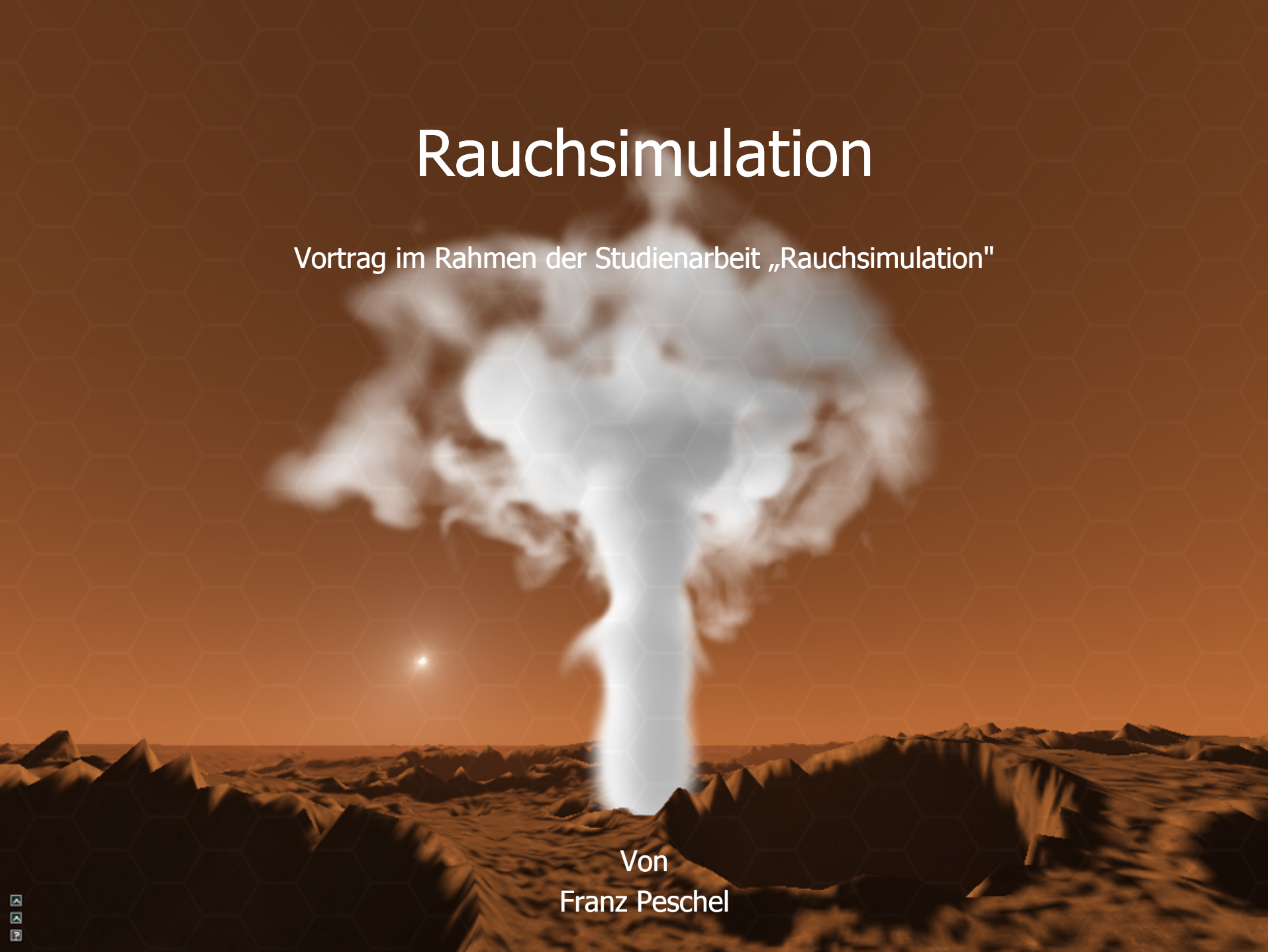


Rauchsimulation

Vortrag im Rahmen der Studienarbeit „Rauchsimulation"

Von
Franz Peschel

Inhalt

Einführung

Motivation:

- Mit Fluid Simulation ...

- ... faszinierend anzuschauende Naturphänomene:
- Rauch, Dampf, Feuer, Wasser

- Große Fülle an neuen Interaktionmöglichkeiten

- Integration in eine Echtzeitanwendungen – was ist heute möglich?

Inhalt

State of the Art

2 Ansätze zu Fluid-Simulation mit Navier-Stokes-Gleichungen

- Rasterbasiertes Verfahren

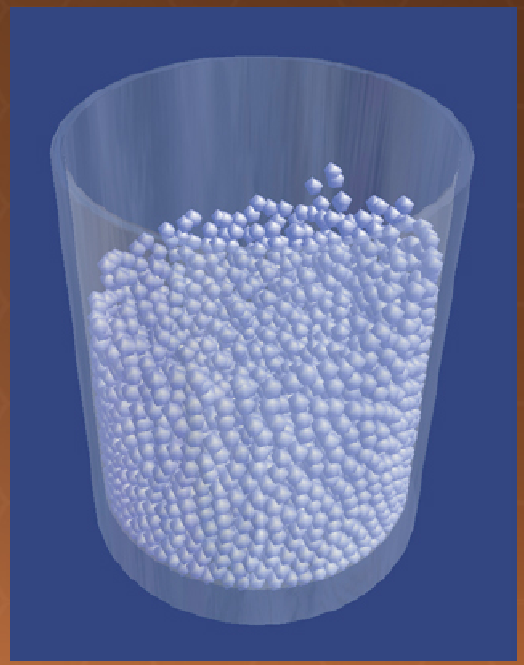
Hellgate London

S.T.A.L.K.E.R. Clear Sky

Quelle: 6,7

State of the Art

- Partikelbasiertes Verfahren
  - Smoothed Particle Hydrodynamics



Quelle: 3

Inhalt

Fluid Simulation

- Lösen der Navier-Stokes Gleichungen rasterbasiert
  - Algorithmus von Stam 1999 und Fedkiw 2001
    - Stabil für beliebige Zeitschritte
      - Damit schnelle Berechnung möglich
    - Kann auf aktuellen GPUs implementiert werden
    - Siehe Harris, GPU Gems 2004

Quelle: 1, 4

Navier-Stokes Gleichungen

- Beschreibt den Fluss eines inkompressiblen Fluids

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p - \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

Advektion | Druck-Gradient | Diffusion | Beschleunigung

$$\nabla \cdot \mathbf{u} = 0 \quad \leftarrow$$

Geschwindigkeit muss divergenzfrei sein

Quelle: 1

Divergenzfrei

- Was bedeutet divergenzfrei?
  - In jedem Fluidelement halten sich die eingehende Geschwindigkeit und die ausgehende Geschwindigkeit die Wage.
    - Ausnahmen: Geschwindigkeitsquellen oder Abflüsse
  - Es gilt die Masse / Impulserhaltung $\nabla \cdot \mathbf{u} = 0$

Quelle: 1

Algorithmus

- Stam 1999
  - Hinzufügen von Kräften $\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$
  - Advektion $\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$
  - Diffusion $\left(\mathbf{I} - \nu\Delta t\nabla^2\right)\mathbf{w}_3 = \mathbf{w}_2$
  - Lösen der Druckgleichung $\nabla^2 p = \nabla \cdot \mathbf{w}_3$
  - Geschwindigkeitsfeld – Gradient(Druckfeld) $\mathbf{u}(\mathbf{x},t+\Delta t) = \mathbf{w}_3 - \nabla p$

Quelle: 1

Algorithmus

- Stam 1999
  - Hinzufügen von Kräften

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$

Quelle: 1

Beschleunigung

- Hinzufügen von Kräften

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$

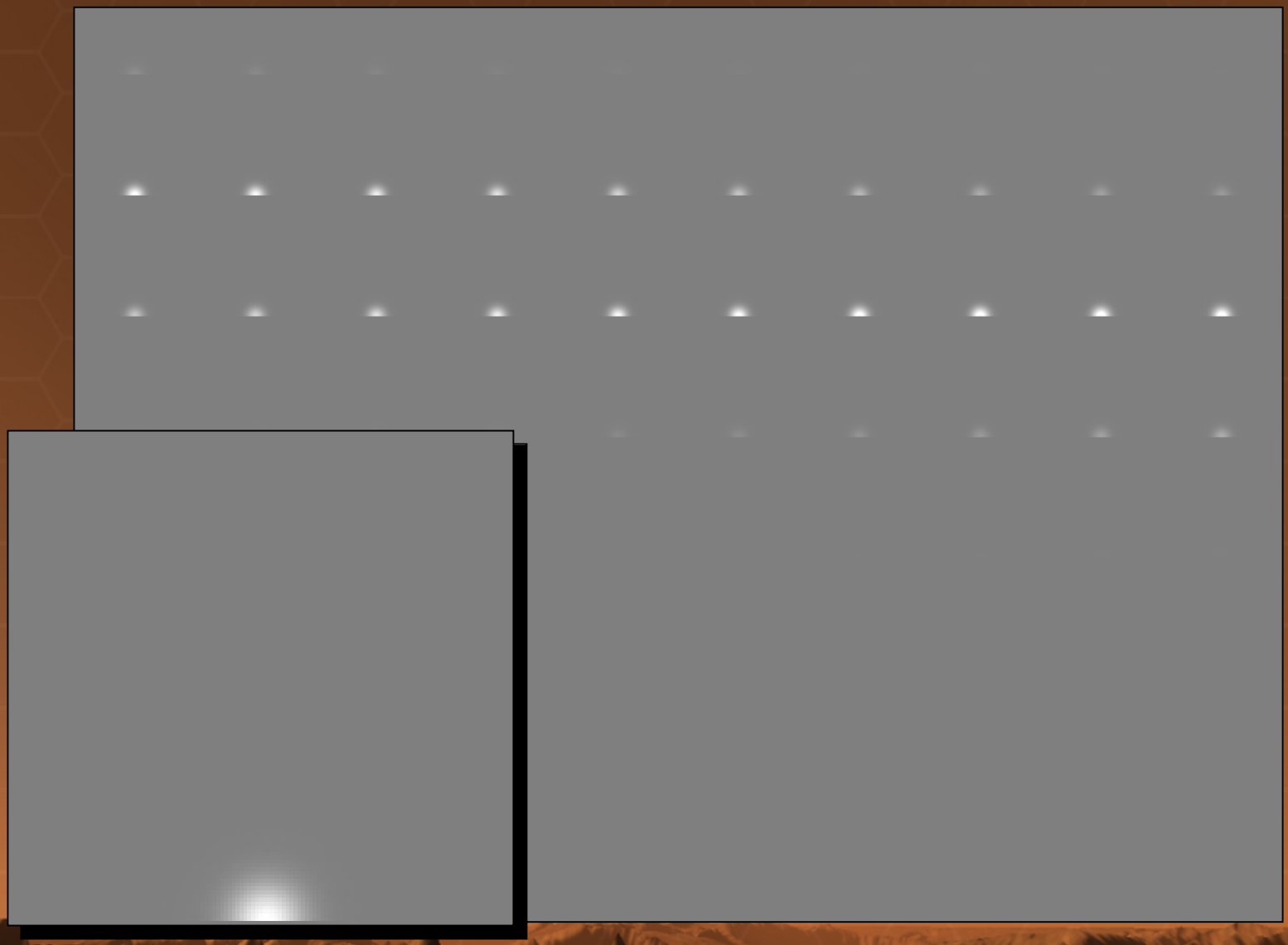
  - Geschwindigkeitsfeld + Kraftquelle * Δt

  - Gauss-geglätteter Splat

    - Farbe bestimmt Richtung und Stärke der Kraft

Quelle: 1

Beschleunigung

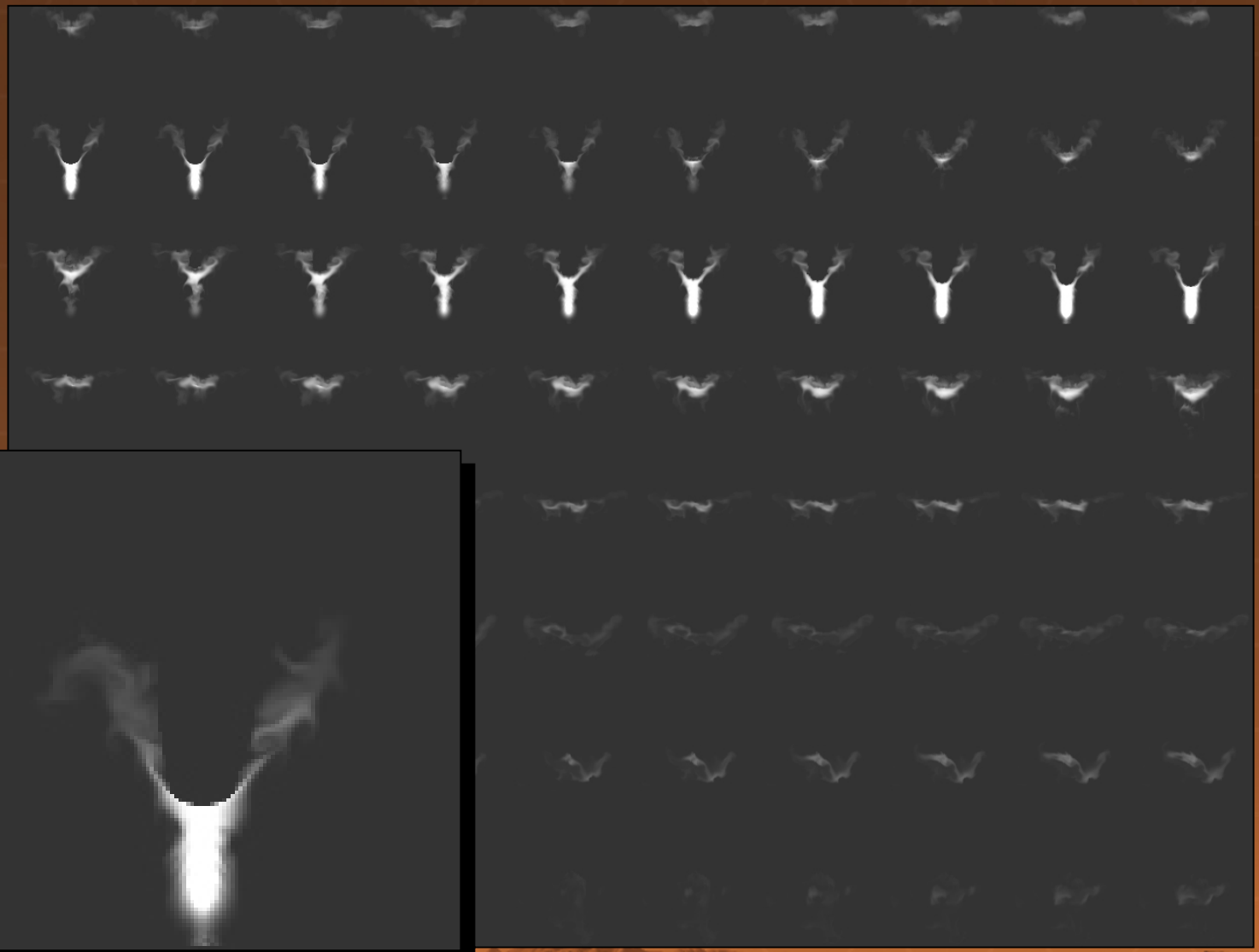


Simulation Flüssigkeiten und Gase

- Einfachste Anwendung einer Fluidsimulation
- Dichtefeld mit skalaren Werten d:
  - Wassertropfen im Dampf
  - Rußpartikel im Rauch
  - Staub
- Transport der Partikel
  - Advektionsoperator auf Dichtefeld d anwenden

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{d}$$

Simulation Flüssigkeiten und Gase



Temperatur Simulation

- Konvektionsströme simulieren
  - Auftrieb berechnen
    - Neues Skalarfeld T
    - Auftriebsoperator F_{buo}:
    - Umgebungstemperatur Tu < lokale Temperatur T:
      - Geschwindigkeitsfeld + T
    - d = Dichte

$$\vec{F}_{buo} = (-\mu d + \sigma (T - T_u)) \vec{k}$$

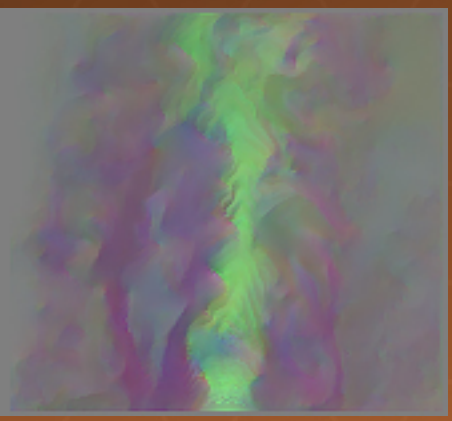
Temperatur Simulation

- Wärmequelle treibt Partikel nach oben. (Grün)
- Die Partikel kühlen ab und fallen durch die Schwerkraft runter. (Lila)

