

Cinema 4D SDK

Eine Einführung

SDK-Team

- Ich bin Ferdinand und arbeite im SDK Team bei Maxon
- Wir entwickeln, pflegen und dokumentieren das Cinema 4D SDK
- Wir leisten auch den Entwickler-Support für Dritte
- Ihr könnt mit uns über

`sdk_support(at)maxon.net`

in Kontakt treten.

Themen

- SDK-Team
- Überblick Cinema 4D API
- Praktisches Beispiel
- Ressourcen

- Q&A für alle Präsentationen

Überblick Cinema 4D API

Cinema 4D API

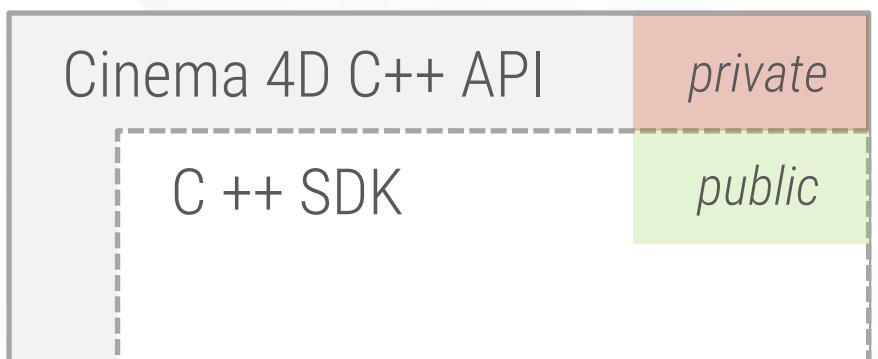
- Cinema 4D ist in C++ 17 geschrieben und diese Codebasis bildet die nicht öffentliche C++ API.

Cinema 4D C++ API

private

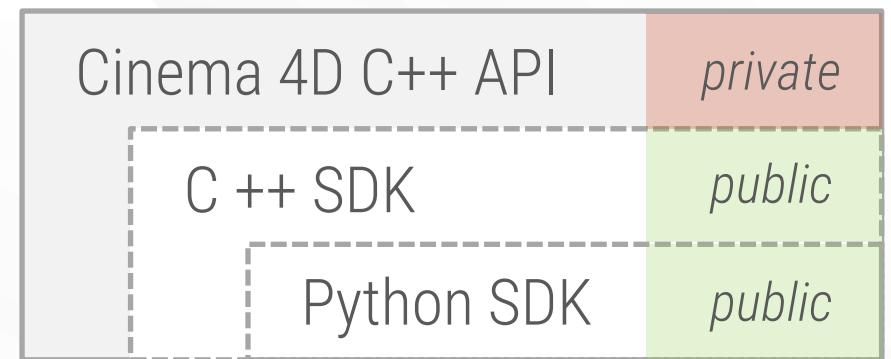
Cinema 4D API

- Cinema 4D ist in C++ geschrieben und diese Codebasis bildet die nicht öffentliche C++ API.
- Das **C++ Service Development Kit** (SDK) stellt eine Teilmenge dieser API für die Plugin-Entwicklung in C++ für Windows und MacOS öffentlich für Dritte zur Verfügung.



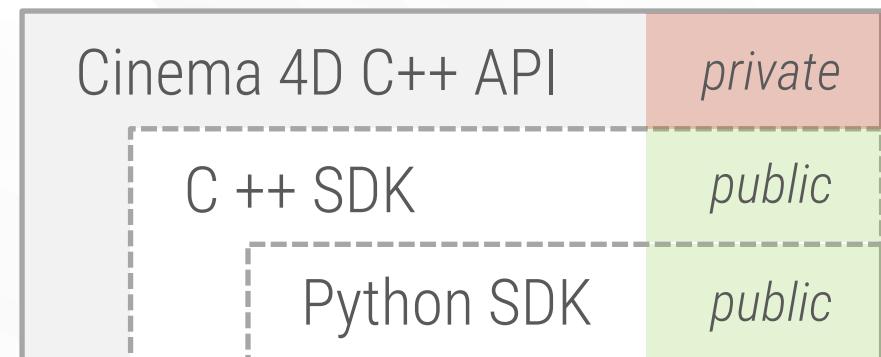
Cinema 4D API

- Cinema 4D ist in C++ geschrieben und diese Codebasis bildet die nicht öffentliche C++ API.
- Das **C++ Service Development Kit (SDK)** stellt eine Teilmenge dieser API für die Plugin-Entwicklung in C++ für Windows und MacOS öffentlich für Dritte zur Verfügung.
- Das **Python SDK** stellt Teilmenge des C++ SDK für die Plugin-Entwicklung in CPython 3.x für Dritte zur Verfügung.



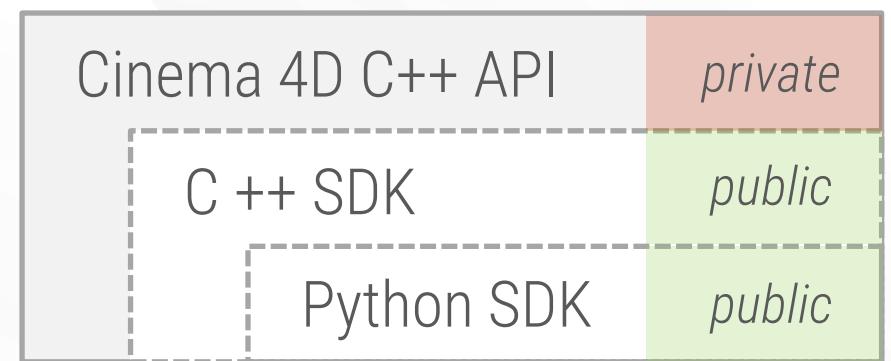
Cinema 4D API

- Das C++ und Python SDK sind primär konzipiert, um es Dritten zu ermöglichen, *Plugins* für Cinema 4D zu entwickeln.



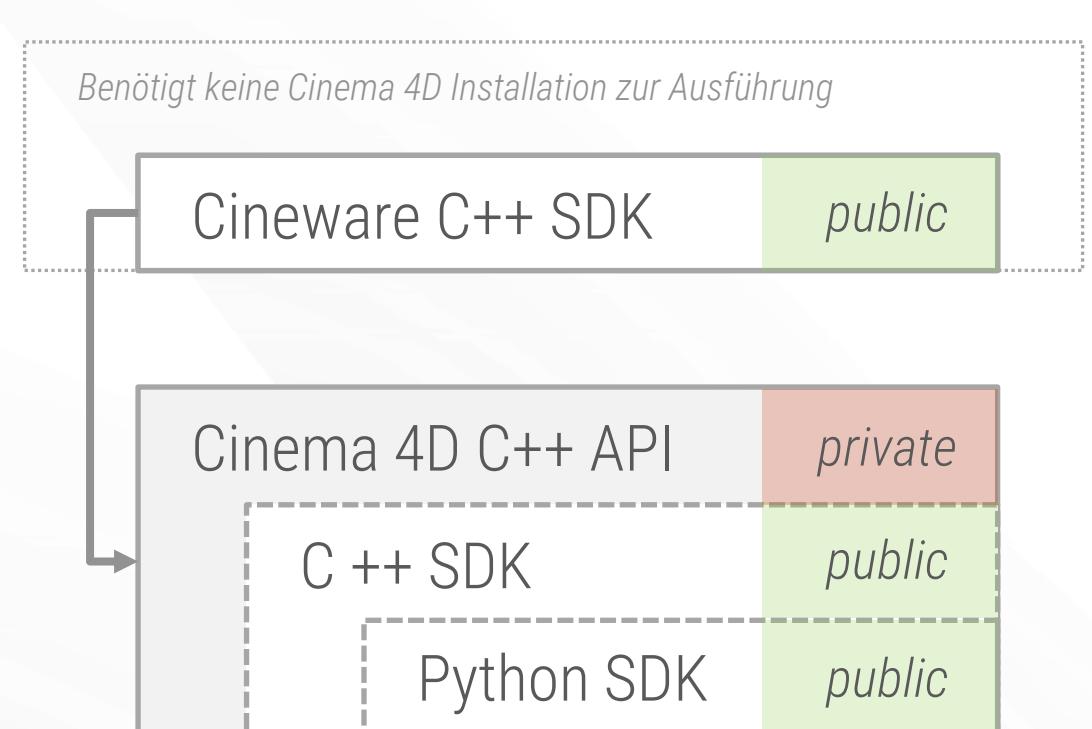
Cinema 4D API

- Das C++ und Python SDK sind primär konzipiert, um es Dritten zu ermöglichen, *Plugins* für Cinema 4D zu entwickeln.
- Plugin: Ein Zusatzfunktionalität für eine Software, die durch Nutzer nachinstalliert werden kann.
- Cinema bietet verschiedene Plugin-Typen an, die durch Dritte umgesetzt werden können: *Objekte*, *Tags* , *Tools*, *Materialen*, *Shader* und viele mehr ...



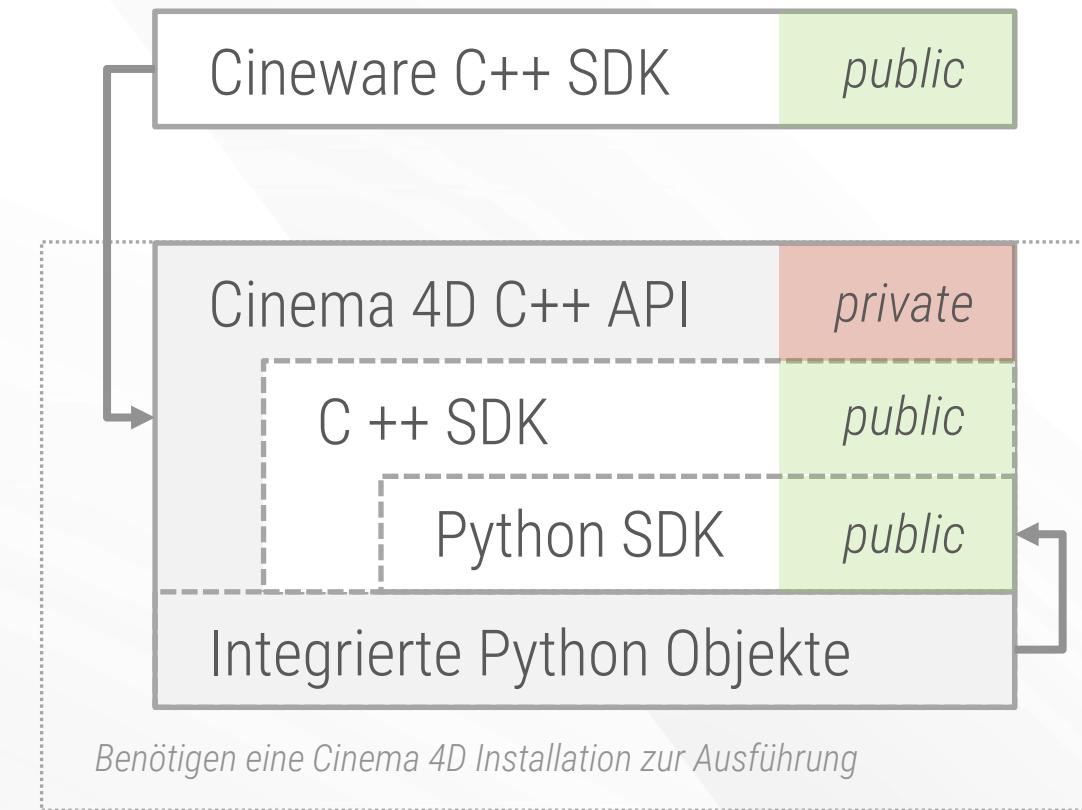
Cinema 4D API

- **Cineware** ist eine C++ Bibliothek zum Laden, Speichern und Erstellen von Dateien im .c4d Format.



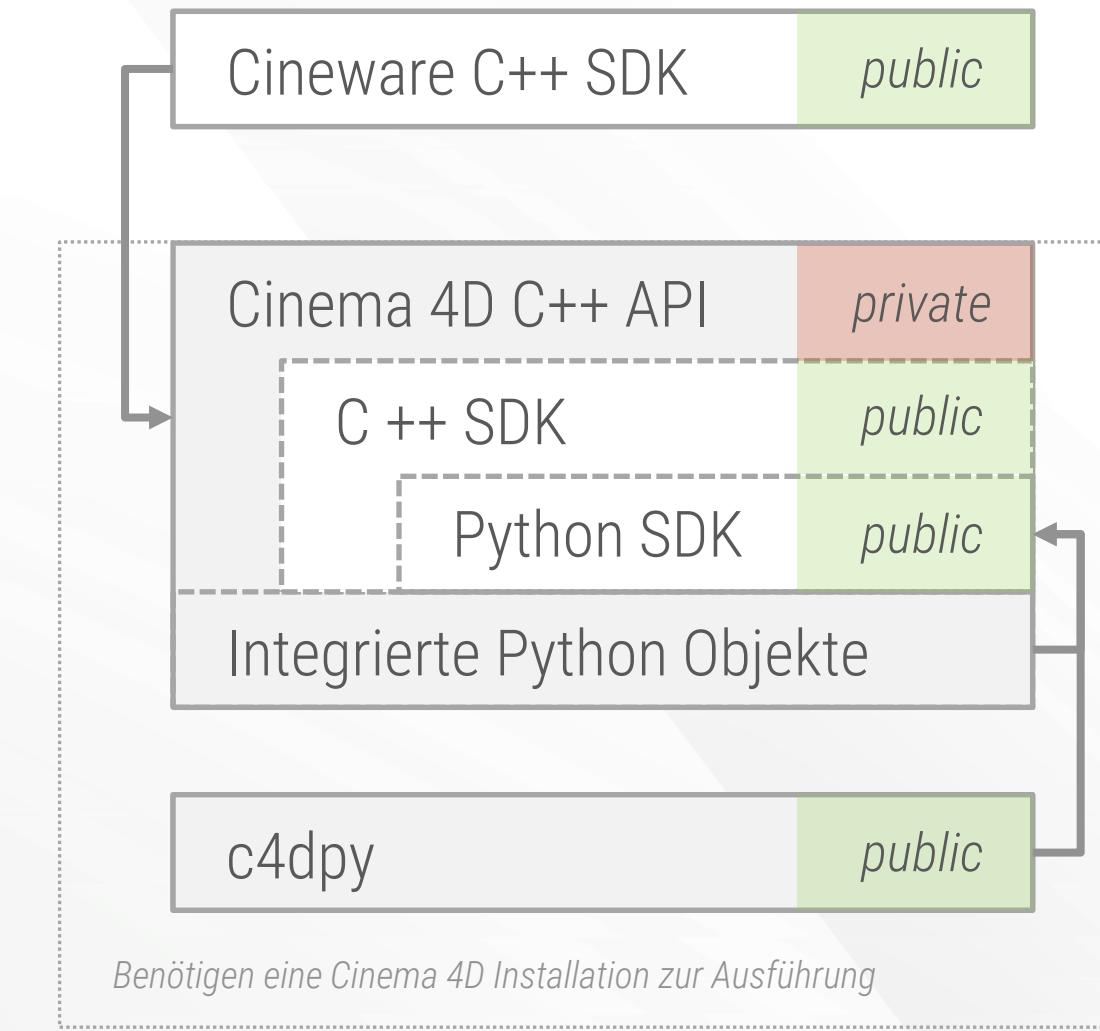
Cinema 4D API

- **Cineware** ist eine C++ Bibliothek zum Laden, Speichern und Erstellen von Dateien im .c4d Format.
- Cinema 4D bietet auch eine Reihe integrierter Python Objekte, die eine direkte Umsetzung wichtiger Arbeitsweisen erlauben.



