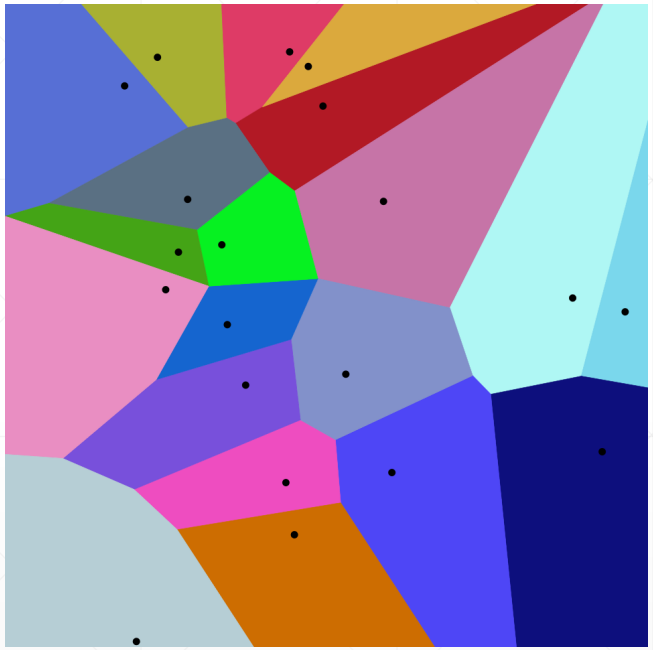
Voronoi Diagrams in 3D Space

Samuel Essig

Fortgeschrittenen Software Praktikum
Hauptseminar "Computergraphik und Visualisierung"
Universität Heidelberg, 15. Dezember 2025

Motivation

- Euklidische Metrik ist intuitiv und allgegenwärtig
- Andere Metriken sind ungewohnt
- Abstände bestimmen Nähe, Nachbarschaft und Struktur im Raum
- Visualisierung der Voronoi Diagramme um das Verhalten einer Metrik besser zu verstehen

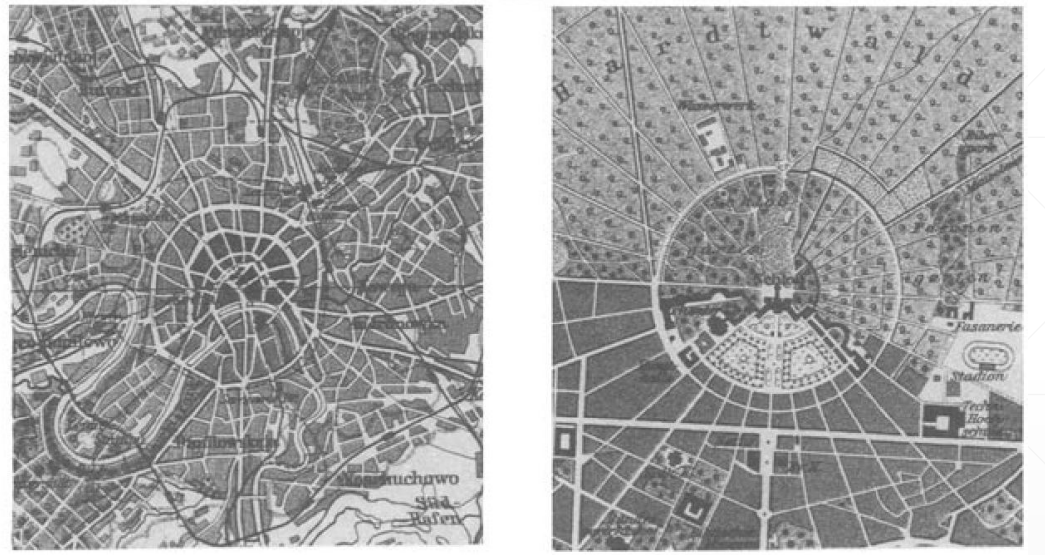


[1]

[1] https://commons.wikimedia.org/wiki/File:Euclidean_Voronoi_diagram.svg

Karlsruhe Metrik (Moskau Metrik)

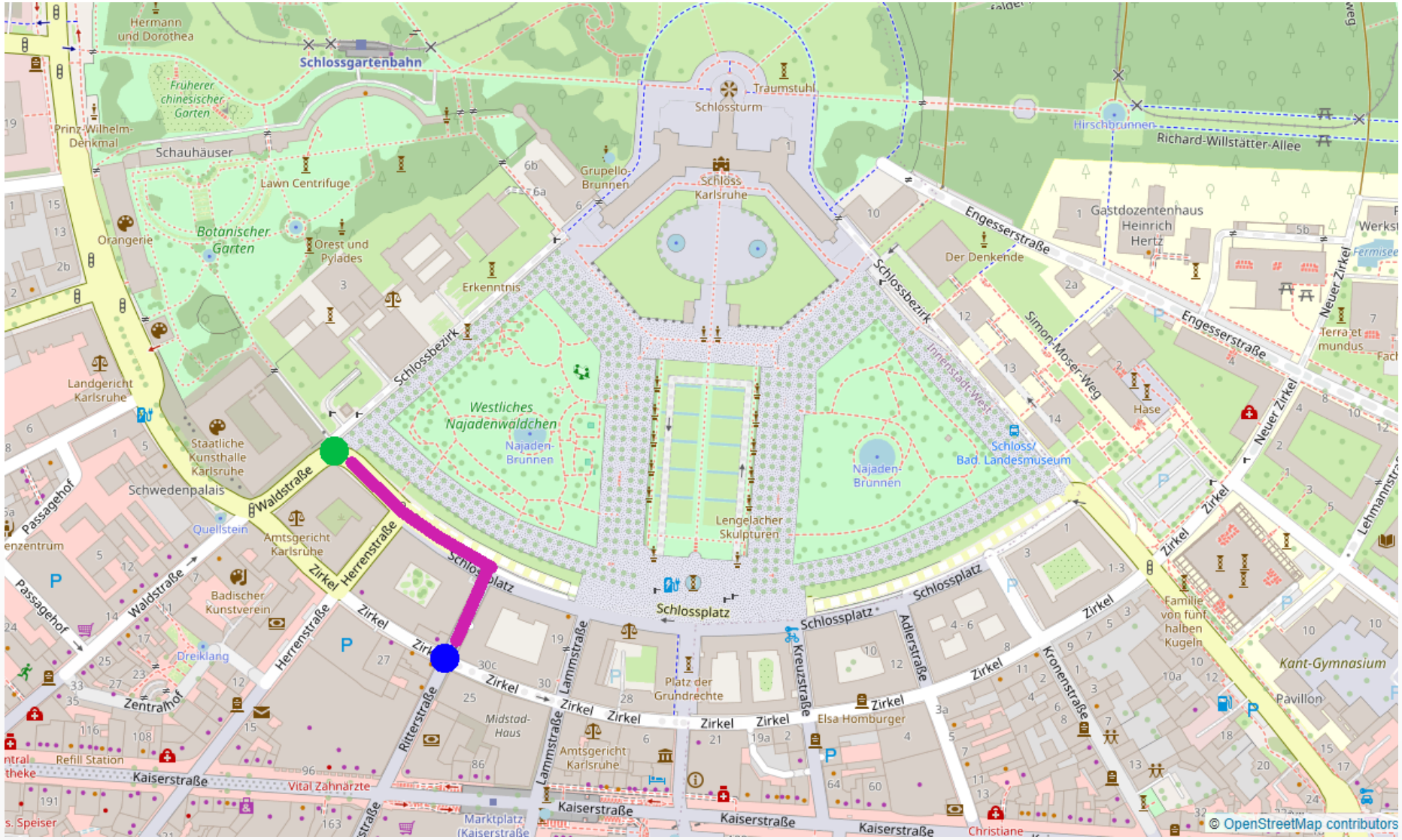
- Inspiriert vom Stadtplan von Karlsruhe
- Kürzester Pfad aus
  - radialen Strahlen vom Ursprung
  - Kreisbögen um den Ursprung
- Stückweise Metrik



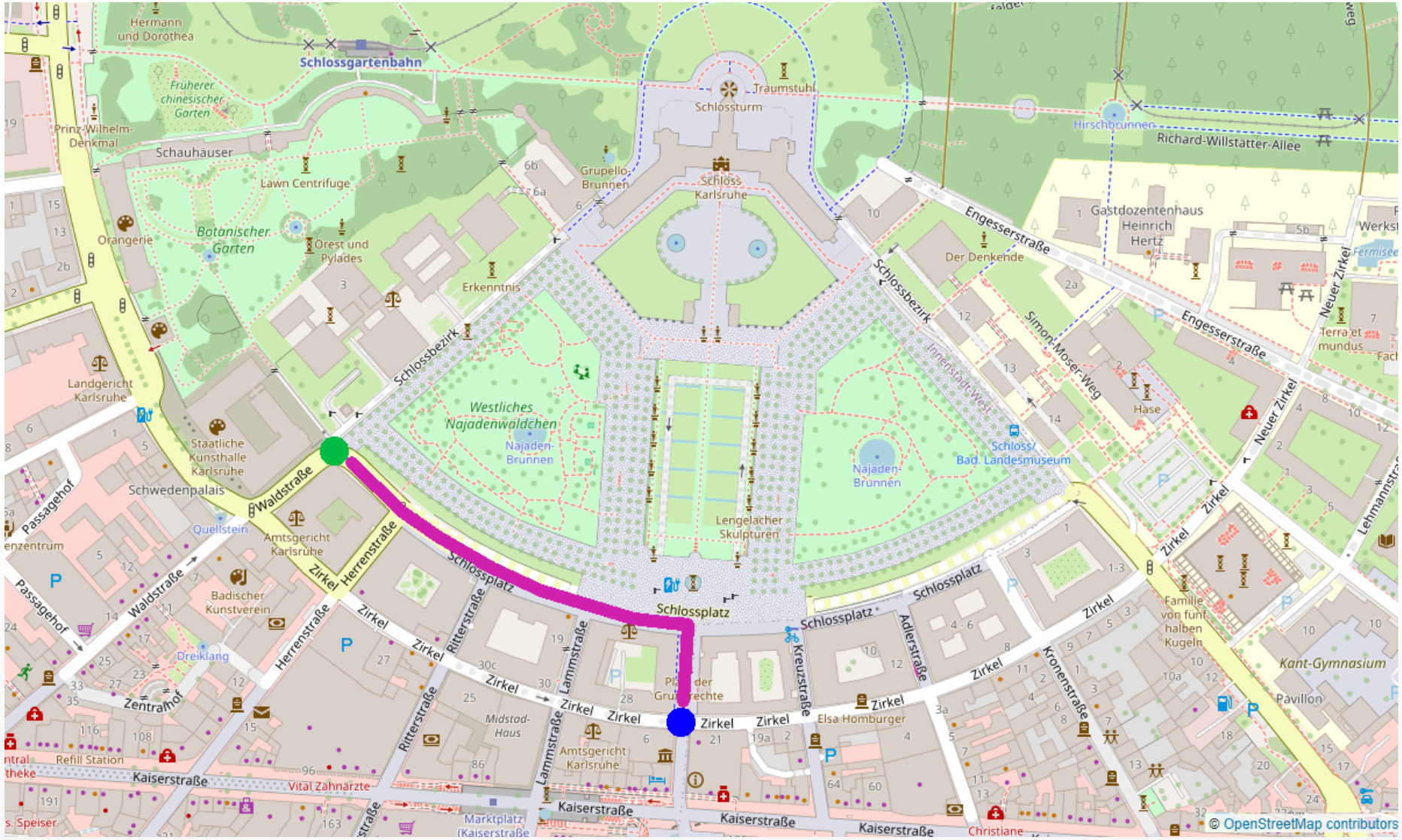
[1]