Dichtequelle

Dichte d

Temperatur T



Geschwindigkeit u

Wirbelstärkenerhaltung

- Detaillierte Verwirbelungen verschwinden durch numerische Verluste der Simulation
  - Wirbelstärkenerhaltung fügt sie wieder dazu

- 3 Schritte

1) Berechne Wirbelstärkefeld aus Geschwindigkeitsfeld

$$\vec{\Phi} = \nabla \times \vec{u}$$

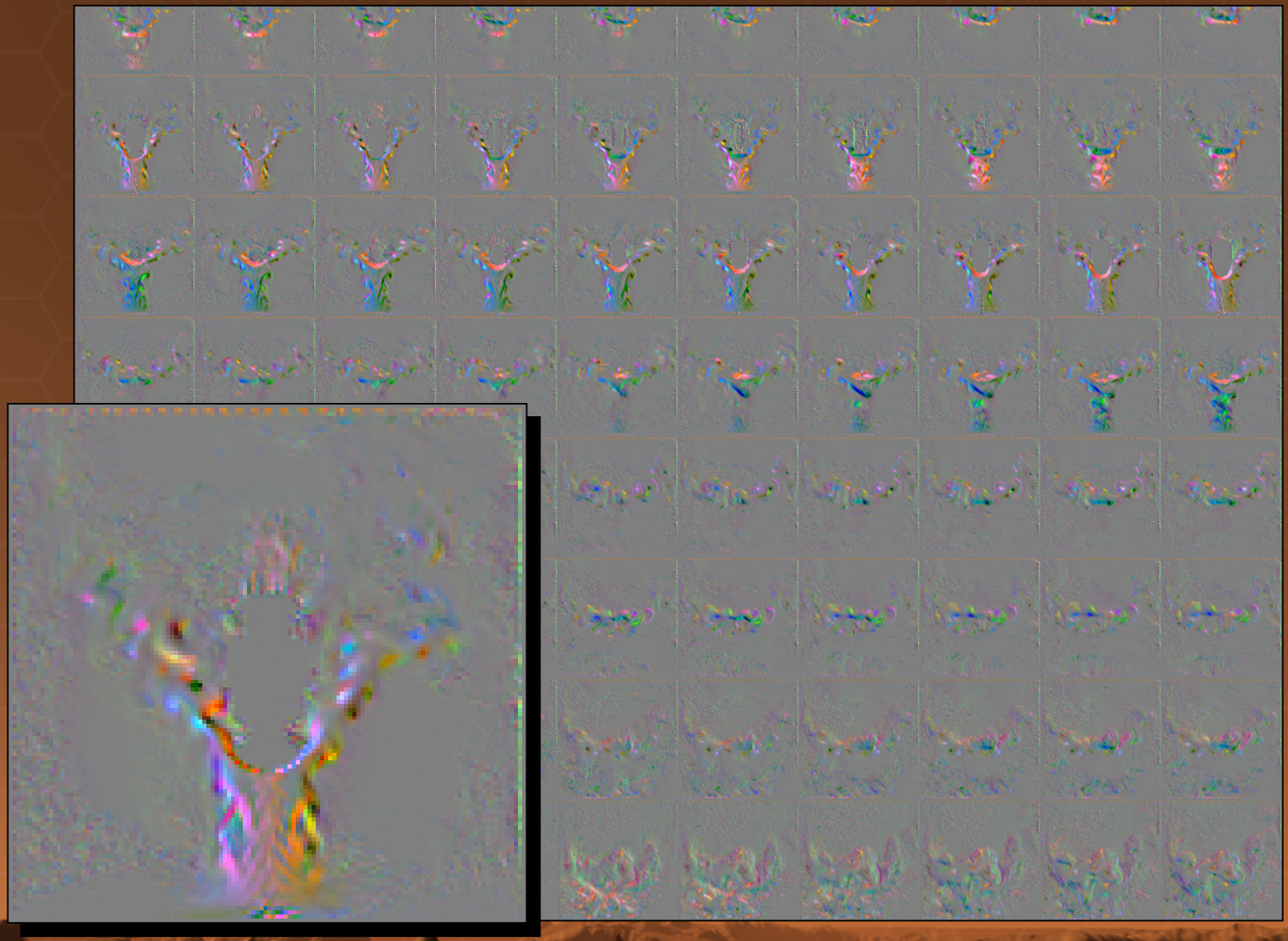
2) Berechne normalisierte Wirbelstärke-Positionsvektoren

$$\vec{X} = \frac{\vec{\omega}}{|\vec{\omega}|}, \vec{\omega} = \nabla |\vec{\Phi}|$$

3) Bestimme Richtung und Skalierung der Wirbelstärkekraft F_{vc}

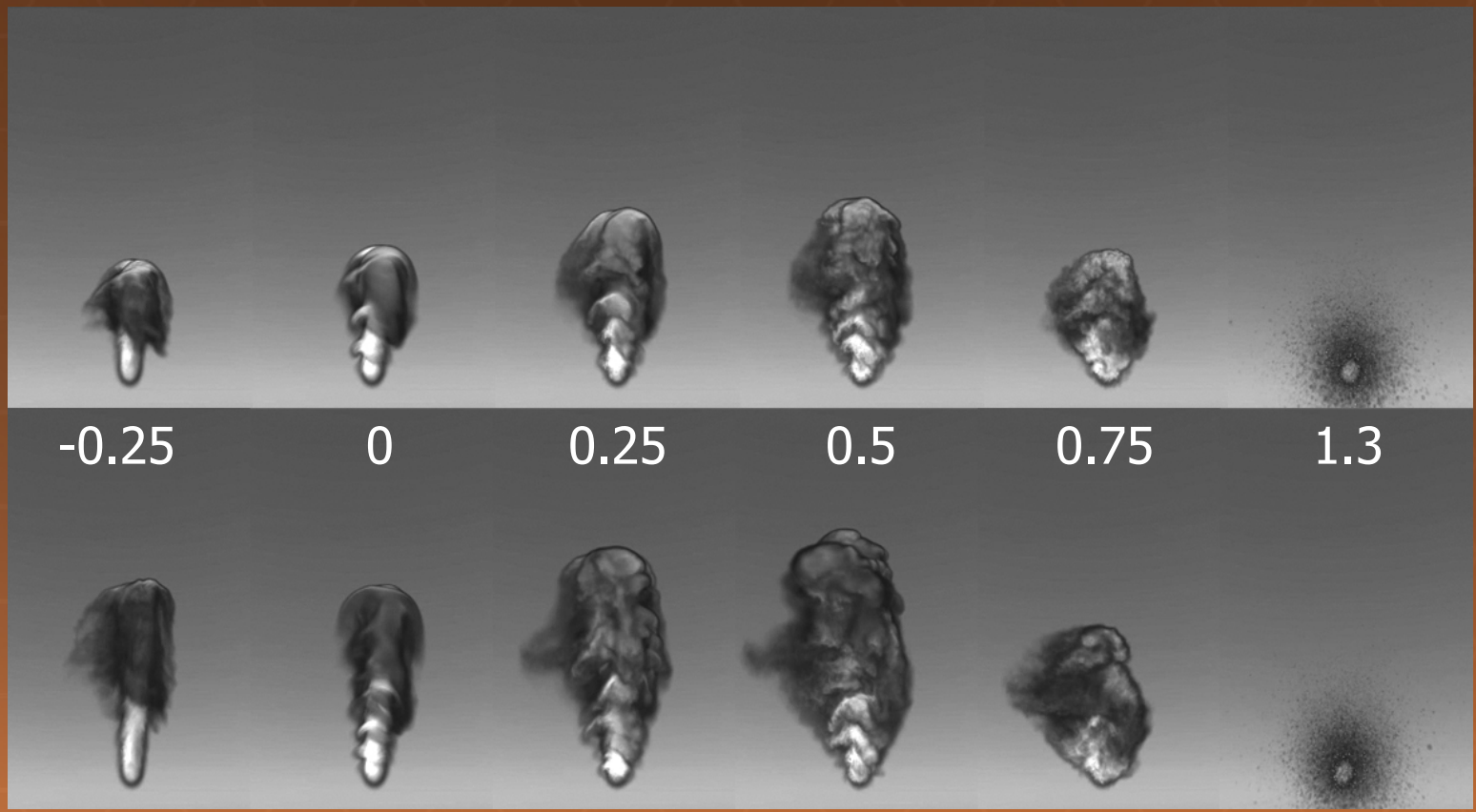
$$\vec{F}_{vc} = \epsilon \left(\vec{X} \times \vec{\Phi} \right)$$

Wirbelstärkenerhaltung



Wirbelstärkenerhaltung

- Geschwindigkeitsfeld + Wirbelstärkenkraft (verschiedene ε)



Algorithmus

- Stam 2000
  - Hinzufügen von Kräften
  - Advektion

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$

$$\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$$

Quelle: 1

Advektion

- Advektion: Elemente im Fluid werden entlang seines Geschwindigkeitsfeldes transportiert

$$\mathbf{w}_2(\mathbf{x}) = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1 \Delta t)$$

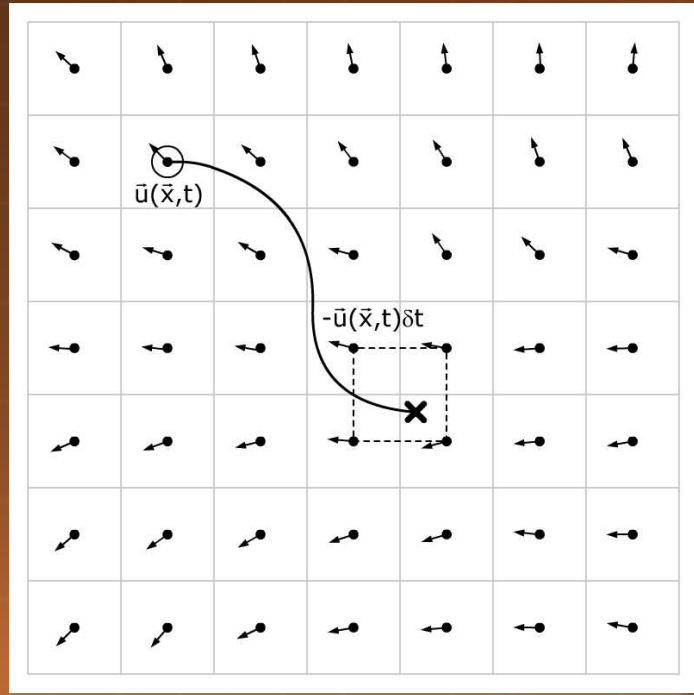
- Geschwindigkeit in Position x zur

  Zeit t + Δt ?

- Folge Geschwindigkeitsfeld zurück zum Zeitpunkt

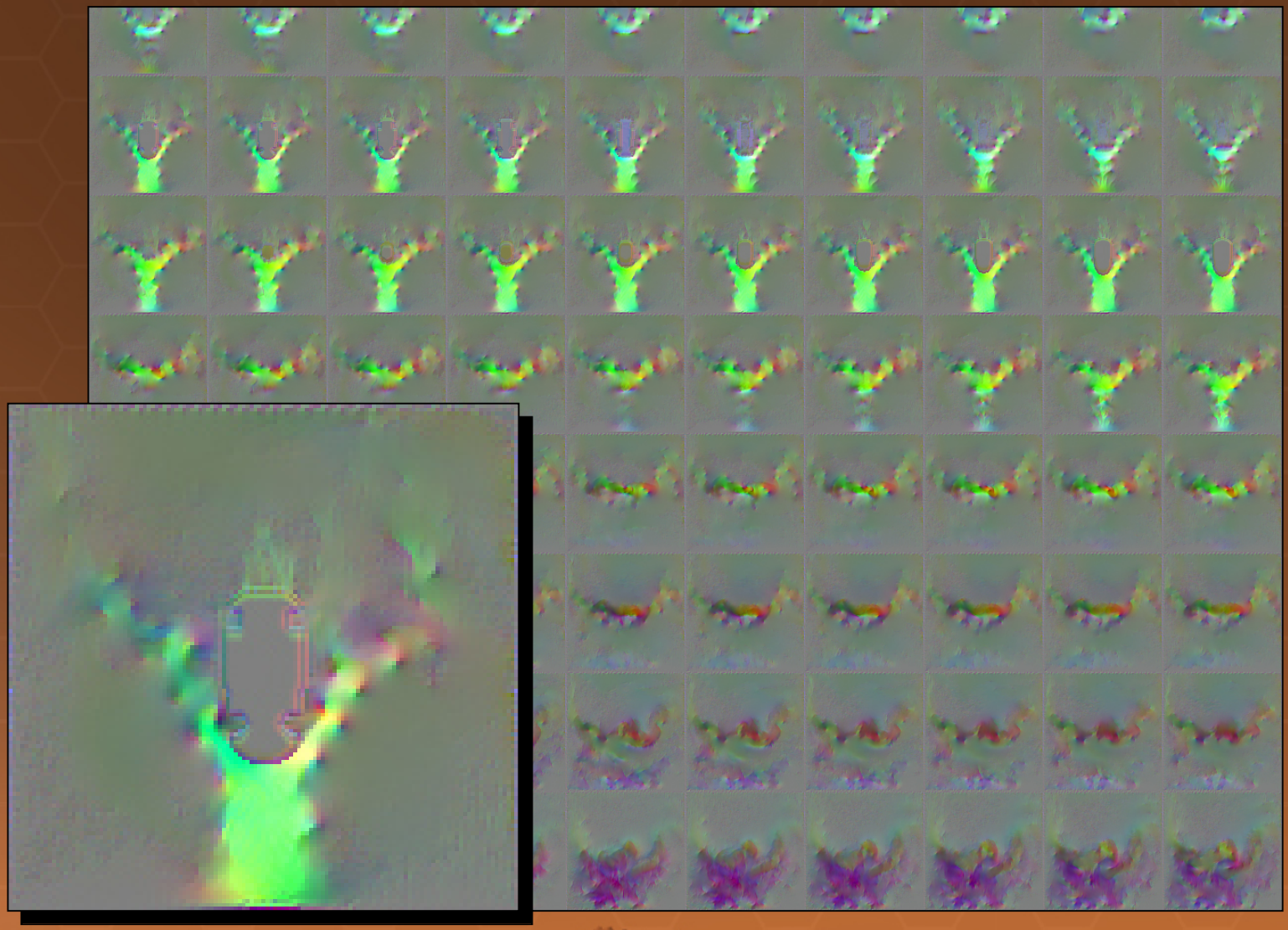
  X : (x-w1 Δt)