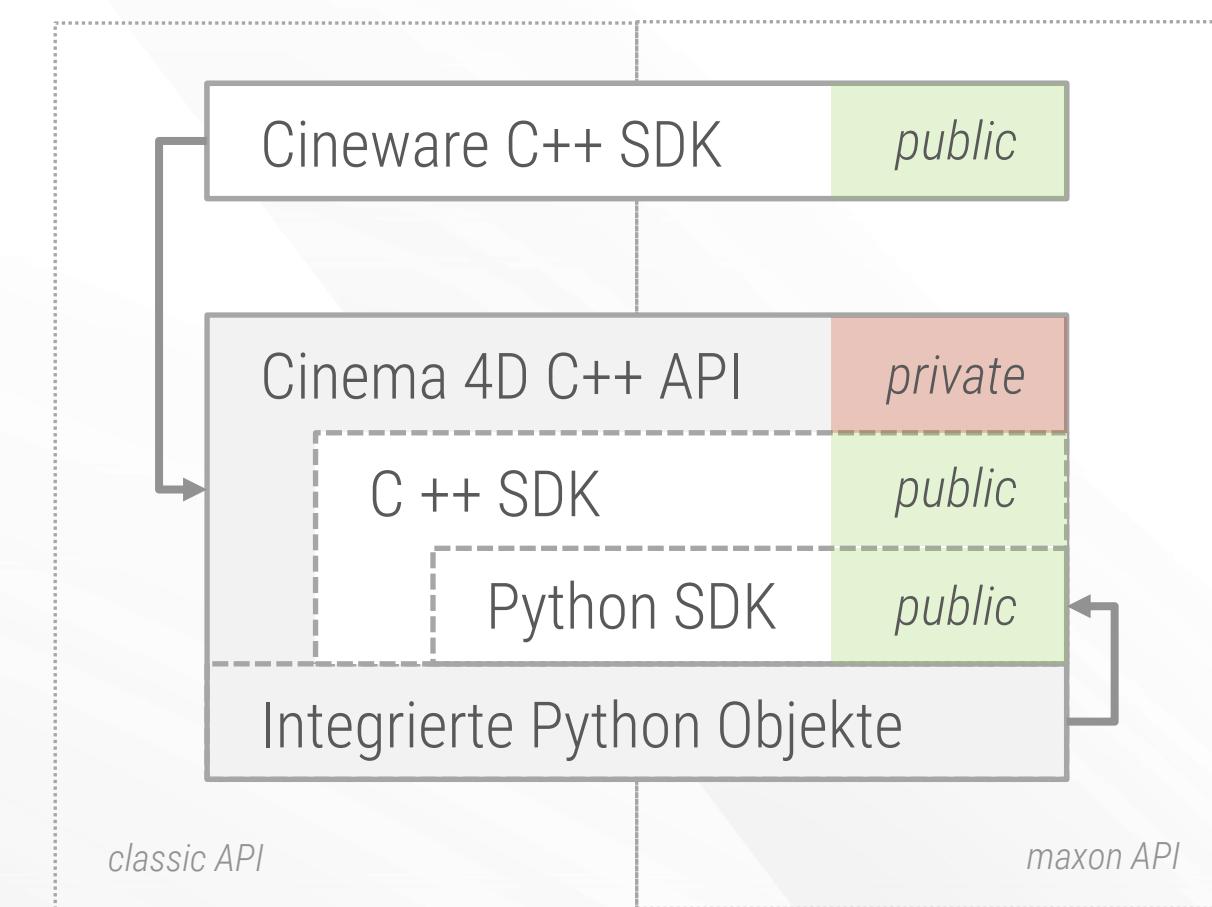
Cinema 4D API

- **Cineware** ist eine C++ Bibliothek zum Laden, Speichern und Erstellen von Dateien im .c4d Format.
- Cinema 4D bietet auch eine Reihe integrierter Python Objekte, die eine direkte Umsetzung wichtiger Arbeitsweisen erlauben.
- Zusätzlich können auch Python Skripte in Cinema und mittels **c4dpy**, einem eigenständig für Cinema konzipierten CPython Interpreter, ausgeführt werden oder ein interaktiver Interpreter gestartet werden.



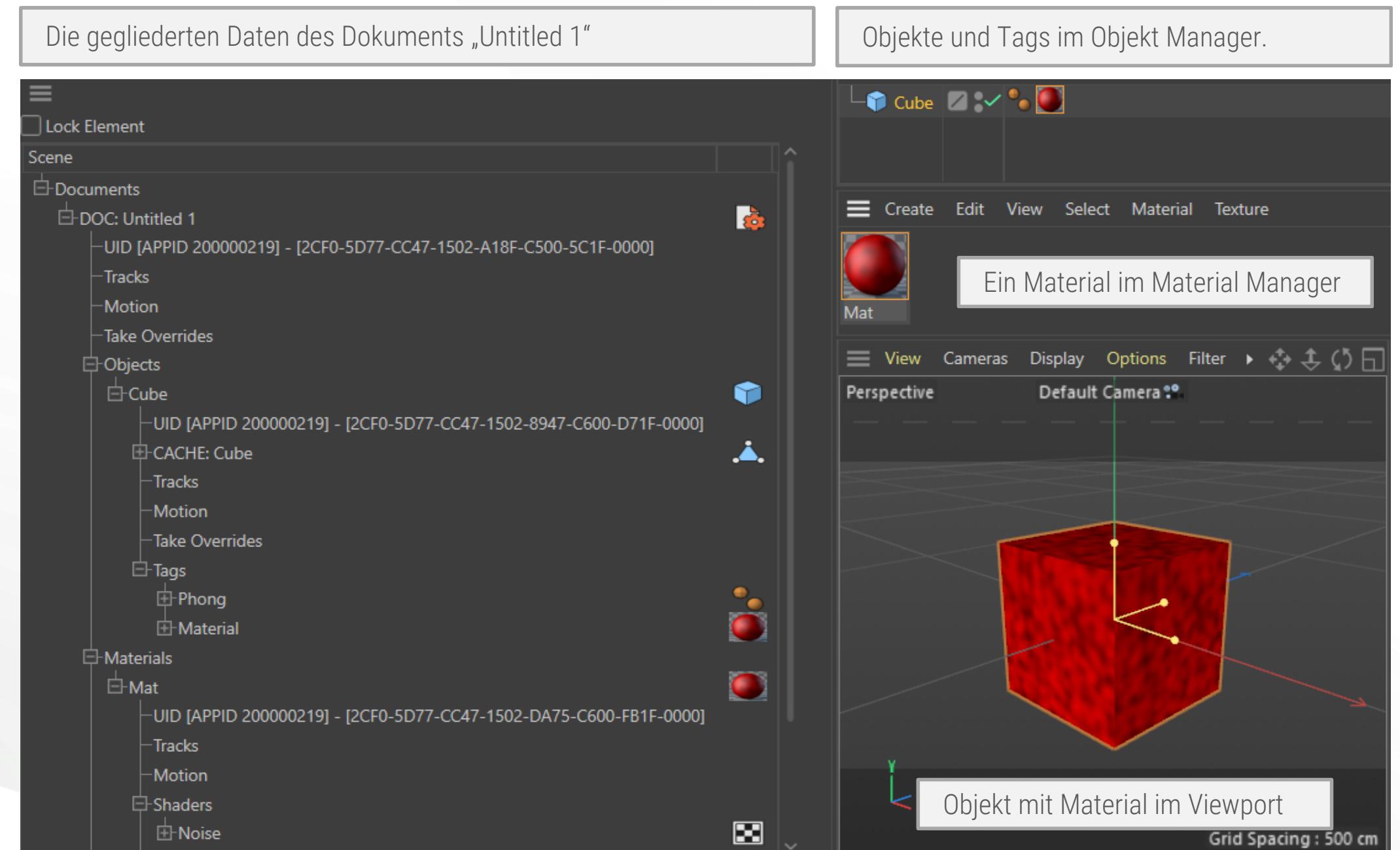
Cinema 4D API

- **Cineware** ist eine C++ Bibliothek zum Laden, Speichern und Erstellen von Dateien im .c4d Format.
- Cinema 4D bietet auch eine Reihe integrierter Python Objekte, die eine direkte Umsetzung wichtiger Arbeitsweisen erlauben.
- Zusätzlich können auch Python Skripte in Cinema und mittels *c4dpy* ausgeführt werden oder ein interaktiver Interpreter gestartet werden.
- Die Codebasis von Cinema 4D ist in die „classic“ und maxon API aufgeteilt.



Cinema 4D Scene Graph

- Große Teile der Daten eines Dokuments, einer Szene, werden in einer hierarchischen Struktur verwaltet.
- Dialog auf der linken Seite ist das *ActiveObject* Plugin aus dem C++ SDK.¹

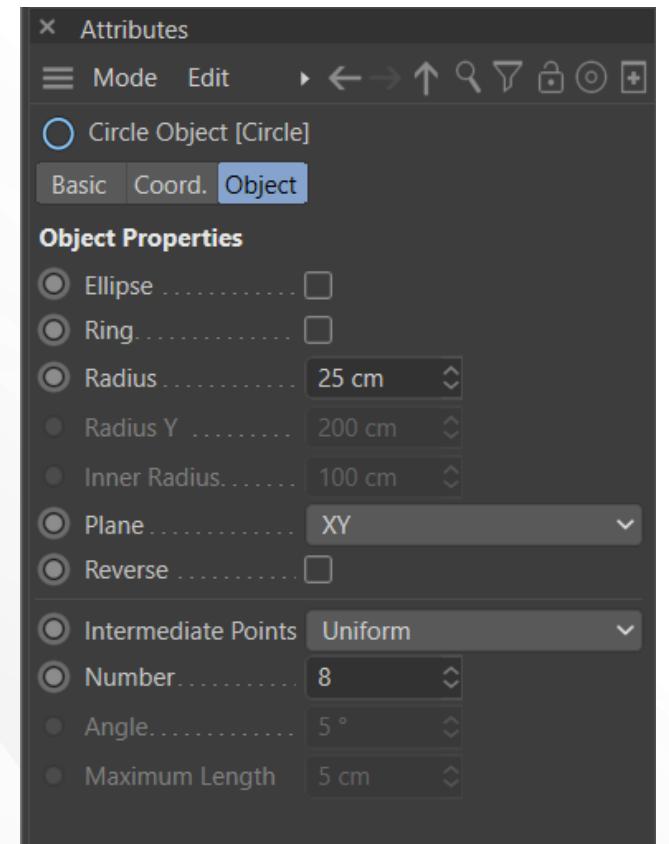


[1] github.com/PluginCafe/cinema4d_cpp_sdk_extended/blob/master/plugins/cinema4dsdk/source/gui/activeobject.cpp

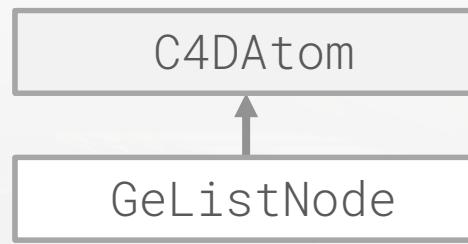
Wichtige Datentypen in der classic API

- C4DAtom ist die Basisklasse für viele Datentypen und setzt:
 - das Nachrichtensystem via C4DAtom.Message()
 - den Parameterzugriff via C4DAtom.GetParameter()
und .SetParameter() und mehr um.

