DevKitchen 2018
Cinema 4D R20 - Features API
Cinema 4D
R20
Features
API

Volumes
Fields
Multi-Instances
Volumes in Cinema 4D R20

The volume system is based on the OpenVDB library.

• Volume Loader: Loads data from vdb files.
• Volume Builder: Creates volume data from scene elements.
• Volume Mesher: Creates a polygon mesh from volume data.
• Volume Object: Stores volume data.
Volumes in Cinema 4D R20

Classic API components in `cinema.framework`:

- `lib_volumeobject.h`: Contains the `VolumeObject` class.
- `lib_volumebuilder.h`: Contains the `VolumeBuilder` class.
Volumes in Cinema 4D R20

MAXON API components defined in `volume.framework`:

- `maxon::VolumeInterface`: Represents volume data.
- `maxon::CommandClasses`: Contain volume commands.
- `maxon::VolumeToolsInterface`: Contains functions to handle volumes.
Creating a VolumeBuilder object

The VolumeBuilder class is defined in **lib_volumebuilder.h**.

- Allocate a new instance with `VolumeBuilder::Alloc();`
- Insert it into the `BaseDocument` as usual.
- Use `VolumeBuilder::AddSceneObject()` to add objects to the list.
Reading Volume Data

VolumeObject is defined in lib_volumeobject.h. VolumeInterface is defined in volume.h.

• Check for object type with Ovolume.
• Cast into VolumeObject.
• Access volume data with VolumeObject::GetVolume().
• Create a GridIteratorRef
• Initialise iterator with GridIteratorInterface::Init() and the volume.
Reading Volume Data

- Iterate the grid with IsNotAtEnd() and StepNext().
- Access coordinates with GetCoords().
- Access value with GetValue().
- To create the world space position use the matrix provided with VolumeInterface::GetGridTransform().
Writing Volume Data

VolumeObject is defined in `lib_volumeobject.h`. VolumeInterface is defined in `volume.h`.

- Create a volume with `VolumeObject::Alloc()`.
- insert it into the BaseDocument as usual.
- Create an empty volume with `VolumeToolsInterface::CreateNewFloat32Volume()`.
- Create a GridAccessorRef.
Writing Volume Data

• Initialise the accessor with the empty volume using GridAccessorRef::Init().
• Write into the volume using GridAccessorRef::SetValue().
• Store the volume in the VolumeObject using VolumeObject::SetVolume().
Using Volume Commands

VOLUMEDATA, VolumeCommandData and CommandClasses are defined in \texttt{volumecommands.h}

- Create a \texttt{LegacyCommandDataRef} context from \texttt{VOLUMEDATA}.
- Create a \texttt{VolumeCommandData} object.
- Set the data and store it in the context using \texttt{SetLegacyData()}.
- Access a command from \texttt{CommandClasses}.
- Invoke that command on the given context using \texttt{Invoke()}. 
Using Volume Commands

• Get the result data from the context using `GetLegacyData()`.
• Get the results from the `VolumeCommandData` structure.
Volume API Documentation

- VolumeObject Manual
- VolumeBuilder Manual
- VolumeInterface Manual
- Volume Tools Manual
Volume API in Python

Defined in `c4d.modules.volume`.

- `VolumeObject` and `VolumeBuilder` available.
- Commands used with `SendVolumeCommand()`.
Volume API in Python

Defined in `maxon.frameworks.volume`.

- VolumeInterface
- GridAccessorInterface
- VolumeToolsInterface.
Fields
Fields System

The Field system replaces the previous “Falloff” system. It allows to sample space to get a scalar value and a colour value. Fields are used in:

- MoGraph Effectors
- Deformers
- Volumes
- Particles
- .....

DevKitchen 2018 – Cinema 4D R20 Features - Fields
Fields System

The system is made of these components:

- Field Objects: Objects in space that define a Field.
- Field List: A list of Field layers.
- Field Layer: Either references a Field object or defines a global function.
The API is defined in `cinema.framework`:

- `FieldObject`: Represents a Field object in the scene.
- `FieldList/FieldLayer`: The “Field” parameter type.
- `FieldData`: Base class for custom Field objects.
- `FieldLayerData`: Base class for custom Field layers.
Sampling a FieldObject

FieldObject is defined in c4d_fielddata.h.

