschiedenen Varianten der künstlichen Düse wurde festgestellt, dass die Luftstrahlsimulation mit Hilfe des UDF-Programms durch die beliebige Vorgabe einer Injektorposition funktioniert hat. Der Luftstahl im stationären Zustand bildet sich wie erwartet kegelförmig und symmetrisch aus. Ebenso hat die Partikelsimulation auch funktioniert. In einem weiteren Schritt wurde die Simulation mit den Ergebnissen aus dem stationären Zustand auch im instationären Zustand durchgeführt. Das Luftstrahl- und Partikelmodell wurde miteinander in einer Simulation getestet. Um die Rechenzeit zu reduzieren wurde die Simulation parallel auf mehreren Rechnen durchgeführt. Die Luftstrahlsimulation im instationären Zustand ohne Partikel funktioniert wie erwartet. Der Luftstrahl bewegte sich auf einer Grade in dem Berechnungsgebiet siehe **Abbildung 52** und**Abbildung 53**.

Bei Verwendung des Partikelmodells zusammen mit dem Luftstrahl funktionierte die Simulation nicht. Es wurden mehrere Berechnungen durchgeführt, die zu fehlerhaften Ergebnisse geführt haben. Die Injektorposition wechselte sich willkürlich in dem Gebiet. Dadurch befanden sich der Luftstrahl und die Partikel nicht an der gleichen Position.

Mit den Ergebnissen aus dieser Diplomarbeit wurde es festgestellt, dass die Luftstrahl- und Partikelsimulation mit Hilfe der Masse- und Impulsquellen direkt aus den Zellen ohne eine zusätzliche Spraydüse in dem Berechnungsgebiet erzeugt werden kann.

Literaturverzeichnis

[1] P. Wulf (2011): Vorlesungskrip CFD. Hochschule für Angewandten Wissenschaften Hamburg.

[2] ANSYS Inc. (2009): FLUENT 12.0 Theory Guide, ANSYS, Inc.

[3] FLUENT Inc. (2006): FLUENT 6.3 UDF Manual.

[4] Oertel und Böhle (2006): Strömungsmechanik, 4. Auflage, Vieweg+Teubner Verlag.

[5] Ferziger, J. H. und Peric, M. (2002): Computational Methods for Fluid Dynamics, 3. Auflage, Springer-Verlag.

Anhang

Anhang A1: UDF-Programm

Anhang A2: DIN-A3 Ergebnisse der Luftstrahlmodelle

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel “Optimierung eines Discrete-Particle-Modells (DPM) zur Strömungssimulation von luftunterstützten Sprays” selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Hamburg, den 29. November 2011

_____

Unterschrift

```
#include "udf.h"
#include "dpm.h"


/************ Eingabe ***************/
real masssource = 0.006125;
real spray_half_angle = 15;
real ziel_y = 40;
real x0[3] = {1.0066,0.402,0.3};  /* Koordinat der Injection Düse */
/*********************************/

real A_C[3] = {0.};
real W_C[3] = {0};
real E_C[3] = {0};
real N_C[3] = {0};
real S_C[3] = {0};
real SW_C[3] = {0};
real SE_C[3] = {0};
real NW_C[3] = {0};
real NE_C[3] = {0};
real MittelP[3] = {0};

cell_t w_c;
cell_t e_c;
cell_t n_c;
cell_t s_c;
cell_t sw_c;
cell_t se_c;
cell_t nw_c;
cell_t ne_c;
cell_t I1;
cell_t I2;
cell_t I3;
cell_t I4;

DEFINE_ADJUST(Gil,d)
{
#if !RP_HOST
Thread *t;
face_t f;
Thread *tf;
face_t f1;
Thread *tf1;
face_t f2;
Thread *tf2;
int n;
int n1;
int n2;

cell_t c;
cell_t a_c;
cell_t Nachbr_c;
real NachbrMpkt[3] = {0.};
real face_cent1[3] = {0.};
real face_cent2[3] = {0.};

real x[3] = {0.};
```