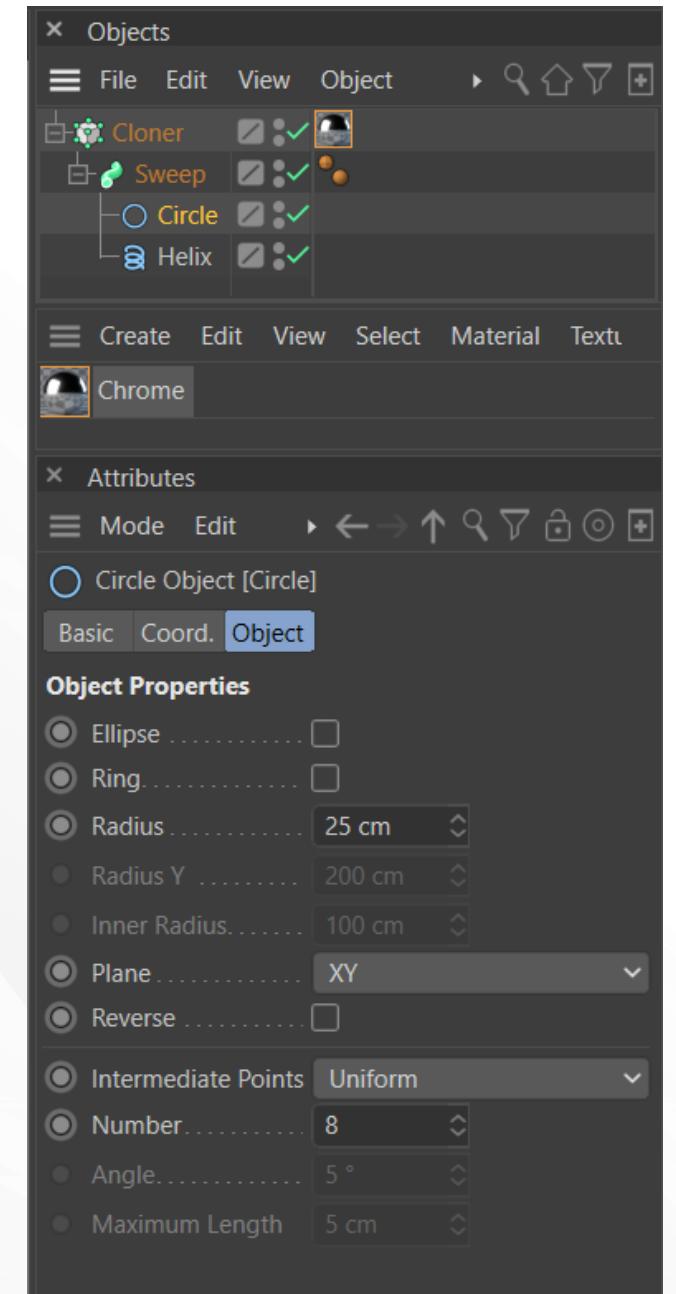
C4DAtom



Wichtige Datentypen in der classic API

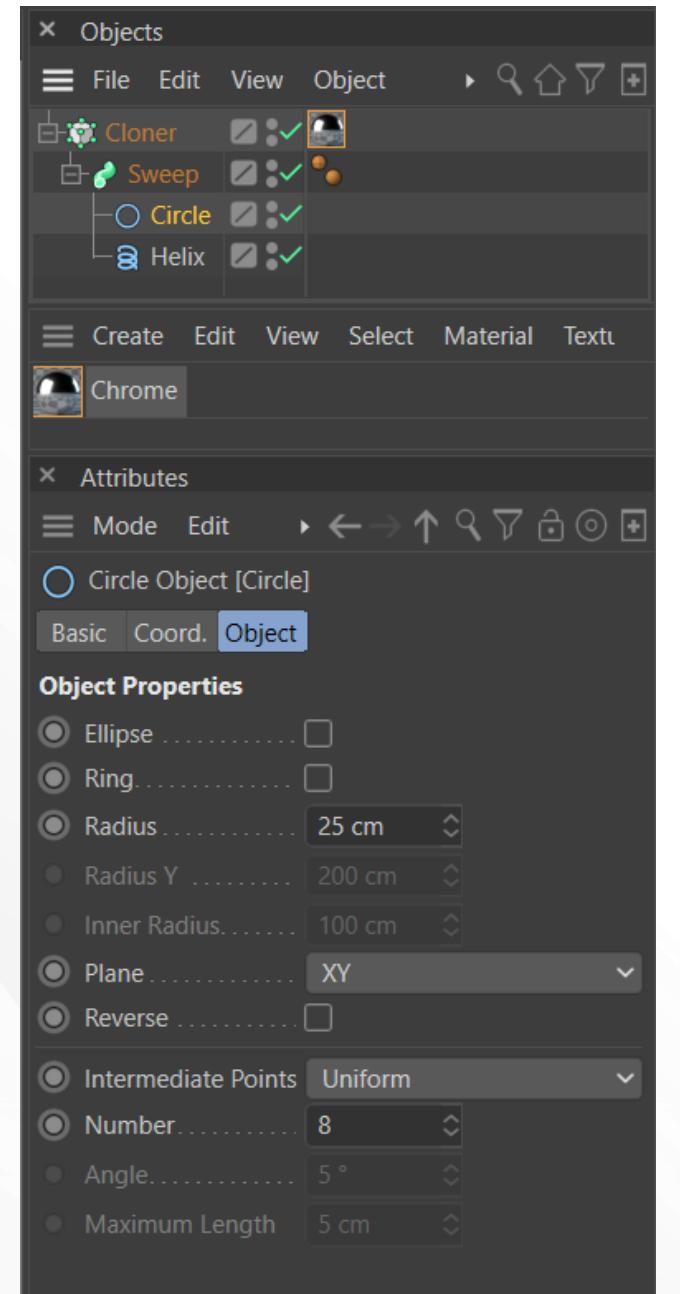
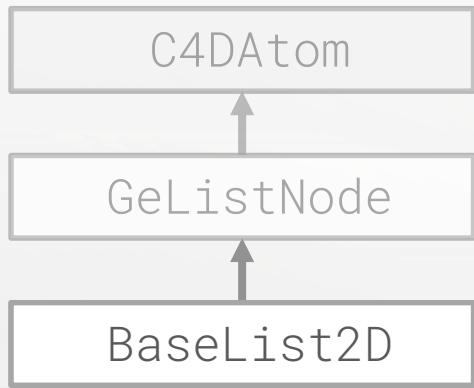


- **C4DAtom** ist die Basisklasse für viele Datentypen und setzt:
 - das Nachrichtensystem via `C4DAtom.Message()`,
 - den Parameterzugriff via `C4DAtom.GetParameter()` und `.SetParameter()` und mehr um.
- **GeListNode** setzt hierarchischen Relationen zwischen Nodes mittels
 - `GeListNode.GetPred()`, `.GetNext()`, `.GetUp()`, `.GetDown()` und mehr um.

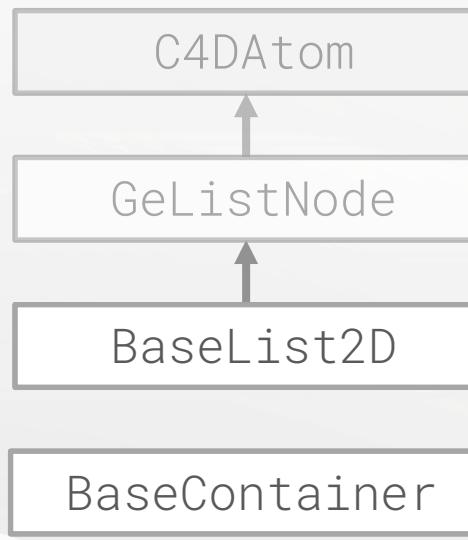


Wichtige Datentypen in der classic API

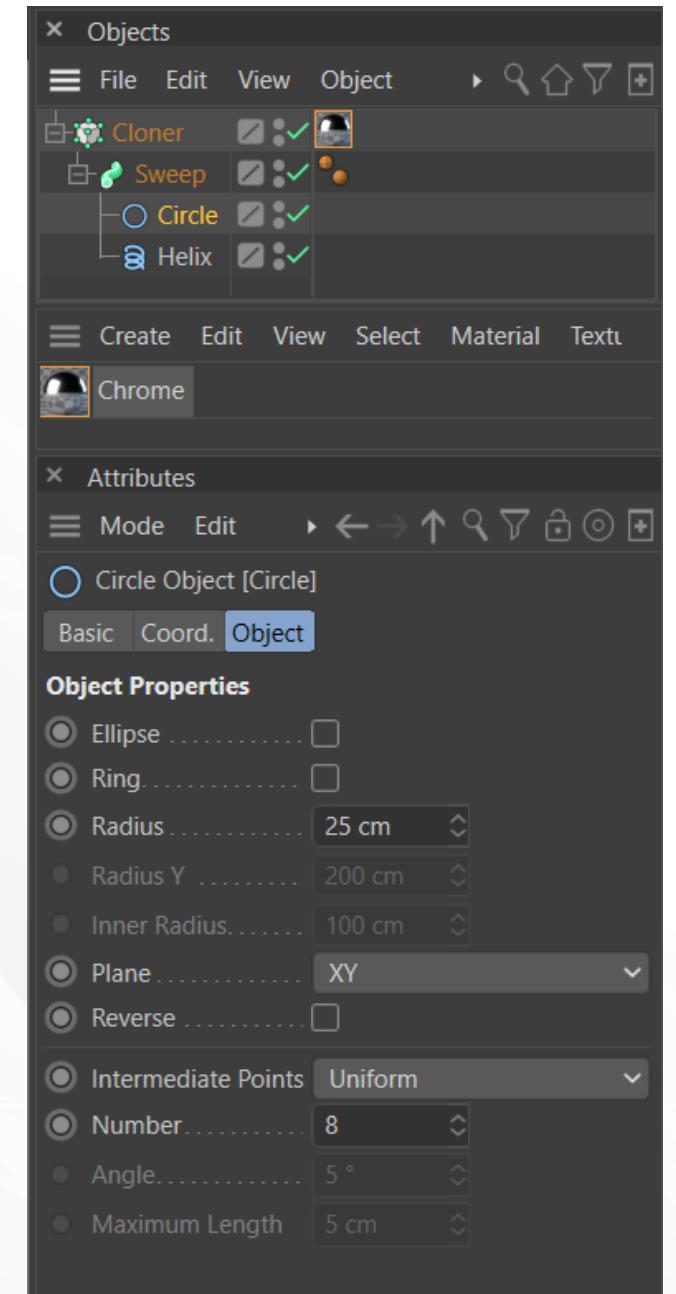
- **BaseList2D** setzt
 - Ebenen, Animationen, ...
 - den Parameterzugriff über BaseContainer um.



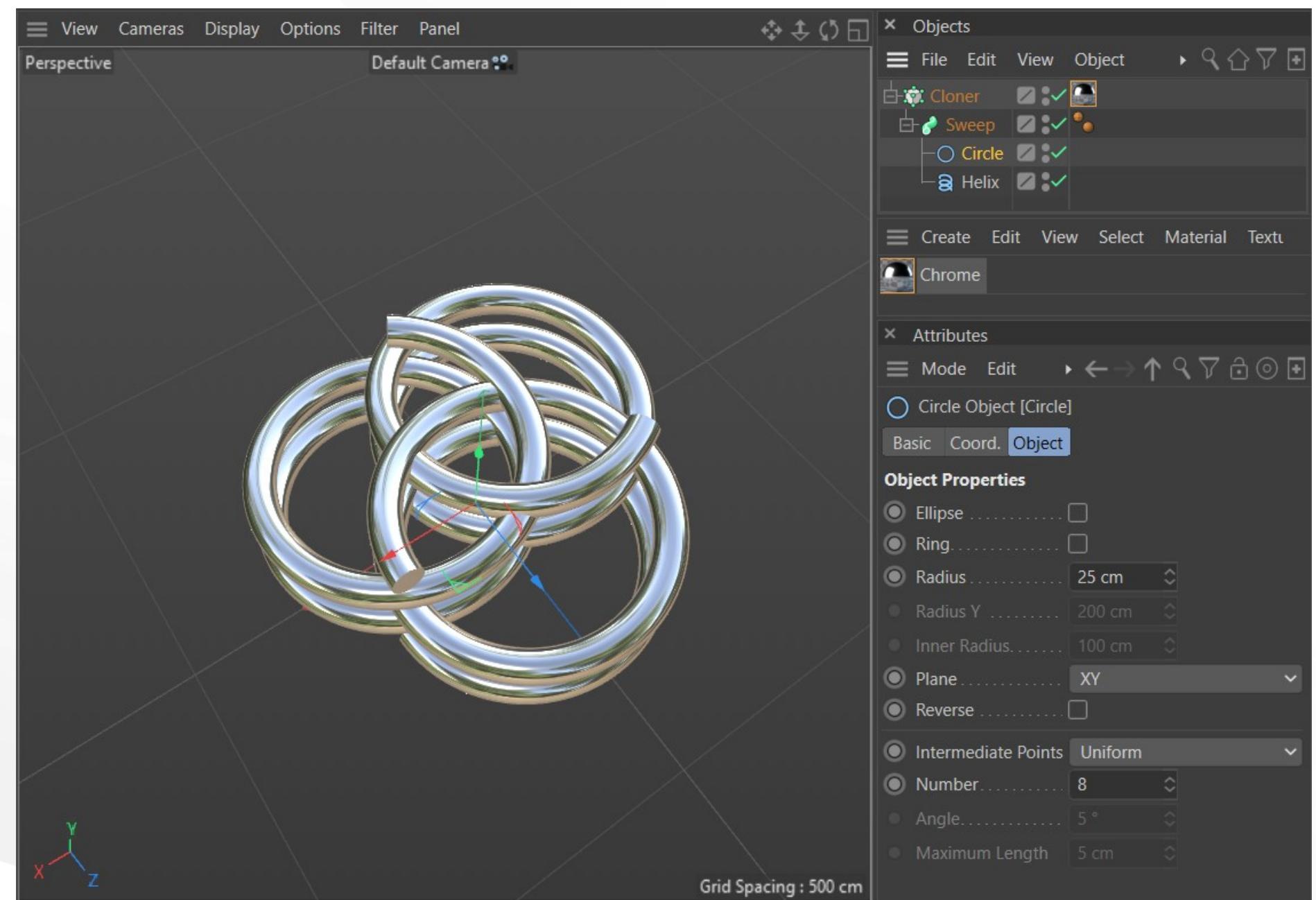
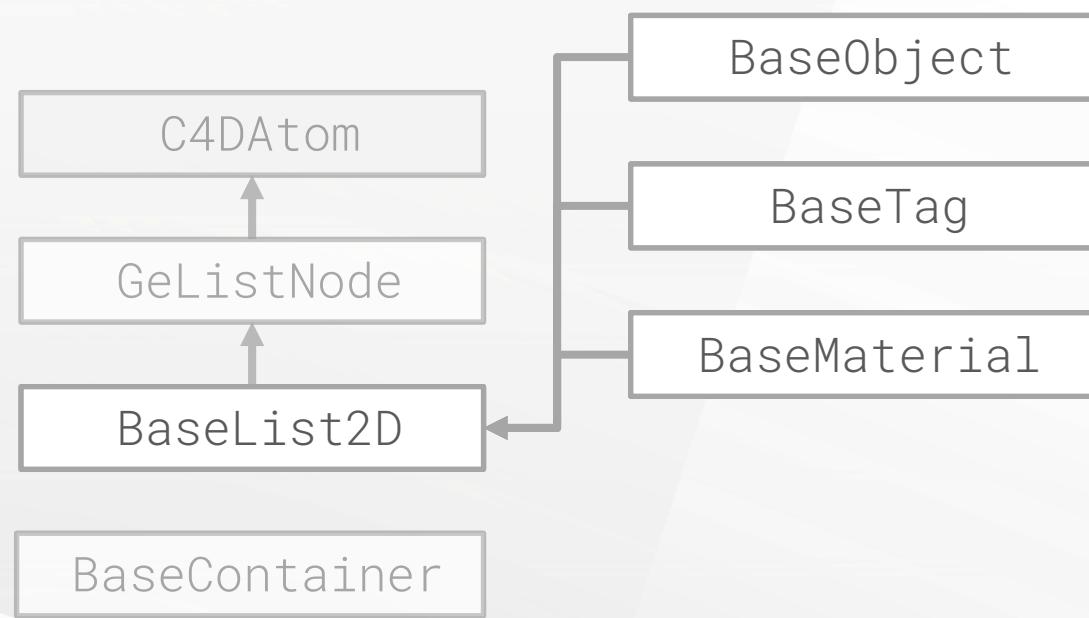
Wichtige Datentypen in der classic API