[1] https://link.springer.com/chapter/10.1007/3-540-50728-0_61

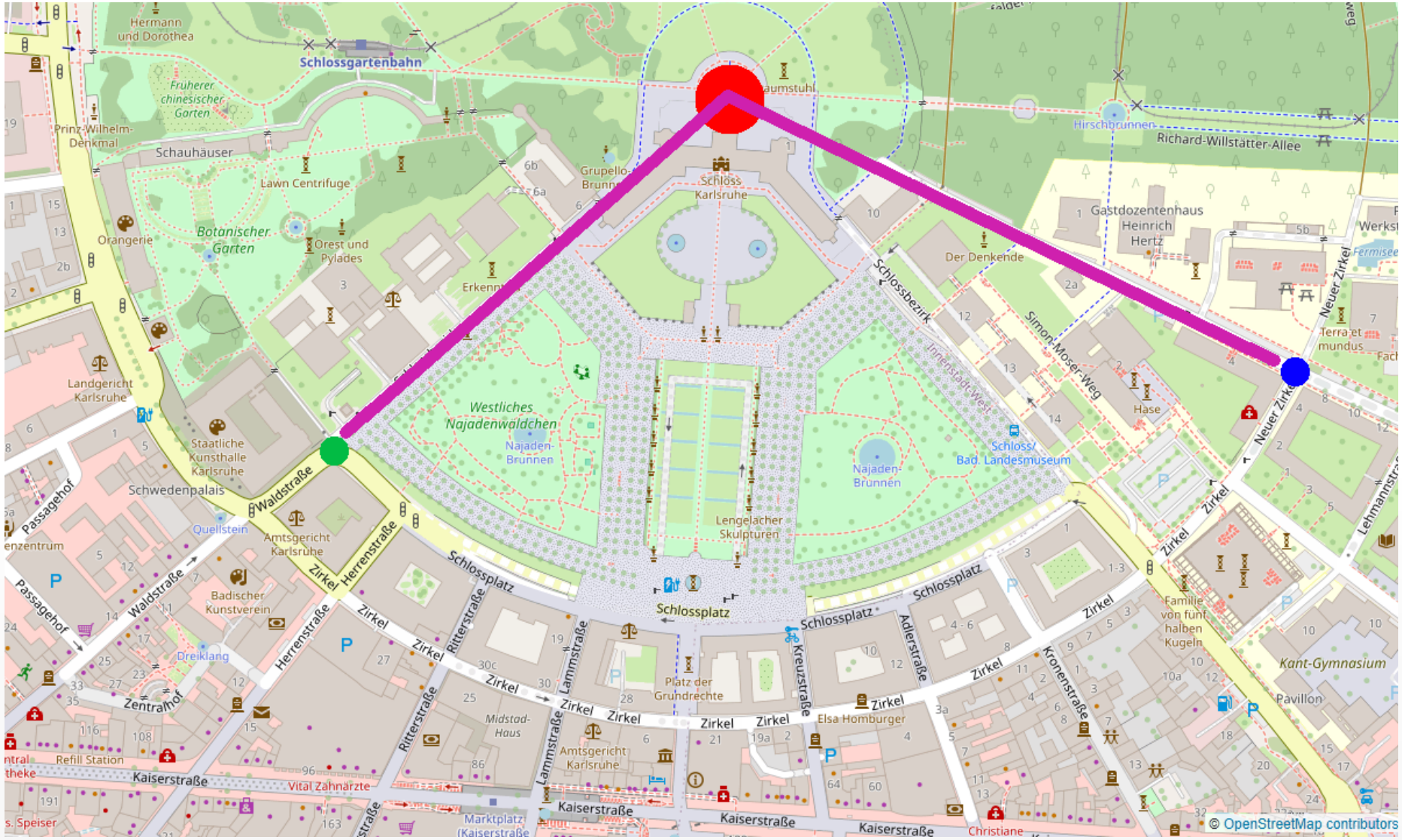
Karlsruhe Metrik



Karlsruhe Metrik



Karlsruhe Metrik

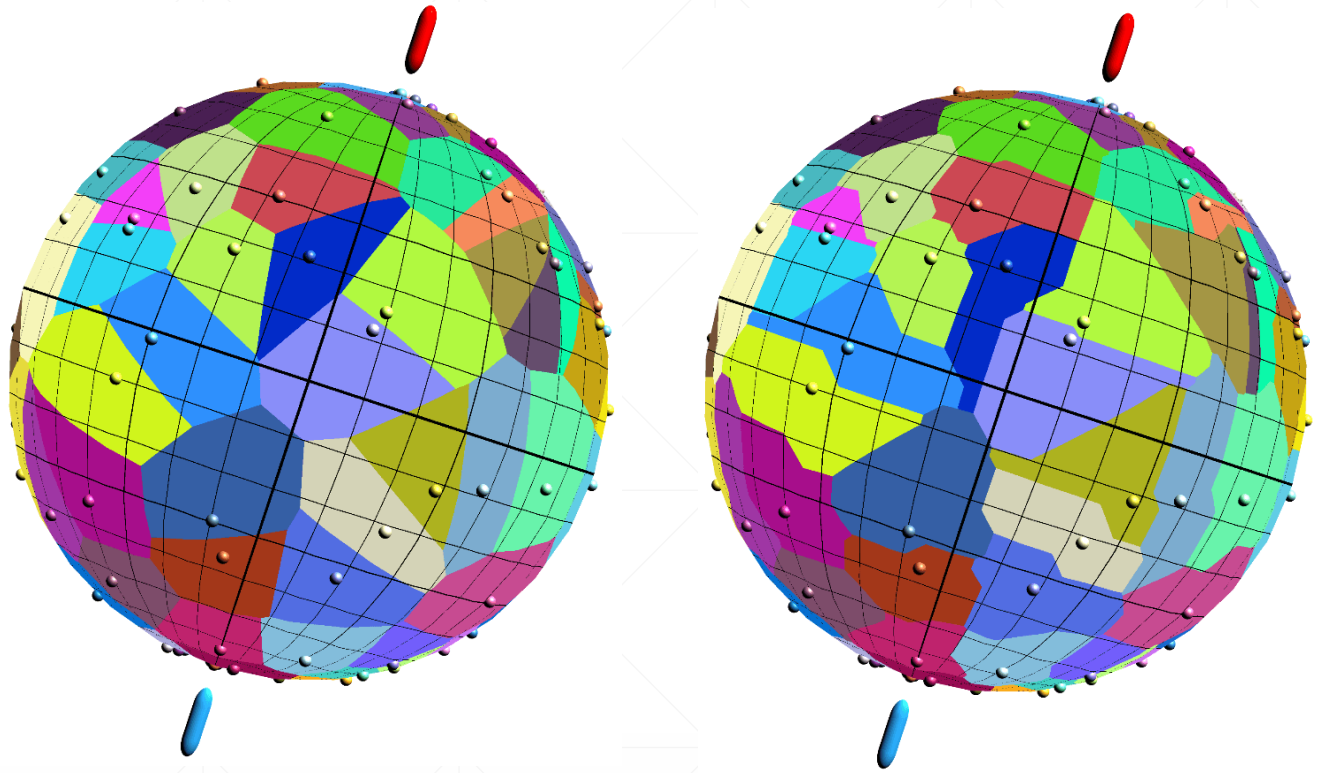


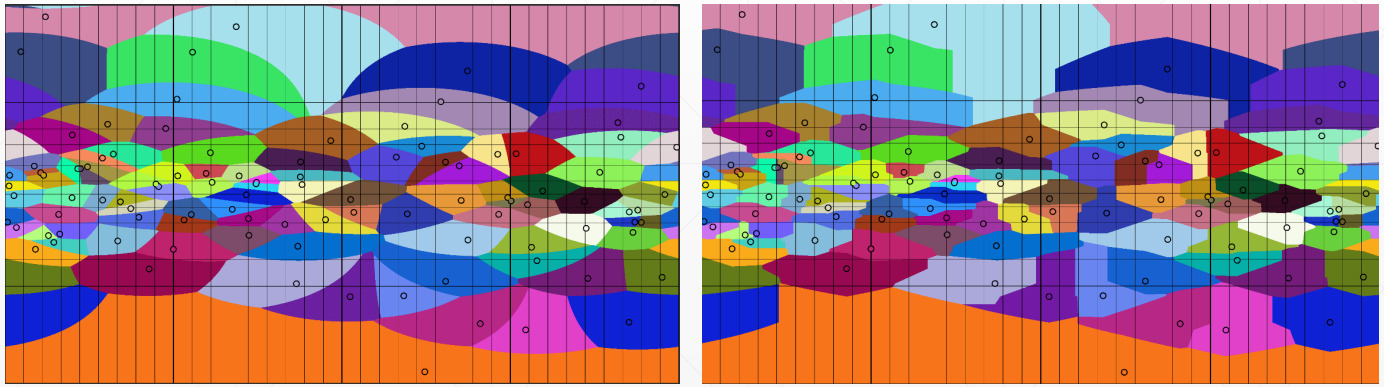
Related Work

- Voronoi Diagramme für die euklidische Metrik sind ausführlich untersucht
- Im 3D Raum liegt der Fokus auf Berechnung und Anwendung
- Keine etablierte Standardvisualisierung in 3D
- Visualisierung in 3D von nicht-euklidischen Metriken ist kaum untersucht

Related Work

- Fortgeschrittenen Praktikum von Sebastian Zins im SS 2025
- Visualisierung von Voronoi Diagrammen auf der Kugeloberfläche
- https://pille.iwr.uni-heidelberg.de/~voronoi01/

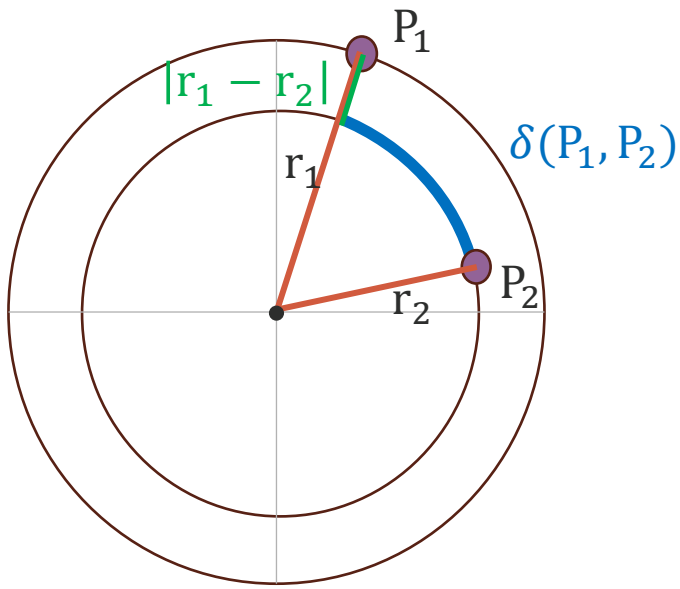




Karlsruhe Metrik

$$d_k(P_1, P_2) = \min \begin{pmatrix} \min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2| \\ r_1 + r_2 \end{pmatrix}$$

$\delta(P_1, P_2)$	Winkelabstand (Radiant)
$\min(r_1, r_2) \cdot \delta(P_1, P_2)$	Kreisbogenlänge



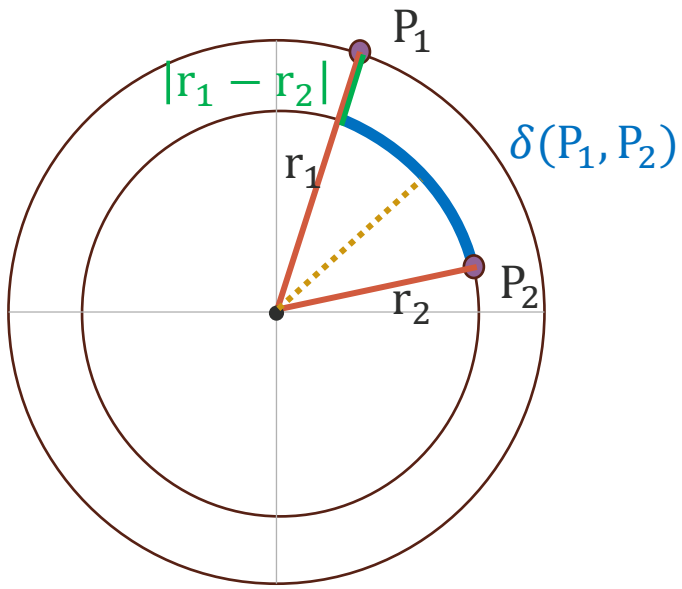
Karlsruhe Metrik

$\delta(P_1, P_2)$ Winkelabstand (Radiant)

$\min(r_1, r_2) \cdot \delta(P_1, P_2)$ Kreisbogenlänge

$$d_k(P_1, P_2) = \min\left(\begin{matrix}\min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2| \\ r_1 + r_2\end{matrix}\right)$$

$$d_k(P_1, P_2) = \begin{cases}\min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2|, & \delta(P_1, P_2) \leq 2 \\ r_1 + r_2\end{cases}$$

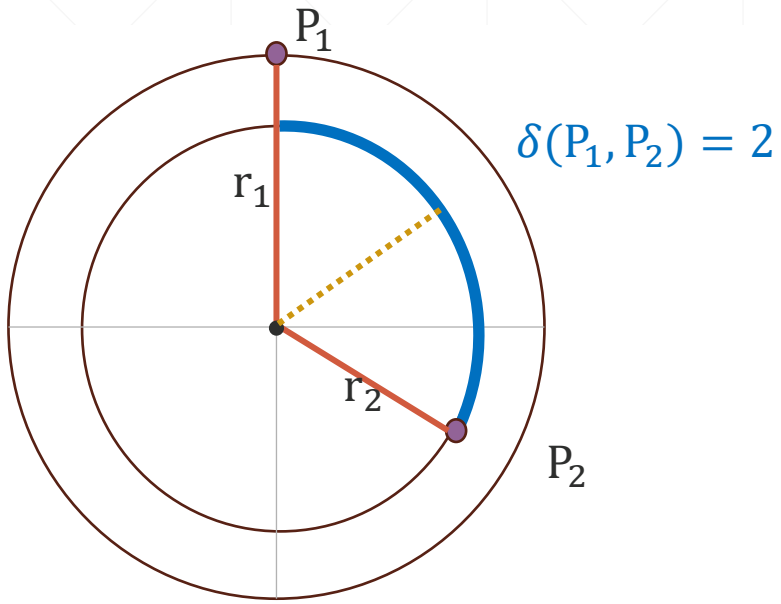


Karlsruhe Metrik

$$d_k(P_1, P_2) = \min\left(\frac{\min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2|}{r_1 + r_2}\right)$$

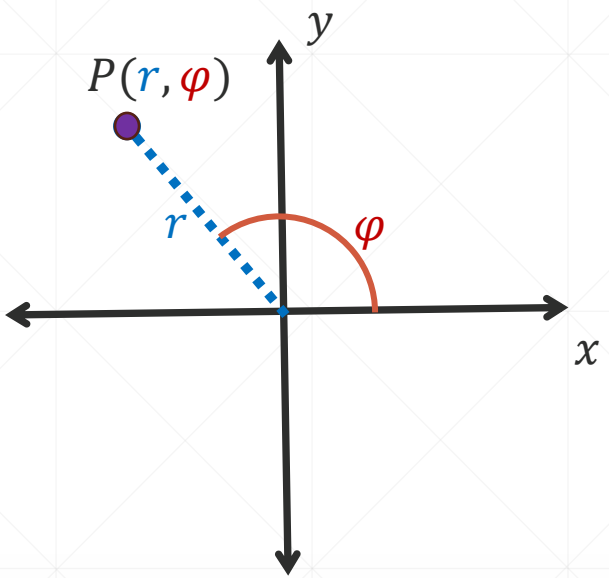
$$d_k(P_1, P_2) = \begin{cases} \dfrac{\min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2|}{r_1 + r_2}, & \delta(P_1, P_2) \leq 2 \end{cases}$$

$r_2 \cdot 2 + r_1 - r_2 = r_1 + r_2$



Karlsruhe Metrik in n-Dimensionen

$$d_k(P_1, P_2) = \begin{cases} \min(r_1, r_2) \cdot \delta(P_1, P_2) + |r_1 - r_2|, & \delta(P_1, P_2) \leq 2 \\ r_1 + r_2 \end{cases}$$



In 2D mit Polarkoordinaten: $P_1(r_1, \varphi_1), P_2(r_2, \varphi_2)$

$$\delta(P_1, P_2) = |\varphi_1 - \varphi_2|$$

In 3D Winkelabstand $\delta(P_1, P_2)$ über das Skalarprodukt:

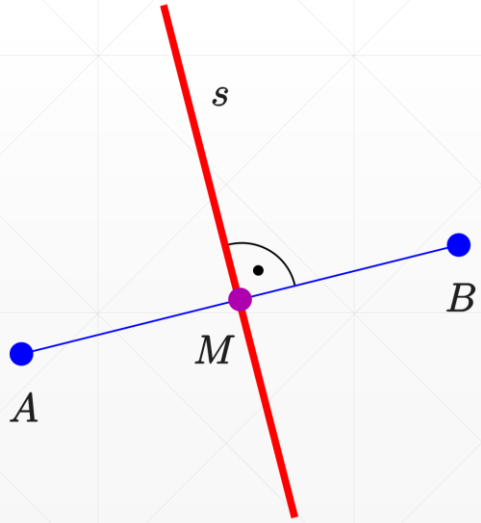
$$\vec{a} \cdot \vec{b} = \| \vec{a} \| \, \| \vec{b} \| \cos\theta$$

$$\theta = \cos^{-1} \frac{\vec{a} \cdot \vec{b}}{\| \vec{a} \| \, \| \vec{b} \|}$$

Bisektoren

$$X \mid dist(P_1, X) = dist(P_2, X)$$

- Voronoi Diagramm für zwei Punkte
- In der euklidischen Metrik: Eine Gerade bzw. Hyperebene unterteilt den Raum



[1]

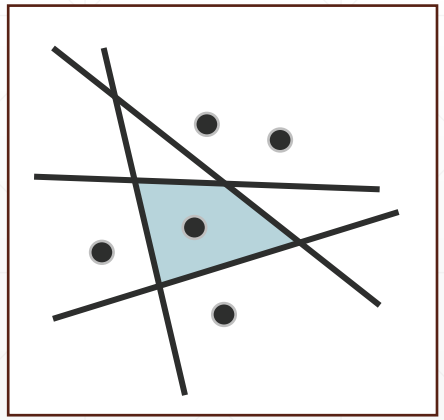
[1] https://commons.wikimedia.org/wiki/File:Mittelsenkr-ab.svg

Voronoi Diagramme

- Voronoi Zelle V_k für eine Menge von Punkte P_k

$$\mathrm{V}_k = \{X \mid dist(\mathrm{P}_k, X) \leq dist(\mathrm{P}_j, X) \text{ für alle } j \neq k\}$$

- Alle Punkte X deren nächster Nachbar P_k ist

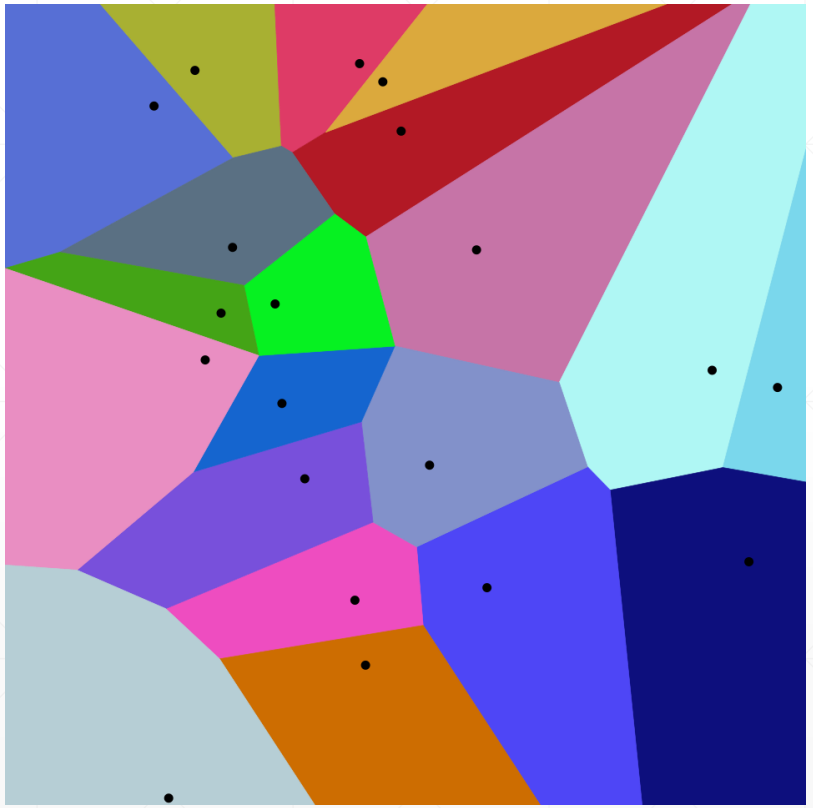


Berechnung von Voronoi Diagrammen

- Alle Mittelsenkrechten berechnen $O(n^2 \log n)$
- Für die euklidische Metrik gibt es effizientere Algorithmen, oft $O(n^2)$
- Sweepline „Wave Front“ „Beach Line“ $O(n \log n)$
- Theoretisch auch für die Karlsruhe Metrik möglich
- Aber: keine Bibliotheken/Implementierungen verfügbar

Berechnung von Voronoi Diagrammen

- Für Visualisierung: Brute Force Approach
  - Jedem Punkt wird eine Farbe (label) zugeordnet
  - Für jedes Pixel wird der Abstand zu allen Punkten berechnet
  - Pixel erhält Farbe des nächsten Nachbarn
- n Abstandsberechnungen pro Pixel



[1]

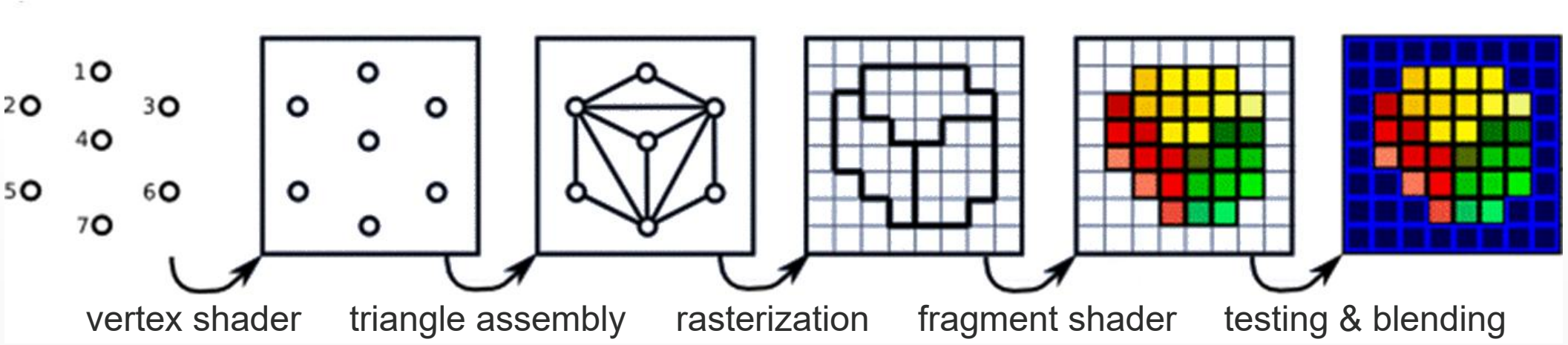
[1] https://commons.wikimedia.org/wiki/File:Euclidean_Voronoi_diagram.svg