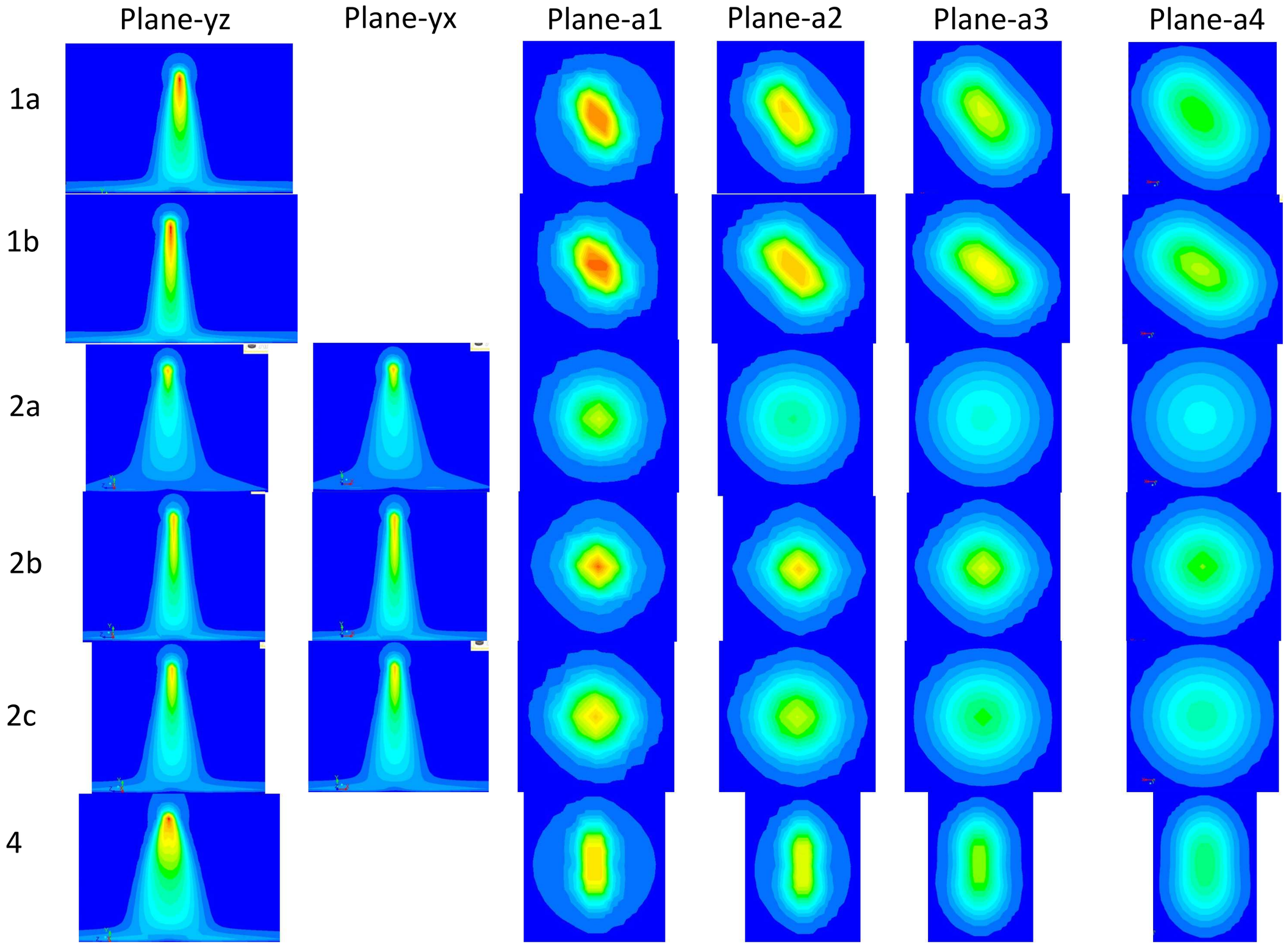
```
real x_x0[3] = {0.};
real abs =0.;
real abs_klein = 0.05; /* Kreisradius um Injector zur Nachbarbestimmung */

real abs_vek[3] = {0.};
real p_i_z1 = 0.;
real p_i_z2 = 0.;
real e[3] = {0,1,0};
real An[3] = {0.};


thread_loop_c(t,d) /*loops over all cell threads in domain*/
{

begin_c_loop(c,t) /* loops over cells in a cell thread */
{
   C_CENTROID(x,c,t)
     NV_VV(x_x0, =, x, -, x0);
     abs = NV_MAG(x_x0);

/* Finde Ursprungscell  */
     if(abs < abs_klein)
          {abs_klein = abs; NV_V(A_C, =, x); a_c = c;
           p_i_z1 = x_x0[2]; /* oder auch A[2] - x0[2]; */
           p_i_z2 = x_x0[0]; /* oder auch A[0] - x0[0];*/

c_face_loop(a_c, t, n) /* loops over all faces of a cell */
  {
f = C_FACE(a_c,t,n); tf = C_FACE_THREAD(a_c,t,n); F_CENTROID(face_cent1,f,tf);
 if(NV_DOT(e, An) == 0.)
   {  if(c == F_C0(f,tf)) Nachbr_c = F_C1(f,tf);
     else Nachbr_c = F_C0(f,tf);}
     C_CENTROID(NachbrMpkt,Nachbr_c,t); NV_VV(abs_vek, =,NachbrMpkt , -, A_C);
      if(abs_vek[2] > 1e-3) {NV_V(W_C,=,NachbrMpkt); w_c= Nachbr_c;}

     else if(abs_vek[2] < -1e-3) {NV_V(E_C,=,NachbrMpkt); e_c= Nachbr_c;}

     if(abs_vek[0] > 1e-3) { NV_V(S_C,=,NachbrMpkt); s_c= Nachbr_c;