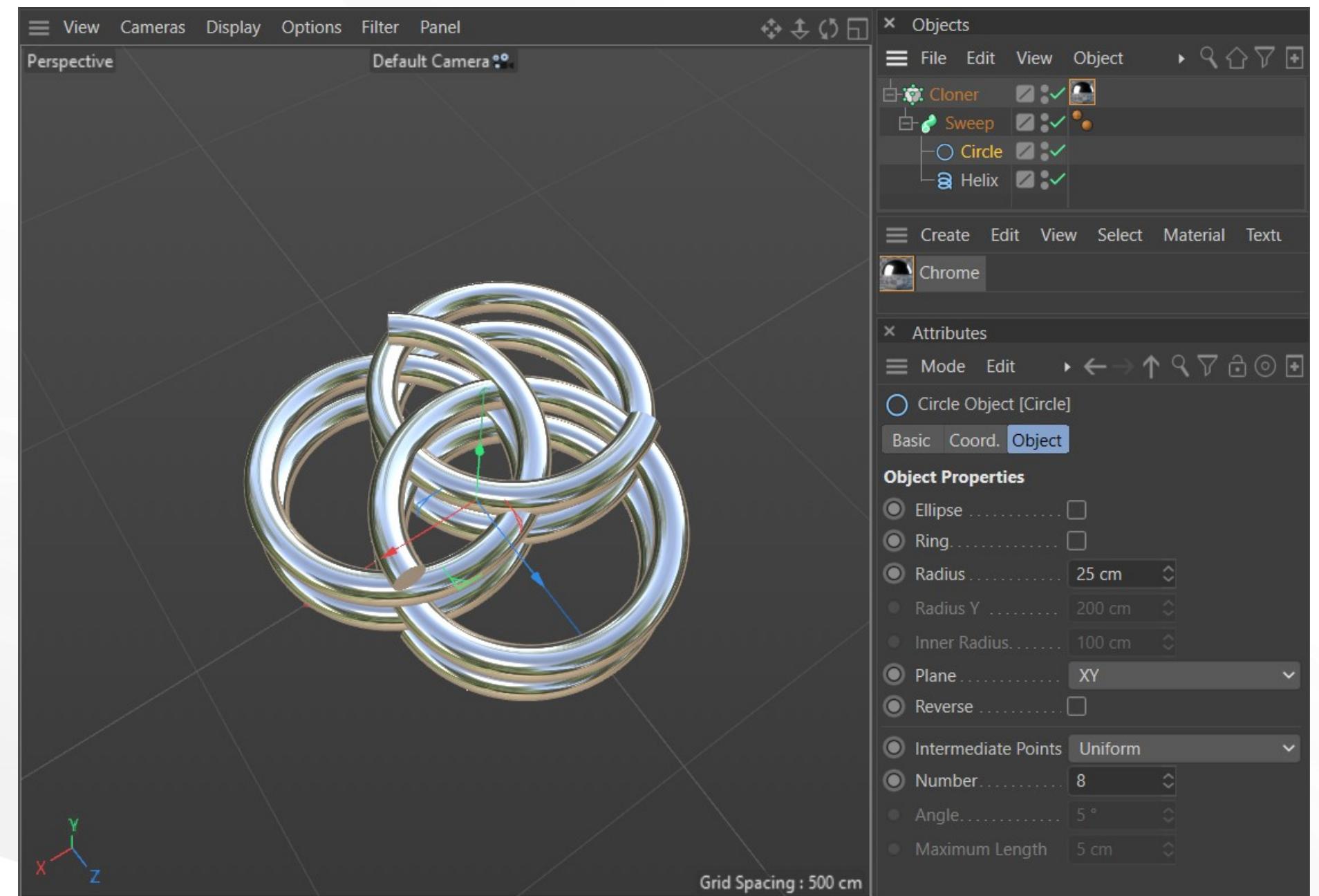
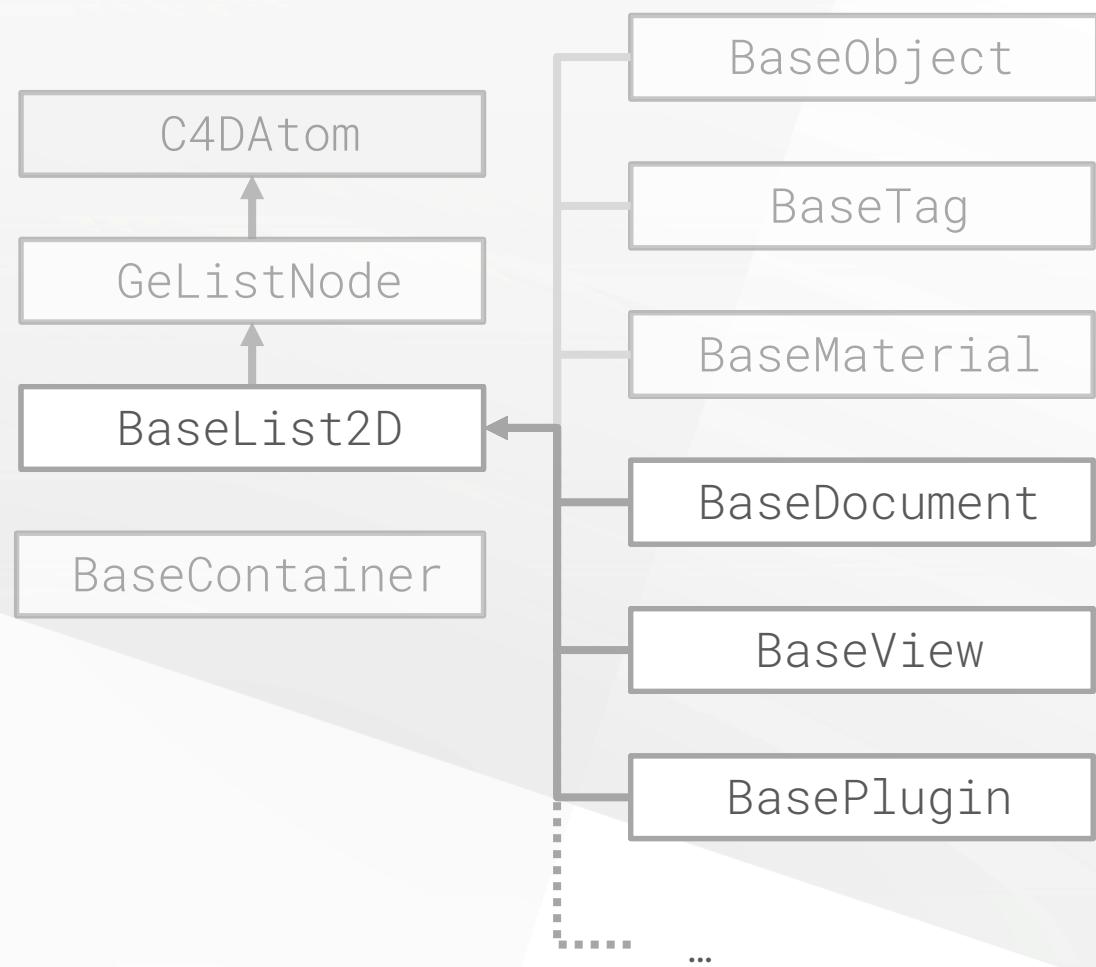
- **BaseList2D** setzt
 - Ebenen, Animationen, ...,
 - den Parameterzugriff über **BaseContainer** um.
- **BaseContainer** machen Schlüssel-Wert Paarsammlungen verfügbar.
 - Via `BaseList2D.GetData()` eine Alternative zum Parameterzugriff über `C4DAtom.GetParameter()` .
- Wer in die Cinema 4D API einsteigen möchte, sollte sich mit diesen vier Typen vertraut machen.



Wichtige Typen in Cinema 4D

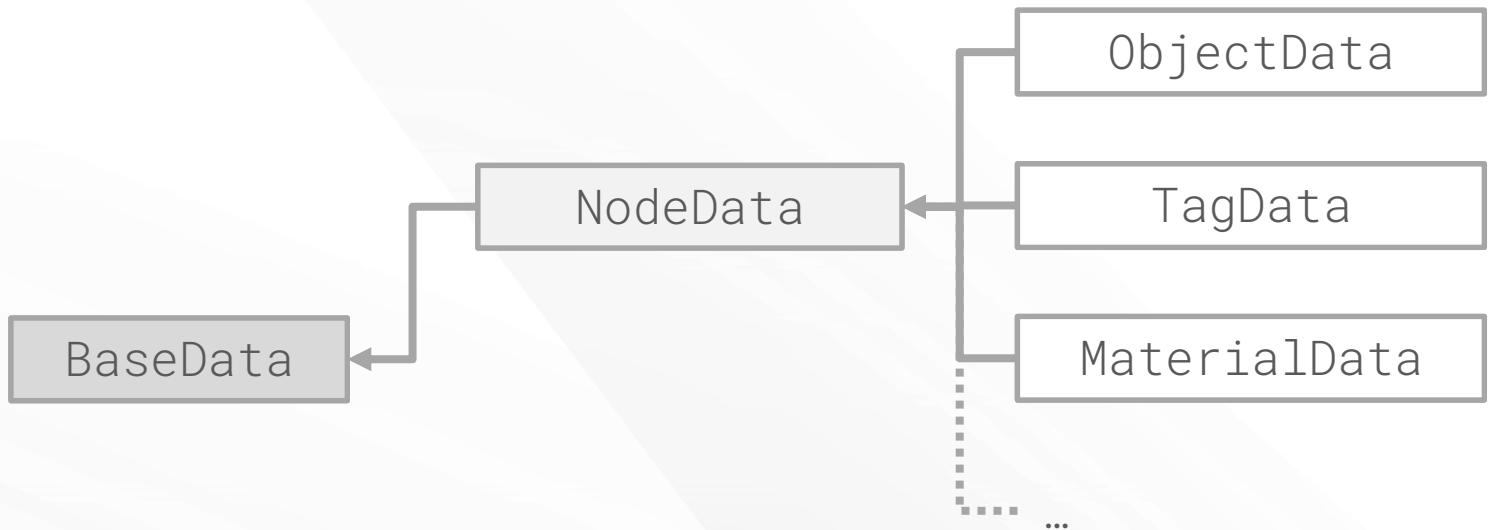


Wichtige Typen in Cinema 4D



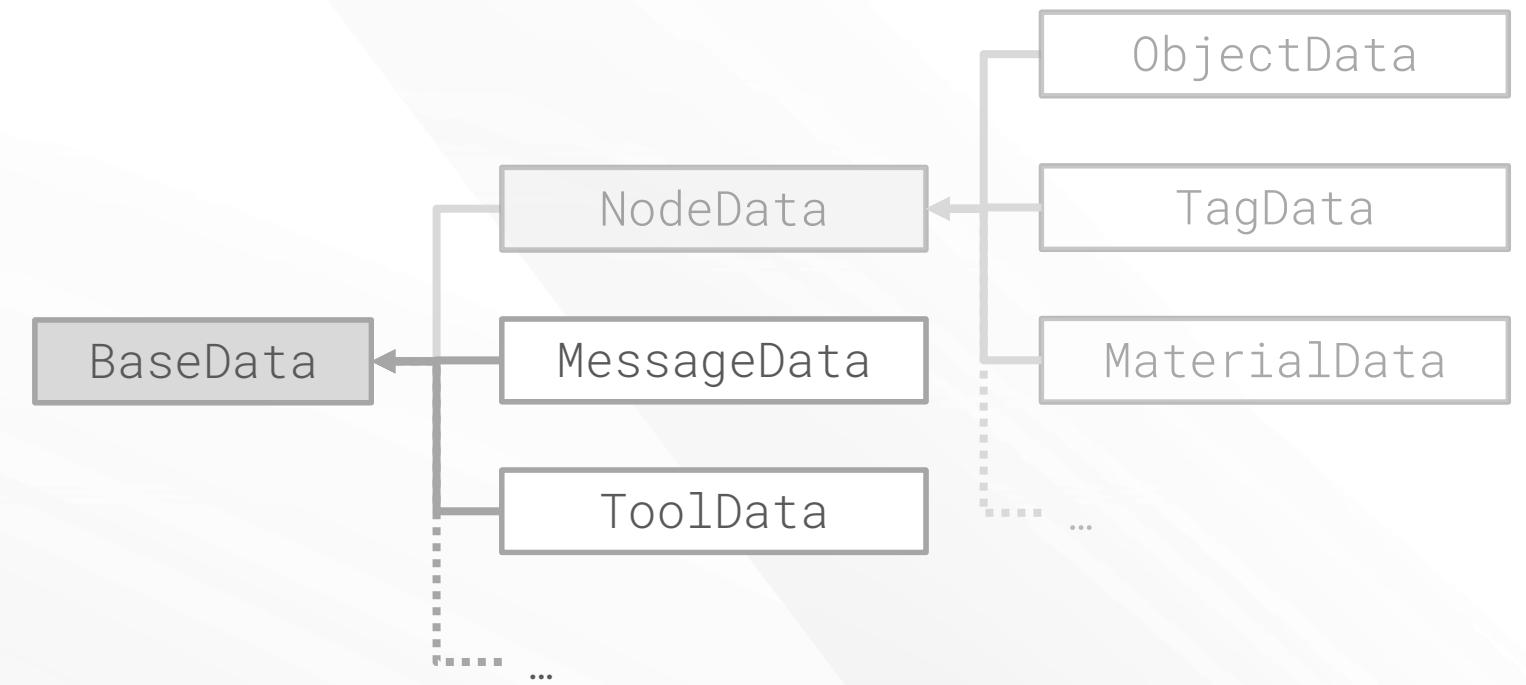
Wichtige Plugintypen in Cinema 4D

- Alle Plugins müssen in Cinema 4D mit einer einzigartigen Plugin ID registriert werden.
- Die meisten Plugintypen sind von NodeData abgeleitet und setzen eine Form von BaseList2D um (ObjectData, TagData, MaterialData, etc.)