- Einfaches Fragment-Programm



Quelle: 1

Advektion



Algorithmus

- Stam 2000
  - Hinzufügen von Kräften
  - Advektion
  - Diffusion

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$

$$\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$$

$$\left(\mathbf{I} - \nu\Delta t\nabla^2\right)\mathbf{w}_3 = \mathbf{w}_2$$

Quelle: 1

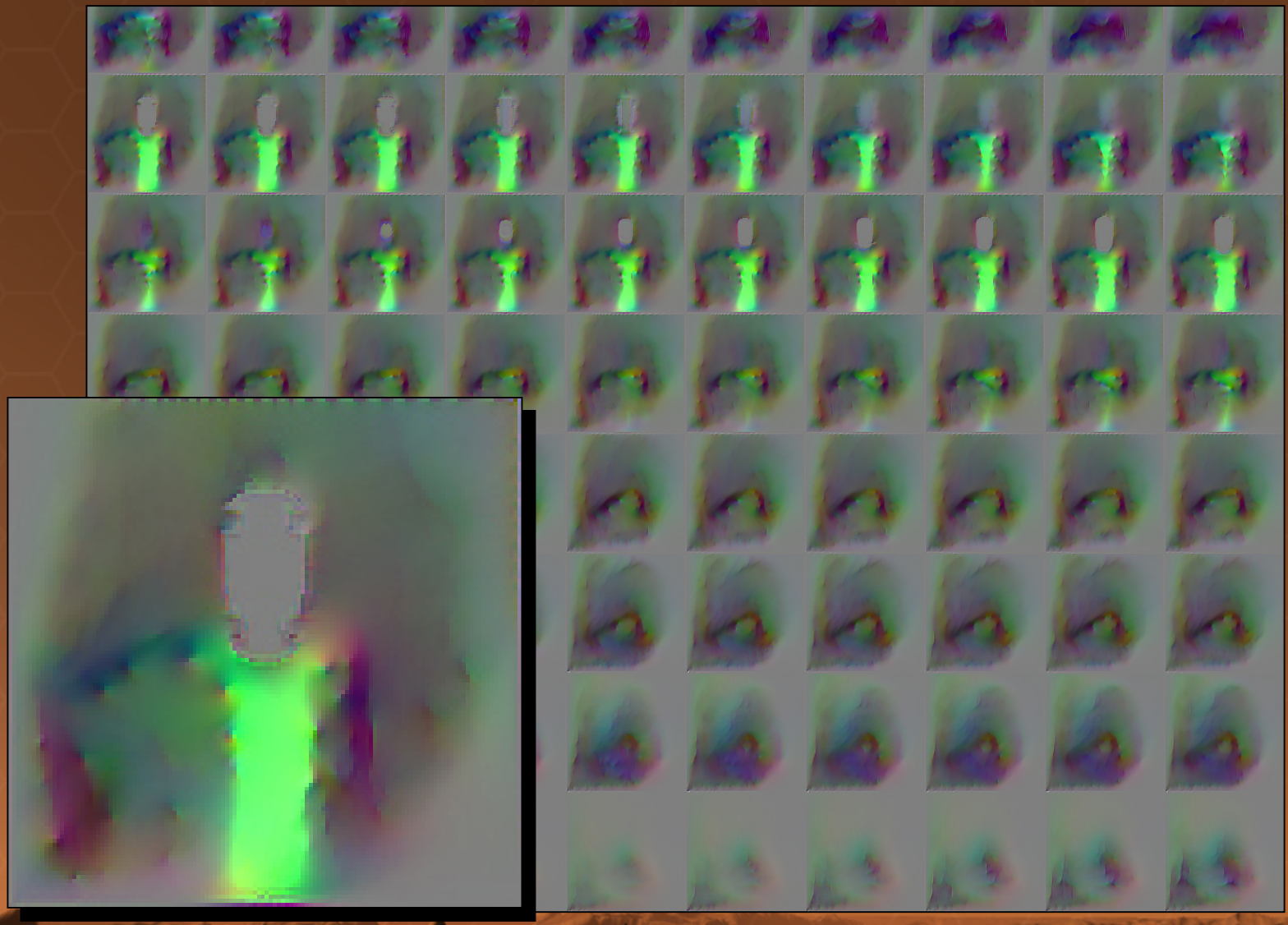
Viskose Diffusion

$$\left(\mathbf{I} - \nu \Delta t \nabla^2\right)\mathbf{w}_3 = \mathbf{w}_2$$

- Ein viskoses Fluid fließt zäh wie Sirup
  - Verursacht durch Diffusion der Geschwindigkeit
- Implizite diskrete Form von $\frac{\partial \mathbf{w}_2}{\partial t} = \nu \nabla^2 \mathbf{w}_2$
  - Explizite Form instabil
- Lösung auch im Druck-Berechnungsschritt auf nächster Folie
- Optionaler Berechnungsschritt

Quelle: 1

Viskose Diffusion



Algorithmus

- Stam 2000
  - Hinzufügen von Kräften $$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$
  - Advektion $$\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$$
  - Diffusion $$\left(\mathbf{I} - \nu\Delta t\nabla^2\right)\mathbf{w}_3 = \mathbf{w}_2$$
  - Lösen der Druckgleichung $$\nabla^2 p = \nabla \cdot \mathbf{w}_3$$

Quelle: 1

Poisson-Druck Lösung

$$\nabla^2 p = \nabla \cdot \mathbf{w}_3$$

- Poisson Gleichung
  - Diskretisierung und Lösen mit Hilfe eines iterativen Lösungsverfahrens (Relaxation)
  - Jacobi, Gauss-Seidel, Multigrid...
    - Jacobi ist leicht auf der GPU implementierbar
    - SA nutzt 40 Jacobi Iterationen

Quelle: 1

Poisson-Druck Lösung

- 2D Jacobi Iteration: Wiederholte Auswertung der Gleichung

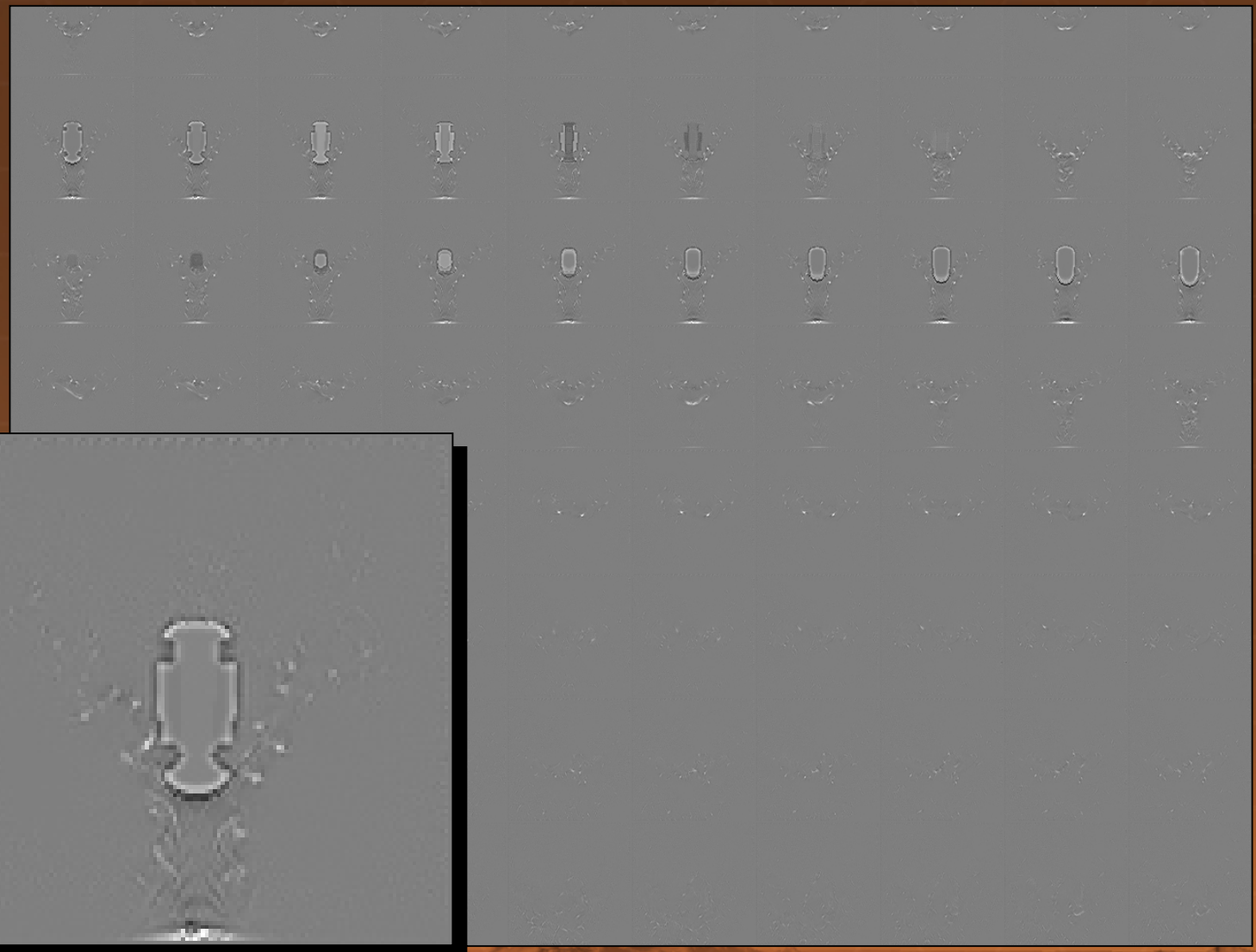
$$p_{i,j}^{n+1} = \frac{1}{4}\left(p_{i+1,j}^{n} + p_{i-1,j}^{n} + p_{i,j+1}^{n} + p_{i,j-1}^{n} - \delta^2(\nabla \cdot \mathbf{w}_3)\right),$$

$$\nabla \cdot \mathbf{w}_3 = \frac{1}{2\delta}(u3_{i+1,j} - u3_{i-1,j} + v3_{i,j+1} - v3_{i,j-1})$$

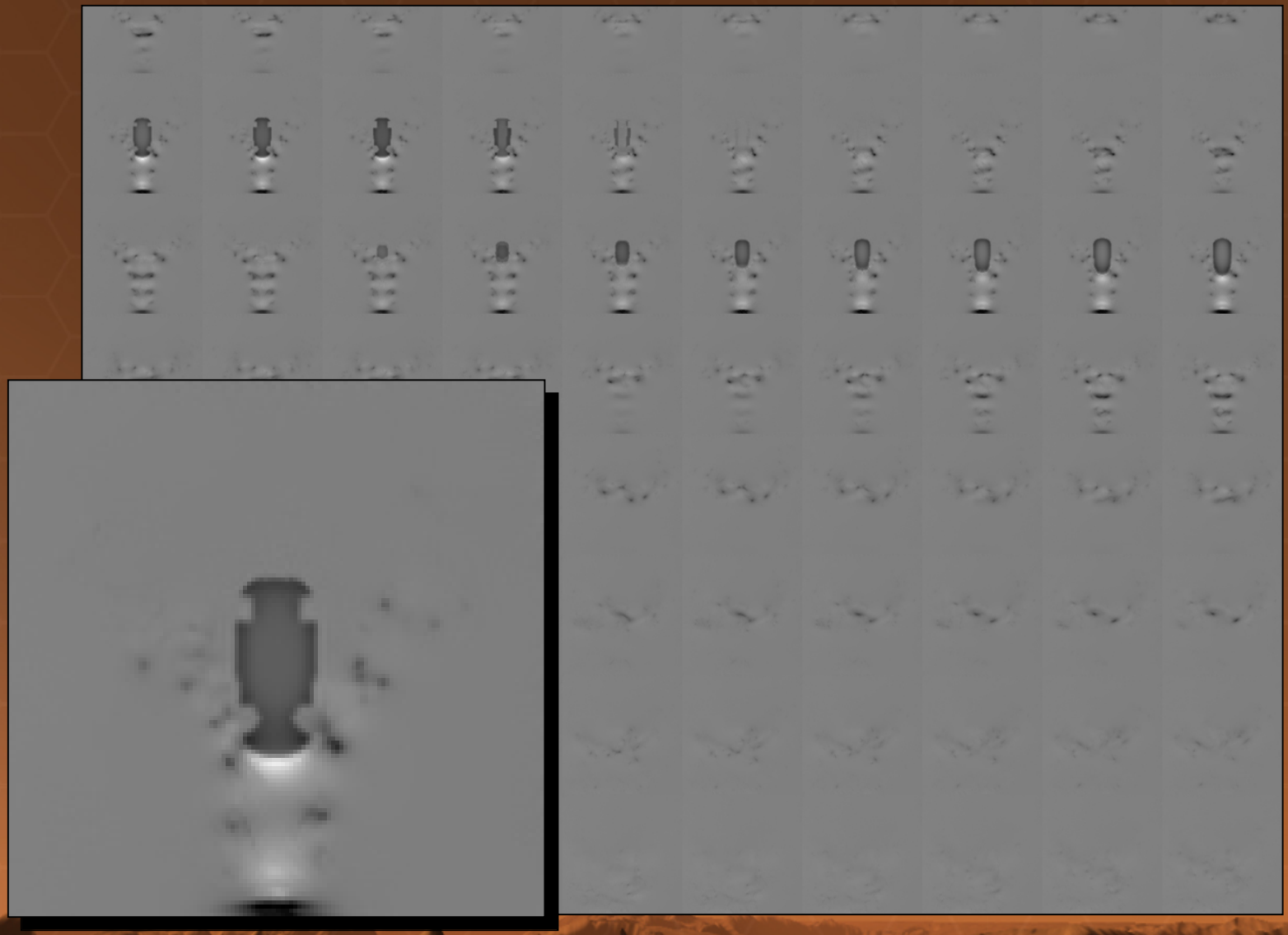
δ = Rasterabstand
u, v = Komponenten von $\mathbf{w}_3$
i, j = Rasterkoordinaten
n = Lösungs-Iteration

Quelle: 1

Divergenzfeld



Druckfeld



Algorithmus

- Stam 2000
  - Hinzufügen von Kräften
  - Advektion
  - Diffusion
  - Lösen der Druckgleichung
  - Ziehe den Gradienten des Druckfeldes vom Geschwindigkeitsfeld ab

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)\Delta t$$

$$\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$$

$$\left(\mathbf{I} - \nu\Delta t\nabla^2\right)\mathbf{w}_3 = \mathbf{w}_2$$

$$\nabla^2 p = \nabla \cdot \mathbf{w}_3$$

$$\mathbf{u}(\mathbf{x},t+\Delta t) = \mathbf{w}_3 - \nabla p$$

Quelle: 1

Druck-Gradient abziehen

$$\mathbf{u}(\mathbf{x}, t + \Delta t) = \mathbf{w}_3 - \nabla p$$

- Letzter Berechnungsschritt eines Zeitschritts
  - u ist jetzt ein divergenzfreies Geschwindigkeitsfeld

$$u_{i,j} = u3_{i,j} - \frac{1}{2}\delta\left(p_{i+1,j} - p_{i-1,j}\right)$$

$$v_{i,j} = v3_{i,j} - \frac{1}{2}\delta\left(p_{i,j+1} - p_{i,j-1}\right)$$

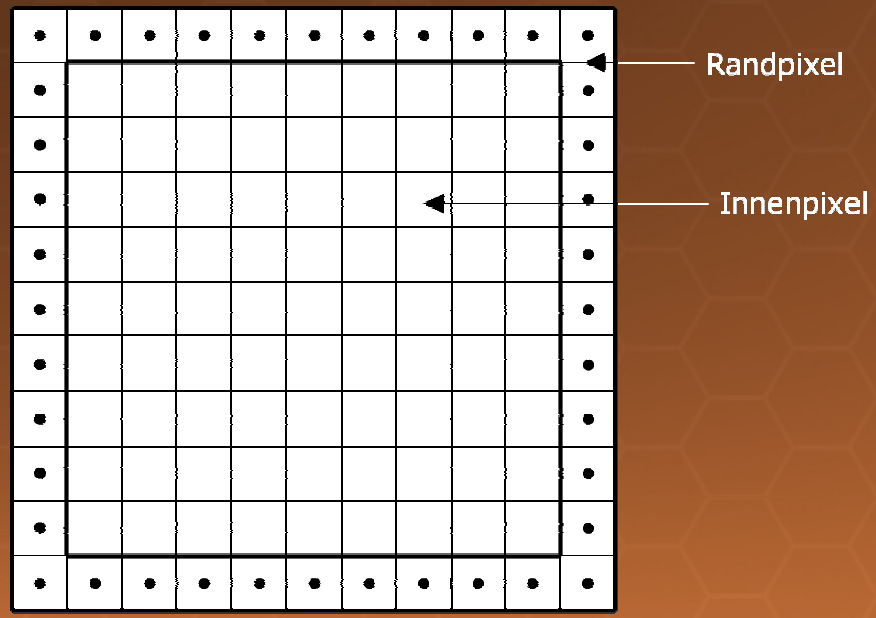
δ = Rasterabstand
i, j = Rasterkoordinaten
u, v = Komponenten von $\mathbf{u}$
$u3$,$v3$ = Komponenten von $\mathbf{w_3}$
p = Druck

Quelle: 1

Algorithmus

Randbedingungen

- Es gibt verschiedene Arten von Randbedingungen
  - No-slip, Free-slip, Neumann
- In SA: Geschwindigkeitsfeld
  Verwendet „no-slip“-Randbedingung
  - Geschwindigkeit u = 0 an den Rändern