      c_face_loop(s_c, t, n1) /* loops over all faces of a cell */
      {f1 = C_FACE(s_c,t,n1); tf1 = C_FACE_THREAD(s_c,t,n1); F_CENTROID(face_cent1,f1,tf1);
      if ((S_C[2] - face_cent1[2]) < -1e-3) { if(s_c == F_C0(f1,tf1)) {sw_c = F_C1(f1,tf1);
      C_CENTROID(SW_C,sw_c,t);} else {sw_c = F_C0(f1,tf1);C_CENTROID(SW_C,sw_c,t);} }
      if ((S_C[2] - face_cent1[2]) >  1e-3) { if(s_c == F_C0(f1,tf1)) {se_c = F_C1(f1,tf1);
      C_CENTROID(SE_C,se_c,t);} else {se_c = F_C0(f1,tf1);C_CENTROID(SE_C,se_c,t);} }
      }

     }

      else if(abs_vek[0] < -1e-3) { NV_V(N_C,=,NachbrMpkt); n_c= Nachbr_c;
      c_face_loop(n_c, t, n2) /* loops over all faces of a cell */
      {f2 = C_FACE(n_c,t,n2); tf2 = C_FACE_THREAD(n_c,t,n2); F_CENTROID(face_cent2,f2,tf2);
         if (N_C[2] - face_cent2[2] < -1e-3) { if(n_c == F_C0(f2,tf2)) {nw_c = F_C1(f2,tf2);
         C_CENTROID(NW_C,nw_c,t);} else {nw_c = F_C0(f2,tf2);C_CENTROID(NW_C,nw_c,t);} }
         if (N_C[2] - face_cent2[2] >  1e-3) { if(n_c == F_C0(f2,tf2)) {ne_c = F_C1(f2,tf2);
         C_CENTROID(NE_C,ne_c,t);} else {ne_c = F_C0(f2,tf2);C_CENTROID(NE_C,ne_c,t);} }
      }
                                  }
```