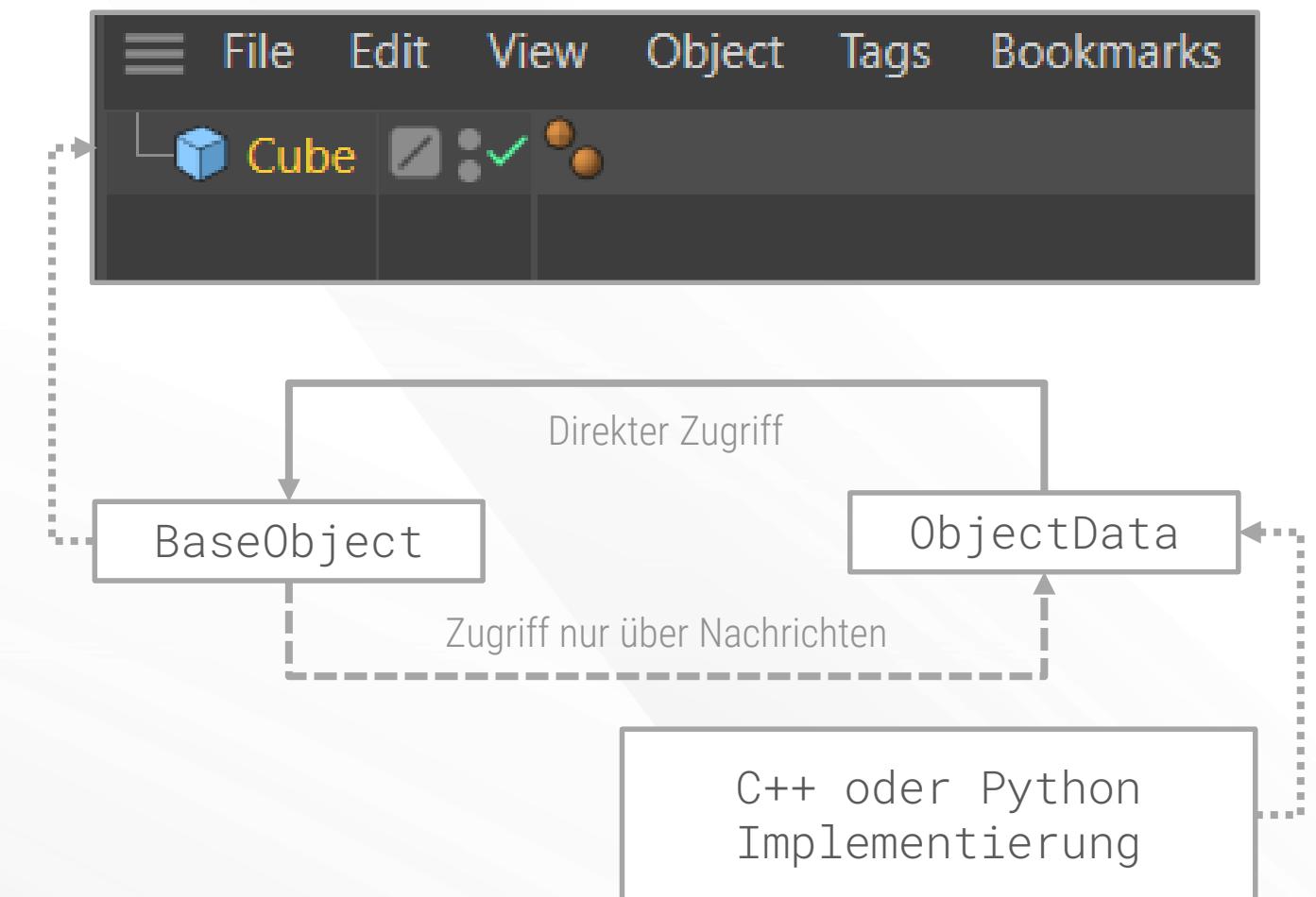
Wichtige Plugintypen in Cinema 4D

- Alle Plugins müssen in Cinema 4D mit einer einzigartigen Plugin ID registriert werden.
- Die meisten Plugintypen sind von NodeData abgeleitet und setzen einen Form von BaseList2D um (ObjectData, TagData, MaterialData, etc.)
- Wichtige nicht-NodeData Plugintypen sind MessageData und ToolData.



Sichtbarkeit von Plugin Typen

- Die Plugin-Implementierungen sind gekapselt in Cinema 4D und Plugins kommen in einem Plugin und UI Layer daher.
 - Pluginseite z.B.: ObjectData, ShaderData, ToolData, etc.
 - UI-Seite z.B.: BaseObject, BaseShader, eine BaseList2D (ToolData), etc.
- Für den Nutzer ist sowohl in der GUI als auch programmatisch nur der UI Layer sichtbar.

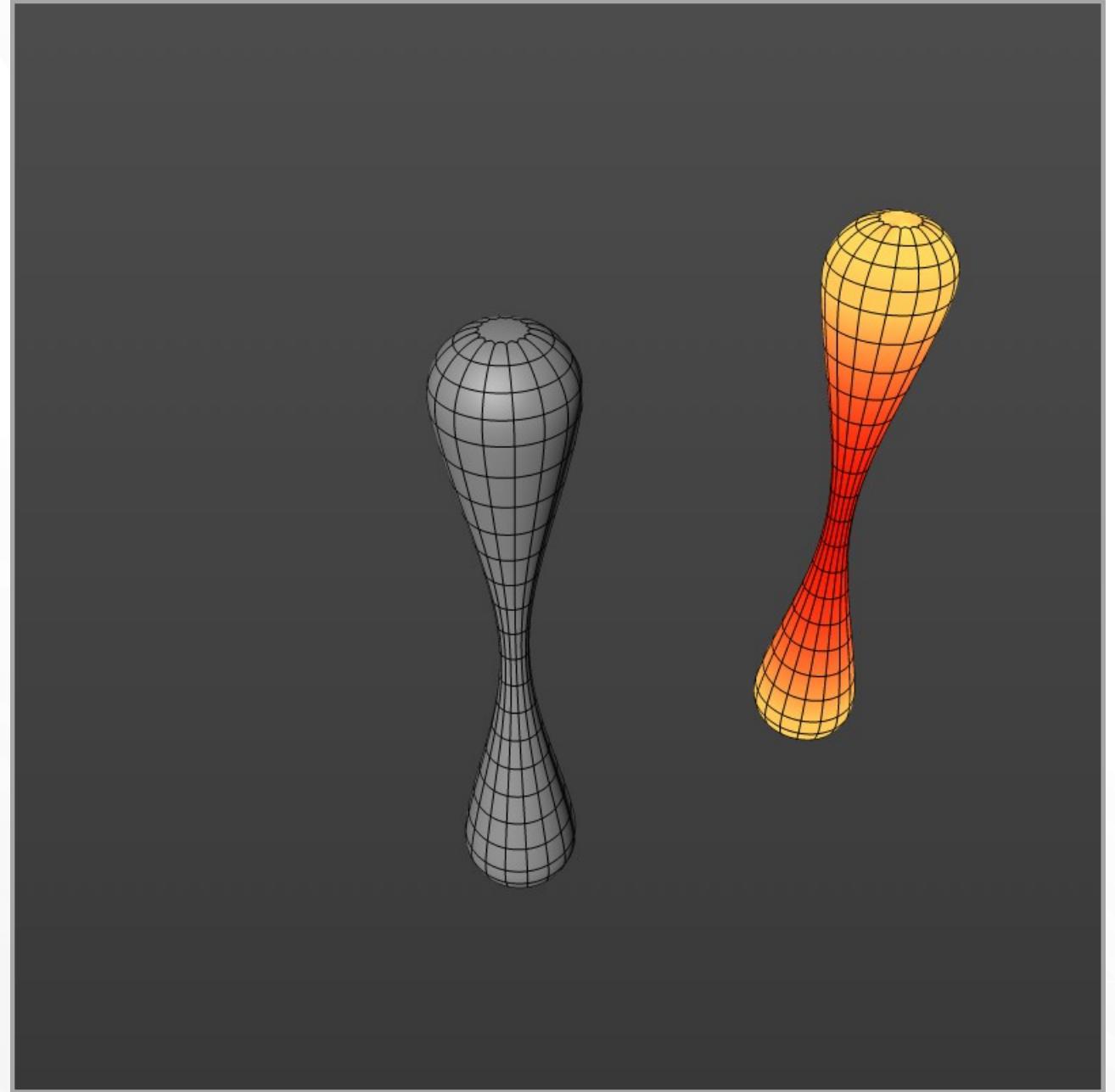


Beispiel

Python Scripting Objekte als Entwicklungshilfe

Problem – Local Thickness

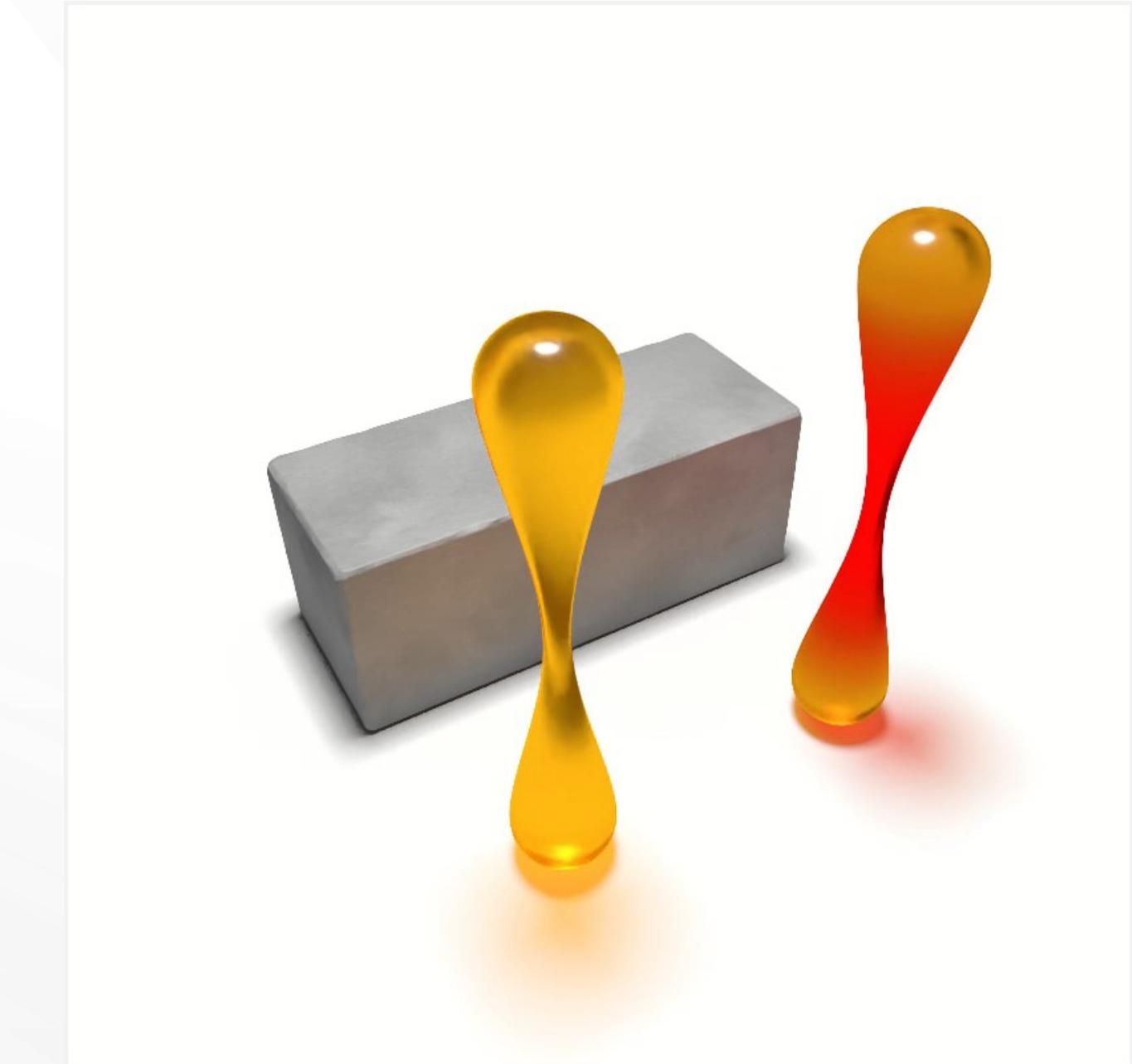
- Ein Problem der algorithmischen Geometrie:
„Wie „dick“ ist eine Körper an einem Punkt?“



Eine Möglichkeit die Dicke des abgebildeten Körpers zu interpretieren. Gelb eingefärbte Bereiche sind von „hoher Dicke“ und rot eingefärbte Bereiche von „geringer Dicke“.