Implementierung

- Weiterentwicklung von Sebastians Praktikum
- Unity: 3D Game Engine
- Zufällig verteilte Punkte in einer Kugel
- Jeder Punkt besitzt eine Farbe
- Pro Pixel: Abstand zu allen Punkten berechnen
- Fragment Shader/Pixel Shader berechnet sowieso den Farbwert für jedes Pixel

Fragment/Pixel Shader

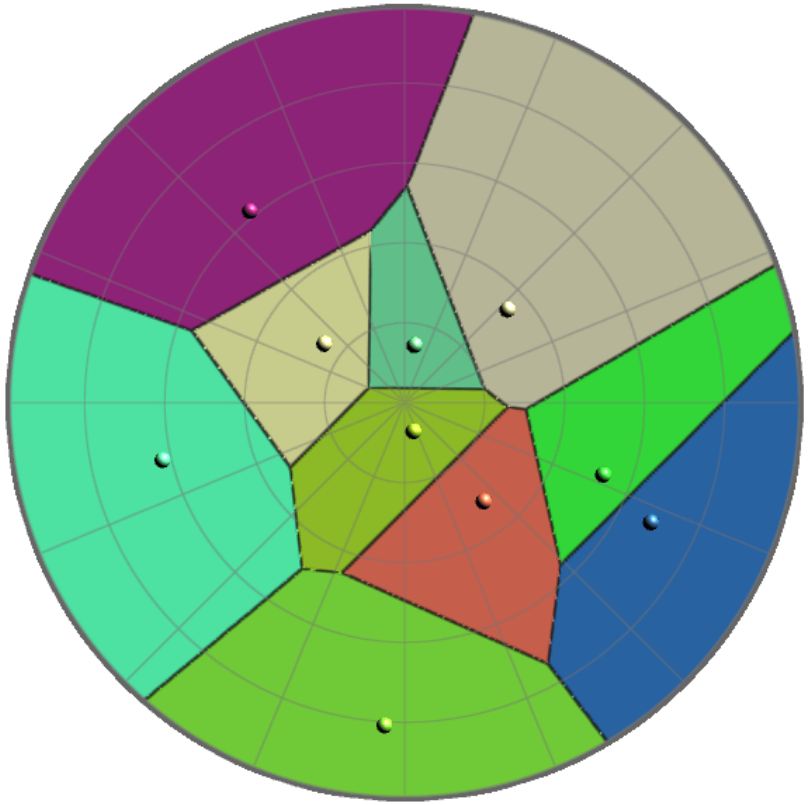
Grafikpipeline z.B. mit OpenGL, Vulkan oder Direct3D



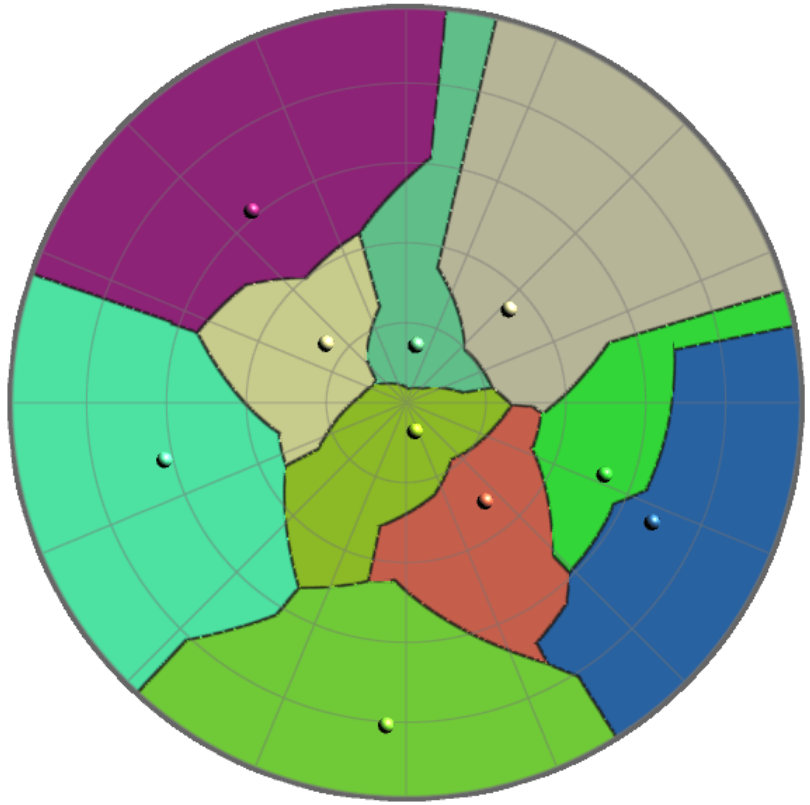
[1]

[1] https://webdocs.cs.ualberta.ca/~pierreb/CMPUT411/

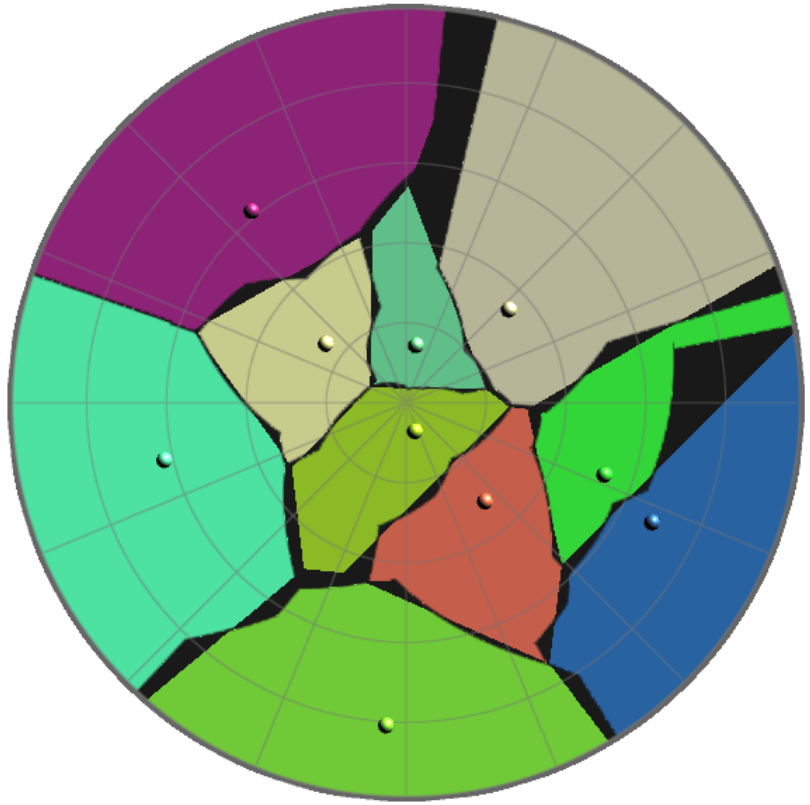
Karlsruhe Metrik in 2D



Euklidische Metrik

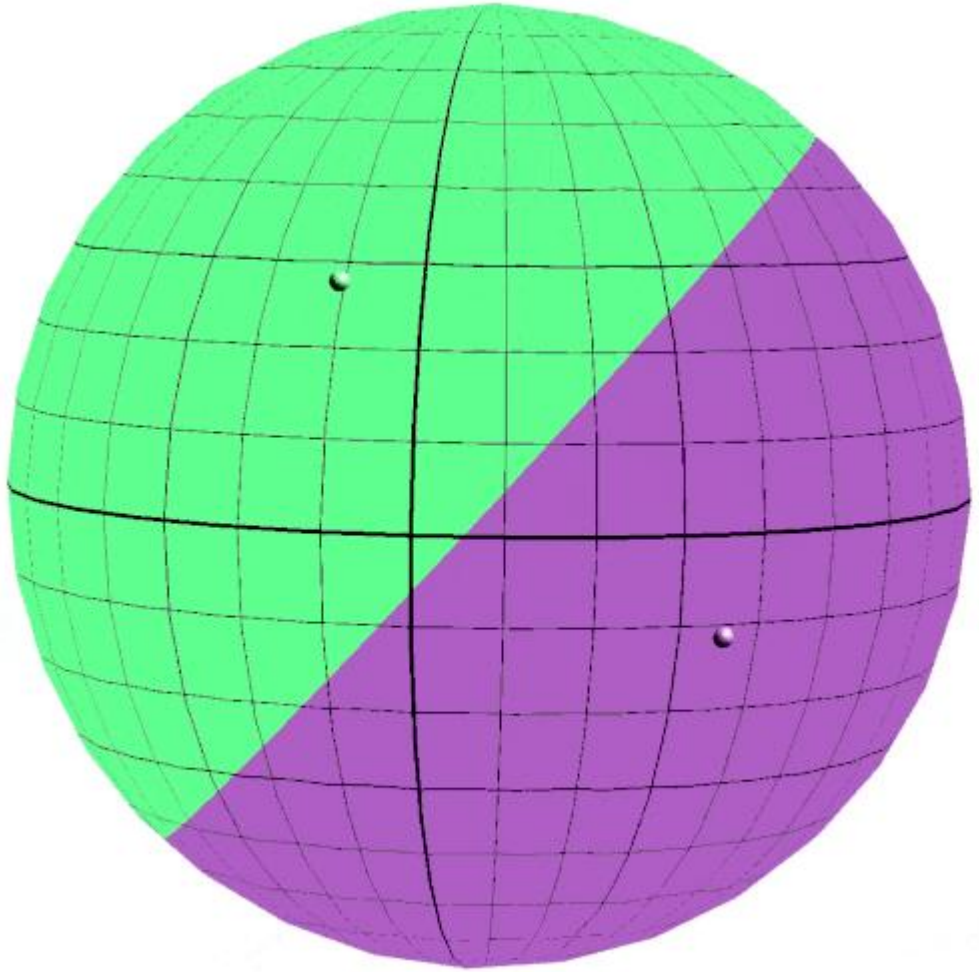


Karlsruhe Metrik



Schnittmenge/Differenz

Karlsruhe Metrik auf der Kugeloberfläche



Karlsruhe Metrik auf der Kugeloberfläche

$$d_{\mathrm{k}}(\mathrm{P}_1, \mathrm{P}_2) = \begin{cases} \min(\mathrm{r}_1, \mathrm{r}_2) \cdot \delta(\mathrm{P}_1, \mathrm{P}_2) + |\mathrm{r}_1 - \mathrm{r}_2|, & \delta(\mathrm{P}_1, \mathrm{P}_2) \leq 2 \\ \mathrm{r}_1 + \mathrm{r}_2 \end{cases}$$

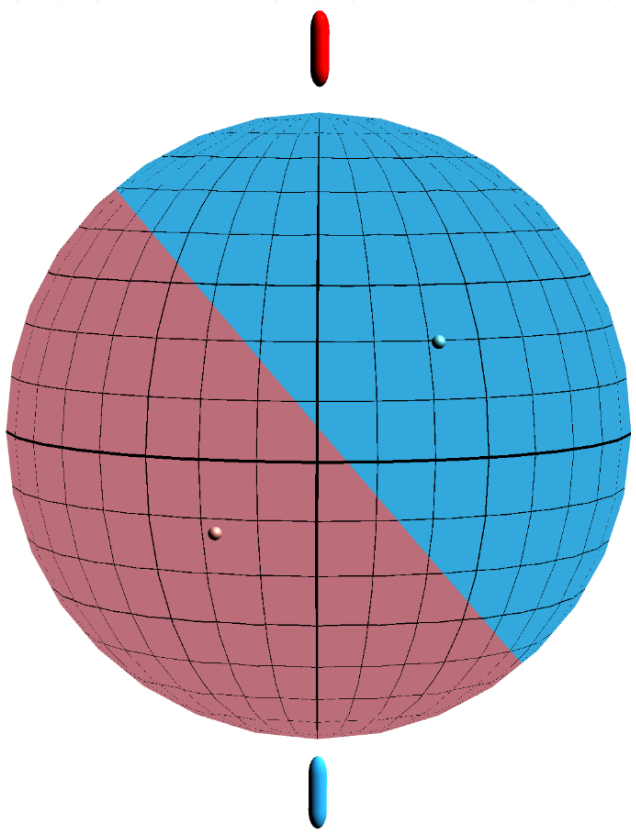
$\mathrm{r}_1 = \mathrm{r}_2 = r$ $\rightarrow$ Punkte auf der Kugeloberfläche

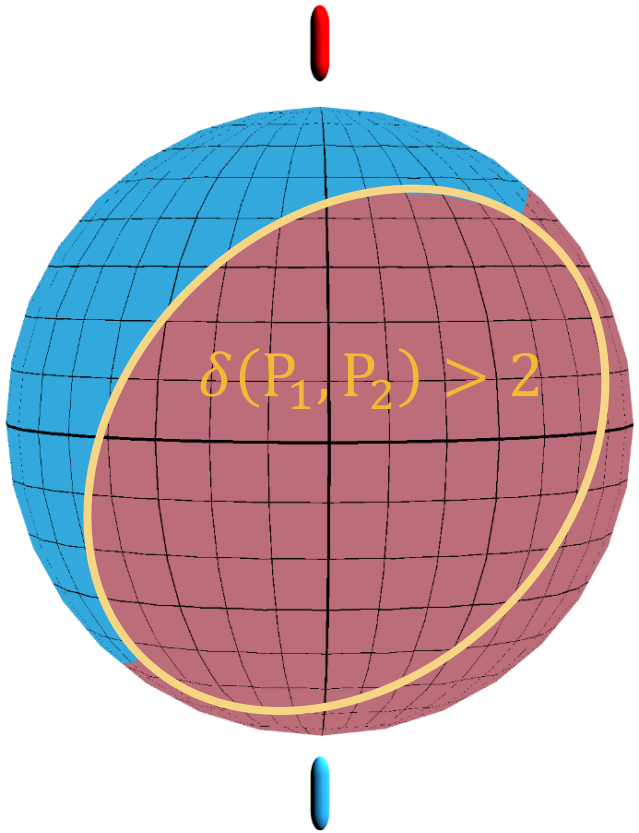
$$d_{\mathrm{k}}(\mathrm{P}_1, \mathrm{P}_2) = \begin{cases} r \cdot \delta(\mathrm{P}_1, \mathrm{P}_2), & \delta(\mathrm{P}_1, \mathrm{P}_2) \leq 2 \\ 2r \end{cases}$$

Geodäte auf Großkreis

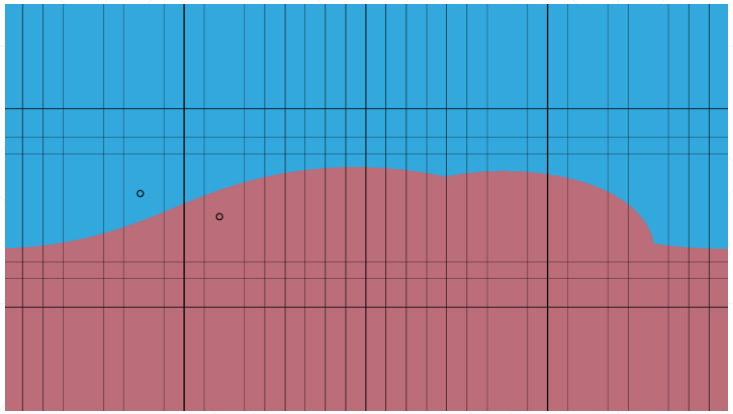
Tunnel durch Kugelmittelpunkt

Karlsruhe Metrik auf der Kugeloberfläche

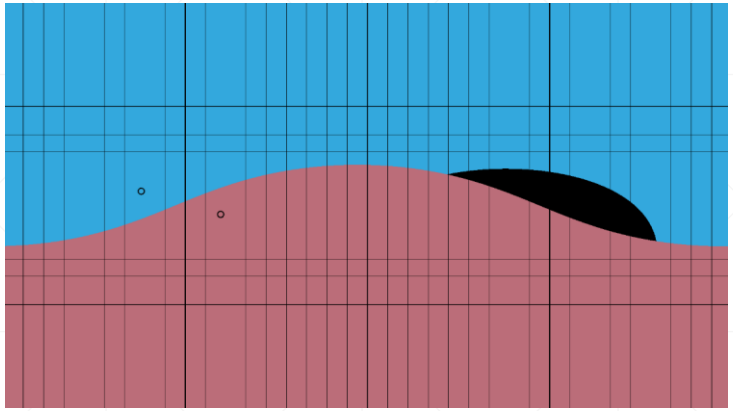




$$d_k(P_1, P_2) = \begin{cases} r \cdot \delta(P_1, P_2), & \delta(P_1, P_2) \leq 2 \\ 2r \end{cases}$$