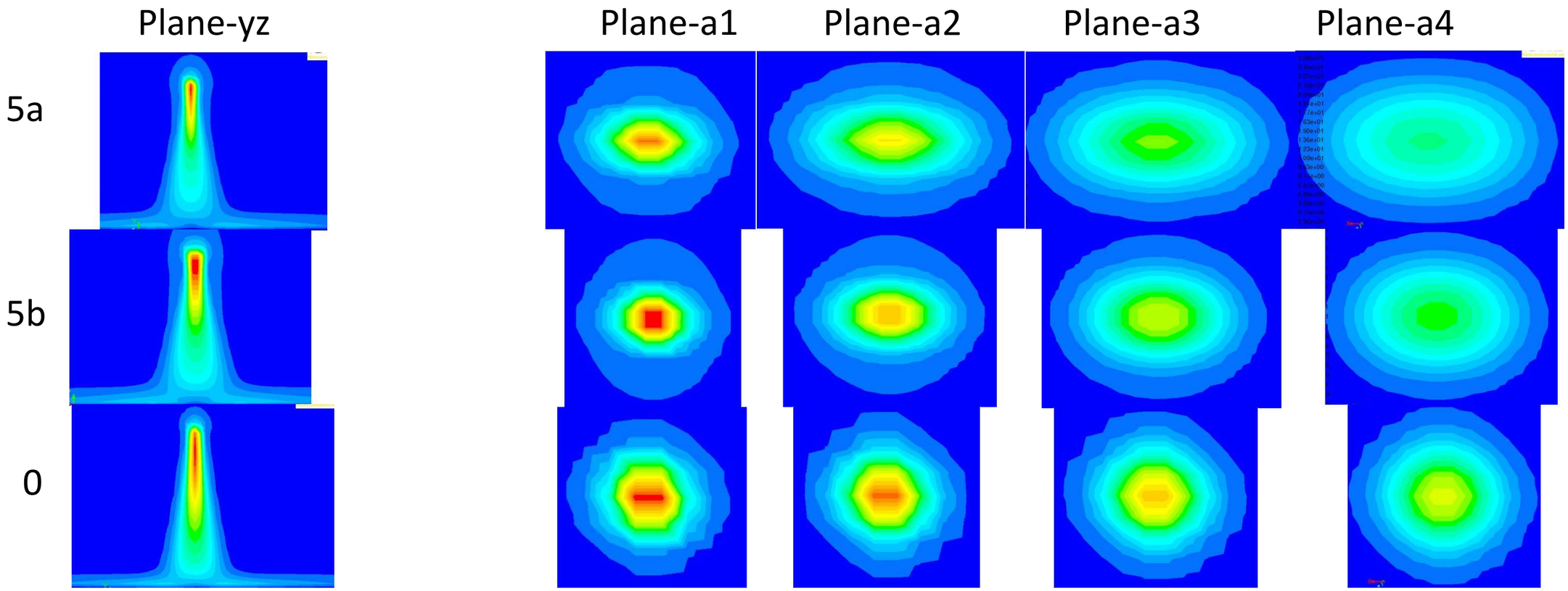
```
  }
          }
} end_c_loop(c,t)
}

if (p_i_z1 >= 0. && p_i_z2 >= 0.) {I1 = a_c ; I2 = n_c; I3 = ne_c; I4 = e_c; NV_VS_VS(MittelP
, =, NE_C, *, 0.75, -, A_C, *, 0.5); printf("\n 1111111111111\n"); } /*Fall I */
if (p_i_z1 < 0. && p_i_z2 >= 0.) {I1 = w_c ; I2 = nw_c; I3 = n_c; I4 = a_c;  NV_VS_VS(MittelP
, =, N_C, *, 0.75, -, W_C, *, 0.5); printf("\n 22222222222222\n");} /* Fall II */
if (p_i_z1 < 0. && p_i_z2 < 0.)   {I1 = sw_c ; I2 = w_c; I3 = a_c; I4 = s_c; NV_VS_VS(MittelP
, =, A_C, *, 0.75, -, SW_C, *, 0.5); printf("\n 33333333333333\n"); } /* Fall III */
if (p_i_z1 >= 0. && p_i_z2 < 0.) {I1 = s_c ; I2 = a_c; I3 = e_c; I4 = se_c;  NV_VS_VS(MittelP
, =, E_C, *, 0.75, -, S_C, *, 0.5); printf("\n 44444444444444\n");}/* Fall IV */
/*
printf("\n Koordinat-Cell-A_C x= %g, y= %g, z= %g\n",  A_C[0],  A_C[1],  A_C[2]);
printf("\n Koordinat-Cell-N_C x= %g, y= %g, z= %g\n",  N_C[0], N_C[1],  N_C[2]);
printf("\n Koordinat-Cell-S_C x= %g, y= %g, z= %g\n",  S_C[0],  S_C[1],  S_C[2]);
printf("\n Koordinat-Cell-W_C x= %g, y= %g, z= %g\n",  W_C[0],  W_C[1],  W_C[2]);
printf("\n Koordinat-Cell-E_C x= %g, y= %g, z= %g\n",  E_C[0],  E_C[1],  E_C[2]);
printf("\n Koordinat-Cell-SW_C x= %g, y= %g, z= %g\n",  SW_C[0],  SW_C[1],  SW_C[2]);
printf("\n Koordinat-Cell-SE_C x= %g, y= %g, z= %g\n",  SE_C[0],  SE_C[1],  SE_C[2]);
printf("\n Koordinat-Cell-NW_C x= %g, y= %g, z= %g\n",  NW_C[0],  NW_C[1],  NW_C[2]);
printf("\n Koordinat-Cell-NE_C x= %g, y= %g, z= %g\n",  NE_C[0],  NE_C[1],  NE_C[2]);*/

#endif /* !RP_HOST */
}



/************************************************************************/

DEFINE_SOURCE(Massource,c,t,dS,eqn)
{
#if !RP_HOST
real source;

if (c == I1)
    {source = masssource/(4*C_VOLUME(c,t)); printf("\n ooooooooouuuuuuuuuuuuuhhhhh I1, %g\n",
     source);}
else if (c == I2)
    { source = masssource/(4*C_VOLUME(c,t)); printf("\n ooooooooouuuuuuuuuuuuuhhhhh_I2, %g\n"
    , source);}
else if (c == I3)
    { source = masssource/(4*C_VOLUME(c,t)); printf("\n ooooooooouuuuuuuuuuuuuhhhhh_I3, %g\n"
    , source);}
else if (c == I4)
    { source = masssource/(4*C_VOLUME(c,t)); printf("\n ooooooooouuuuuuuuuuuuuhhhhh_I4, %g\n"
    , source);}
else
source = 0.;

return source;
}


/************************************************************************/
```

```
/* Momentum in x-Richtung) */

real ccI1= 0; real I1_vx = 0.;
real ccI2= 0; real I2_vx = 0.;
real ccI3= 0; real I3_vx = 0.;
real ccI4= 0; real I4_vx = 0.;

DEFINE_SOURCE(xmom_source,c,t,dS,eqn)