Problem – Local Thickness

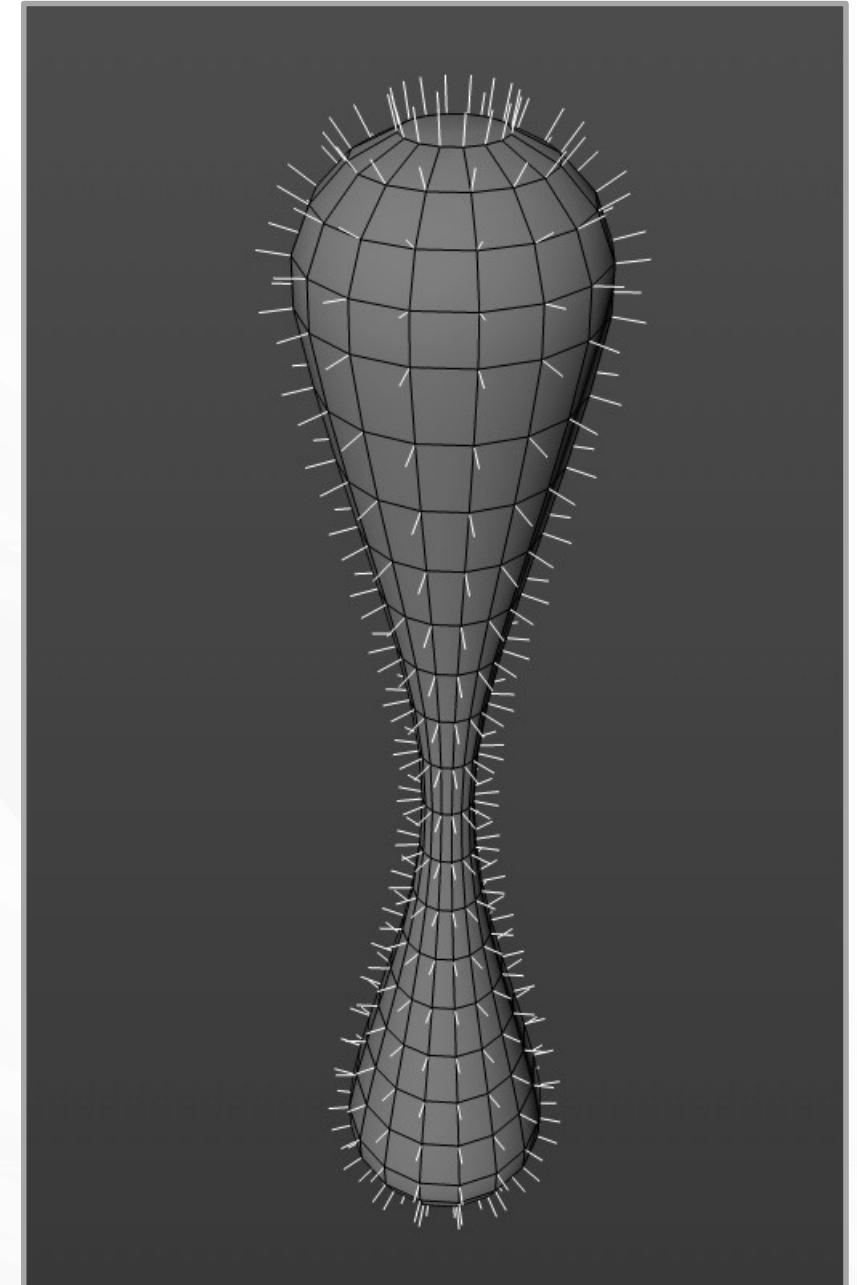
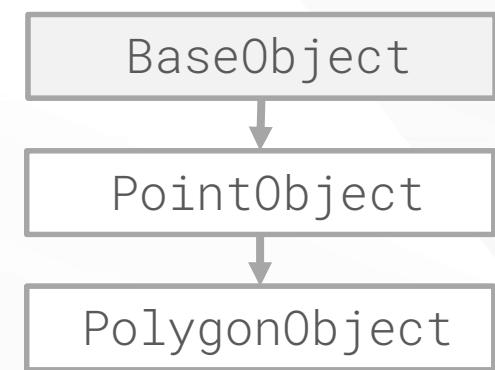
- Ein Problem der algorithmischen Geometrie:
„Wie „dick“ ist eine Körper an einem Punkt?“
- Hat einige direkte praktische Anwendungen:
 - In der *Texturierung*, um beispielsweise Schmutz automatisch an bestimmten Stellen aufzutragen.
 - Im *Rendering*, um beispielsweise Transluzenz vorauszuberechnen.
 - In *Simulationen*, etwa in einer Softbody-Simulation.



Die Dicke des rechten Körpers wird zur Texturierung und Steuerung der Steifigkeit einer Softbody-Simulation einsetzt.

Ein Lösungsansatz – Raycasting

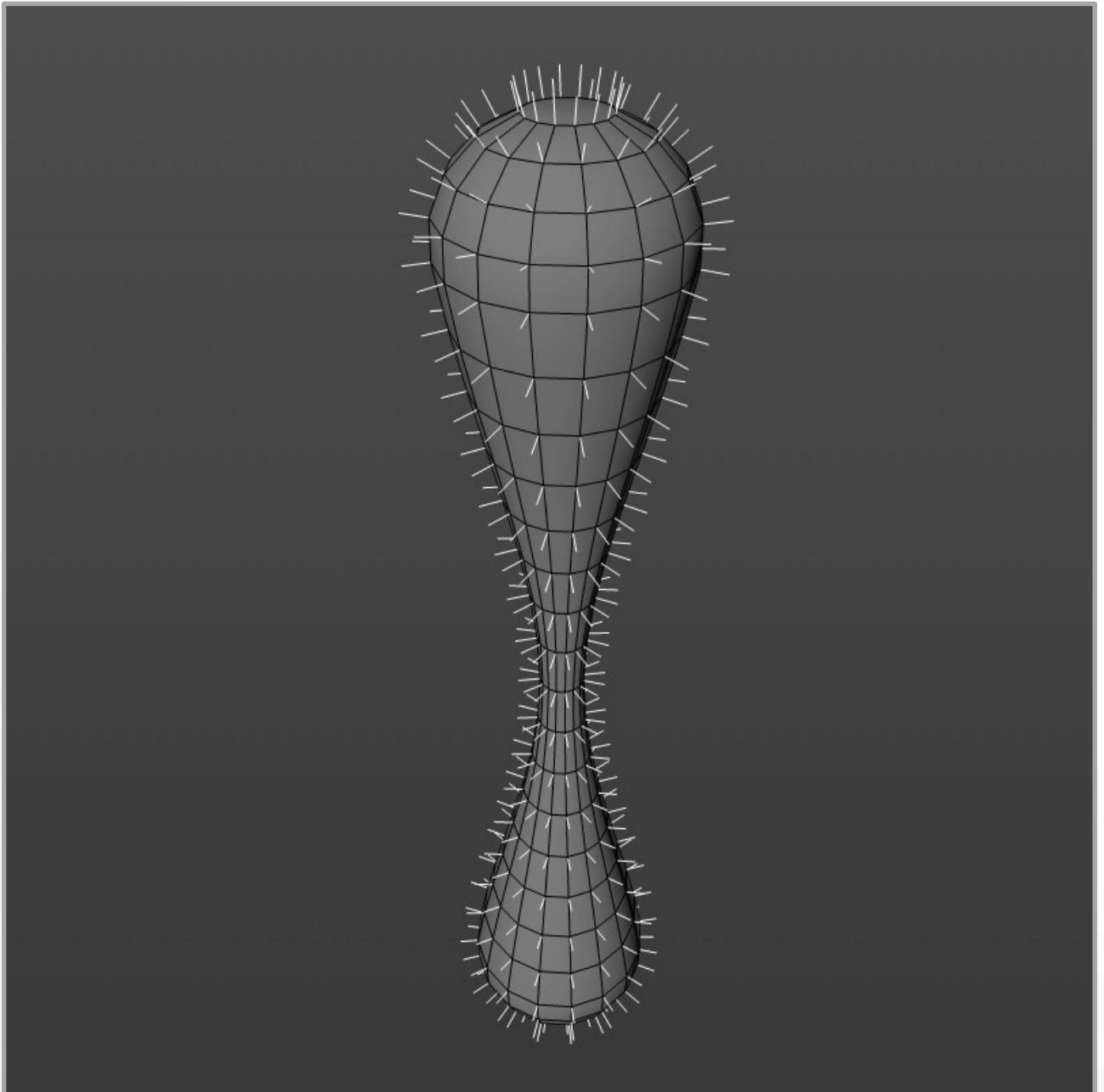
- Es gibt viele Lösungsansätze für das Problem, wir nähern uns hier naiv.
- In Cinema 4D haben wir es nahezu ausschließlich mit *polygonaler*, d.h., diskreter, Geometrie zu tun.
- Die Beispielgeometrie wird über eine endliche Anzahl *vertices* definiert, denen wir jeweils einer Normale zuordnen können.
 - Was die Punkte festlegt, an denen wir die Dicke testen können.



Die Polygone und Vertexnormalen der Beispielgeometrie.

Ein Lösungsansatz – Raycasting

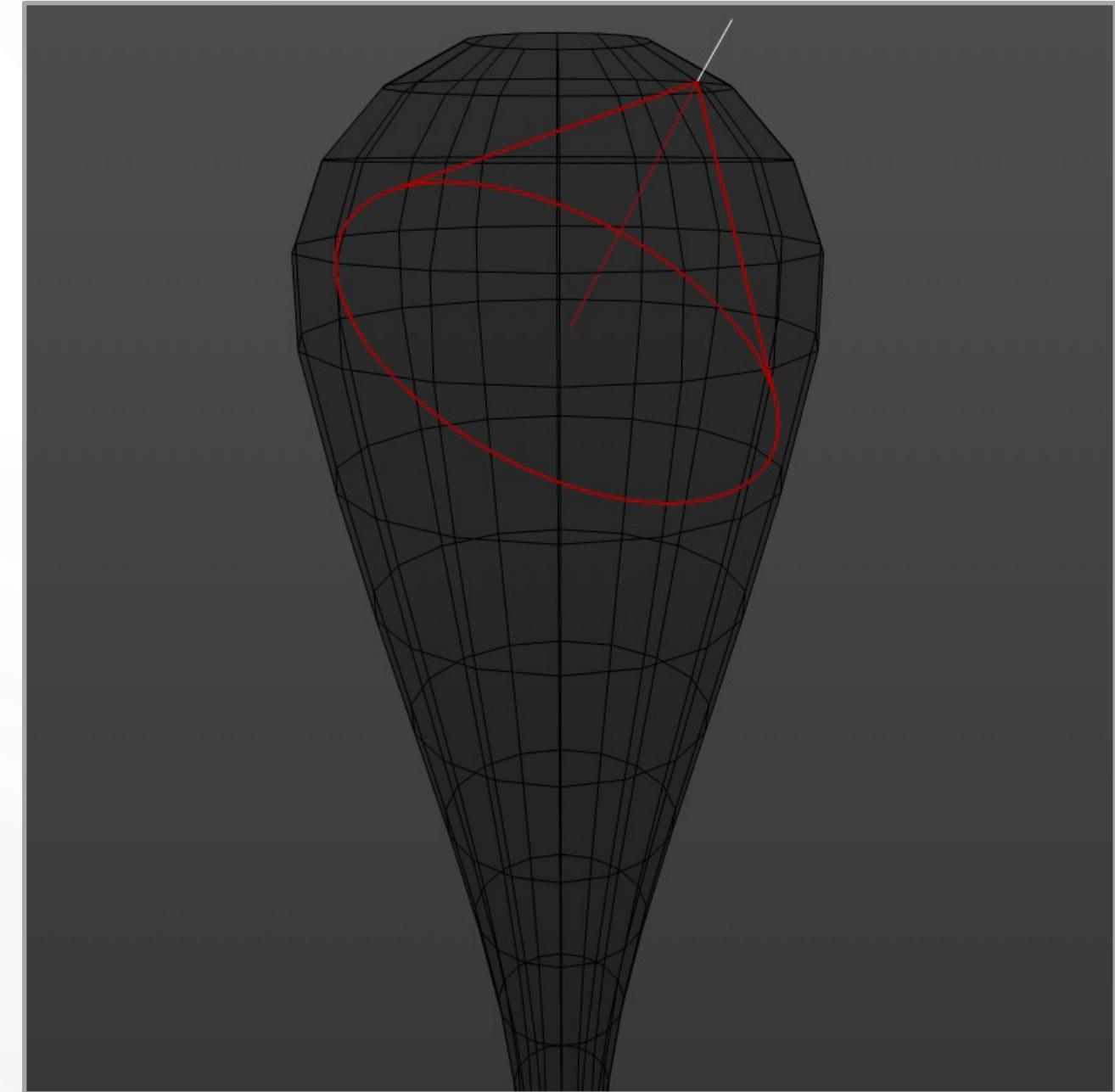
- Für jeden Vertex ro und seine Normale n des Körpers K konstruieren wir einen Raumwinkel Ω entlang $-n$.



Der Blick auf einen einzelnen Vertex.