• Check the object type with Ofield.
• Cast into FieldObject.
• Prepare position values to sample at.
• Store information on the samples in a FieldInput object.
• Prepare result data in a FieldOutput object.
• Create a FieldInfo context object that references the FieldInput object.
Sampling a FieldObject

- Prepare sampling with `InitSampling()`.
- Sample the `FieldObject` with `Sample()`.
- Free data with `FreeSampling()`.
- Sampling results are now stored in the `FieldOutput` / `FieldOutputBlock` structure.
Sampling a FieldList

FieldList is defined in `customgui_field.h`
FIELDS is defined in `ofalloff_panel.h`

- Access the parameter FIELDS.
- Get the custom data type with `GetCustomDataType()`.
- Data type ID is `CUSTOMDATATYPE_FIELDLIST`.
- Cast into a FieldList object.
Sampling a FieldList

- Prepare points to sample.
- Store data in FieldInput object.
- Sample the list with SampleListSimple().
- The result is stored in the returned FieldOutput object.
Implementing a Custom FieldObject

FieldData is defined in `c4d_fieldplugin.h`.

- Implement `InitSampling()` to store data used while sampling.
- Implement `FreeSampling()` to free data used while sampling.
- Implement `Sample()` to define the object’s behaviour.
Implementing a custom FieldObject

- Transformation information is stored in `info._inputData._transform` and the object’s world space matrix (`GetMg()`).
- The input positions are stored in `inputs._positions`.
- The result values are written to `outputs._value`.
- The plugin is registered with `RegisterFieldPlugin()`.
Implementing a custom FieldLayer

FieldLayerData is defined in `c4d_fieldplugin.h`.

- Implement `InitSampling()` to store data used while sampling.
- Implement `FreeSampling()` to free data used while sampling.
- Implement `Sample()` to define the layer’s behaviour.
Implementing a custom FieldLayer

- The input positions are stored in `inputs._positions`.
- The result values are written to `outputs._value`.
- The plugin is registered with `RegisterFieldLayerPlugin()`.
- Set `FIELDLAYER_DIRECT` for layers that modify the existing values (modifier layer).
Fields API Documentation

- FieldList Manual
- FieldLayerData Manual
- FieldObject Manual
- FieldData Manual
Fields API in Python

Defined in `c4d.modules.mograph`.

- `FieldObject`
- `FieldList`
Python in Fields

Python can be used to program these components:

- Python Field
- Python Field Layer
Multi-Instances
Multi-Instances

Multi-Instances are an extension of the existing Instance object.

• An object can be used as a position source (particles, MoGraph Matrix).
• Position data can be written into the object using the API.
Multi-Instances

The InstanceObject class is defined in `cinema.framework`.

- `lib_instanceobject.h` contains the class.
- `oinstance.h` contains parameter IDs.
Creating an InstanceObject

InstanceObject is defined in `lib_instanceobject.h`.

- Create a new instance with `InstanceObject::Alloc()`.
- Insert the instance into the `BaseDocument`.
- Set the referenced object using `SetReferenceObject()`.
- Set the parameter `INSTANCEOBJECT_RENDERINSTANCE_MODE`. 

Creating an InstanceObject

- If using multi-instances, prepare position and colour data.
- Set the positions using `SetInstanceMatrices()`.
- Set the colours using `SetInstanceColors()`.
Reading Position Data from an InstanceObject

• Check the object type with Oinstance.
• Cast into InstanceObject.
• Get the number of instances with GetInstanceCount().
• Get the matrix for a given instance index with GetInstanceMatrix().
InstanceObject API Documentation

• InstanceObject Manual
InstanceObject API in Python

Defined in the \texttt{c4d} namespace.

\begin{itemize}
\item InstanceObject
\end{itemize}
3D FOR THE
REAL WORLD.

MAXON Computer GmbH
Max-Planck-Straße 20
D-61381 Friedrichsdorf
Germany

E info@maxon.de
T +49 (0) 6172 5906-0
F +49 (0) 6172 5906-30

maxon.net