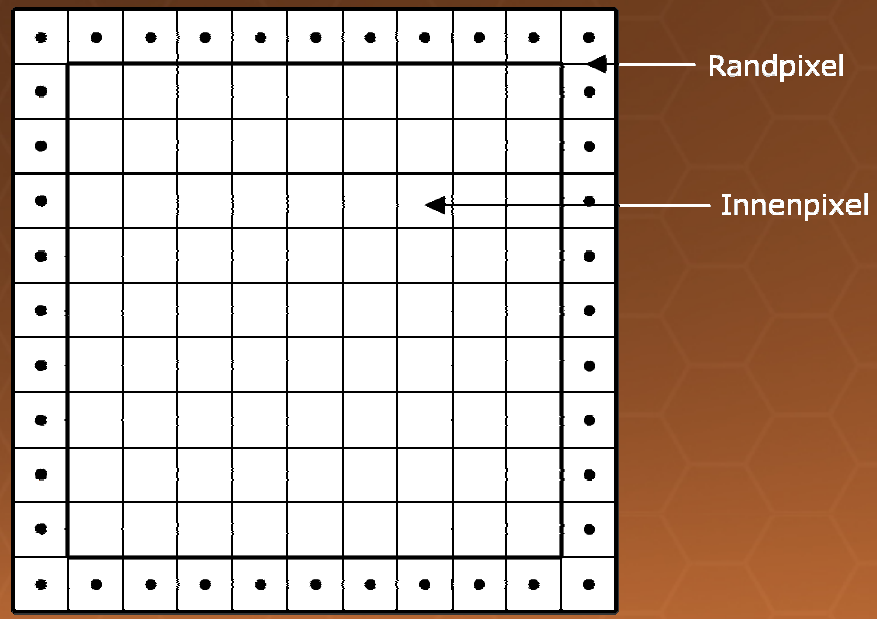
Randbedingungen

- Druck: Neumann Randbedingung

$$\frac{\partial p}{\partial n} = 0$$

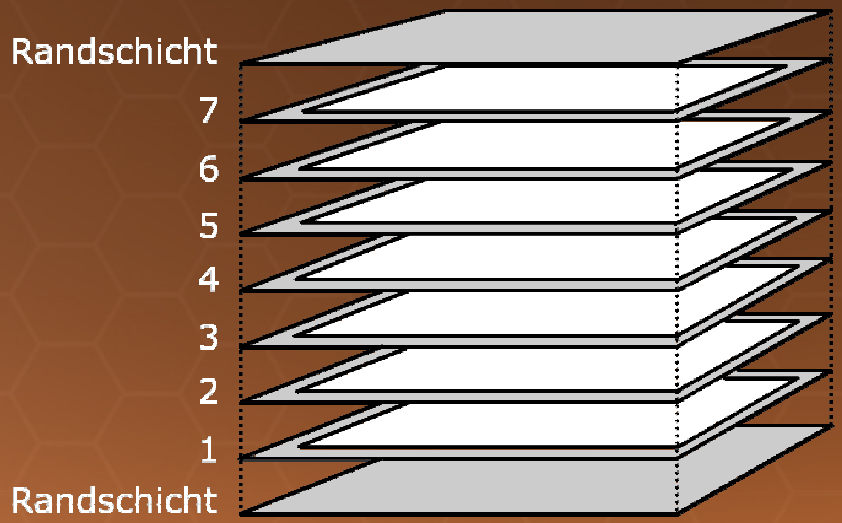
  - Wird in jedem Schritt der Iteration neu gesetzt

  - P(Rand) = p(nächster Nachbar, der kein Randvoxel ist)

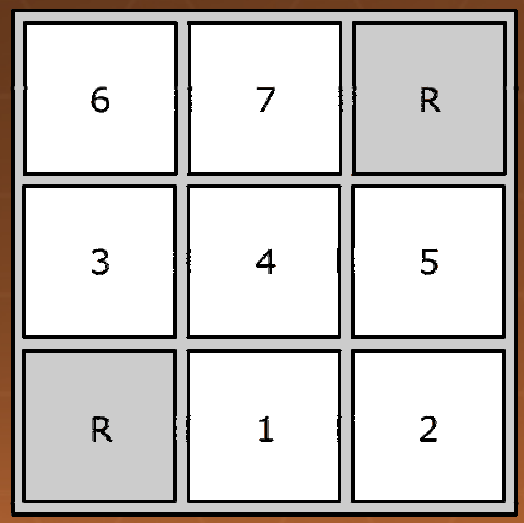


Flat 3D Textures

- Vorteile
  - Ein Textur-Update pro Operation
  - Bessere Nutzung der Parallelität der GPU -> schnell
  - Gut zu debuggen, Simulationsvorschau
  - Abwärtskompatibel



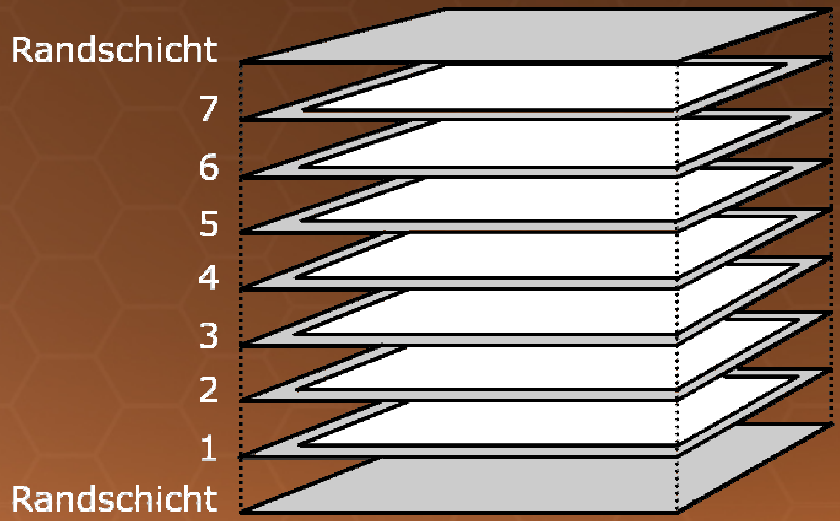
3D-Textur



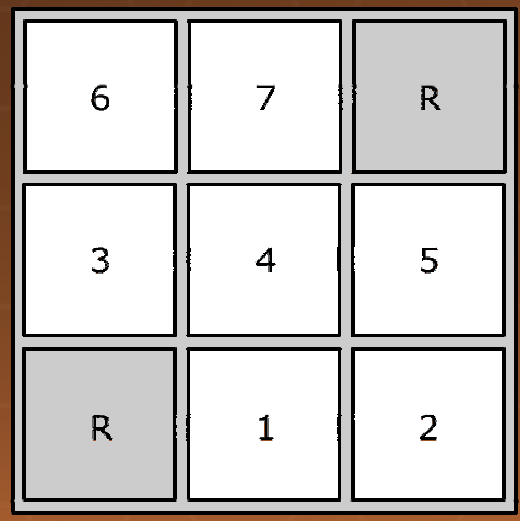
Flat 3D-Textur

Flat 3D Textures

- Nachteil
  - Komplexe Datenstruktur
  - Textur-Offsets müssen berechnet werden



3D-Textur

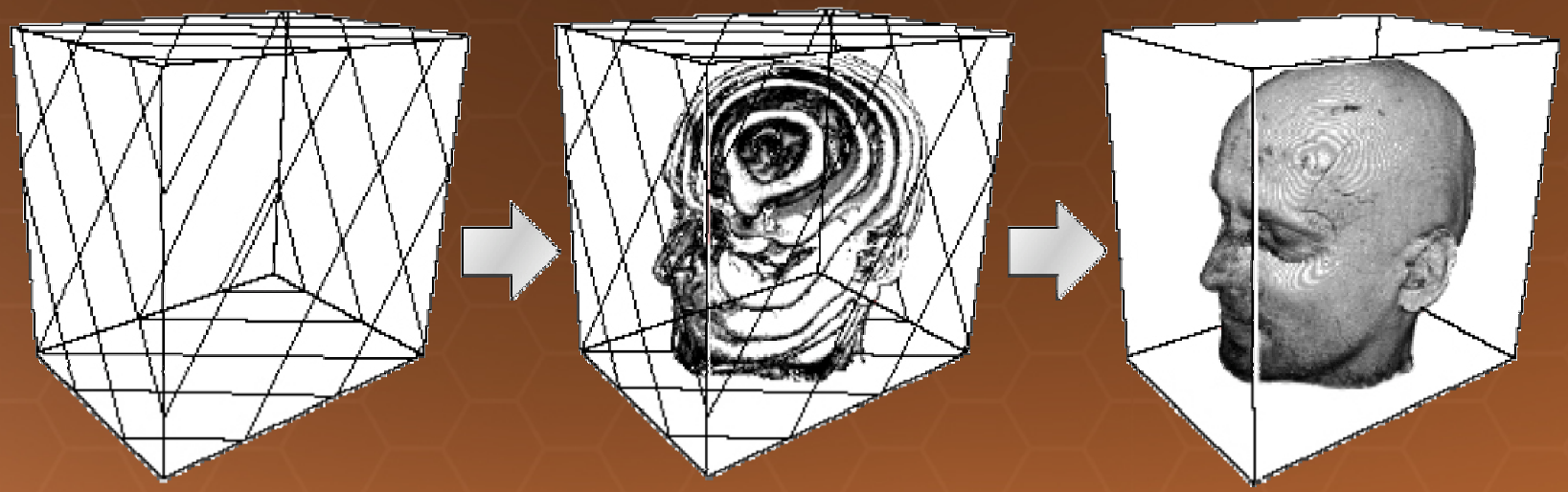


Flat 3D-Textur

Inhalt

Volumen Rendering

- View-Aligned Volume Slicing, in SA implementiert
  - Vorteil
    - kaum Artefakte, schnell



Polygon Slices 3D Textur Final Image

Quelle: 5

Inhalt

Hindernis-Erkennung

- Hindernisse verändern Flussverhalten des Fluids
  - Partikeldichte im Hindernisobjekt = Null
  - Geschwindigkeit entspricht der Geschwindigkeit des Objekts

Hindernis-Erkennung

- „Free slip“-Randbedingung verhindert eindringen des Fluids in das Objekt

  - Aktueller Geschwindigkeitswert Randvoxel (Geschwindigkeit Objekt + Normale)

    =

    Geschwindigkeit des Nachbarvoxels im Fluid

- Hinzufügen zum Geschwindigkeitsfeld
- Voxelisierung des Objekts notwendig

Hindernis-Erkennung

Inhalt

Voxelisierung

- 3D-Flat Textur: speichert Innenvoxel des Objekts

  - Render Objekt in Schleife über alle Schichten des Fluidvolumens mit Hilfe einer orthografischen Projektion

  - Far-clipping plane auf unendlich
  - Near-clipping plane Tiefenwert der aktuellen Schicht

  - Pro Schicht wird Objekt zweimal gerendert
    1. Mit backface culling: schreibe –1 in Alpha-Kanal
    2. Mit frontface culling: schreibe 1 in Alpha Kanal

    - Ergebnis: Alpha Kanal = 1: Voxel innerhalb des Objekts
      Alpha Kanal = 0: Voxel ausserhalb des Objekts

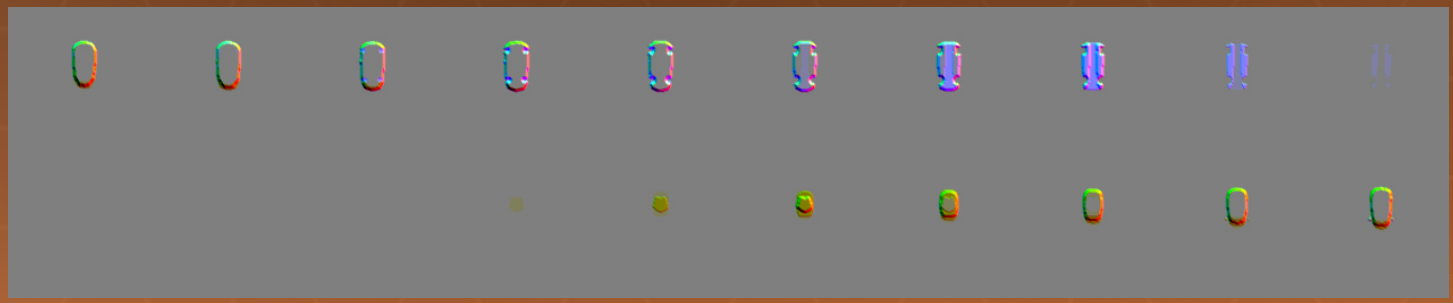
Voxelisierung

Voxelisierung

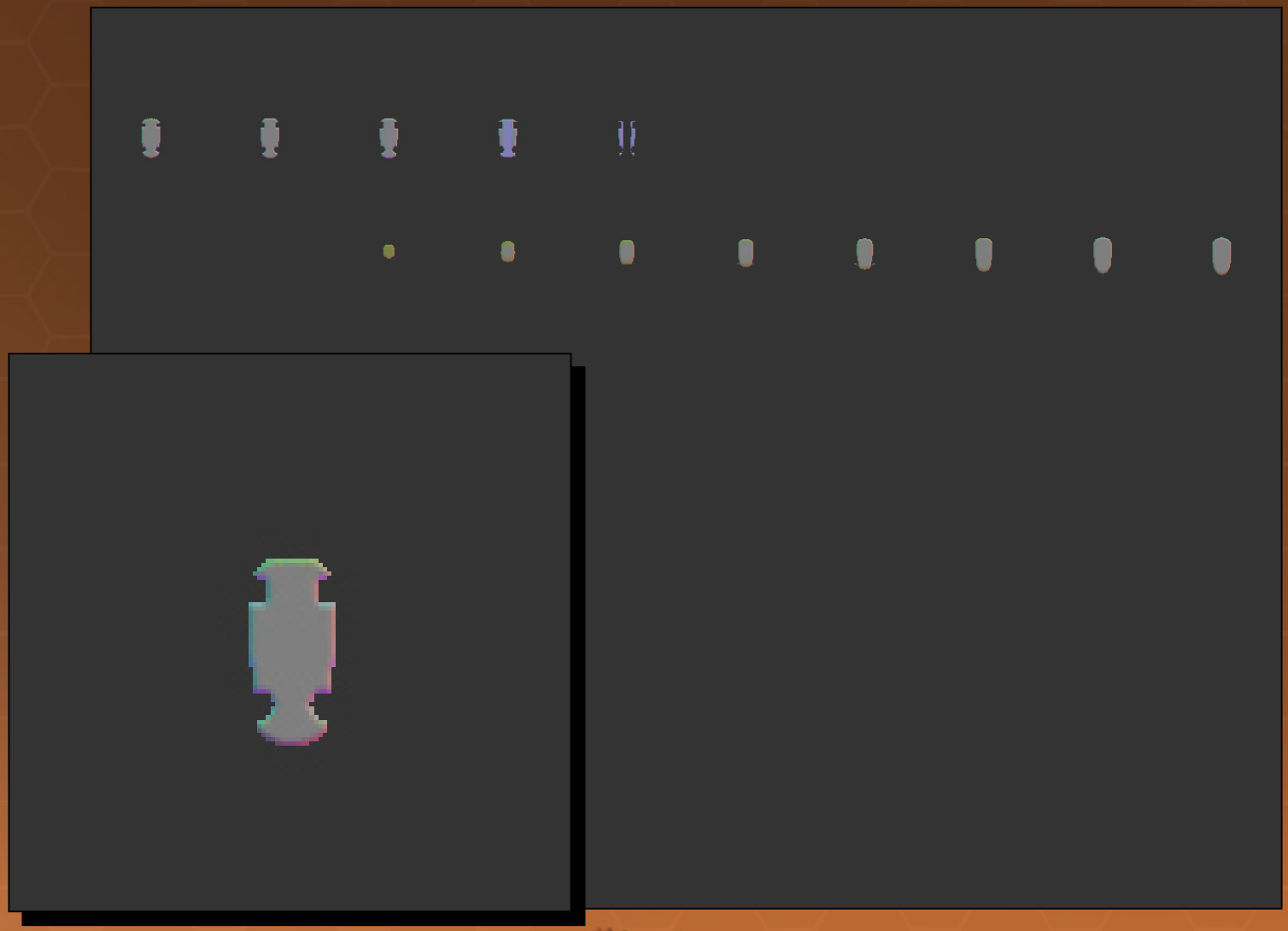
- Gradient des Alpha-Kanals der Voxel-Textur liefert Normalen