Ein Lösungsansatz – Raycasting

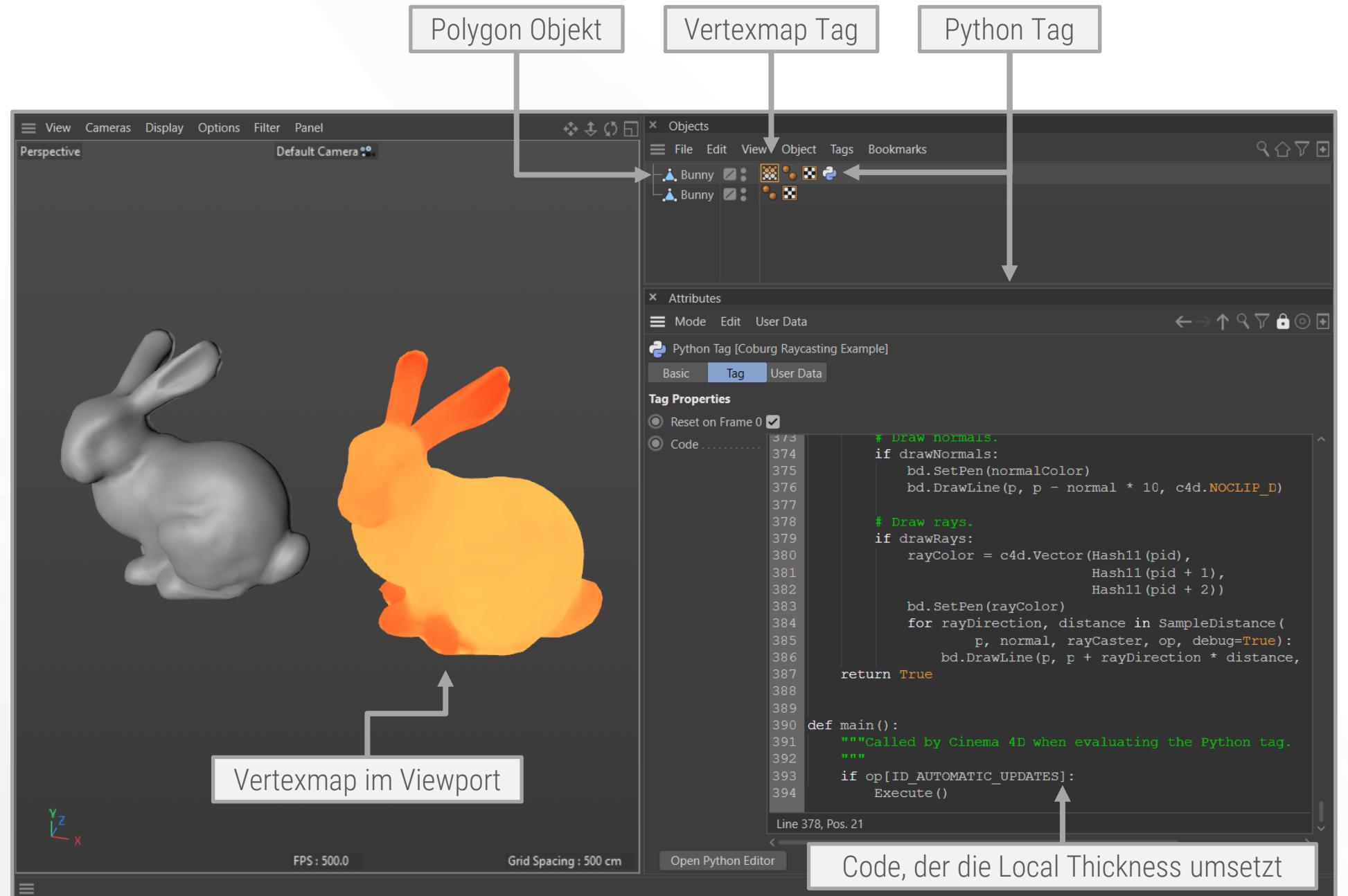
- Für jeden Vertex ro und seine Normale n des Körpers K konstruieren wir einen Raumwinkel Ω entlang $-n$.
- Innerhalb von Ω konstruieren wir cnt Vektoren rd .
- Für jedes Tupel (ro, rd) führen wir einen *ray-triangle* Überschneidungstest für jedes Dreieck T in G aus, um so die Distanz zwischen jedem Treffer h und ro in einer Liste $samples$ zu sammeln.
 - In Cinema 4D können wir GeRayCollider für diese Überschneidungstest verwenden.
- Mittels der Funktion $f(samples)$ errechnen wir die finale geschätzte Dicke des Körpers an dem Vertex ro .



Wir raycasten in einem Raumwinkel ausgerichtet an dem Invers einer Vertexnormale.

Umsetzung – Python Tag

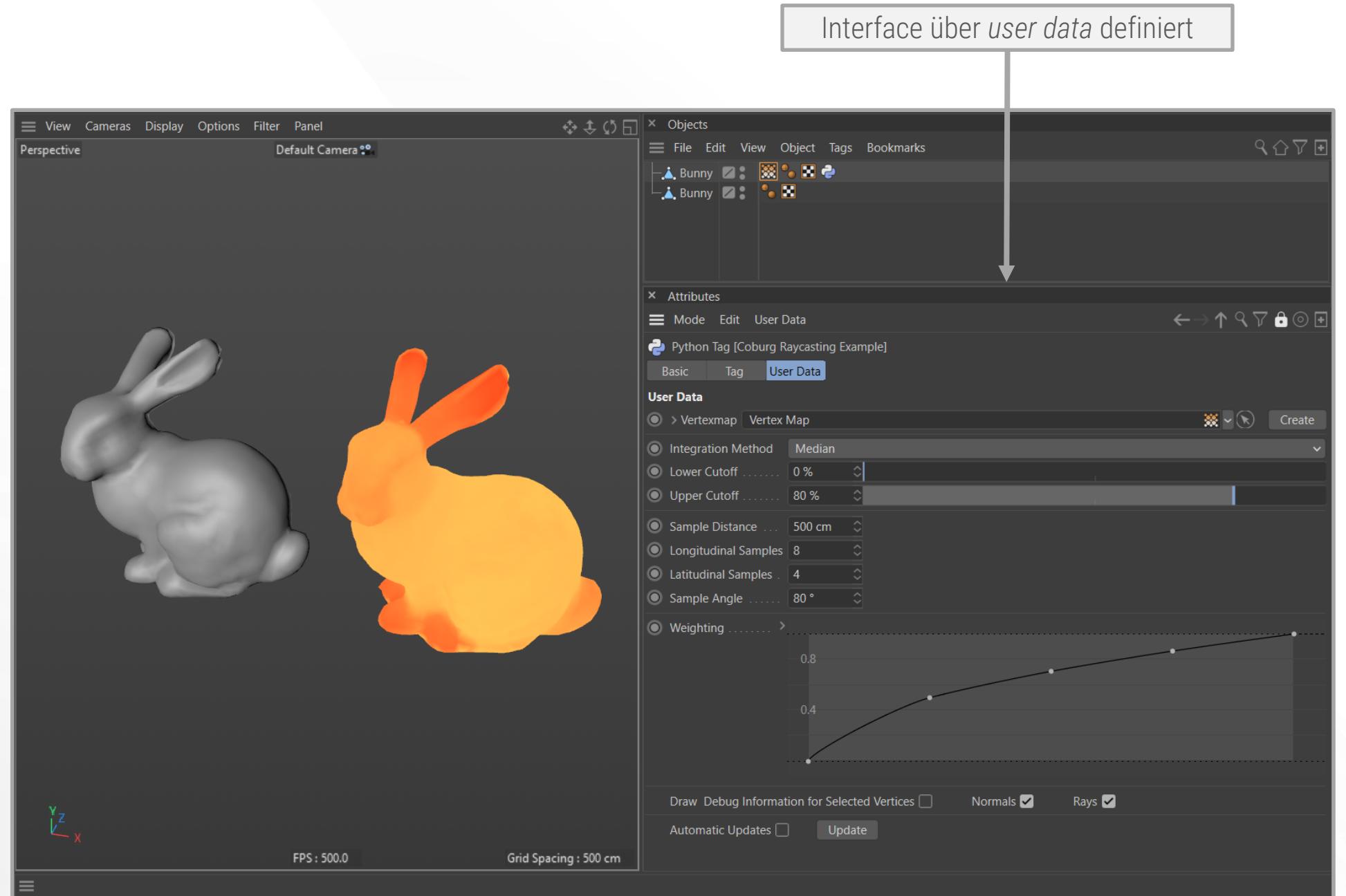
- Zur Umsetzung verwenden wir einen *Python Tag* und *Vertexmaps*.



Ein Python Tag setzt hier die Berechnung der Local Thickness des Objektes um, dem er anhängig ist. Die Daten werden in eine Vertexmap geschrieben.

Umsetzung

- Zur Umsetzung verwenden wir einen *Python Tag* und *Vertexmaps*.
- Der Vorteil von Python Scripting Objekten ist,
 - dass wir keine Plugin ID benötigen und
 - *user data* Parameterdefinitionen für uns vereinfachen.

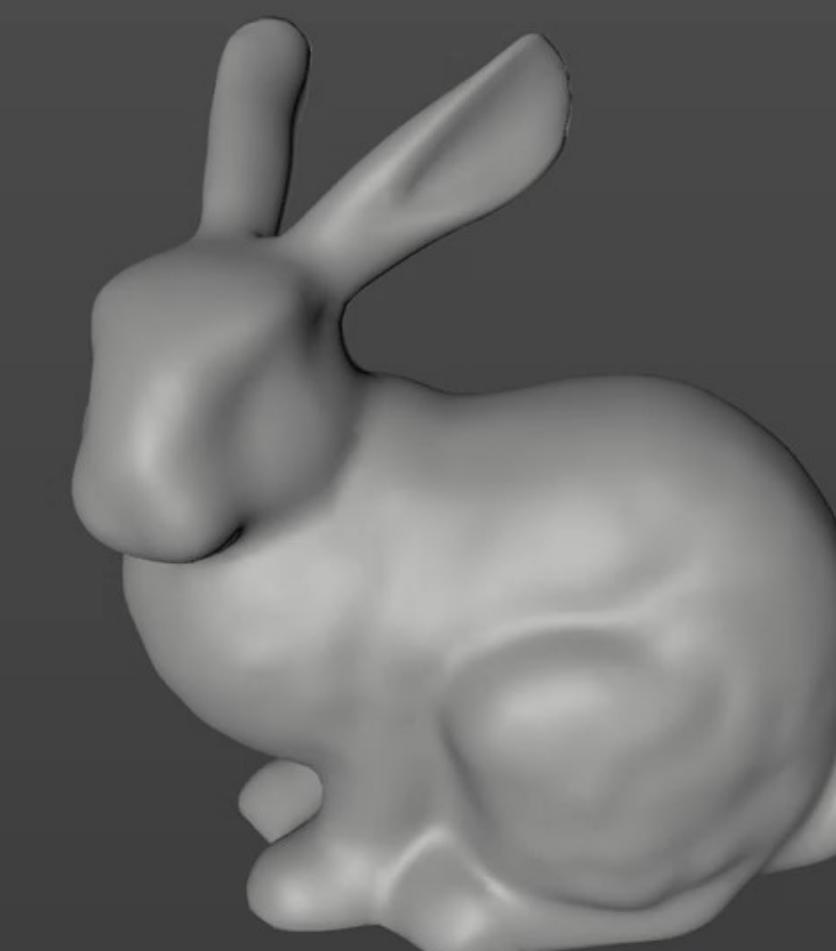


Python Scripting Objekte können GUIs haben, die den GUIs von Plugins in nur wenig nachstehen.



View Cameras Display Options Filter Panel

Perspective Default Camera

Y
Z
X

FPS : 333.3

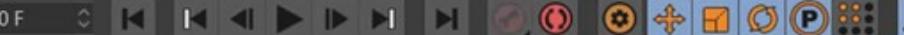
Grid Spacing : 50 cm

0 5 10 15 20 25 30 35 36 40 45 50 55 60 65 70 75 80 85 90 36 F

0 F

90 F

150 F



File Edit View Object Tags Bookmarks

SDS

Bunny

Mode Edit User Data

Python Tag [Coburg Raycasting Example]

Basic Tag User Data

User Data

Vertexmap Vertex Map

Integration Method Harmonic Mean

Lower Cutoff 0 %

Upper Cutoff 80 %

Sample Distance ... 500 cm

Longitudinal Samples 2

Latitudinal Samples 2

Sample Angle 80 °

Weighting

Draw Debug Information for Selected Vertices Normals Rays

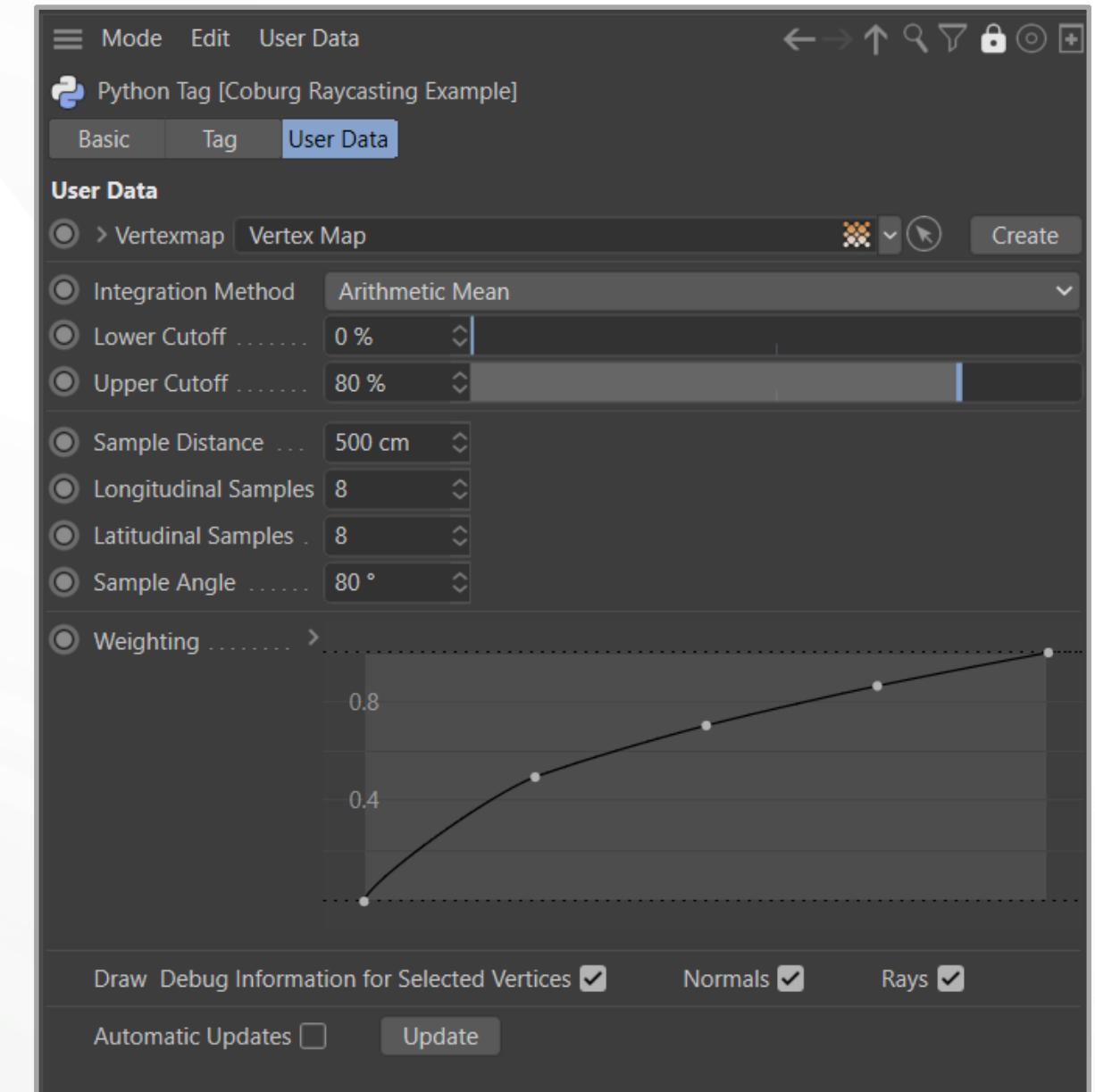
Automatic Updates Update

A graph editor window showing a curve that starts at approximately (0, 0.2) and ends at approximately (1, 0.8), representing a weighting function for ray casting.