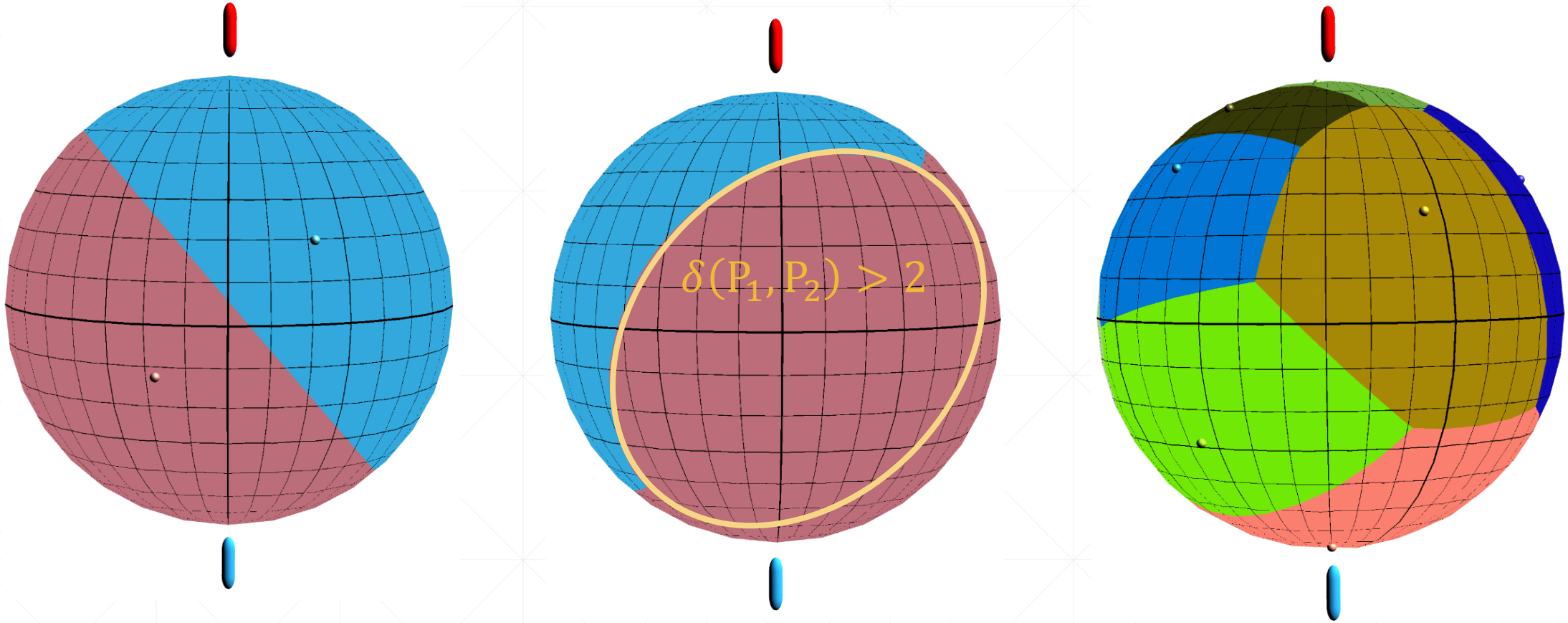


Mercator-Projektion



Unterschied zur geodätischen Distanz auf Großkreisen

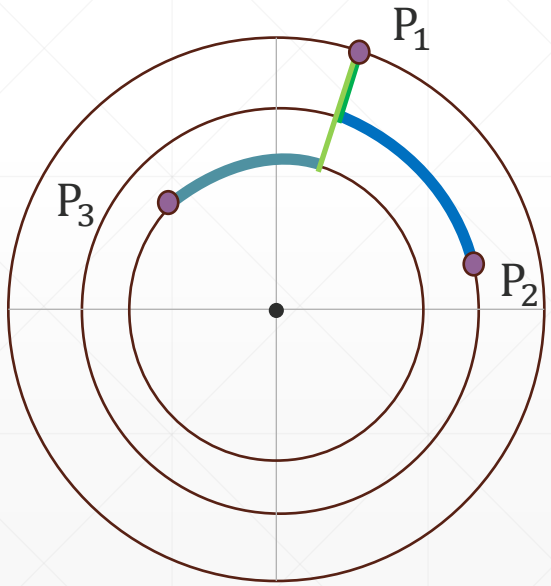
Karlsruhe Metrik auf der Kugeloberfläche



Bei gleichem Abstand „gewinnt“ der zuletzt getestete Punkt

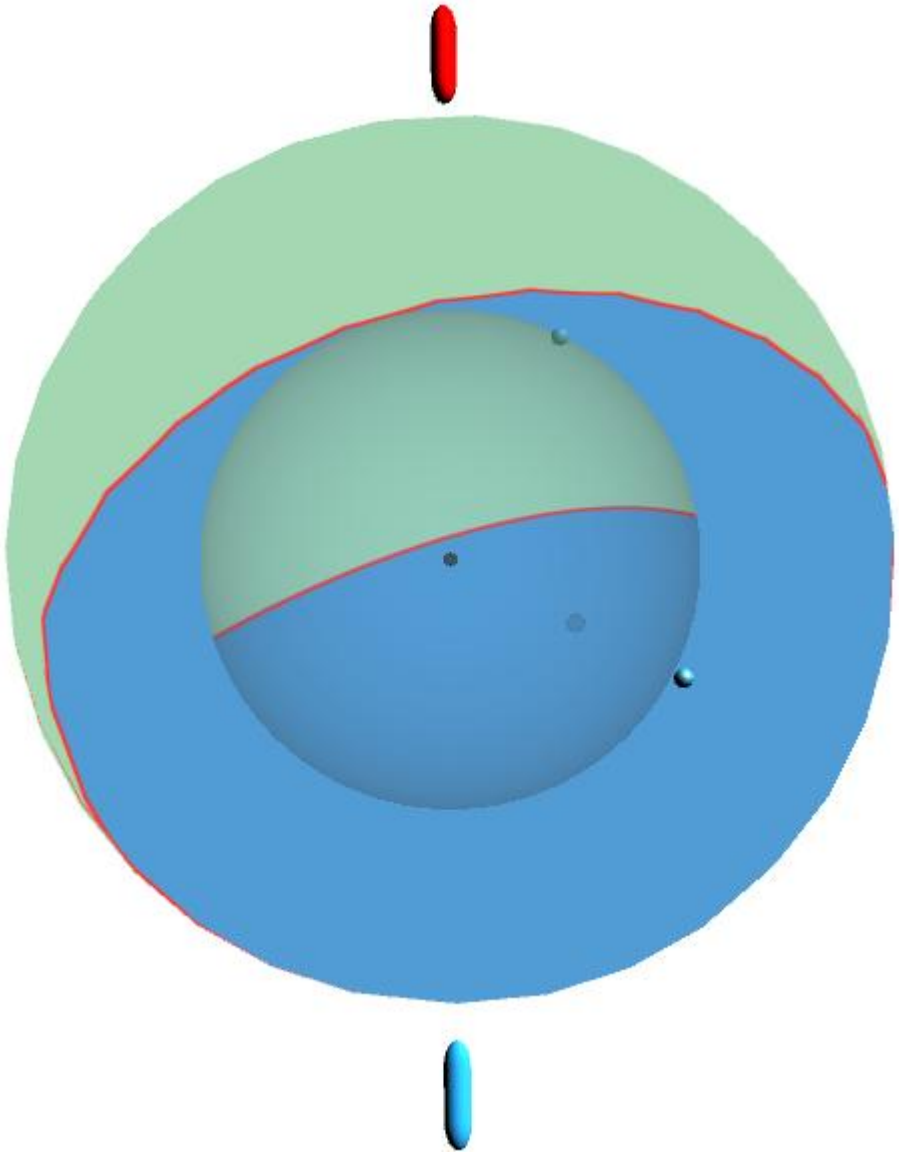
Voronoi in 3D

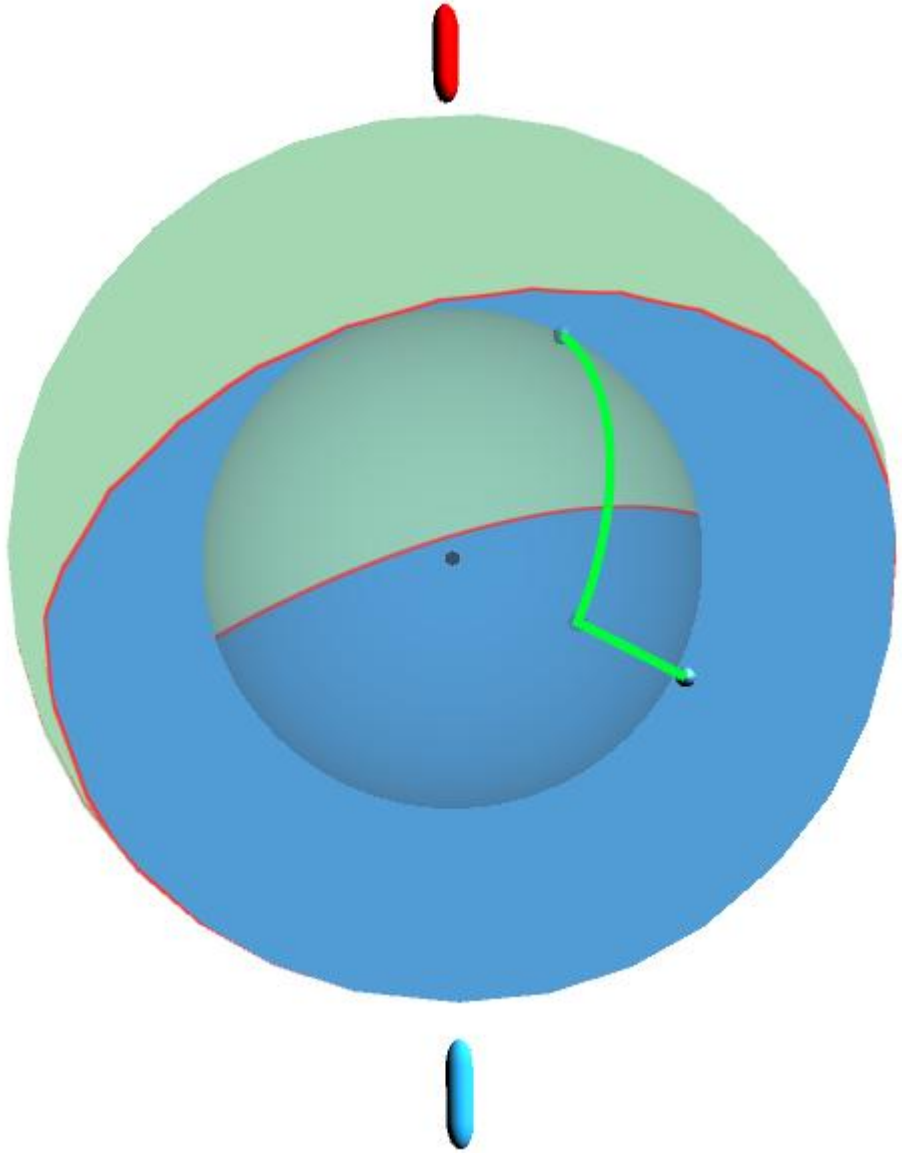
- Punkte zufällig **in der Kugel** verteilt
- Kugel ist der Raum, notwendig für Fragment Shader