{
real source = 0;
real lambda = 2000;
real ziel_x = ziel_y * tan(spray_half_angle/180*3.14159);

if (c == I1)
    {source = ccI1+lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eqn] = 0; printf("\n
    ccccccccccccccccxxxxxx, sourceI1= %g, Vel_I1_x= %g\n", source, C_U(c,t)); ccI1 = source;
     I1_vx = C_U(c,t); printf("\n ziel_x = %g\n", ziel_x);}
else if (c == I2)
    {source = ccI2-lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eqn] = 0;  printf("\n
    ccccccccccccccccxxxxxx, sourceI2= %g, Vel_I2_x= %g\n", source, C_U(c,t)); ccI2 = source;
     I2_vx = C_U(c,t);}
else if (c == I3)
    {source = ccI3-lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eqn] = 0;  printf("\n
    ccccccccccccccccxxxxxx, sourceI3= %g, Vel_I3_x= %g\n", source, C_U(c,t)); ccI3 = source;
     I3_vx = C_U(c,t);}
else if (c == I4)
    {source = ccI4+lambda*(ziel_x-fabs(C_U(c,t))) ; dS[eqn] = 0; printf("\n
    ccccccccccccccccxxxxxx, sourceI4= %g, Vel_I4_x= %g\n", source, C_U(c,t)); ccI4 = source;
     I4_vx = C_U(c,t);}



else
source = 0.;

return source;
#endif /* !RP_HOST */
}


/*************************************************************************/
/* Momentum in y-Richtung) */

real aaI1 = 0; real I1_vy = 0.;
real aaI2 = 0; real I2_vy = 0.;
real aaI3 = 0; real I3_vy = 0.;
real aaI4 = 0; real I4_vy = 0.;

DEFINE_SOURCE(ymom_source,c,t,dS,eqn)
{
#if !RP_HOST
real source =0;
real lambda =3000;

if (c == I1)
    {source = aaI1-lambda*(ziel_y-fabs(C_V(c,t))) ; dS[eqn] = 0; printf("\n
    aaaaaaaaaaaaaaaayyyyyy, sourceI1= %g, Vel_I1_y= %g\n", source, C_V(c,t)); aaI1= source;
```

```
    I1_vy = C_V(c,t);}
else if (c == I2)
    {source = aaI2-lambda*(ziel_y-fabs(C_V(c,t))) ; dS[eqn] = 0; printf("\n
    aaaaaaaaaaaaaaaaayyyyyy, sourceI2= %g, Vel_I2_y= %g\n", source, C_V(c,t)); aaI2= source;
    I2_vy = C_V(c,t);}
else if (c == I3)
    {source = aaI3-lambda*(ziel_y-fabs(C_V(c,t))) ; dS[eqn] = 0; printf("\n
    aaaaaaaaaaaaaaaaayyyyyy, sourceI3= %g, Vel_I3_y= %g\n", source, C_V(c,t)); aaI3= source;
    I3_vy = C_V(c,t);}
else if (c == I4)
    {source = aaI4-lambda*(ziel_y-fabs(C_V(c,t))) ; dS[eqn] = 0; printf("\n
    aaaaaaaaaaaaaaaaayyyyyy, sourceI4= %g, Vel_I4_y= %g\n", source, C_V(c,t)); aaI4= source;
    I4_vy = C_V(c,t);}


else
source = 0.;

return source;
#endif /* !RP_HOST */
}

/*****************************************************************************/
/* Momentum in z-Richtung) */

real bbI1= 0; real I1_vz = 0.;
real bbI2= 0; real I2_vz = 0.;
real bbI3= 0; real I3_vz = 0.;