  - Geschwindigkeit des Objekts in RGB-Werten

  - Geschwindigkeit = Δ (VertexPosition) / Δt



Voxelisierung

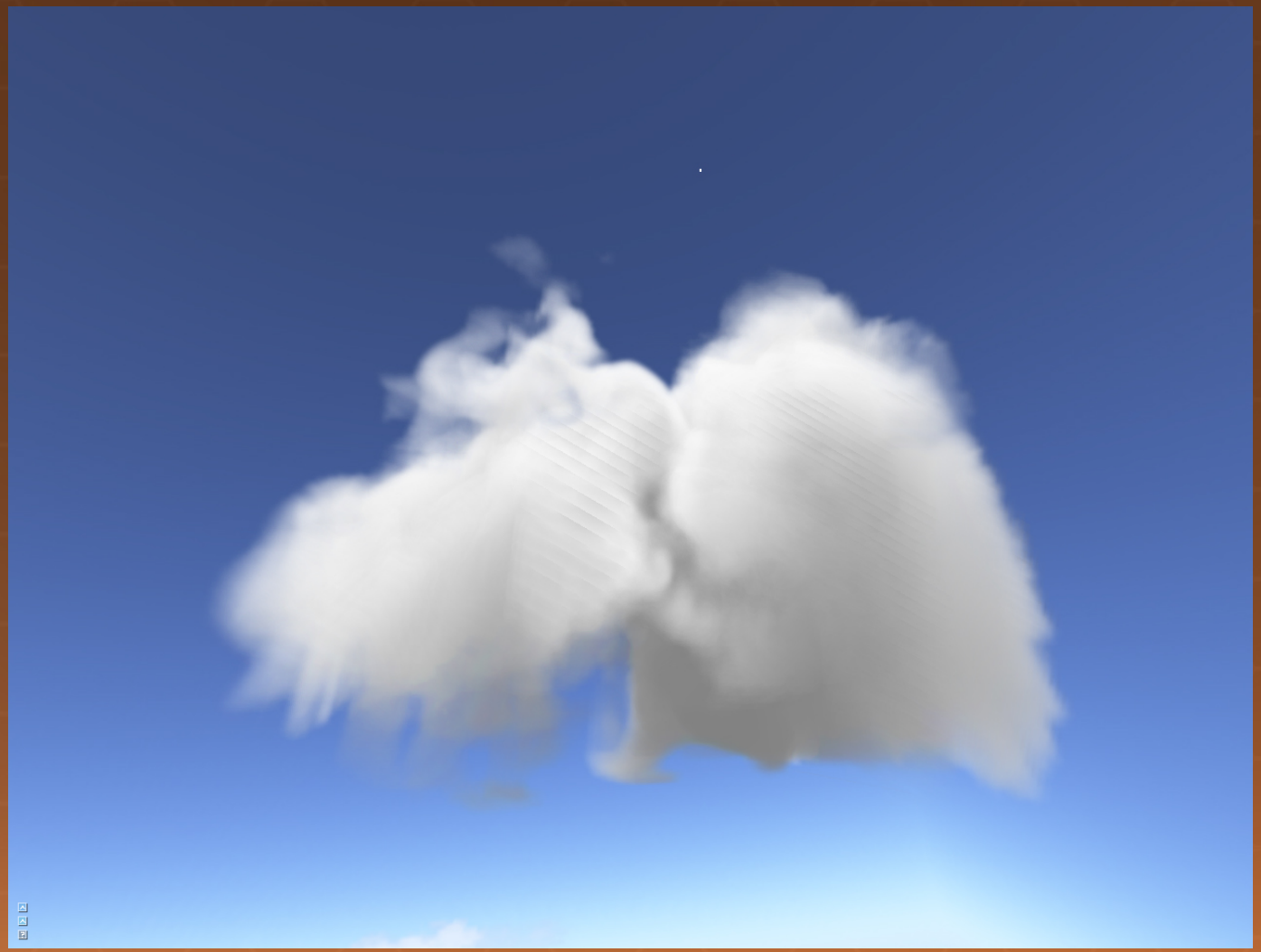


Inhalt

Beleuchtung

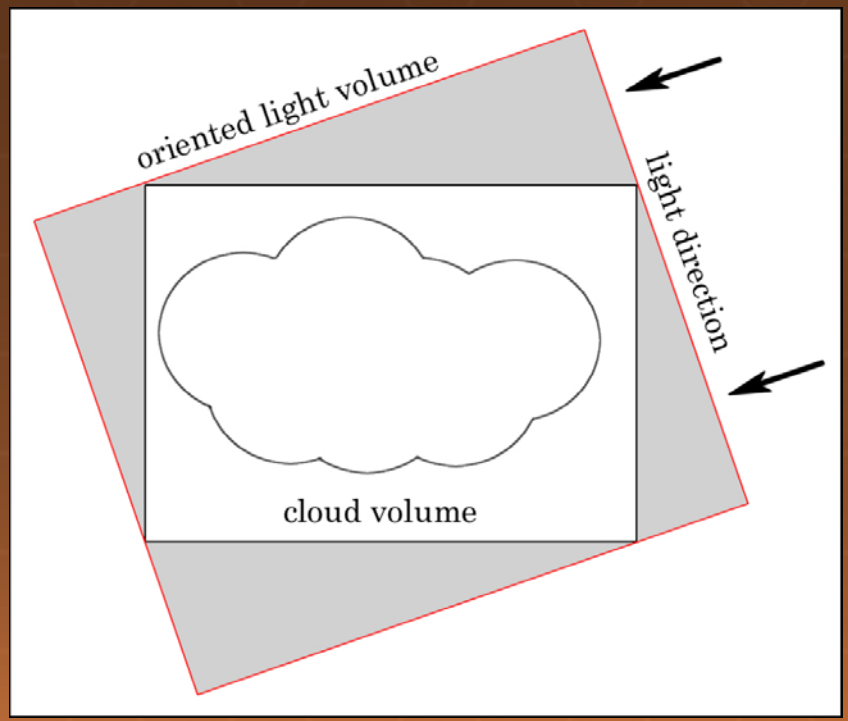
- Dringt Licht in eine Wolke ein, wird es vielfach gestreut (multiple scattering).
- Die Wolke scheint ein "Eigenleuchten" zu besitzen
- Schatten an der sonnenabgewandten Seite ist viel schwächer als bei soliden Objekten
- Problem: Multiple Scattering sehr teuer
  - Lösung: Annähern mit Oriented Light Volume und Inverse Shear Warp - Methode

Beleuchtung



Beleuchtung

- Erstelle neue 3D-Flat Textur “Oriented Light Volume”
- Speichert die Dämpfung des Lichts durch das Dichtevolumen

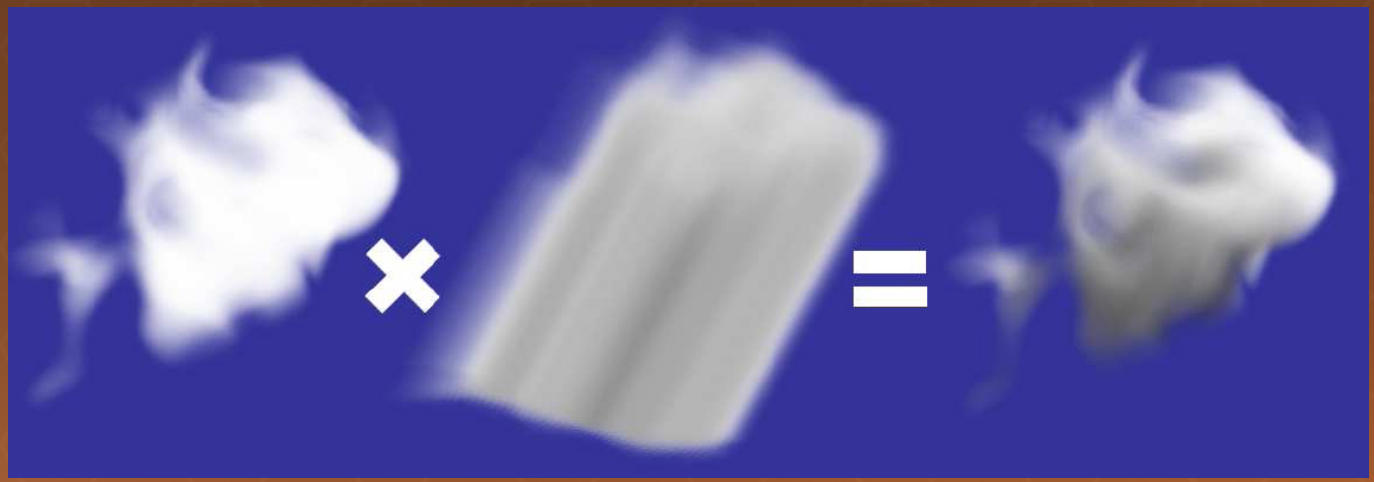


Quelle: 2

Beleuchtung

- Beleuchte Dichtevolumen:

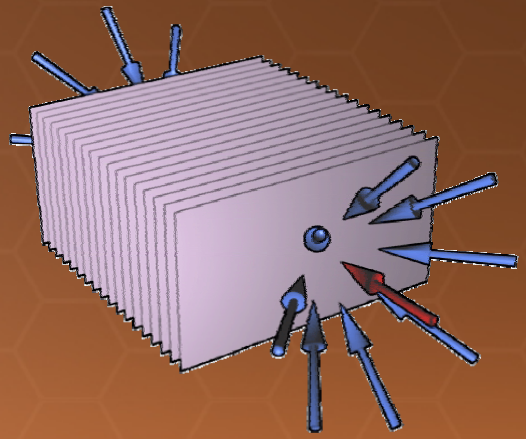
  - Multipliziere das Dichtevolumen mit dem “Oriented Light Volume”



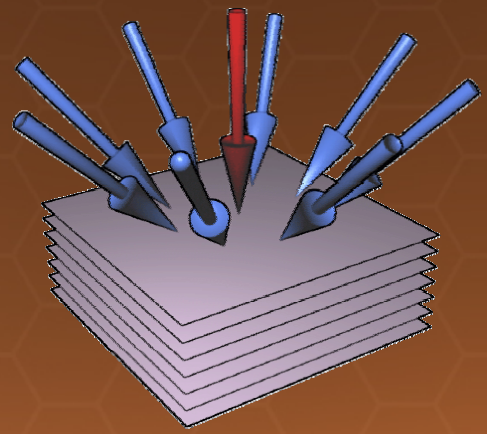
Quelle: 1

Beleuchtung

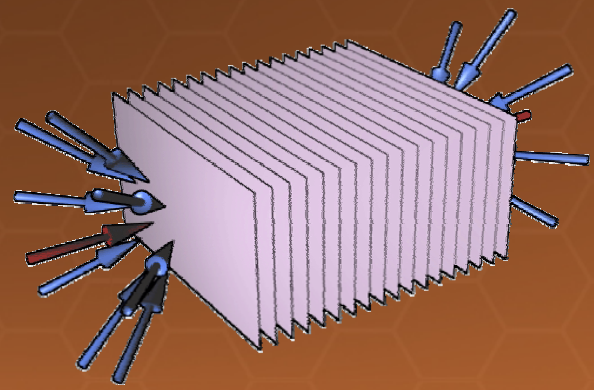
- Wo ist die Lichtquelle?
  - 3 Fälle:
    - max 45° um X-Achse
    - max 45* um Y-Achse
    - max 45° um Z-Achse



45° um X-Achse



45° um Y-Achse



45° um Z-Achse

Quelle: 2

Beleuchtung

- Position Kamera auf entsprechende Achse
- Blick auf Fluidzentrum

- Rendern die Schichten des Dichtevolumens

  - Starte mit der Schicht, die am nächsten zur Lichtquelle ist.
  - Schreibe Ergebnis in ping-pong-Schreibtextur

  - Tausche ping-pong Schreib- und Lesetextur

  - Render die nächste Schicht plus der ping-pong Lesetextur

  - Attenuationsoperator: Berechne Abschwächung des Lichts (Attenuation) in einem Fragment Shader

Beleuchtung

- Abschwächung des Lichts:

  - D_t = aktueller Dichtevolumen Alpha-Wert
  - μ = Abschwächungsrate (Attenuation Ratio)
  - D_{t-1} = Ergebnis des vorigen Rechenschritts

  - Abschwächung (Attenuation) = $1 - (1 - \mu) * D_t$

  - D_{t+1} = Attenuation * D_{t-1}

Beleuchtung

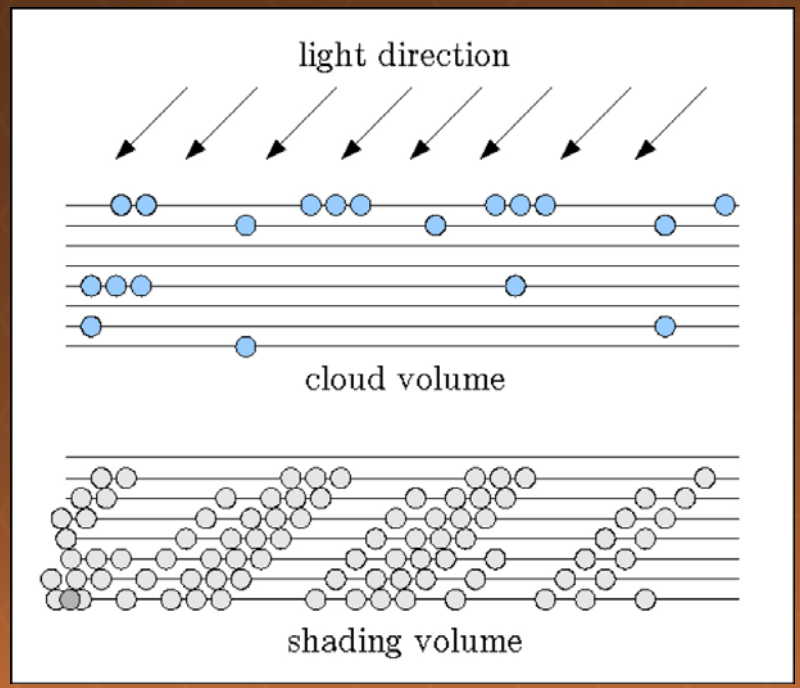
- Beliebige Lichtquellen-Position:

- Fall Y-Achse:
  - Lichtquellen-Position (x,y,z) um 45° zu (0,1,0)

  - Verschiebe Schichten in X-Richtung um x/y

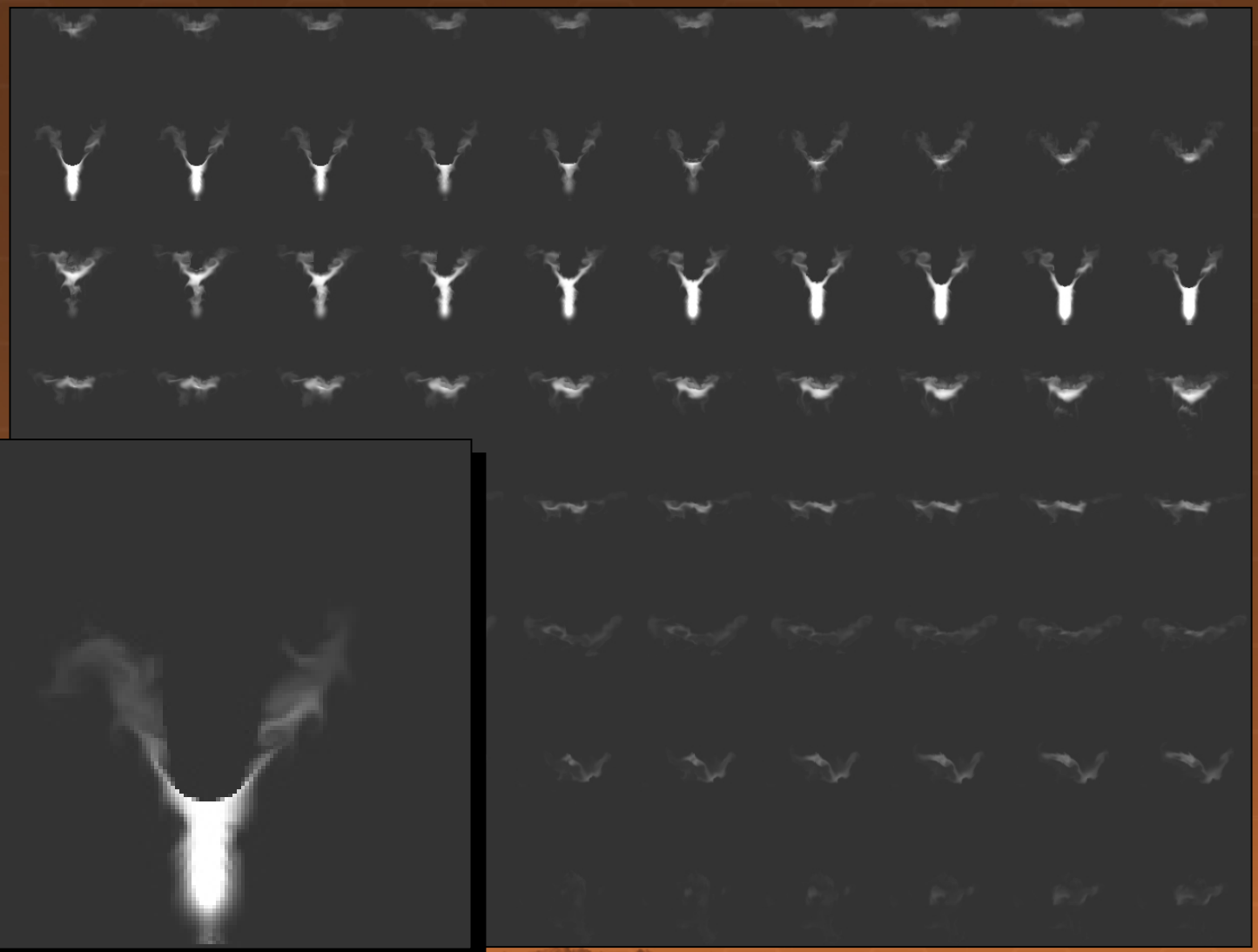
  - Verschiebe Schichten in Z-Richung um z/y