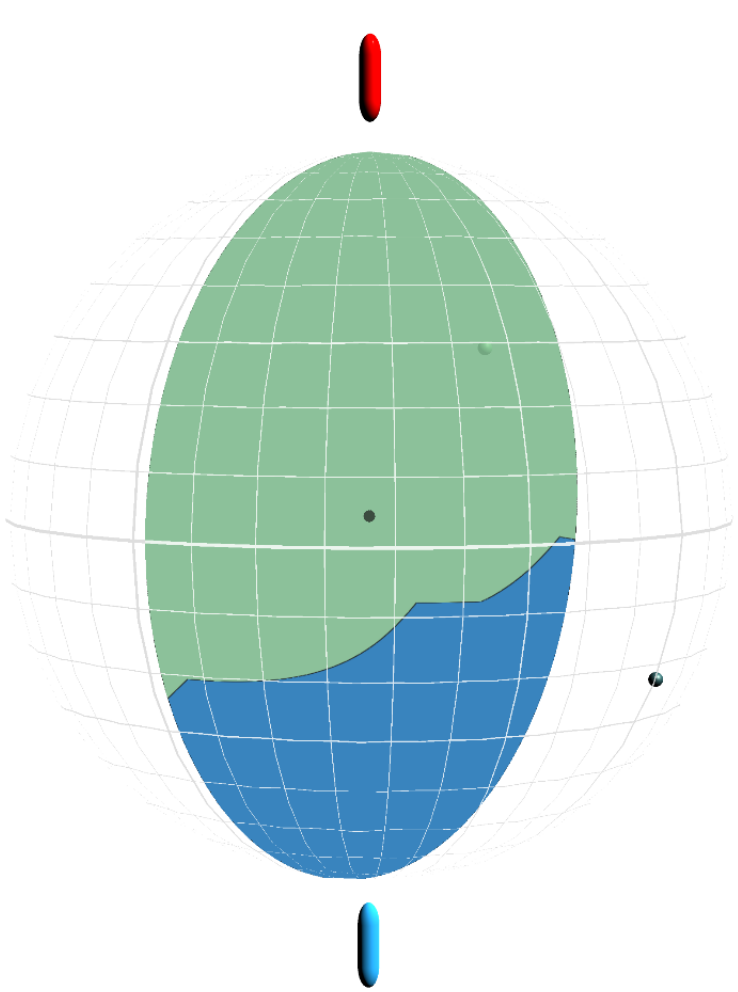
„Zwiebelschalen“

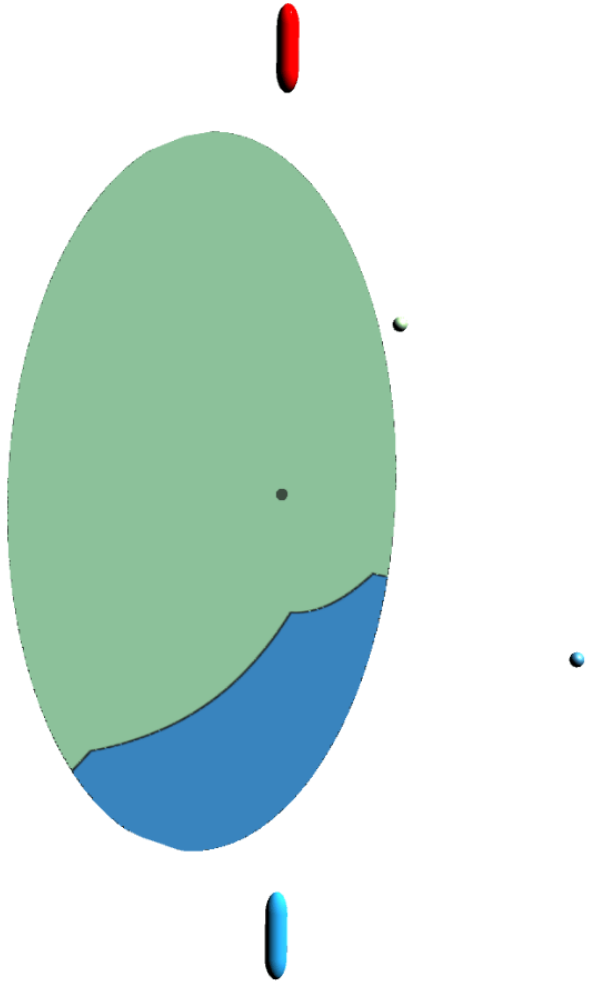
„Zwiebelschalen“



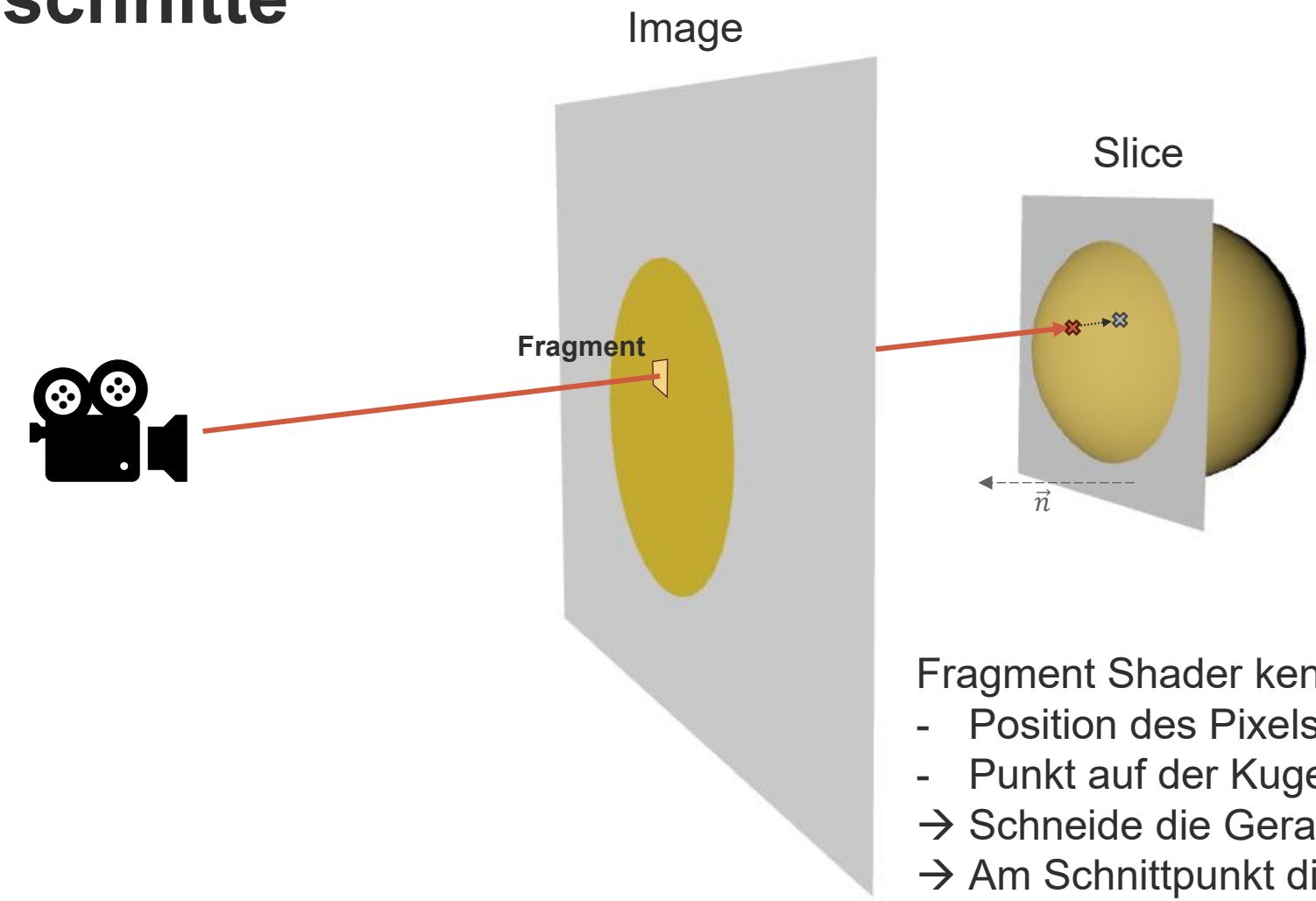


Querschnitte





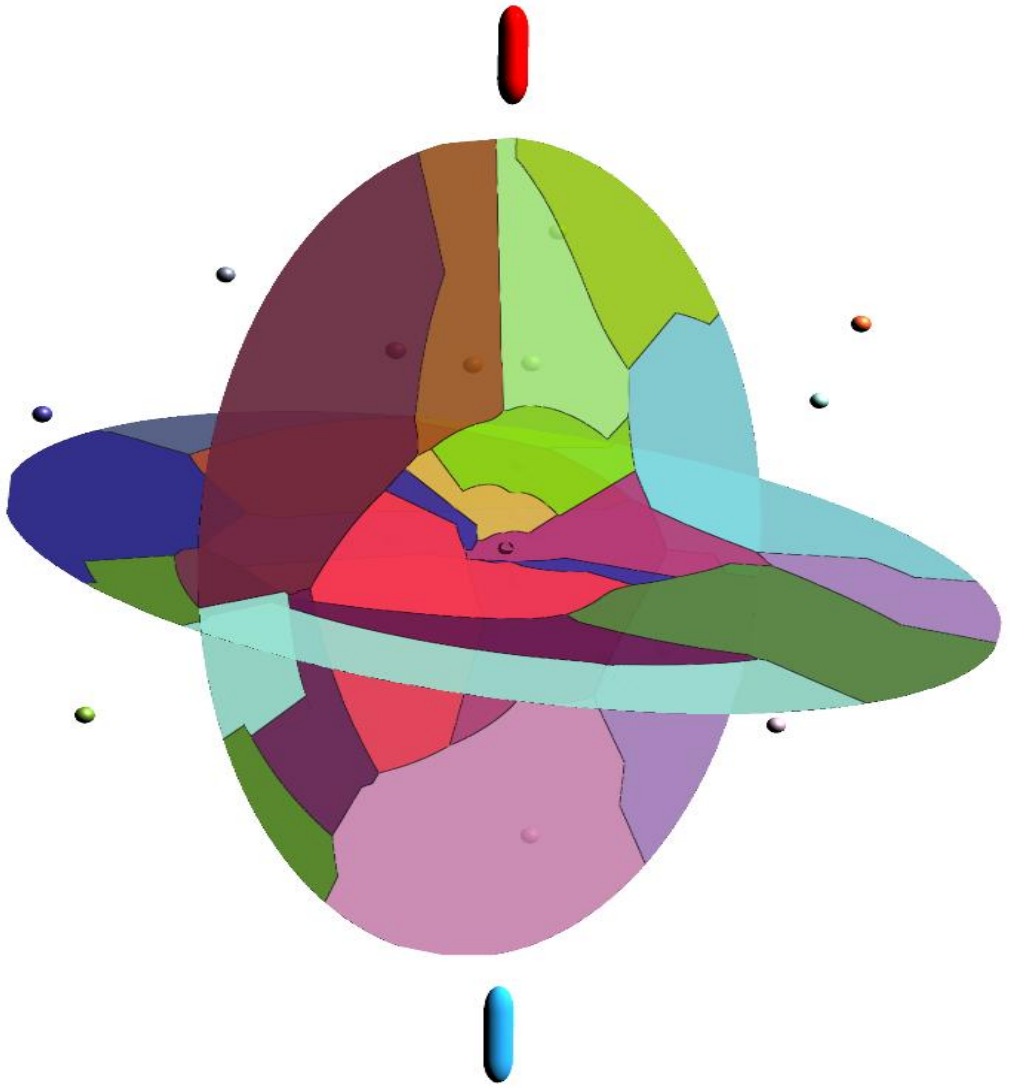
Querschnitte



Fragment Shader kennt:
- Position des Pixels
- Punkt auf der Kugeloberfläche

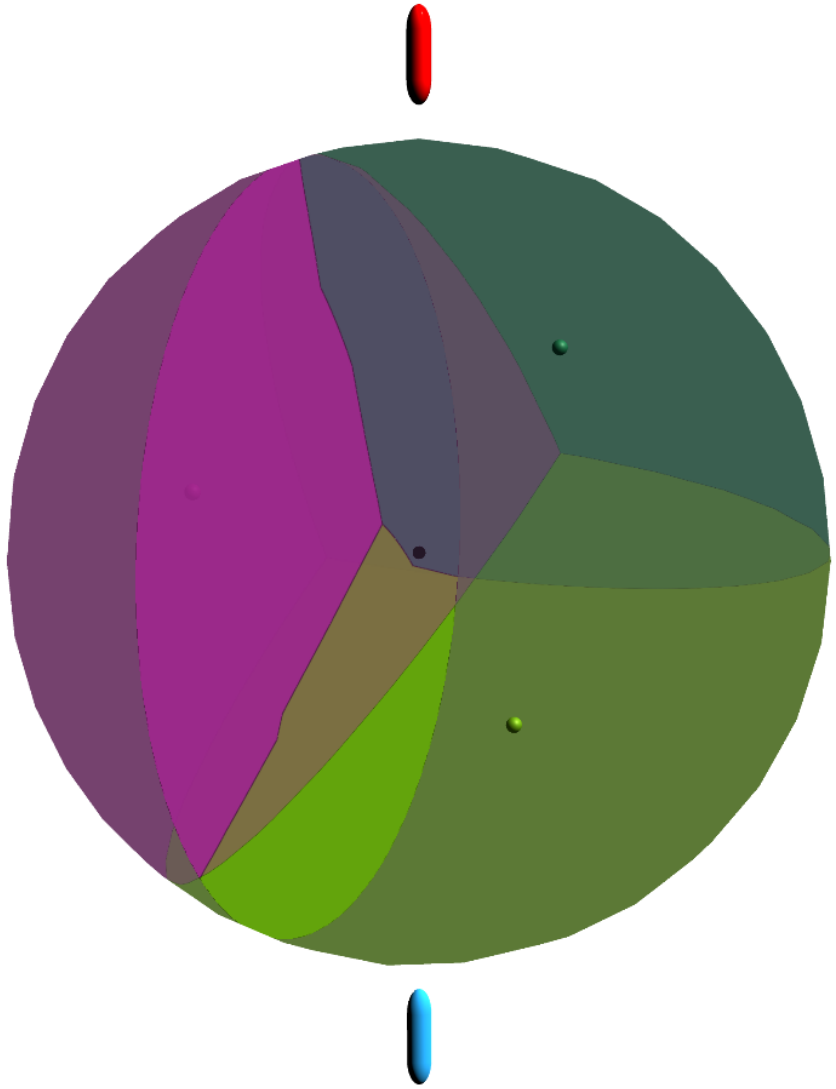
→ Schneide die Gerade mit der Slice-Ebene
→ Am Schnittpunkt die Farbe des nächsten Punktes finden

Querschnitte

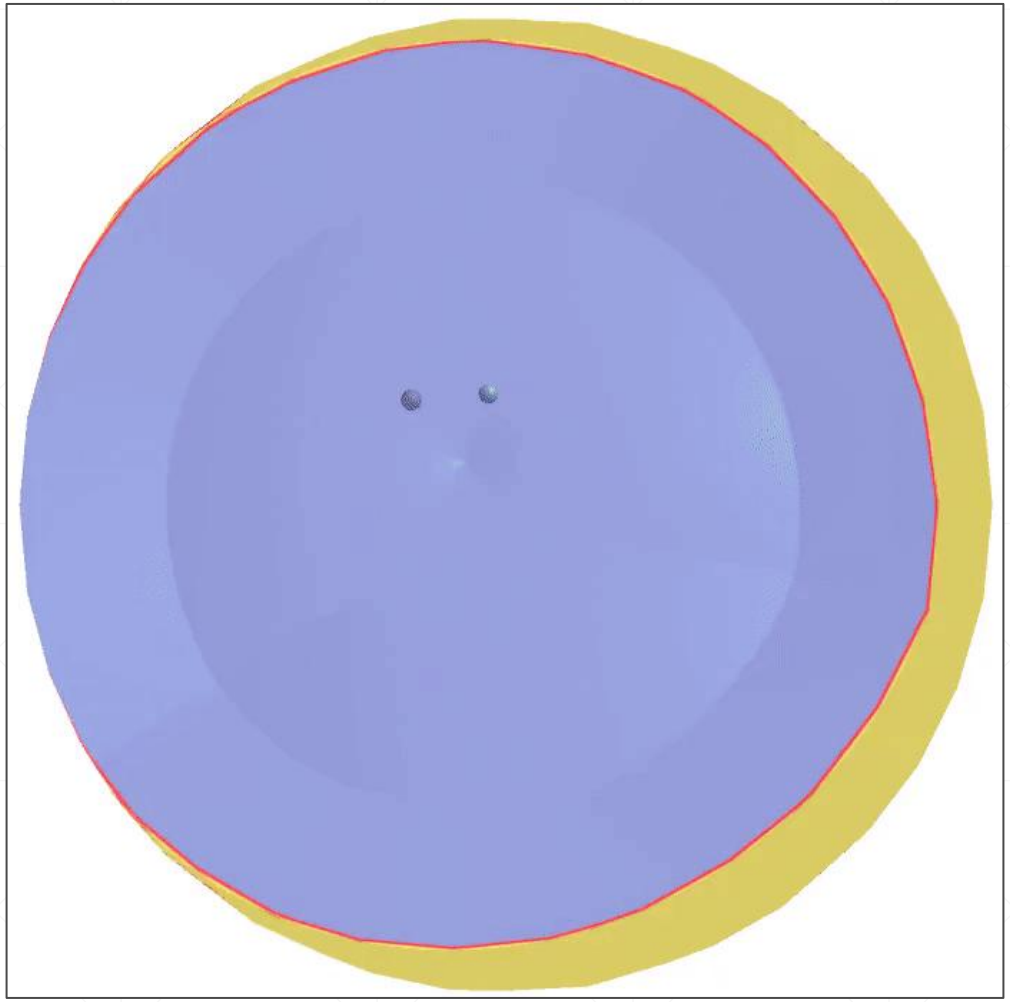


Querschnitte

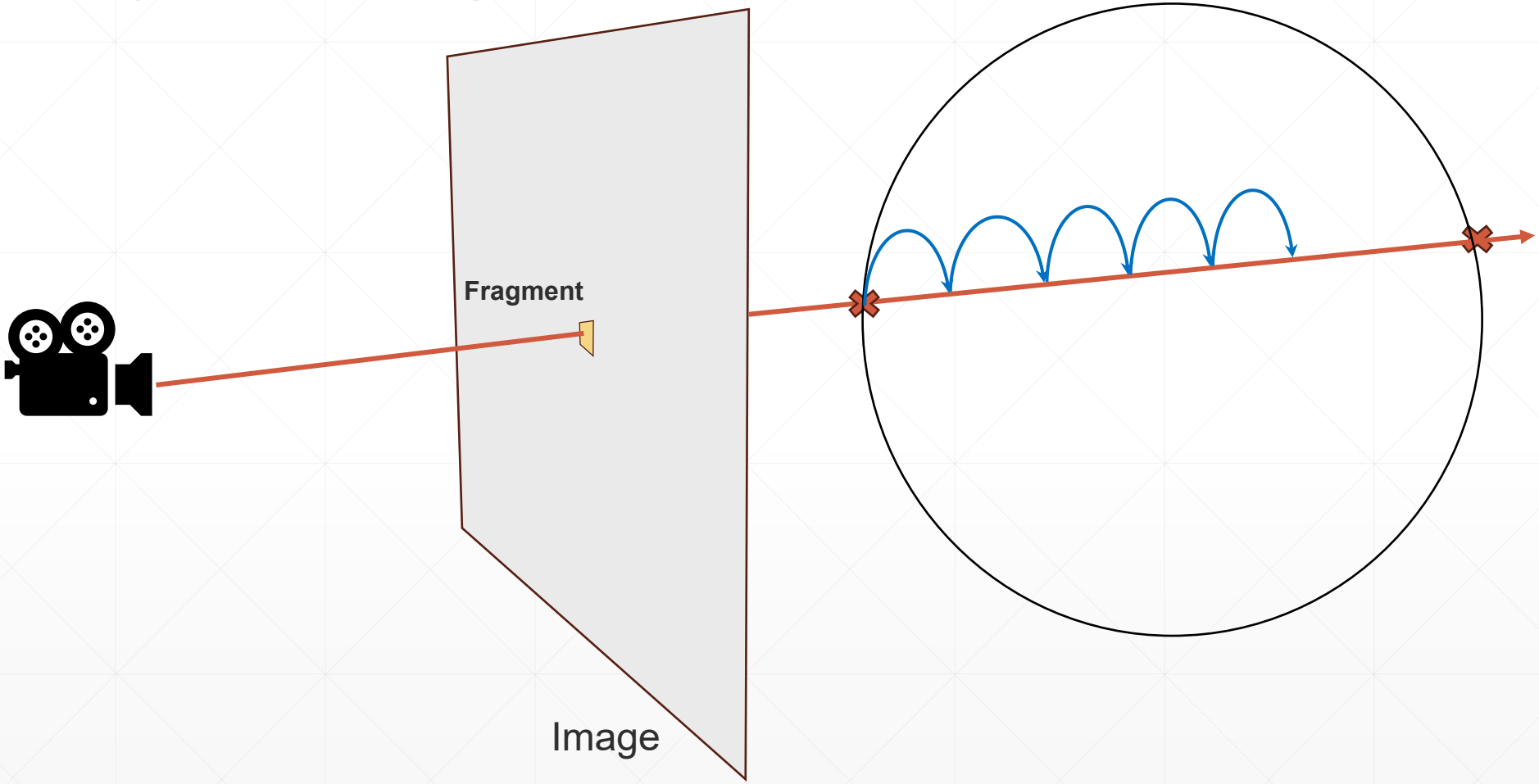
- Transparenz ist schwer zu interpretieren
- Überlagerung von mehr als zwei transparenten Objekten führt zu Mehrdeutigkeit
- Zwei transparente Objekte z. B. zwei Querschnitte, noch verständlich
- Drei transparente Ebenen → visuell überladen
  z.B. zwei Querschnitte und transparente Kugeloberfläche



Bisector Fläche mit Ray Marching



Ray Marching

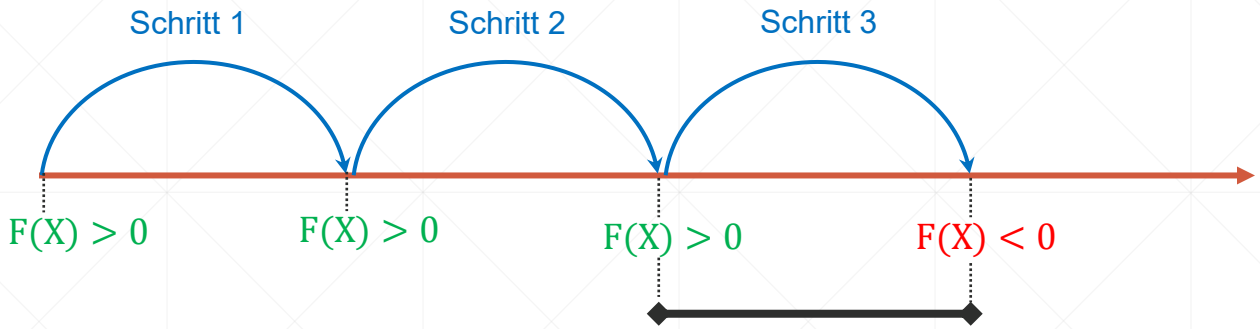


Bisector Fläche mit Ray Marching

Bisektor zwischen P_1 und P_2 :