Takes Content Browser Attributes Layers Structure

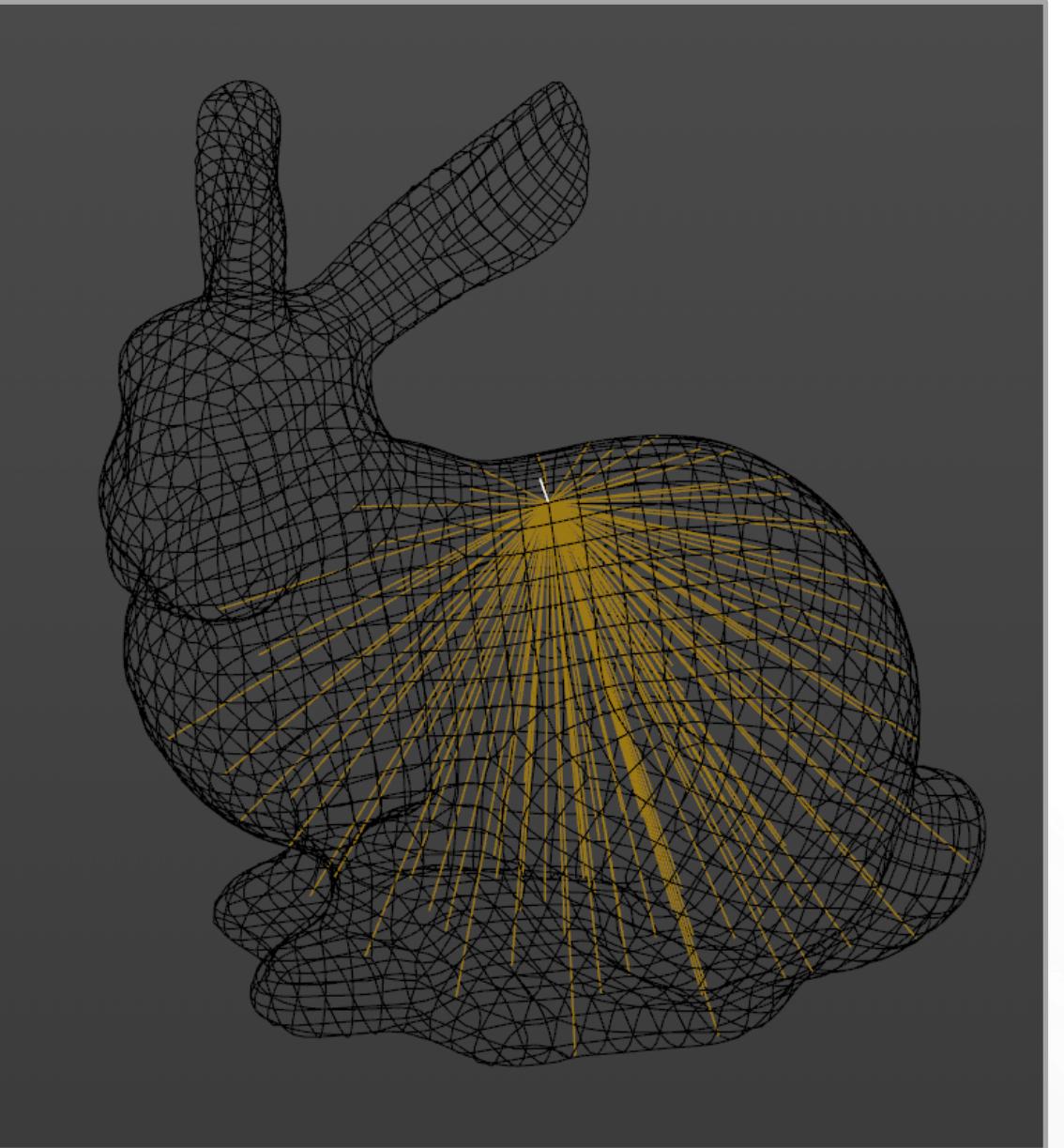
Umsetzung – Python Tag

- `main()`
 - Gegenstück zu `TagData.Execute()`
 - Muss implementiert werden und wird von Cinema aufgerufen, um den Tag auszuführen.
 - Führt in unserem Beispiel nur Code aus, wenn „Automatic Updates“ aktiviert ist.
- `message()`
 - Gegenstück zu `NodeData.Message()`
 - Die Umsetzung ist optional und wird von Cinema aufgerufen, um Nachrichten an den Tag weiterzuleiten.
 - Im Beispiel verwenden wir `message()`, um auf Interfaceingaben zu reagieren.



Umsetzung – Python Tag

- `draw()`
 - Gegenstück zu `TagData.Draw()`
 - Die Umsetzung ist optional und wird von Cinema aufgerufen, um den Tag Informationen in einen Viewport zeichnen zu lassen.
 - Im Beispiel verwenden wir `draw()`, um die Debug Informationen anzuzeigen.



Schlussfolgerungen

- Wir können hier nicht auf alle Details eingehen ...
- Auf GitHub könnt Ihr den narrativen und kompakten Code und die Szenen für die Präsentation finden.
- Scripting Objekte eignen sich gut,
 - um sich mit der Cinema 4D API vertraut zu machen,
 - um schnell einen Prototypen zu entwickeln,
 - oder um Ideen zu visualisieren.

```
def SampleDistance(p, normal, rayCaster, tag, debug=False):
    """Computes the approximated local thickness for a vertex and its mesh.

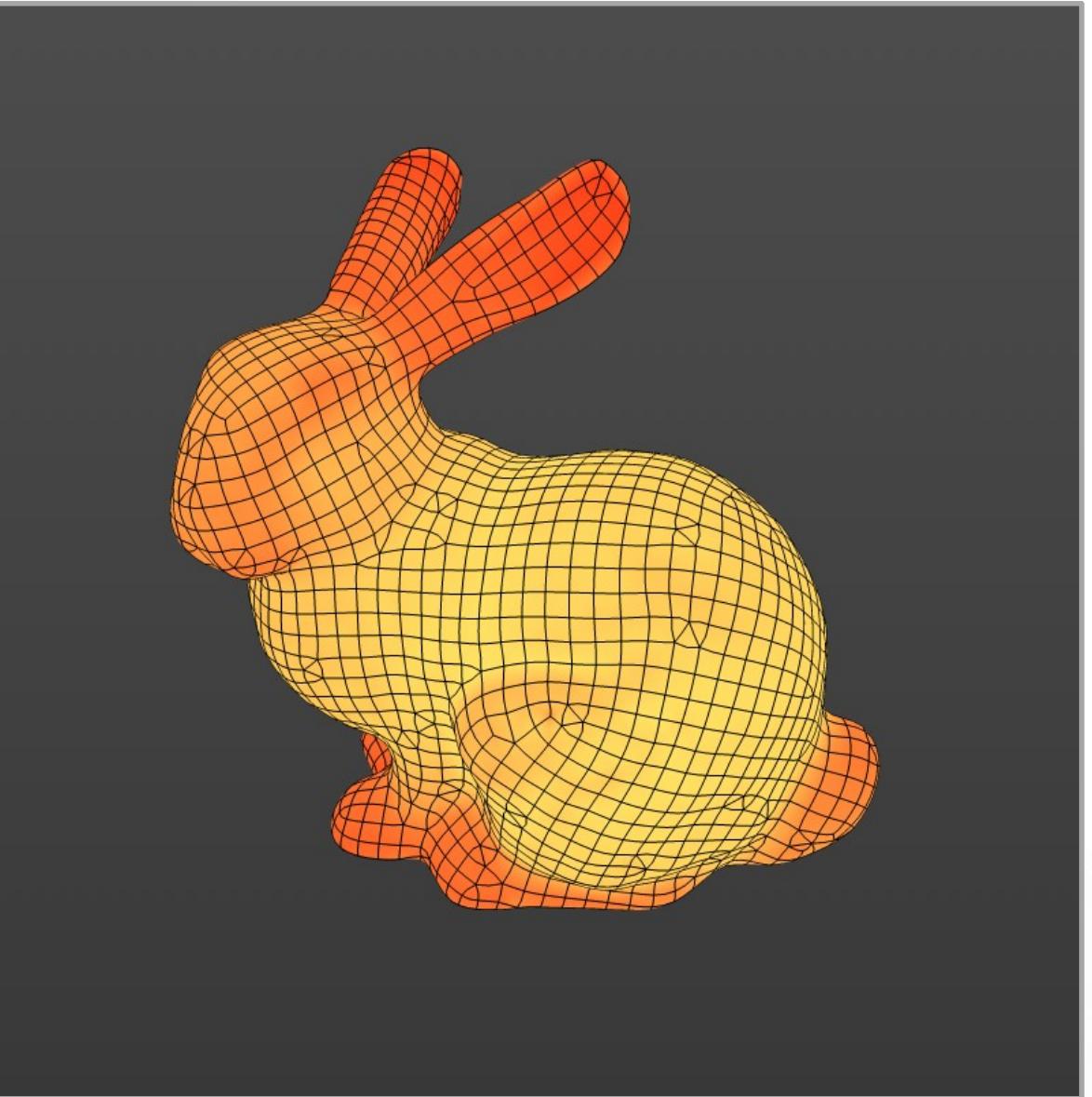
    Args:
        p (c4d.Vector): The point to for the vertex to sample.
        normal (c4d.Vector): The normal to for the vertex to sample.
        rayCaster (c4d.utils.GeRayCollider): A ray casting object.
        tag (c4d.BaseTag): The Python scripting tag holding the user data.
        debug (bool, optional): If True, the function won't normalize the
            distances and also will return the ray direction vectors. Used by
            the debug drawing functionality of draw().

    Returns:
        float or tuple[c4d.Vector, float]: The computed distances for p if
            debug is False. Otherwise the rays and normalized distances.
    """
    result = []
    sampleDistance = tag[ID_SAMPLES_DISTANCE]
    # For each ray direction within the solid angle aligned with the normal:
    for rayDirection in NormalsInSolidAngle(normal,
                                              theta=tag[ID_SAMPLES_ANGLE],
                                              samples=(tag[ID_SAMPLES_LONG],
                                                       tag[ID_SAMPLES_LAT])):
        # The ray origin for the ray casting operation. We nudge the ray
        # origin a little bit inwards to make it easier for us.
        rayOrigin = p + rayDirection * .05
        # We carry out the raycasting, GeRayCollider does all the heavy
        # lifting for us, we do not have to deal with the individual
        # triangles of our mesh.
        does_intersect = rayCaster.Intersect(rayOrigin, rayDirection,
                                              sampleDistance)
        dist = sampleDistance
        # We found no intersections, we simply assume that there is an
        # intersection beyond our maximum ray intersection distance and just
        # append that to our results.
        if not does_intersect:
            result.append((rayDirection, dist) if debug else dist)
            continue
        # We found intersections and now go through all of them to find the
        # shortest intersection distance that occurred.
        for index in range(rayCaster.GetIntersectionCount()):
            rayData = rayCaster.GetIntersection(index)
            if (rayData["backface"] == True and
                rayData["distance"] < dist):
                dist = rayData["distance"]
        result.append((rayDirection, dist) if debug else dist)
    return result
```

Beispiel für den narrativem Code der `SampleDistance()` Funktion, die die Kernlogik umsetzt.

Schlussfolgerungen

- Es gibt aber natürlich gewisse Grenzen besonders hinsichtlich der Performance, bei denen ein Python oder ein C++ Plugin sinnvoller sind.
- Ich hoffe, ich habe Euch ein wenig neugierig gemacht, auf das, was Ihr selbst mit Cinema 4D und seinen APIs umsetzen könnt.
- Wir würden uns über eine Besuch von Euch in unserem Entwicklerforum, dem Plugin Café, freuen.



Ressourcen

Ressourcen

Funktion	Adresse	Details
Development Ressourcen	developers.maxon.net	<ul style="list-style-type: none"> ▪ Links zu allen wichtigen Ressourcen ▪ C++ API, Python API und Cineware Dokumentation ▪ Development Blog-Posts & News
Support Forum	plugincafe.maxon.net	<ul style="list-style-type: none"> ▪ Support für C++ und Python Plugin-Entwicklung ▪ Registrierung von Plugin IDs
SDK Code Beispiele	github.com/PluginCafe	<ul style="list-style-type: none"> ▪ Beispiele für C++ und Python Plugins, Python Skripte und Scripting Objekte
Bildungslizenzen	maxon.net/de/educational-licenses	<ul style="list-style-type: none"> ▪ Bildungslizenzen für Maxon Produkte
Präsentationsdaten	github.com/PluginCafe/praxiswochen	<ul style="list-style-type: none"> ▪ Slides, Beispielszenen und Code für diese Präsentation

Q&A

Präsentationen & Namen

Maxon Einführung

Jonas

Maxon Research

Bernd

Programmierern für Programmierer

Fritz & Maik

Cinema 4D SDK Einführung

Ferdinand