  - X-Achse und Z-Achse analog

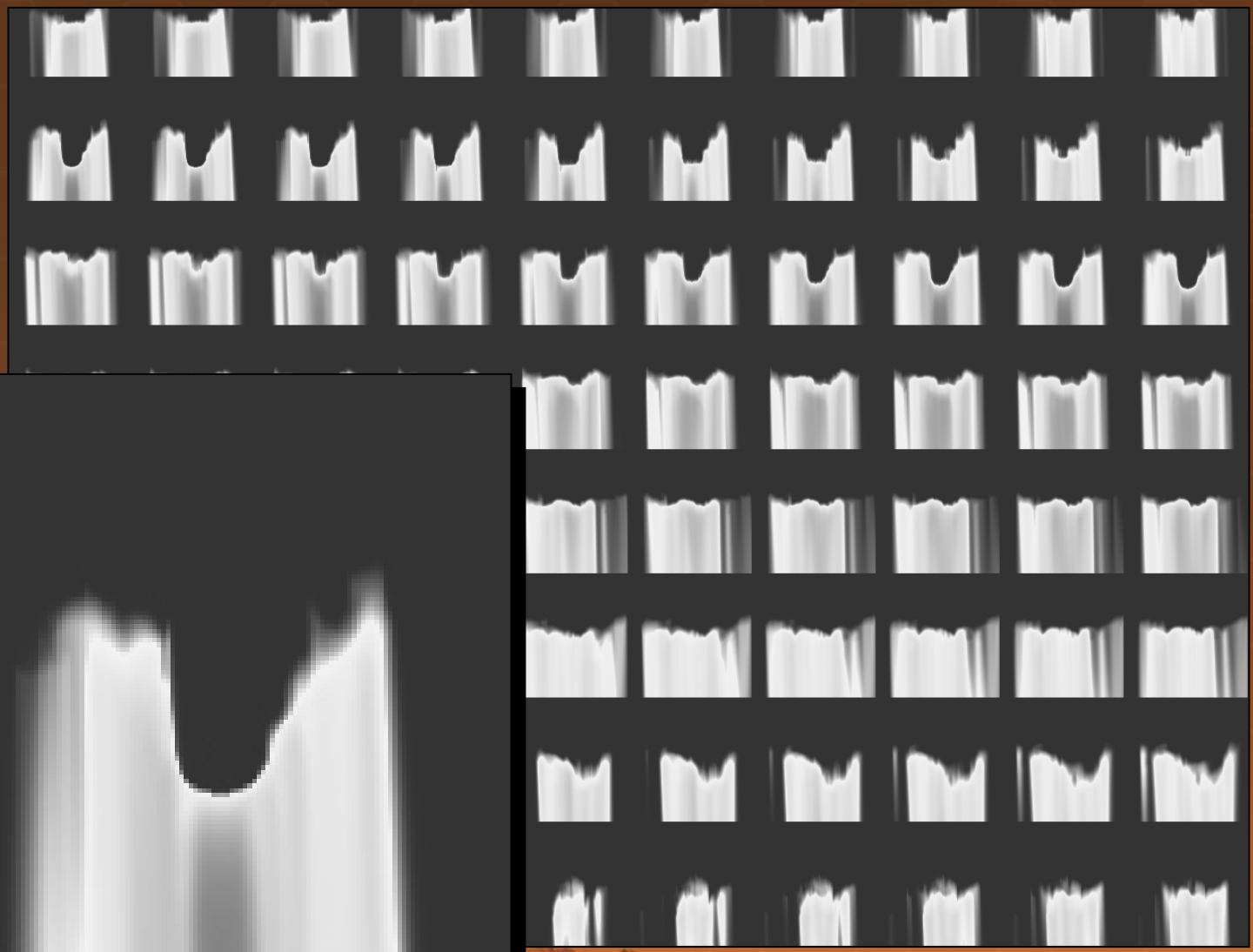


Quelle: 2

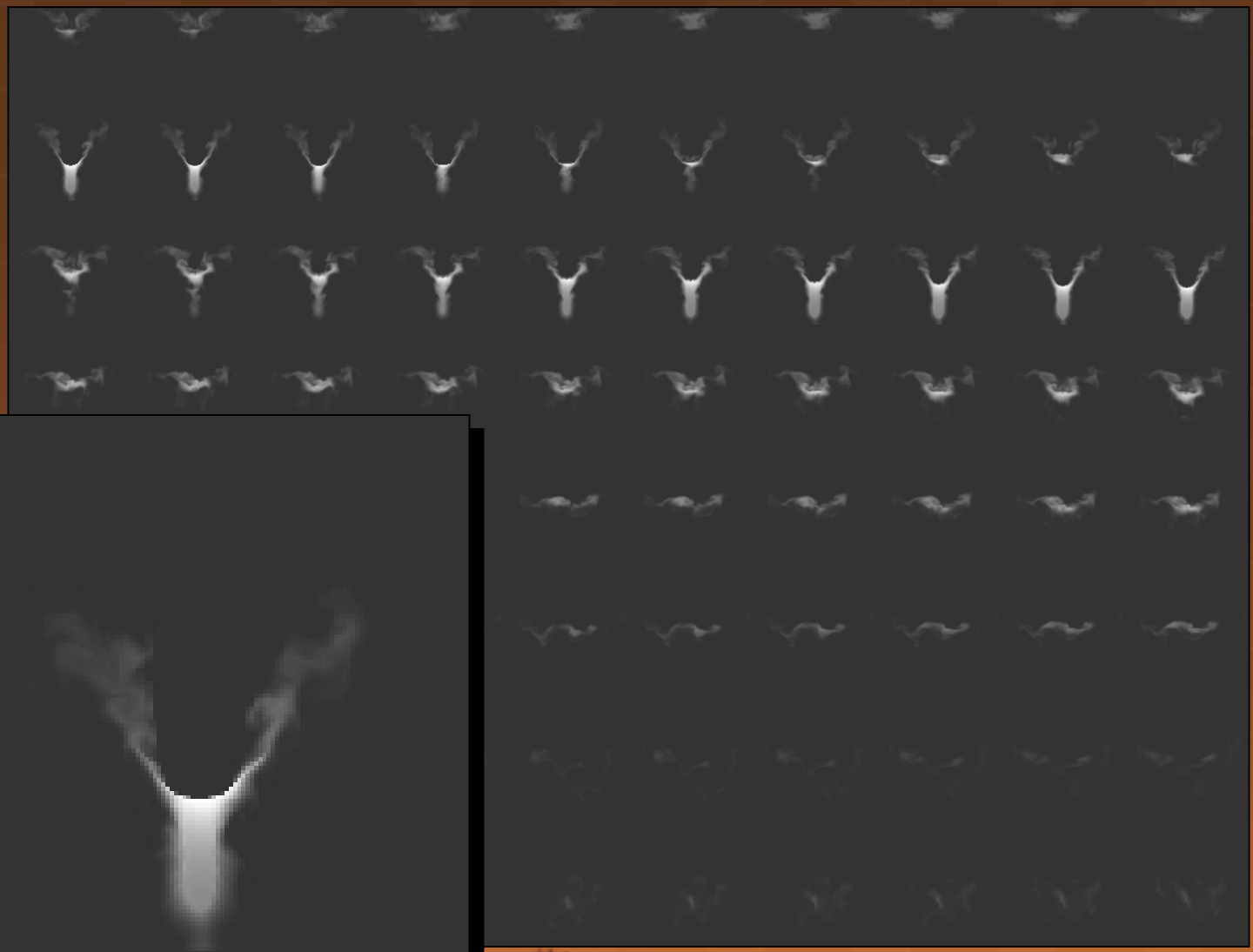
Beleuchtung



Beleuchtung

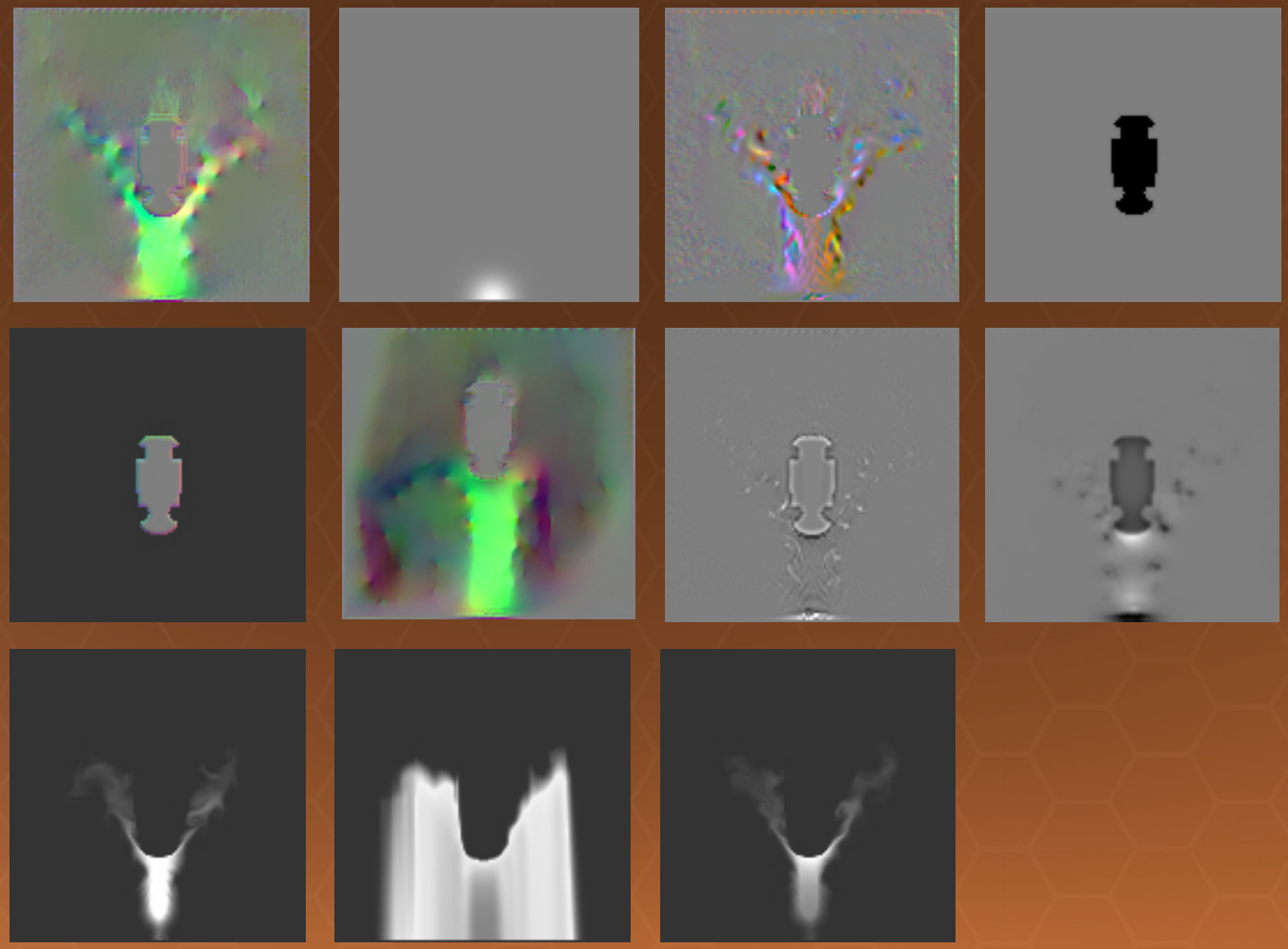


Beleuchtung

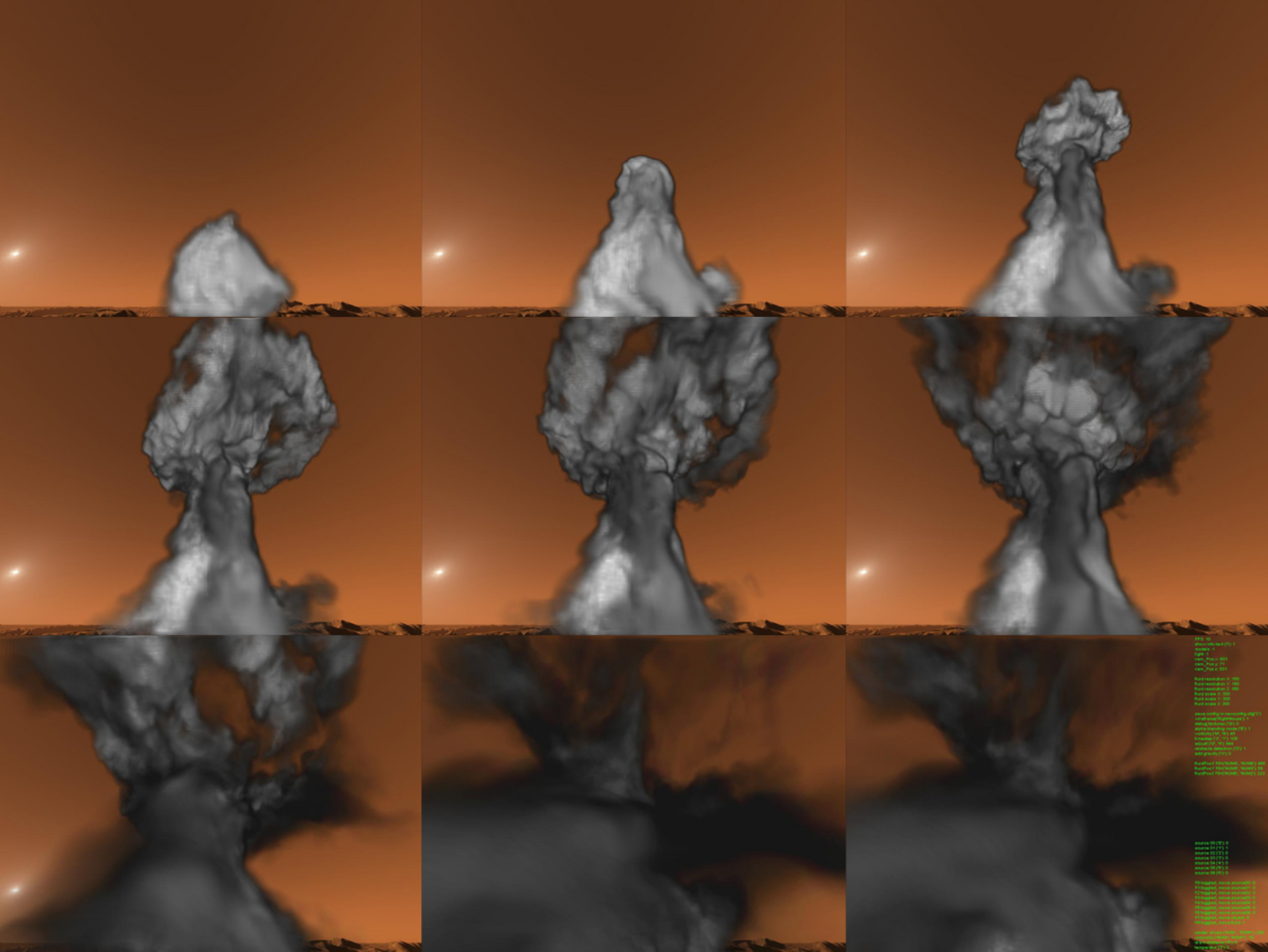


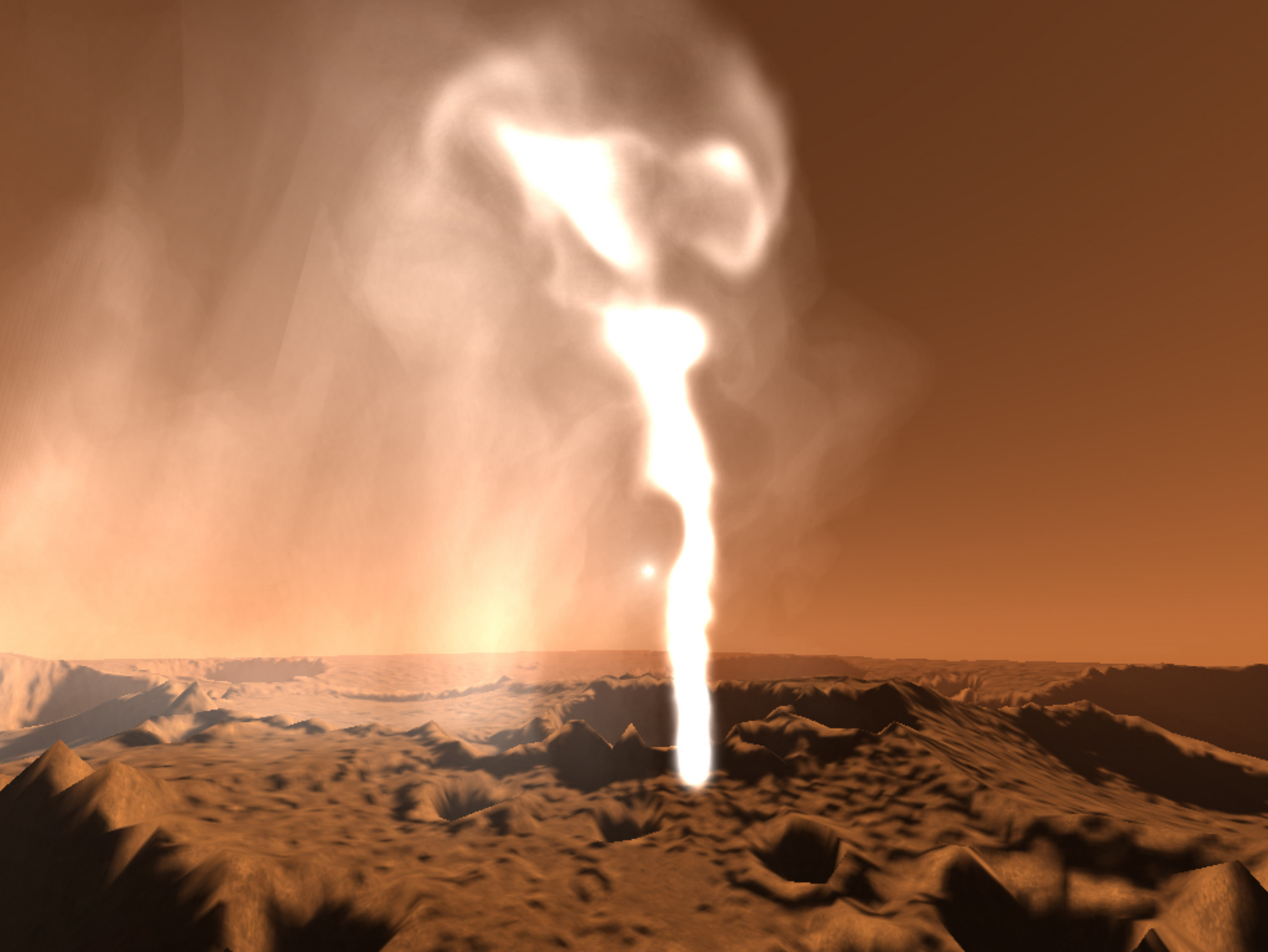
Inhalt

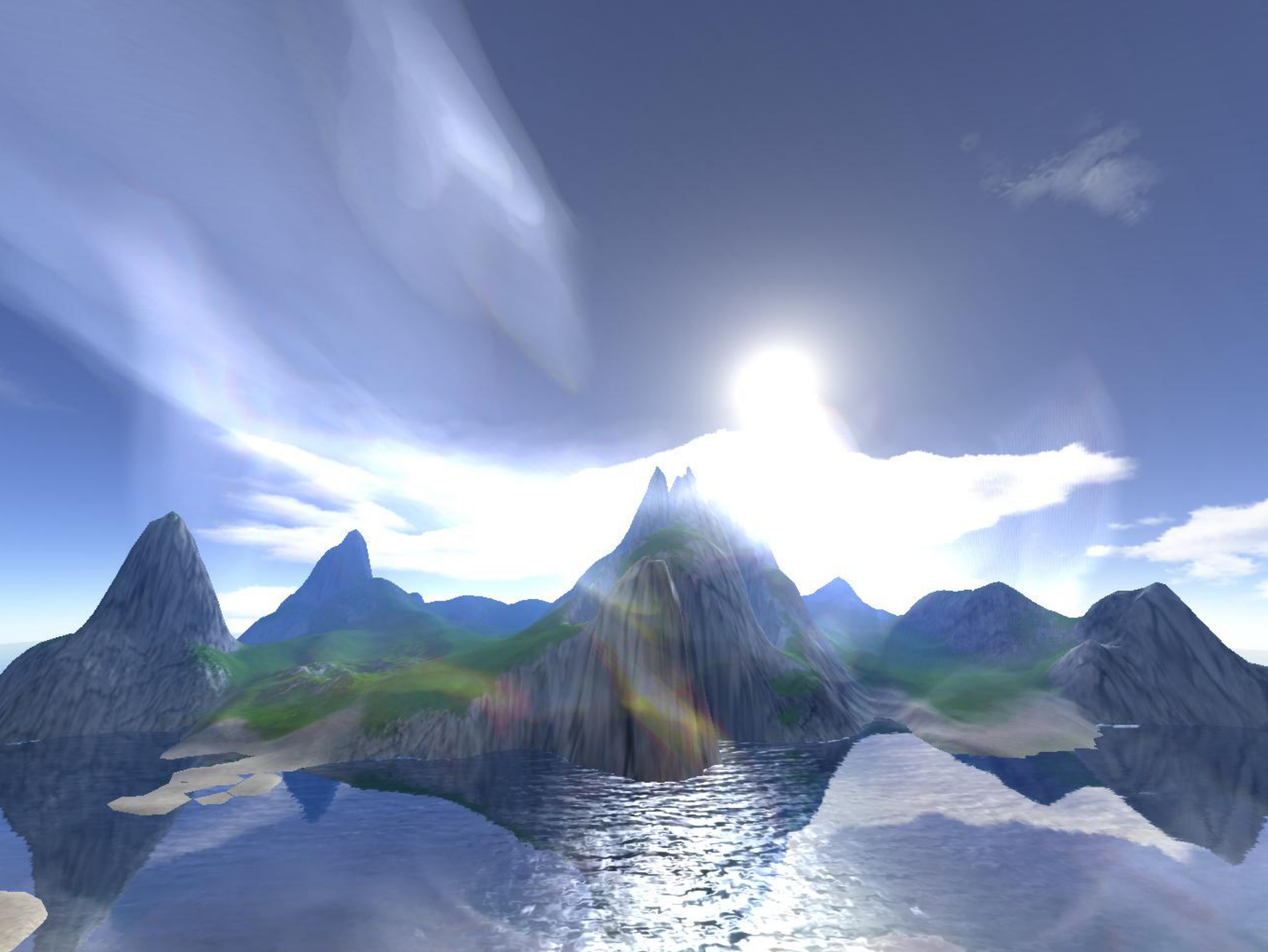
Simulation Überblick

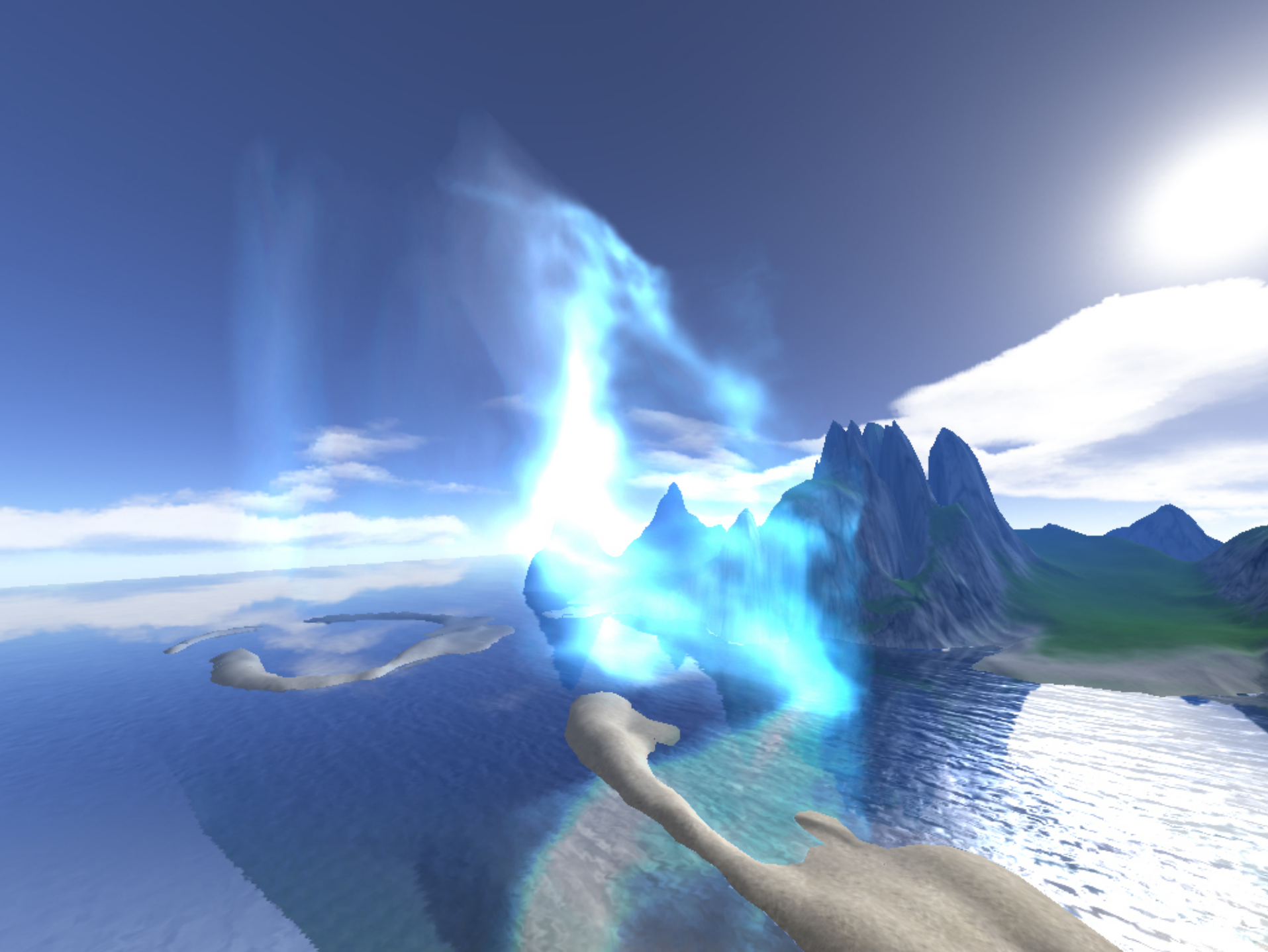


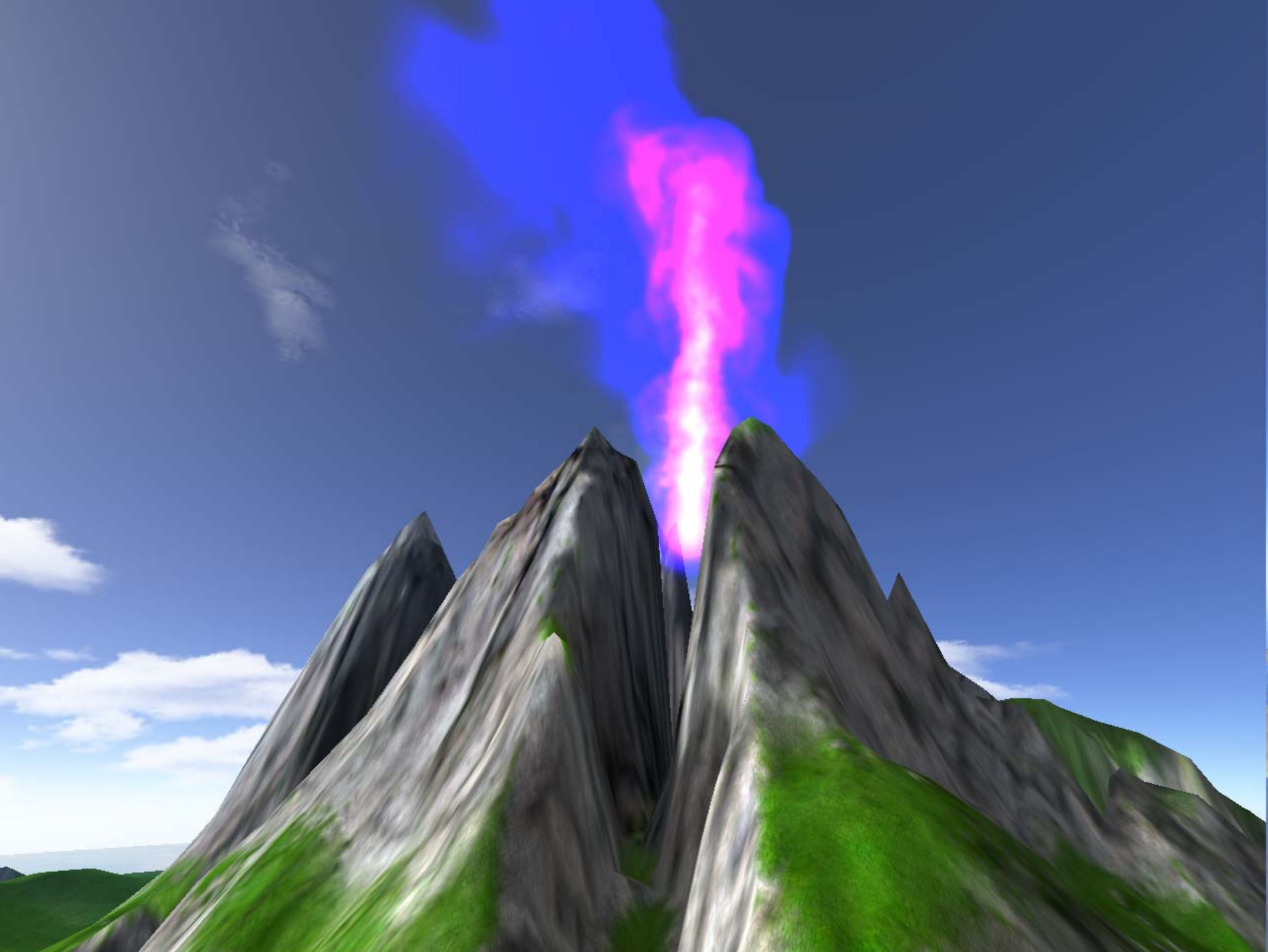
Ergebnisse

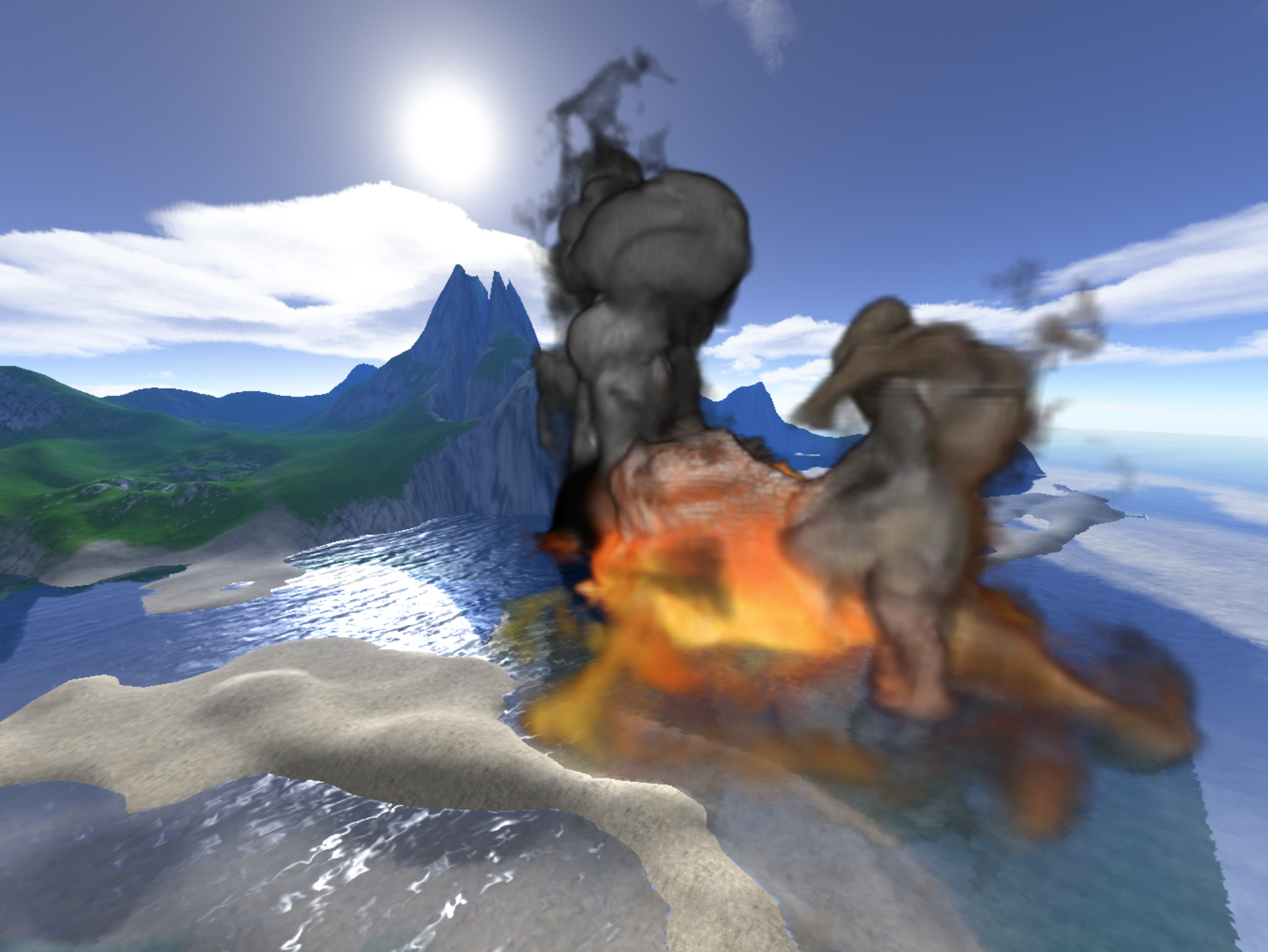


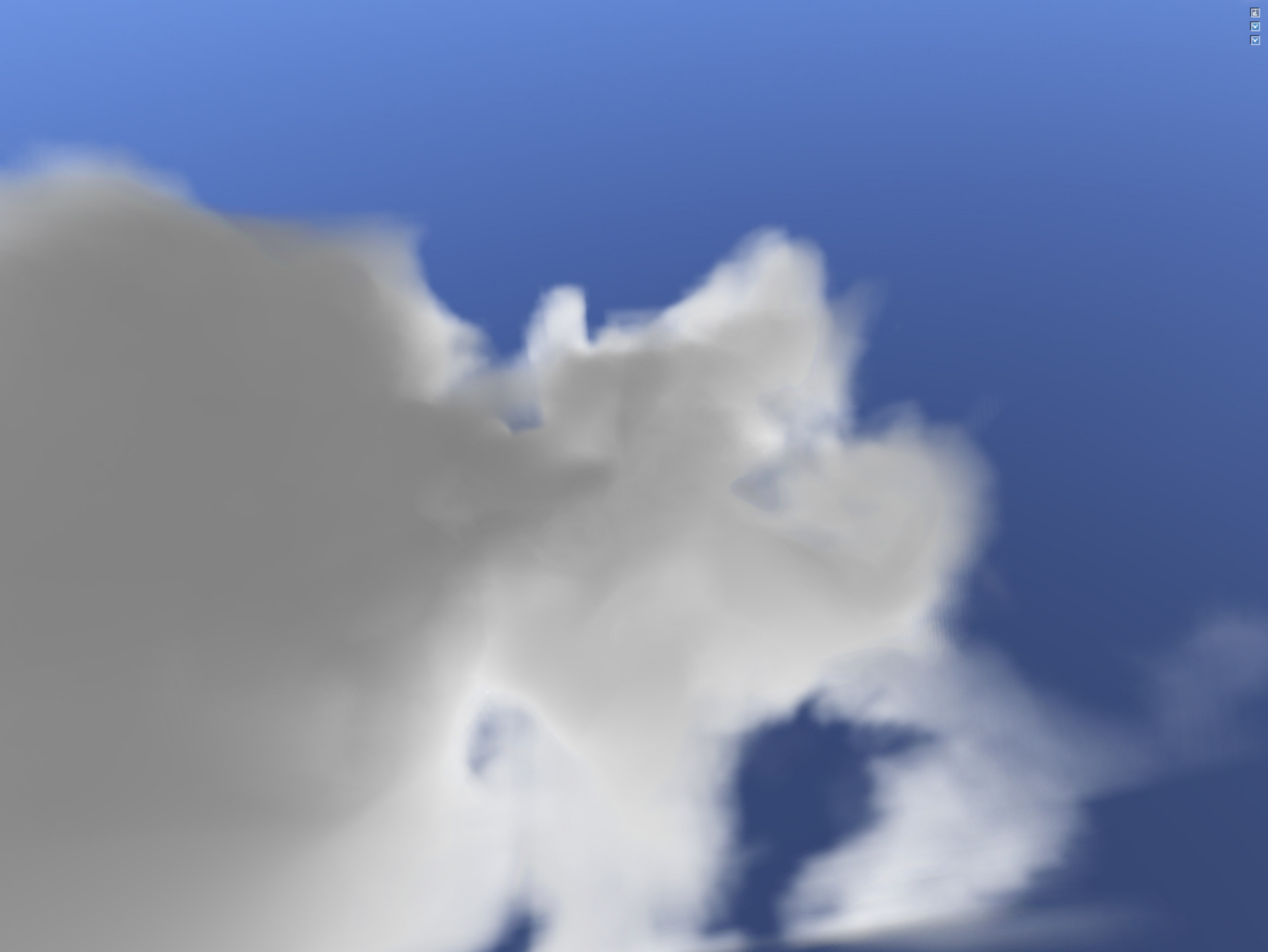


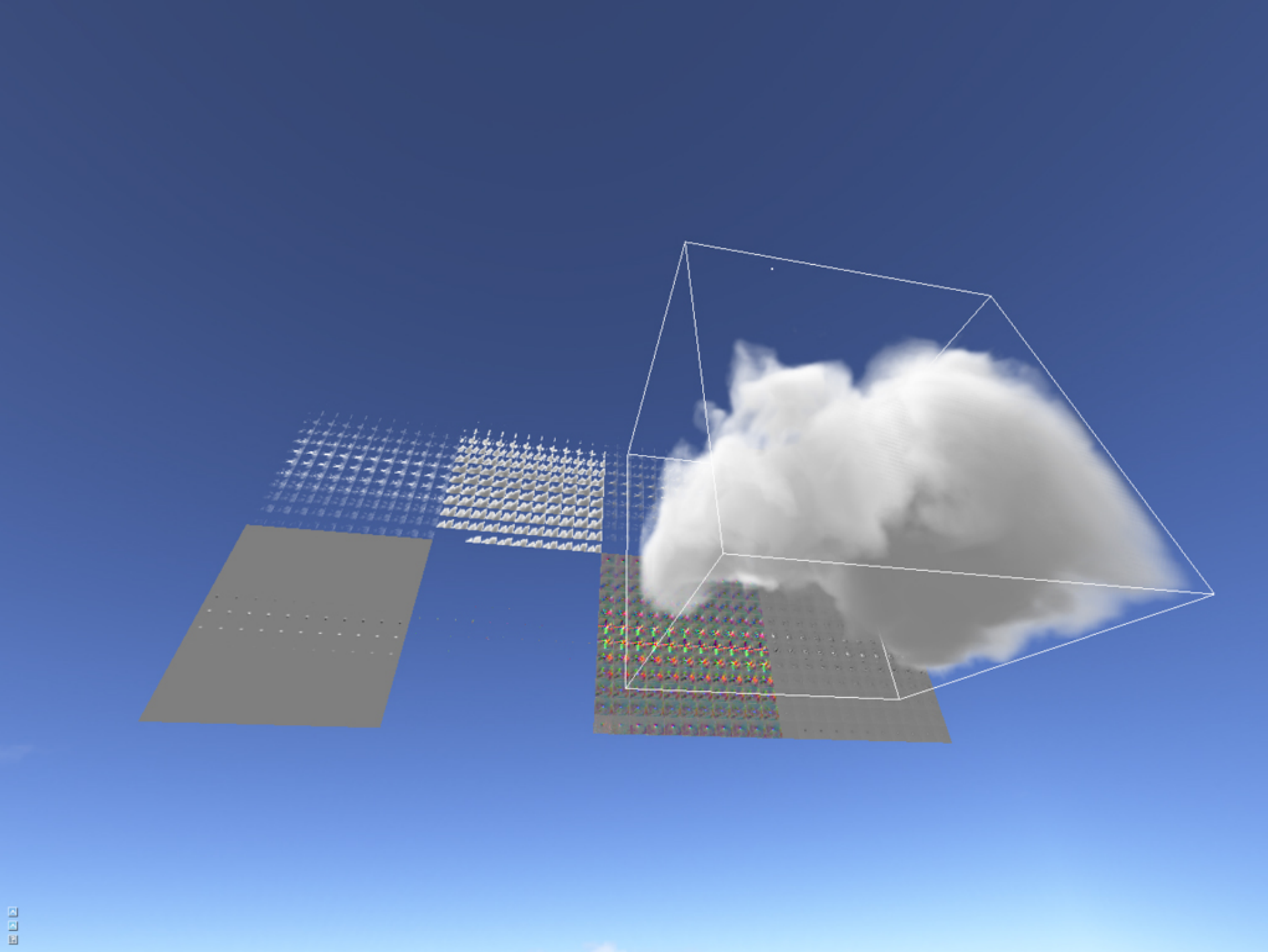


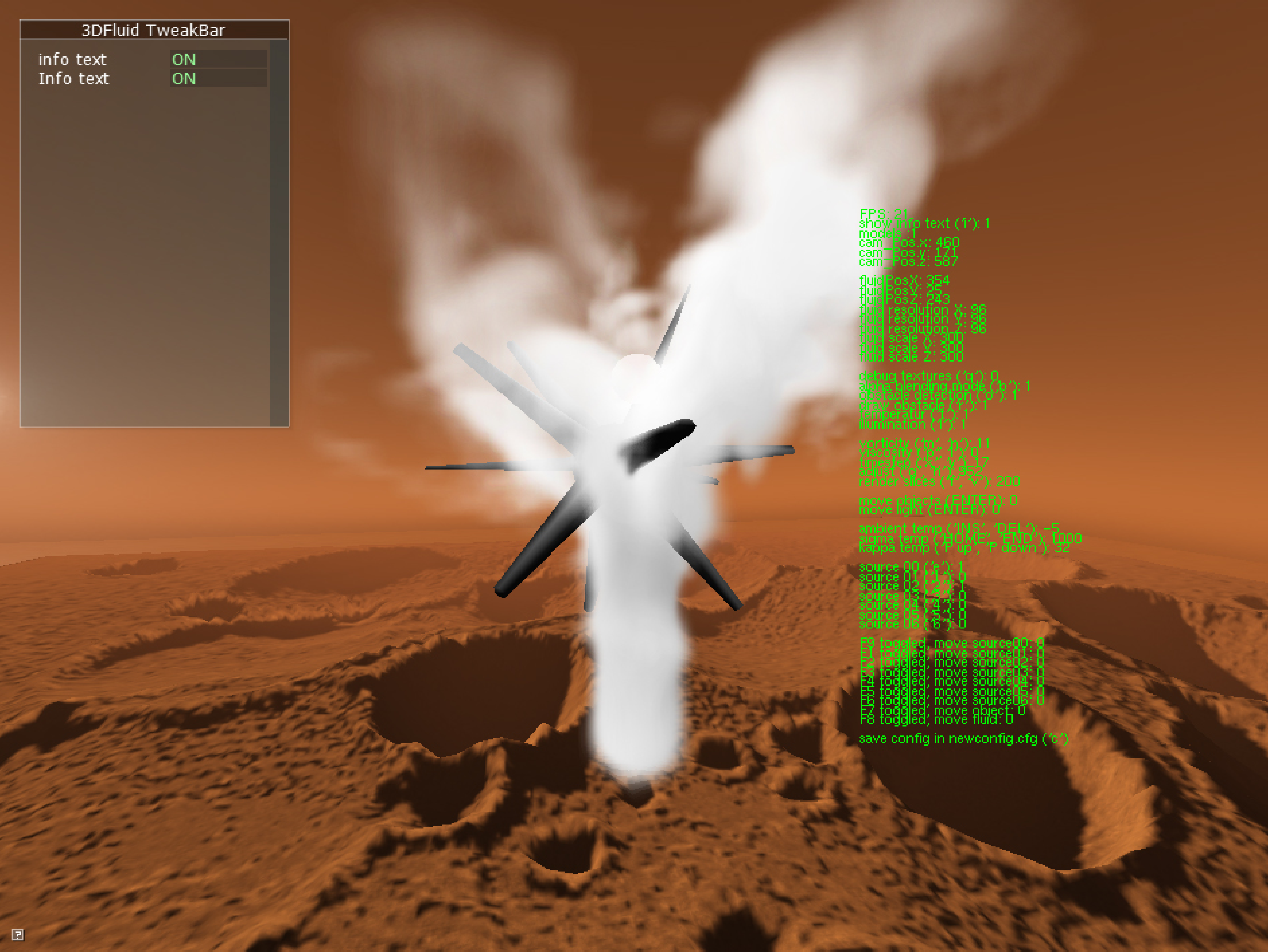











3DFluid TweakBar
info text ON
Info text ON
FPS: 21
show info text ('1'): 1
models :1
cam_Pos.x: 460
cam_Pos.y: 171
cam_Pos.z: 587
fluidPosX: 354
fluidPosY: 25
fluidPosZ: 243
fluid resolution X: 96
fluid resolution Y: 96
fluid resolution Z: 96
fluid scale X: 300
fluid scale Y: 300