$X \mid dist(P_1, X) = dist(P_2, X)$

$F(X) = dist(P_1, X) - dist(P_2, X) = 0$

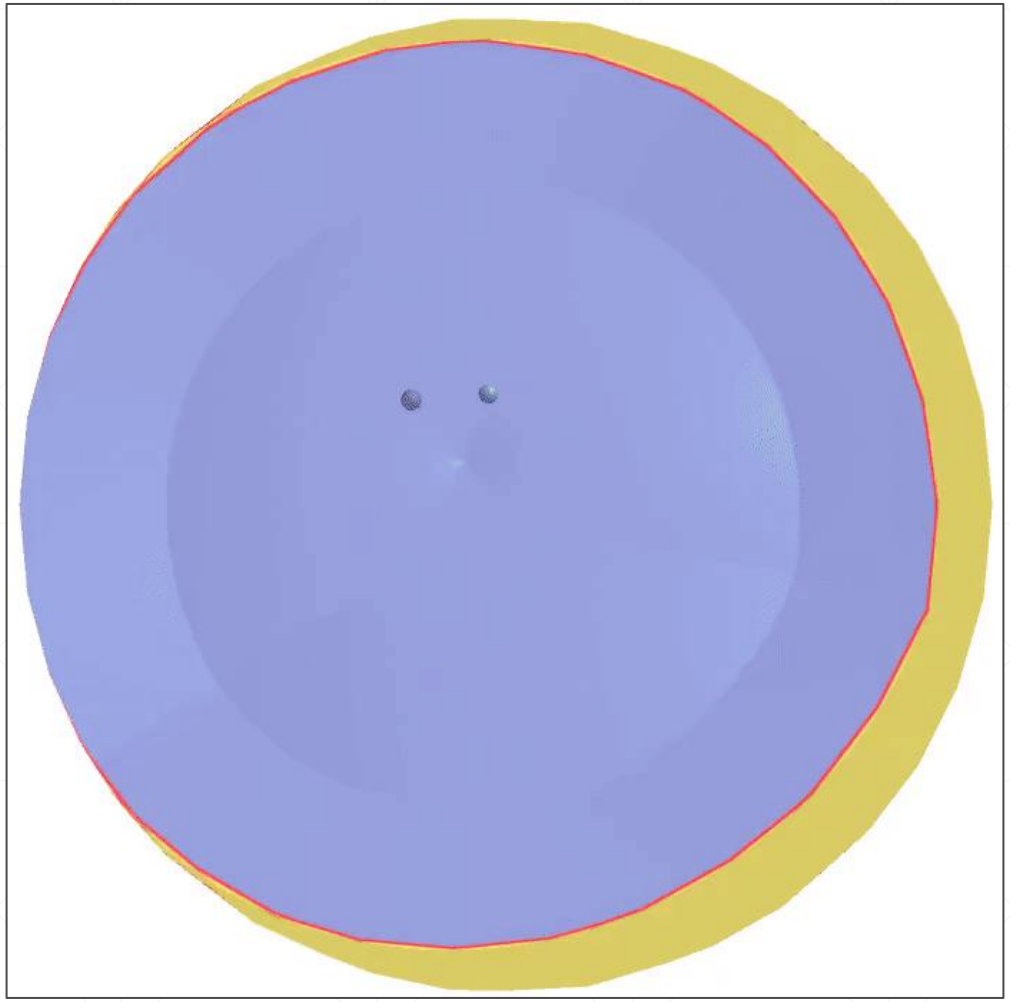


Laufe entlang dem Strahl, bis sich das Vorzeichen von $F(X)$ ändert.

Die Nullstelle muss sich im letzten Schritt befinden.

Mit numerischen Verfahren kann die Nullstelle weiter angenähert werden.
Zum Beispiel: Lineare Interpolation, Newtonverfahren

Bisector Fläche mit Ray Marching



Bisektoren bei gleichem Radius

Bisektor: Menge der Punkte X mit $d_{\mathrm{k}}(\mathrm{P}_1, X) = d_{\mathrm{k}}(\mathrm{P}_2, X)$

$$d_{\mathrm{k}}(\mathrm{P}_j, X) = \begin{cases} \min(\mathrm{r}_j, \mathrm{r}_x) \cdot \delta(\mathrm{P}_j, X) + |\mathrm{r}_j - \mathrm{r}_x|, & \delta(\mathrm{P}_j, \mathrm{P}_x) \leq 2 \\ \mathrm{r}_j + \mathrm{r}_x \end{cases}$$

$$\min(\mathrm{r}_1, \mathrm{r}_x) \cdot \delta(\mathrm{P}_1, X) + |\mathrm{r}_1 - \mathrm{r}_x| = \min(\mathrm{r}_2, \mathrm{r}_x) \cdot \delta(\mathrm{P}_2, X) + |\mathrm{r}_2 - \mathrm{r}_x|$$

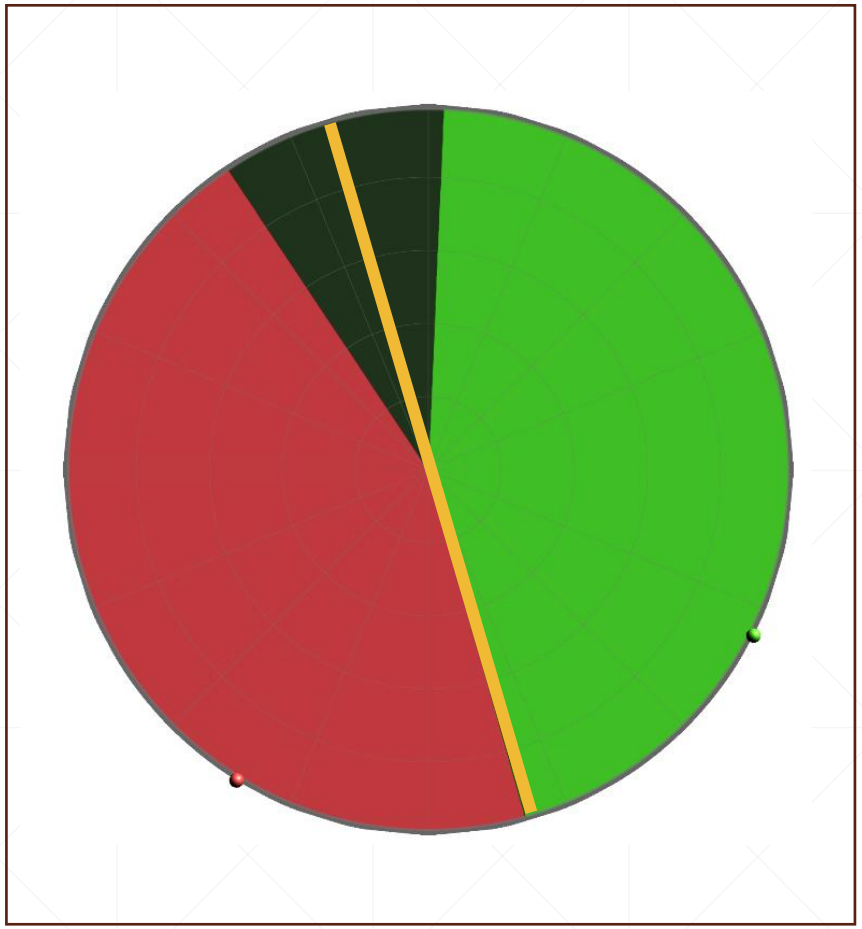
$$\mathrm{r}_1 = \mathrm{r}_2 = r$$

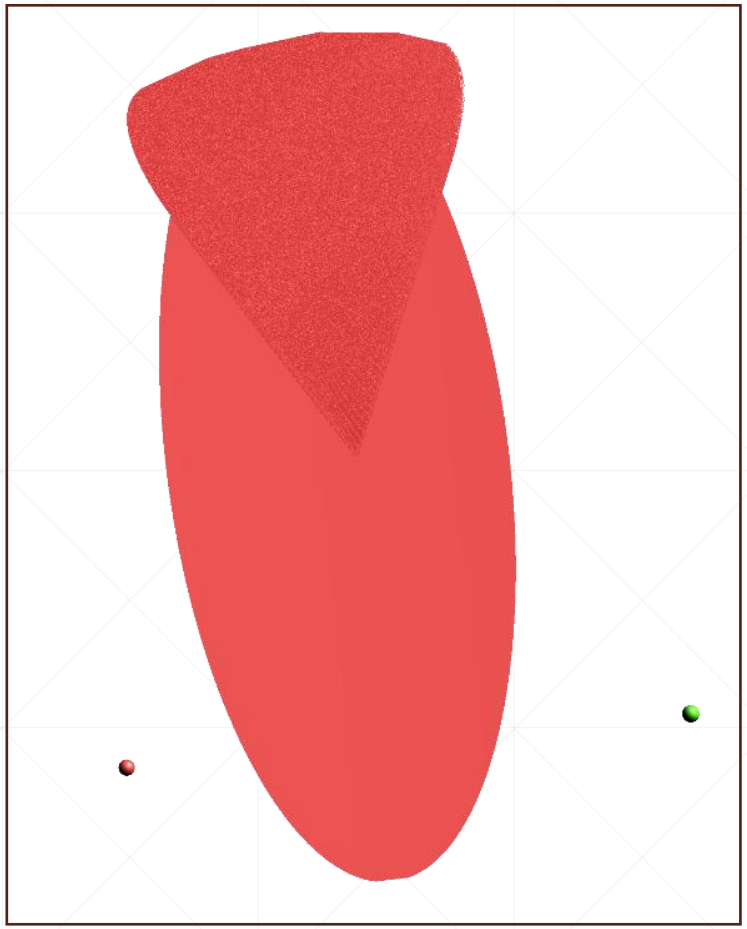
$$\min(r, \mathrm{r}_x) \cdot \delta(\mathrm{P}_1, X) + |r - \mathrm{r}_x| = \min(r, \mathrm{r}_x) \cdot \delta(\mathrm{P}_2, X) + |r - \mathrm{r}_x|$$

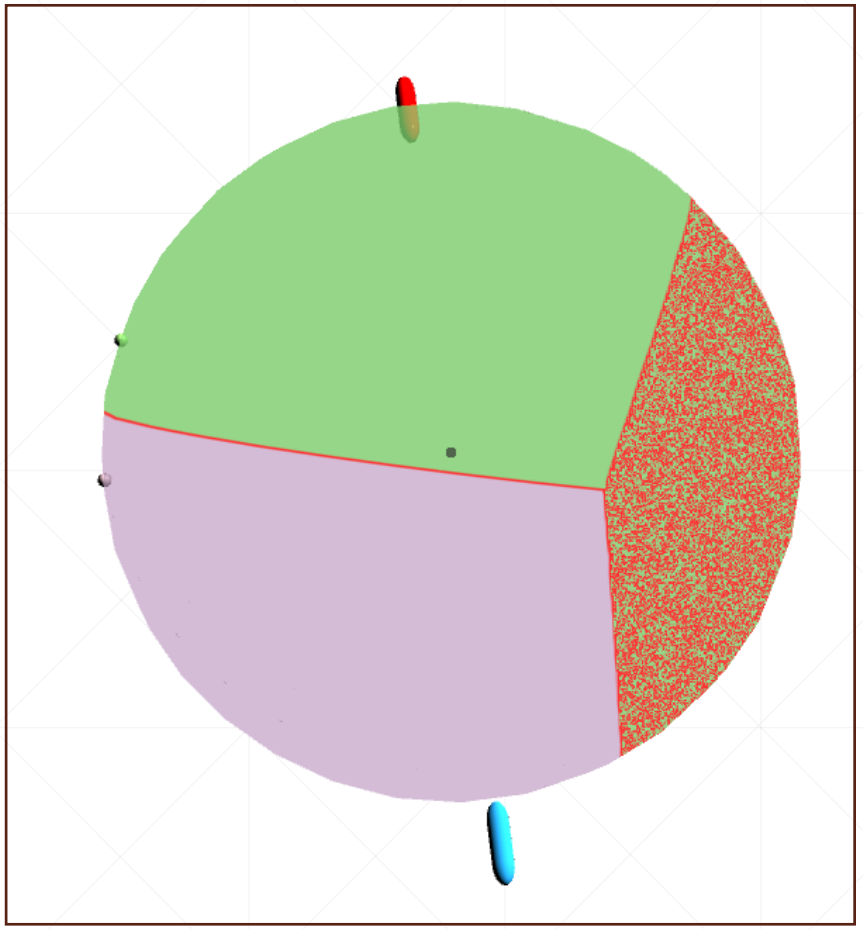
$$\delta(\mathrm{P}_1, X) = \delta(\mathrm{P}_2, X)$$

