

CINEMA 4D R11.5 SDK

- Introduction
- Double Precision
- Render Instances
- Deprecated Functions
- Bucket Rendering
- Videopost Effects
- Memory Management
- Threading

- RayHitID
- RenderDocument
- UVWTag
- MoData/MDArray
- MovieLoader
- Parser/ParserCache

Introduction

CINEMA 4D R11.5 SDK no longer backwards compatible for:

- Channel Shaders
- Materials
- Videopost Effects

- Deprecated functions (functions with Ex) need to be replaced

- Due to the change of several data structures a hybrid SDK is no longer possible

- [GetEnabling/GetDEnabling](#) parameter [t_data](#) is const

Double Precision

- Double precision throughout the whole render API to avoid several reoccurring problems.
- All geometric points and vectors data should be held in double precision (`LReal`, `LVector`, `Lmatrix`).
- Double precision is not necessary for uncritical parts like color calculation.
- `Vector` to `Lvector` conversion with `LV()`
- `Lvector` to `Vector` conversion with `SV()`
- `Matrix` to `LMatrix` conversion with `LM()`
- `Lmatrix` to `Matrix` conversion with `SM()`
- All ray and intersection calculations must be done in double precision.

Render Instances

- A Render Instances is a 'regular' [RayObject](#) which points to the same data as the instanced [RayObject](#). That means they share [paddr](#) and [vadr](#) arrays etc.
- Only textures and materials can be different.
- Only geometry can be instanced (no lights etc.)
- [RayObjects::instance](#) != NULL for Render Instances
- [RayObjectInstanceData](#) holds all necessary data for Render Instances

```
struct RayObjectInstanceData
{
    LONG          instance_of_index;
    Lmatrix       transform;
    Lmatrix       transform_inverse;
    Lmatrix       transform_tensor;
    BaseObject    *link;
};
```

- [instance_of_index](#) is the [RayObject](#) index of the original object
- [transform](#) is the matrix to transform world space points from the original object to the instanced object (still world space)
- [transform_inverse](#) is the matrix to transform world space points from the instanced object to the original object (still world space)
- [transform_tensor](#) allows you to transform normals

Transformation from original object's point to instanced object:

```
new_point = old_point * instanced_rayobject->instance->transform;
```

Transformation from instanced object's point to original object:

```
new_point = old_point * instanced_rayobject->instance->transform_inverse;
```

Transformation of normal from original object to instanced object:

```
new_normal = !(old_normal^instanced_rayobject->instance->transform_tensor);
```

- [link](#) stores the Instance object (of type [Oinstance](#)) that this instanced [RayObject](#) belongs to.

Deprecated Functions

- Most 'Ex' routines have been replaced by a newer one which does the same.
- Use the new routines wherever possible!

Bucket Rendering

- CINEMA 4D R11.5 renders images in buckets.
This should have little to no effect on your source code.

Videopost Effects

- Videopost effects which use the structure `PixelPost` need to be aware that instead of full image lines they are now presented with lines of a bucket. Make sure you take the members `xmin` and `xmax` into account.
- The `MSG_VIDEOPOST_CREATERAY` message has been replaced by the `CreateRay` method of `VideoPostData`.

Memory Managment

- In the past the CINEMA 4D SDK overloaded the operators `new` and `delete`. This is no longer the case. `new` and `delete` will allocate and free memory from The operating system, to allow easier linking of DLLs and DYLIBs.
- CINEMA 4D data structures where control is passed on to CINEMA 4D (so that you don't free it yourself anymore) must be allocated with `gNew` and `gDelete`, or `bNew` and `bDelete` for arrays.
- In general, we recommend to make use of `gNew/gDelete` wherever possible.