fluid scale Z: 300
debug textures ('q'): 0
alpha blending mode ('b'): 1
obstacle detection ('o'): 1
draw obstacle ('r'): 1
temperatur ('t'): 1
illumination ('i'): 1
vorticity ('m', 'n'): 11
viscosity ('p', 'l'): 0
timestep ('x', 'y'): 17
adjust ('g', 'h'): 952
render slices ('f', 'v'): 200
move objects (ENTER): 0
move light (ENTER): 0
ambient temp ('INS', 'DEL'): -5
sigma temp ('HOME', 'END'): 1000
kappa temp ('P up', 'P down'): 32
source 00 ('e'): 1
source 01 ('1'): 0
source 02 ('2'): 1
source 03 ('3'): 0
source 04 ('4'): 0
source 05 ('5'): 0
source 06 ('6'): 0
F9 toggled, move source00: 0
F1 toggled, move source01: 0
F2 toggled, move source02: 0
F3 toggled, move source03: 0
F4 toggled, move source04: 0
F5 toggled, move source05: 0
F6 toggled, move source06: 0
F7 toggled, move object: 0
F8 toggled, move fluid: 0
save config in newconfig.cfg ('c')

Quellen

[1] Harris, Mark J.: Real-Time Cloud Simulation and Rendering, University of North Carolina at Chapel Hill, Diss., 2003

[2] Fizimayer, Robert: A Real-Time Cloud Animation and Illumination Method, Technische Universität Wien.

[3] Müller, Matthias; Charypar, David; Gross, Markus: Particle-based fluid simulation for interactive applications. In: SCA `03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2003

[4] Fedkiw, Ronald; Stam, Jos; Wann Jensen Henrik: Visual Simulation of Smoke. In: Proceedings of ACM SIGGRAPH 2001, 2001

[5] Willkomm, Dennis: Visuelle Effekte mit volumetrischen Shadern – Studienarbeit an der Universität Koblenz-Landau

[6] Tariq, Sarah; Llamas, Ignacio: Real-Time Volumetric Smoke using D3D10. Proceedings of the Game Developer Conference 2007, März 2007. – Nvidia Developer Technology

[7] GSC Game World: S.T.A.L.K.E.R.: Clear Sky. Version: 2008. - S.T.A.L.K.E.R.: Clear Sky HD DirectX10 Feature Demo Video. URL: http://stalker.deepsilver.com

Fragen...