Bisektorbedingung ist unabhängig vom Radius r_x

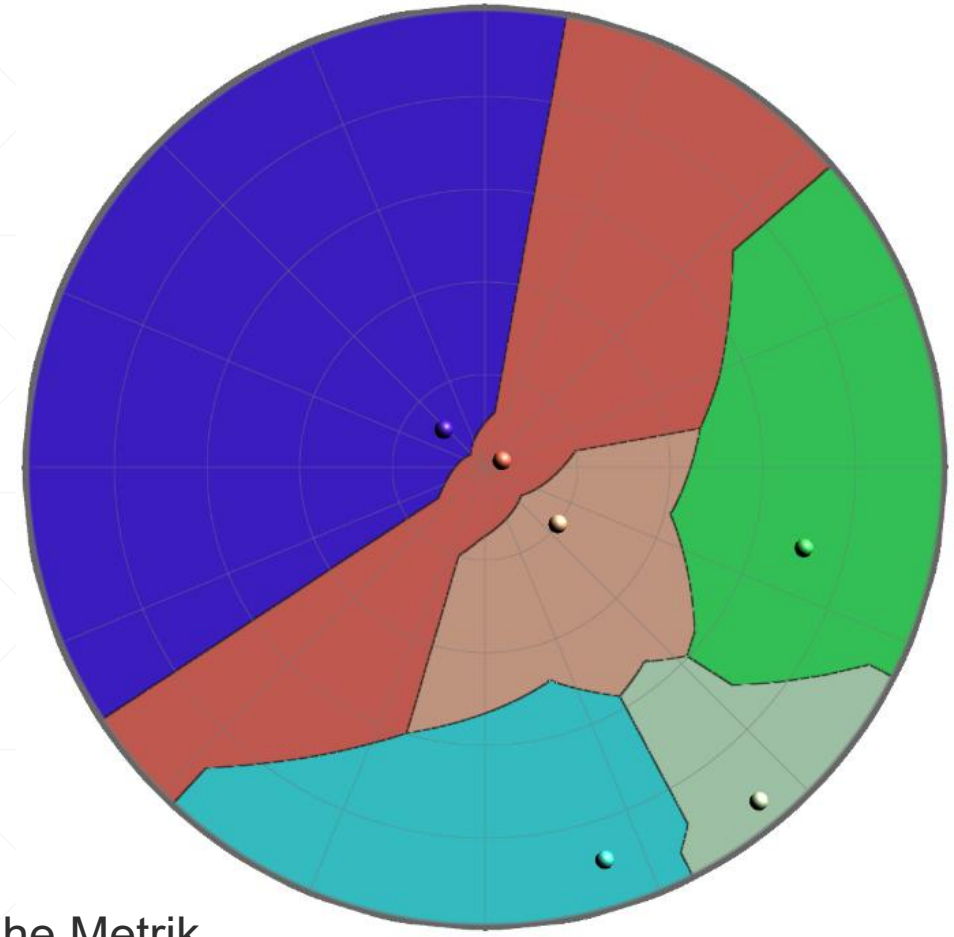
Bisektoren bei gleichem Radius



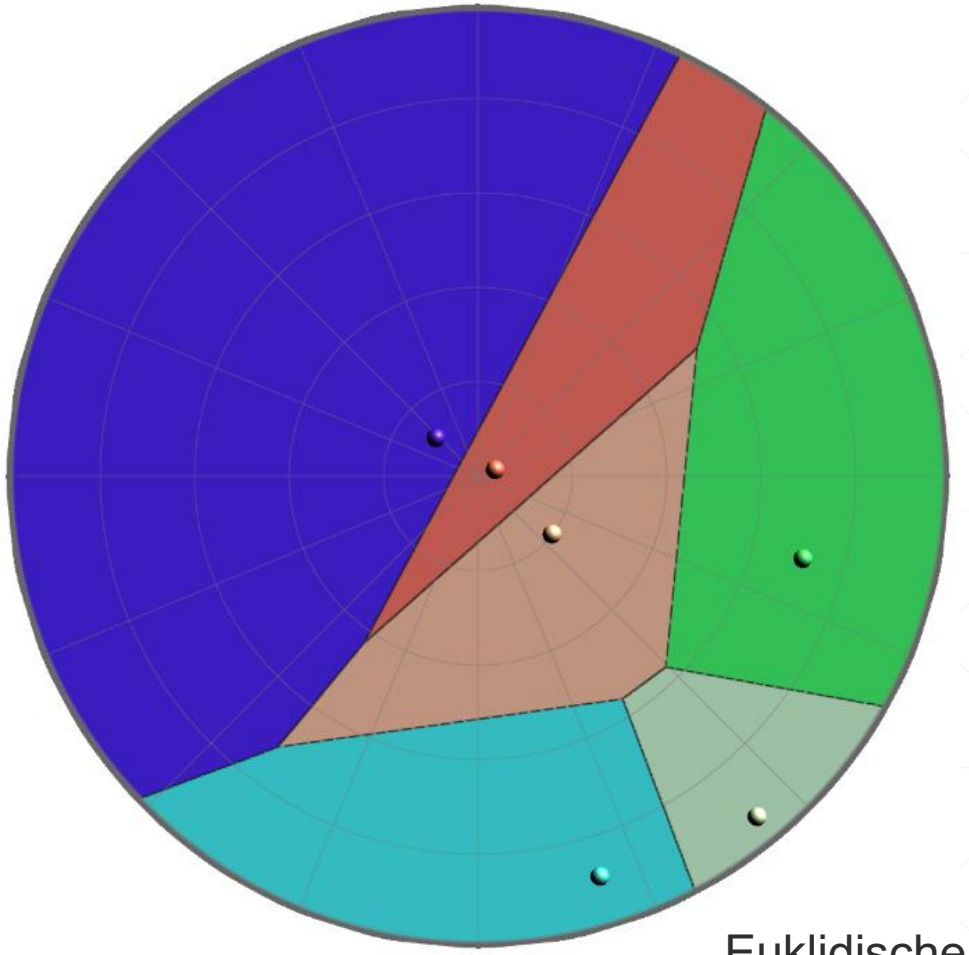




Eigenschaften: Translationsinvarianz

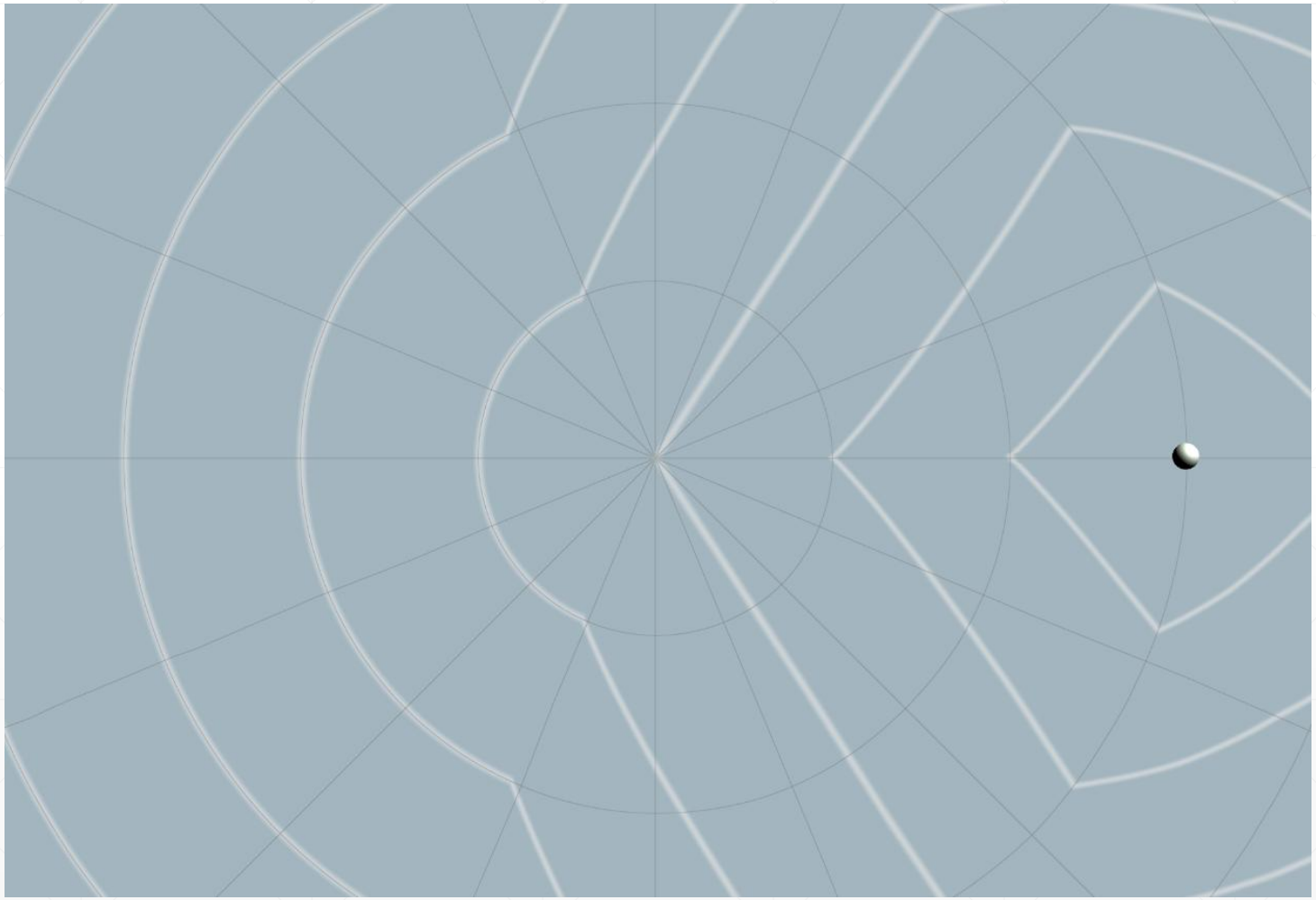


Karlsruhe Metrik



Euklidische Metrik

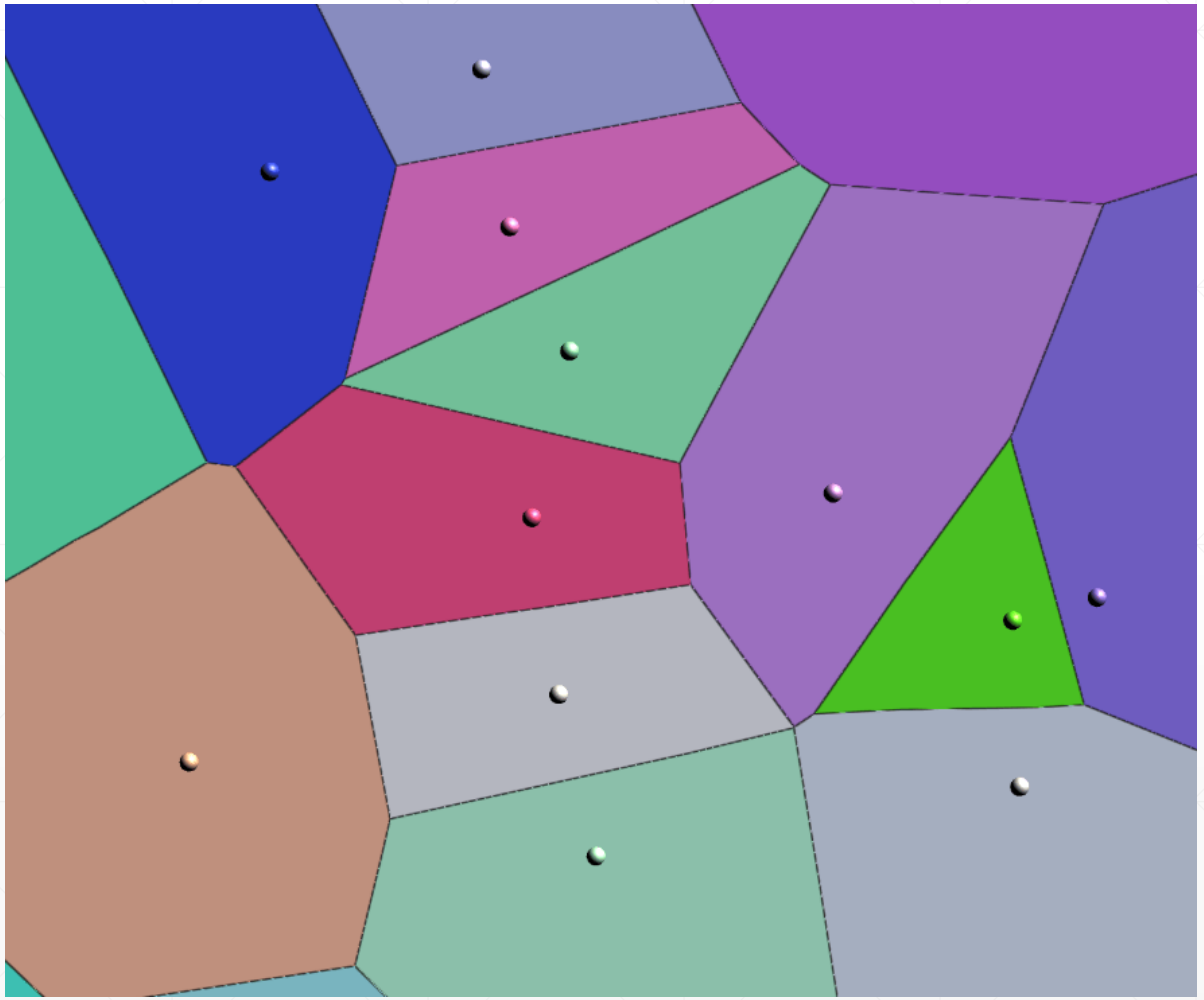
Einheits “Kreise”



Eigenschaften: Konvexe Voronoi Zellen

Euklidische Metrik: **Ja**

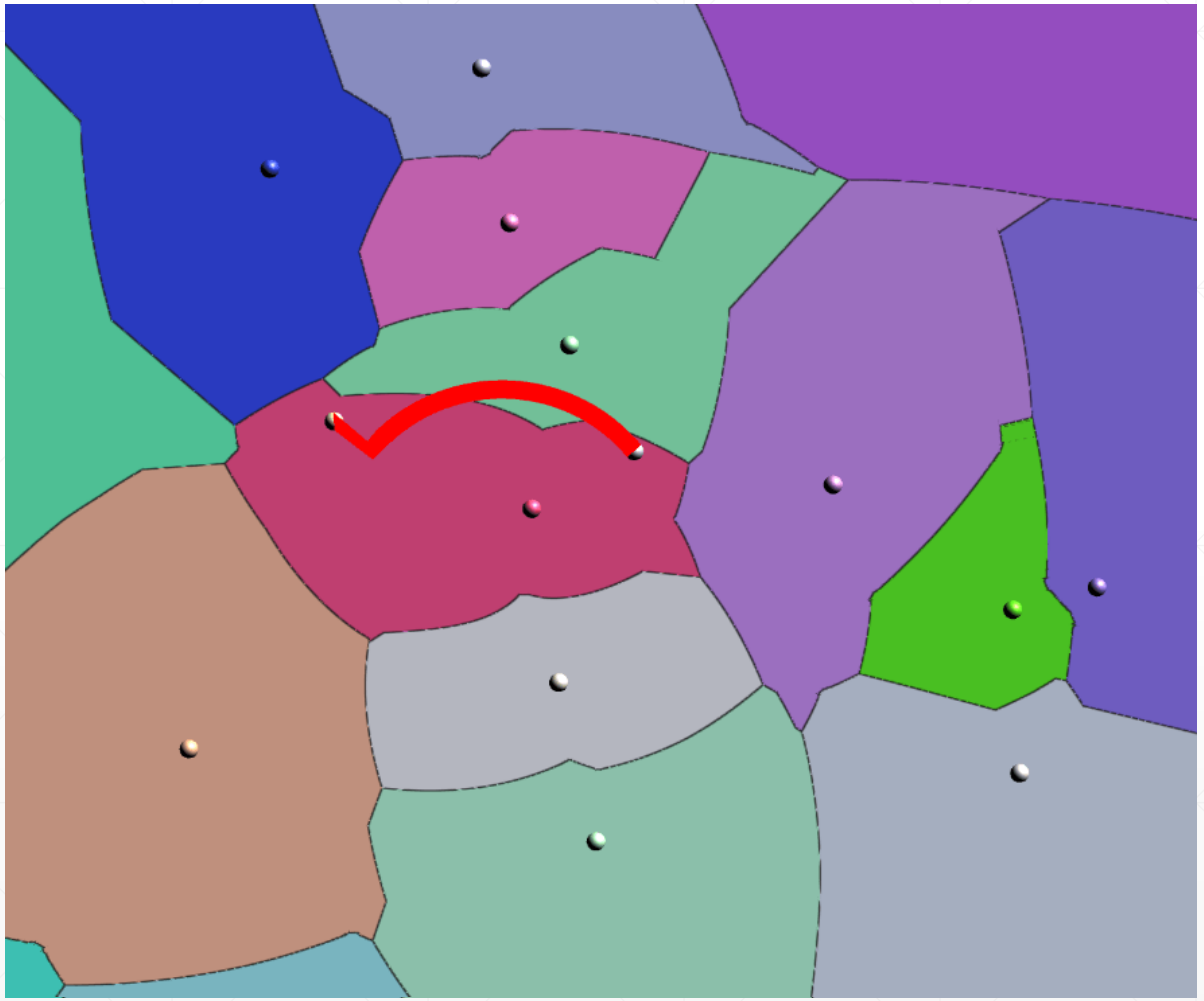
Für je zwei Punkte in einer Voronoi Zelle liegt der euklidische Pfad in der Zelle.



Konvexe Voronoi Zellen in der Karlsruhe Metrik?

Für je zwei Punkte in einer Voronoi Zelle liegt der euklidische Pfad in der Zelle? **Nein**

Für je zwei Punkte in einer Voronoi Zelle liegt der Karlsruhe-Pfad in der Zelle? **Nein**



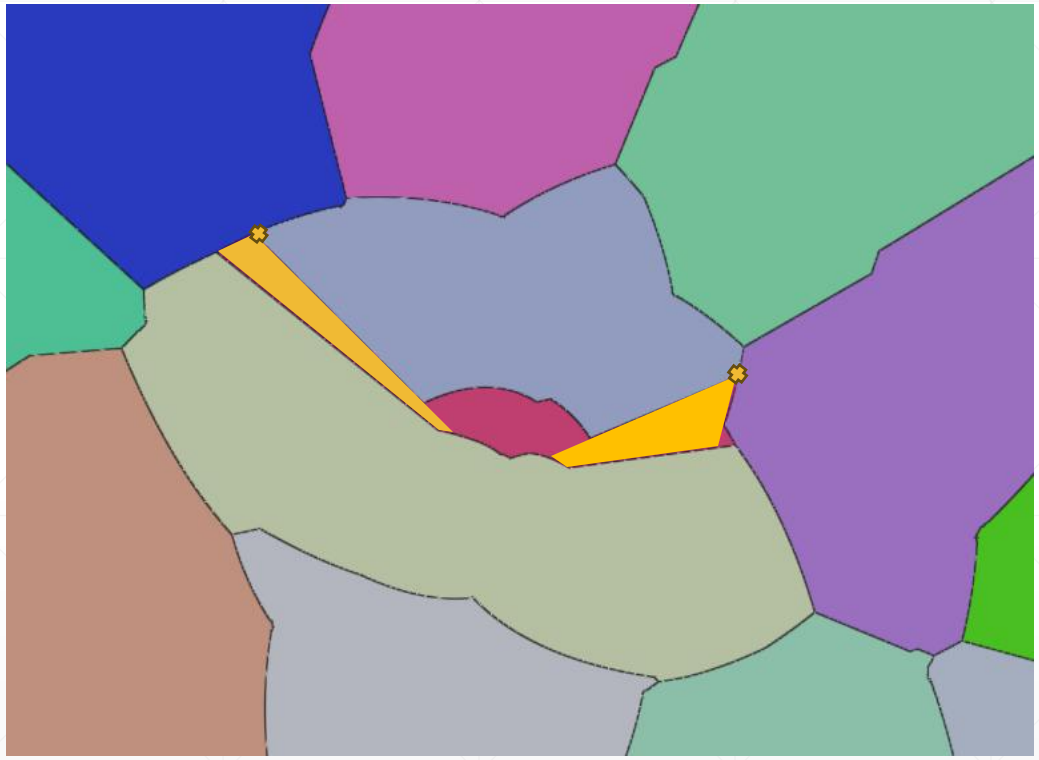
Sternförmig / Starshaped

Eine Menge *M* von Punkten heißt **sternförmig**, wenn es einen Punkt (Sternzentrum) gibt, von dem aus alle Punkte der Menge „sichtbar“ sind.