Threading

- CINEMA 4D R11.5 can run up to 64 render threads simultaneously.
This should have little to no effect on your source code.
- [AutoLock](#)/[AutoRWLock](#) classes for scope based thread-safe data access.
Example of a thread-safe class on the next side:

```

class MyClass
{
    private:
        //Make the values you want threadsafe private just for safety, if you attempt to
        //read/write "value" directly then the compiler wont let you.
        //If you could the value wouldn't be threadsafe as you could access it from an
        //outside thread while it was being read/written in another!

        GE_SPINLOCK lock;
        GE_AUTOLOCK alock;

        LONG value;

    public:
        MyClass(void) { lock = NULL; alock = NULL; }
        ~MyClass(void) { }

        //To access in a threadsafe way use "Getter" and "Setter" functions :

        LONG GetValue(void)
        {
            AutoLock al(&lock, &alock);

            return value;
        }

        void SetValue(LONG v)
        {
            AutoLock al(&lock, &alock);

            value = v;
        }

        Bool RecurseDoSomething(LONG number_of_recurions)
        {
            AutoLock al(&lock, &alock);

            if (!number_of_recurions) return TRUE;

            SetValue(GetValue() * number_of_recurions); //No threadlocks here!

            if (value > 1000) return FALSE;

            return RecurseDoSomething(number_of_recurions -1); //Nor here!
        }
};

```

RayHitID

- Replaces the old LONG value that stored information of an object/polygon. Allows you to set objects with a polygon index, or read this information.

```
struct RayHitID
{
    public:
        RayHitID();
        RayHitID(_DONTCONSTRUCT DC);
        RayHitID(const RayHitID &other);
        RayHitID(RayObject *t_rayobject, LONG t_polygon, Bool second);

        inline Bool IsEqual(const RayHitID &snd) const;

        inline Bool Content() const;
        inline void Clear();

        inline void Set(RayObject *t_rayobject, LONG t_polygon, Bool second);

        inline RayObject *GetObject(VolumeData *vd) const;

        inline LONG GetPolygon() const;
        inline Bool GetSecond() const;

        inline void ClearSecond();
        inline void SetSecond();

        inline void SetPrivateData(LONG t_rayobject, LONG t_polygon);
        inline void GetPrivateData(LONG *t_rayobject, LONG *t_polygon) const;
};
```

- [SetSecond/GetSecond](#) determine for a quadrangle which part of the quadrangle was stored (A-B-C or A-C-D).

- Instead of `if (lhit==0)` you need to write `if (lhit.Content())`

- Instead of `lhit=0` you need to write `lhit.Clear()`

- The routines [ID_to_Obj](#) and [Obj_to_ID](#) have been removed, use the methods of the [RayHitID](#) class instead.

RenderDocument

- `RenderDocument` no longer has the parameters `left/top/right/bottom`.
- Instead following values of the `rdata` container have to be set:
`RDATA_RENDERREGION_LEFT`
`RDATA_RENDERREGION_TOP`
`RDATA_RENDERREGION_RIGHT`
`RDATA_RENDERREGION_BOTTOM`
- Set `RDATA_RENDERREGION` to `TRUE` to render only a part of the image.

UVWTag

·Old [Get/Set/Cpy](#) methods have been replaced by faster routines (they are still available as [GetSlow/SetSlow/CpySlow](#))

·Example of the new [Get](#) method:

```
const void *dataptr = uvwtag->GetDataAddressR();

for (LONG i=0; i<cnt; i++)
{
    UVWStruct res
    UVWTag::Get(dataptr, i, res);

    //do something with res
}
```

MoData/MDArray

- [MoData](#) has been completely rewritten.
The old [MoData](#) structure is still available as [MoDataEx](#) but it is recommended to use the new [MoData](#) class.
- [MDArray](#) is a template class for arrays filled by [MoData](#).

MovieLoader

New MovieLoader class to simplify loading frames of movie clips.

```
class MovieLoader
{
    public:
        static MovieLoader *Alloc(void);
        static void Free(MovieLoader *&ml);

        LONG Open(const Filename &fn);
        void Close(void);

        BaseBitmap* Read(LONG new_frame_idx = -1, LONG *_result = NULL);

        LONG GetInfo(Real *_fps);
};
```

Parser/ParserCache

- The [Parser](#) class has been greatly improved, both feature- and speed-wise.
- The parser expression evaluation is much more correct and supports for instance upper and lower case expressions.
- It is possible to cache parser expressions to speed-up expression evaluation. There can exist several caches ([ParserCache](#)).