real bbI4= 0; real I4_vz = 0.;

DEFINE_SOURCE(zmom_source,c,t,dS,eqn)
{
#if !RP_HOST
real source = 0;
real lambda =2000;
real ziel_z = ziel_y * tan(spray_half_angle/180*3.14159);

if (c == I1)
    {source = bbI1+lambda*(ziel_z-fabs(C_W(c,t))) ; dS[eqn] = 0; printf("\n
    bbbbbbbbbbbbbbbbzzzzzz, sourceI1= %g, Vel_I1_z= %g\n", source, C_W(c,t)); bbI1 = source;
    I1_vz = C_W(c,t);printf("\n ziel_z = %g\n", ziel_z);}
else if (c == I2)
    {source = bbI2+lambda*(ziel_z-fabs(C_W(c,t))) ; dS[eqn] = 0; printf("\n
    bbbbbbbbbbbbbbbbzzzzzz, sourceI2= %g, Vel_I2_z= %g\n", source, C_W(c,t)); bbI2 = source;
    I2_vz = C_W(c,t);}
else if (c == I3)
    {source = bbI3-lambda*(ziel_z-fabs(C_W(c,t))) ; dS[eqn] = 0;  printf("\n
    bbbbbbbbbbbbbbbbzzzzzz, sourceI3= %g, Vel_I3_z= %g\n", source, C_W(c,t)); bbI3 = source;
    I3_vz = C_W(c,t);}
else if (c == I4)
    {source = bbI4-lambda*(ziel_z-fabs(C_W(c,t))) ; dS[eqn] = 0;  printf("\n
    bbbbbbbbbbbbbbbbzzzzzz, sourceI4= %g, Vel_I4_z= %g\n", source, C_W(c,t)); bbI4 = source;
    I4_vz = C_W(c,t);}


else
source = 0.;
```

```
return source;
#endif /* !RP_HOST */
}


/************************************************************************/
real korrektur[3]={0}; /* Vektor zur Verschiebung von Partikeln in MittelP */

DEFINE_EXECUTE_AT_END(Bewegung)
{
#if !RP_HOST
FILE *fpI1; FILE *fpI2; FILE *fpI3;FILE *fpI4;
real time;
real timestep;
int i1 = 0;

i1++;

fpI1 = fopen("I1", "a"); fpI2 = fopen("I2", "a"); fpI3 = fopen("I3", "a"); fpI4 = fopen("I4",
 "a");
fprintf(fpI1, "%i %g %g %g \n",i1, I1_vx, I1_vy, I1_vz);
fprintf(fpI2, "%i %g %g %g \n",i1, I2_vx, I2_vy, I2_vz);
fprintf(fpI3, "%i %g %g %g \n",i1, I3_vx, I3_vy, I3_vz);
fprintf(fpI4, "%i %g %g %g \n",i1, I4_vx, I4_vy, I4_vz);
fclose(fpI1); fclose(fpI2); fclose(fpI3); fclose(fpI4);

timestep = CURRENT_TIMESTEP;
time = CURRENT_TIME;
if (time > 0.15)
x0[2] = x0[2] + 0.5*timestep;

NV_VV(korrektur, =, x0, -, MittelP);
#endif /* !RP_HOST */
}


DEFINE_DPM_INJECTION_INIT(init_partikel,I)
{
#if !RP_HOST
Particle *p;
real time = CURRENT_TIME;
real timestep = CURRENT_TIMESTEP;

loop(p,I->p_init) /* Standard ANSYS FLUENT Looping Macro to get particle streams in an
Injection */
{
/* P_INIT_POS(p)[2] = x0[2];*/

if (time >0.15 ){
 P_INIT_POS(p)[2] = P_INIT_POS(p)[2] + 0.5*timestep;
 P_POS(p)[2] = P_POS(p)[2] - korrektur[2];
 P_POS(p)[0] = P_POS(p)[0] - korrektur[0];}
}
#endif /* !RP_HOST */
}
```


Plane-yz
Plane-yx
Plane-a1
Plane-a2
Plane-a3
Plane-a4
1a
1b
2a
2b
2c
4


Plane-yz
Plane-a1
Plane-a2
Plane-a3
Plane-a4
5a
5b
0