„Sichtbar“ – der euklidische Pfad liegt in der Menge

m-sternförmig bezüglich der Metrik m:

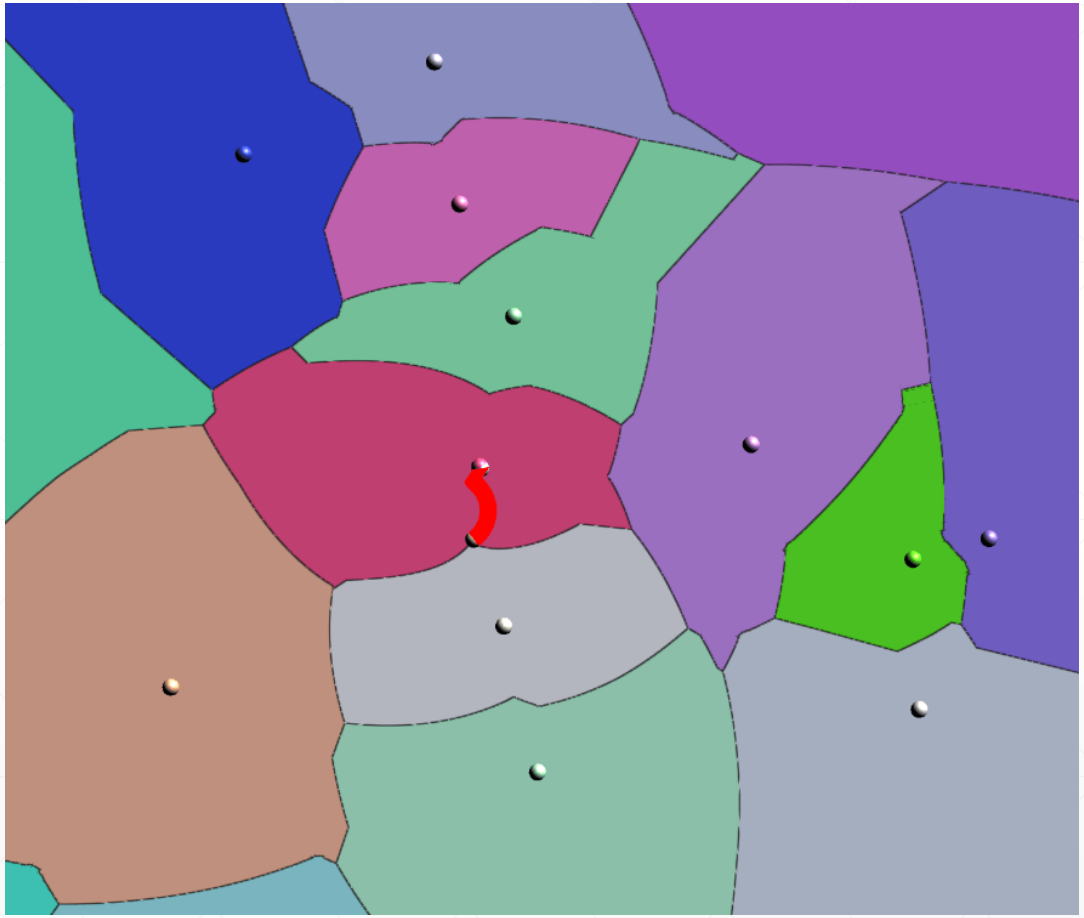
Eine Menge *M* von Punkten heißt **m-sternförmig**, wenn es einen Punkt gibt, für den ein kürzester Pfad in der Metrik m zu einem beliebigen Punkt x ∈ *M* vollständig in *M* liegt.



m-sternförmig / m-starshaped

m-sternförmig bezüglich der Metrik m:

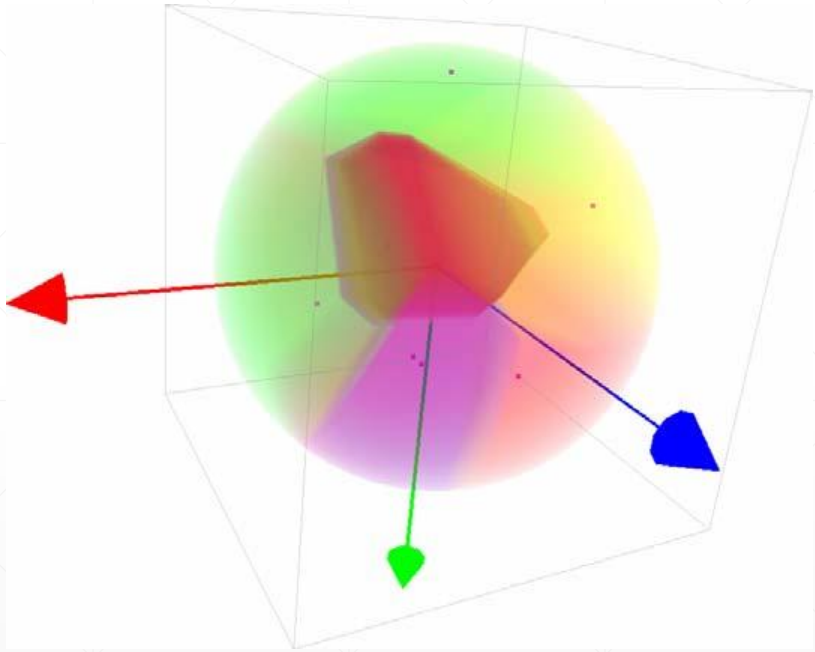
Eine Menge *M* von Punkten heißt **m-sternförmig**, wenn es einen Punkt gibt, für den ein kürzester Pfad in der Metrik m zu einem beliebigen Punkt x ∈ *M* vollständig in *M* liegt.



Dehne, F., Klein, R. “The big sweep”: On the power of the wavefront approach to Voronoi diagrams. Algorithmica 17, 19-32 (1997).

Future Work

- In 2D: Wavefront Algorithmus
- In 3D: Volume Rendering mit Ray Marching
- Andere nicht-euklidische Metriken?



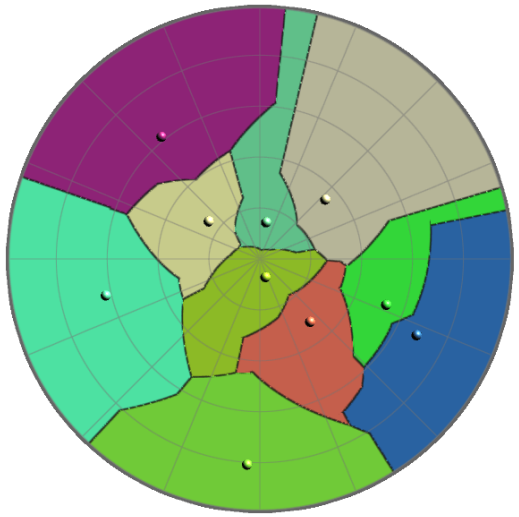
[1]

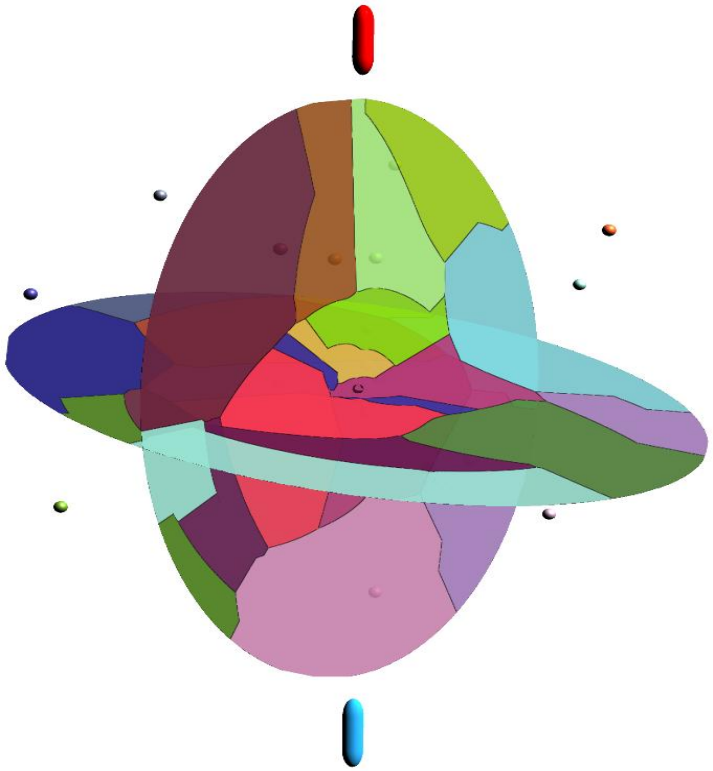
[1] https://www.lix.polytechnique.fr/~nielsen/GPUQuantumVoronoiDiagrams-draft.pdf

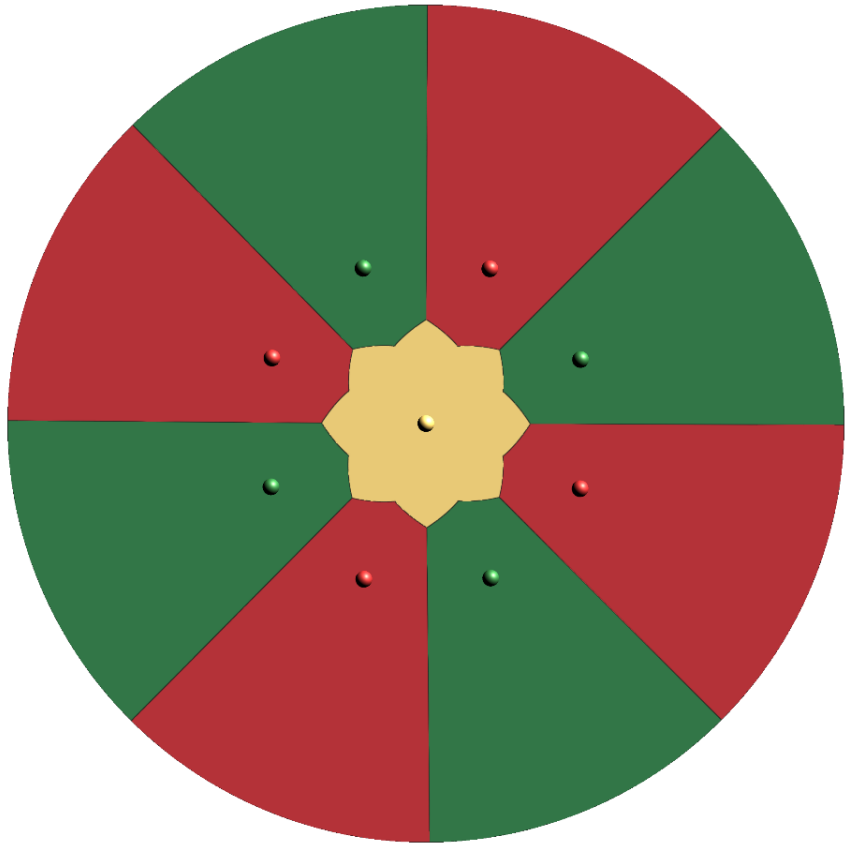
Zusammenfassung

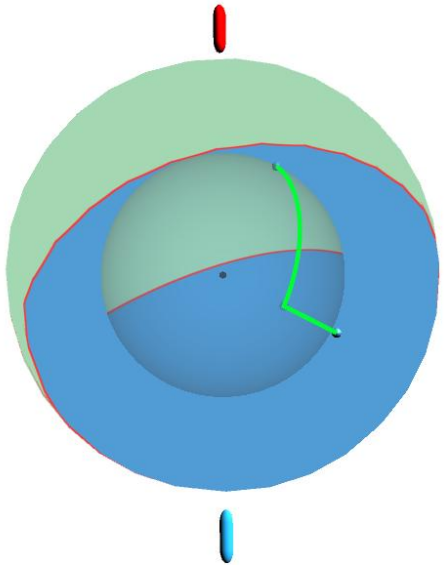
- Direkte Visualisierung statt expliziter Berechnung
- GPU statt spezieller Algorithmen
- Shader liefert direktes Feedback: z.B. Punkte verschieben, Kugel rotieren
- 3D braucht Interaktion, nicht nur Abbildungen
- Transparenz bleibt schwierig

Fragen?





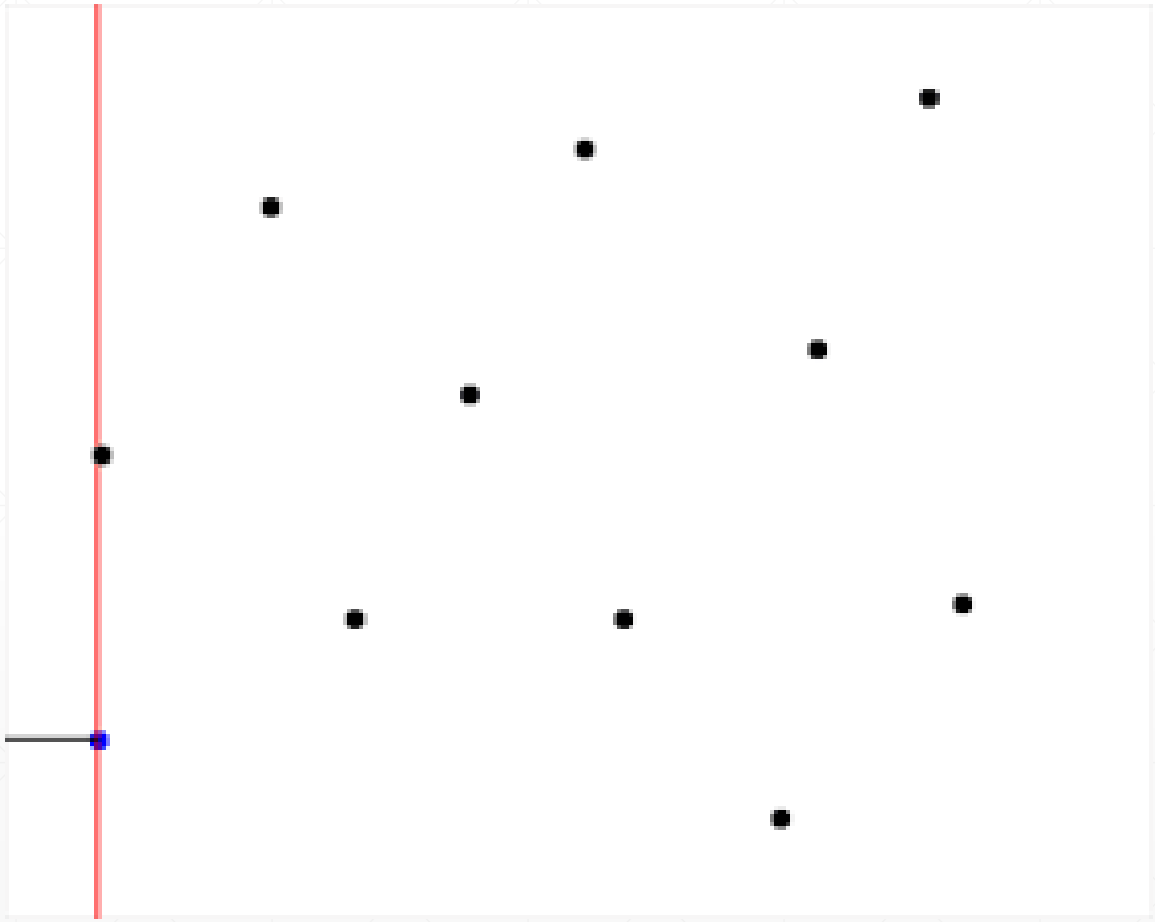




Literatur

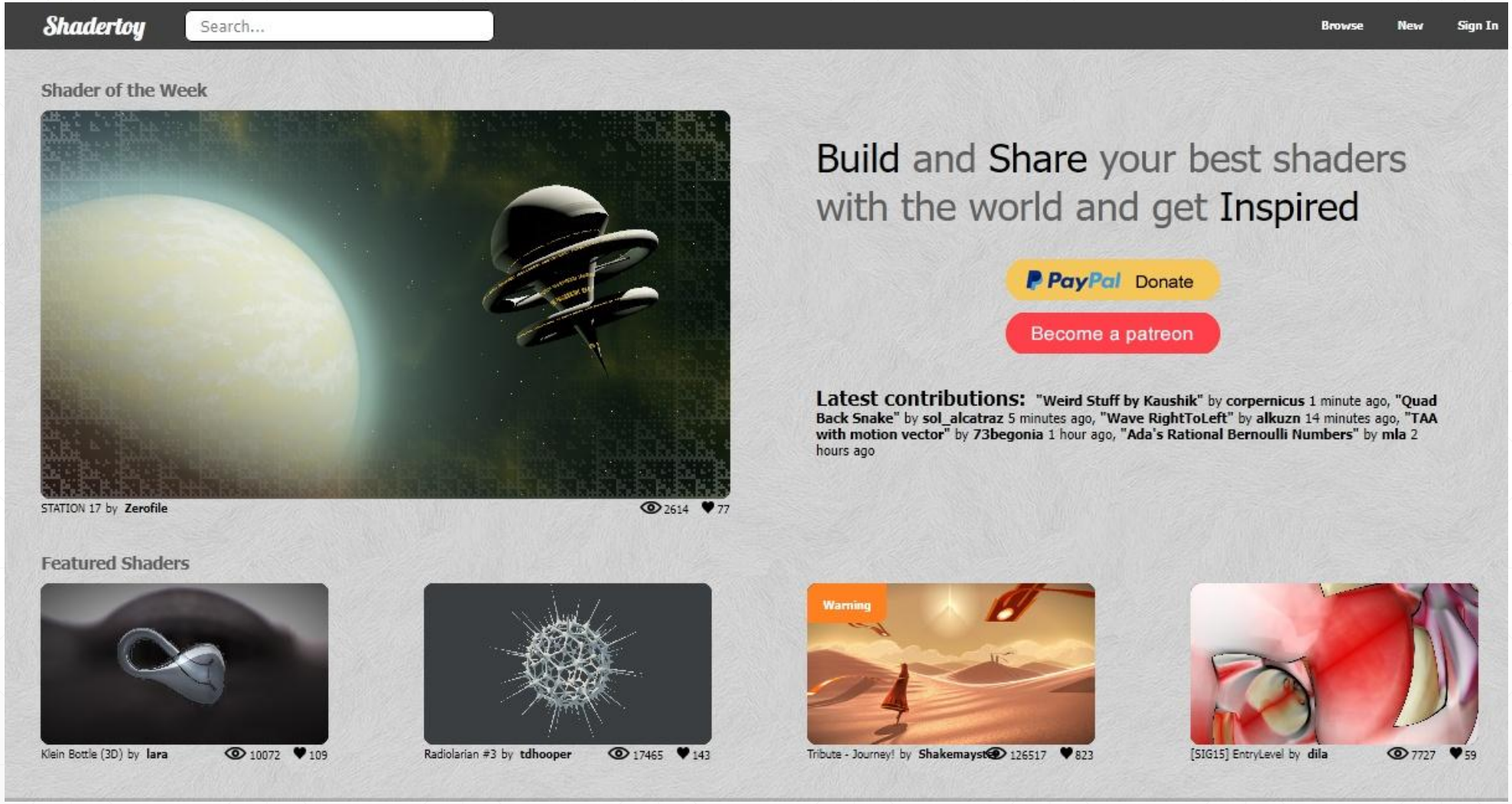
- Rolf Klein (1988).
  Voronoi Diagrams in the Moscow Metric.
  Graph-Theoretic Concepts in Computer Science. WG 1988.
- Atsuyuki Okabe, Barry N. Boots, Kokichi Sugihara (1992).
  Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.
  John Wiley & Sons.
- Steven Fortune (1986).
  A Sweepline Algorithm for Voronoi diagrams.
  Proceedings of the Second Annual Symposium on Computational Geometry.

Fortune‘s Algorithm „Wavefront“ „Beachline“



[1]

[1] https://commons.wikimedia.org/wiki/File:Fortunes-algorithm-slowed.gif



www.shadertoy.com