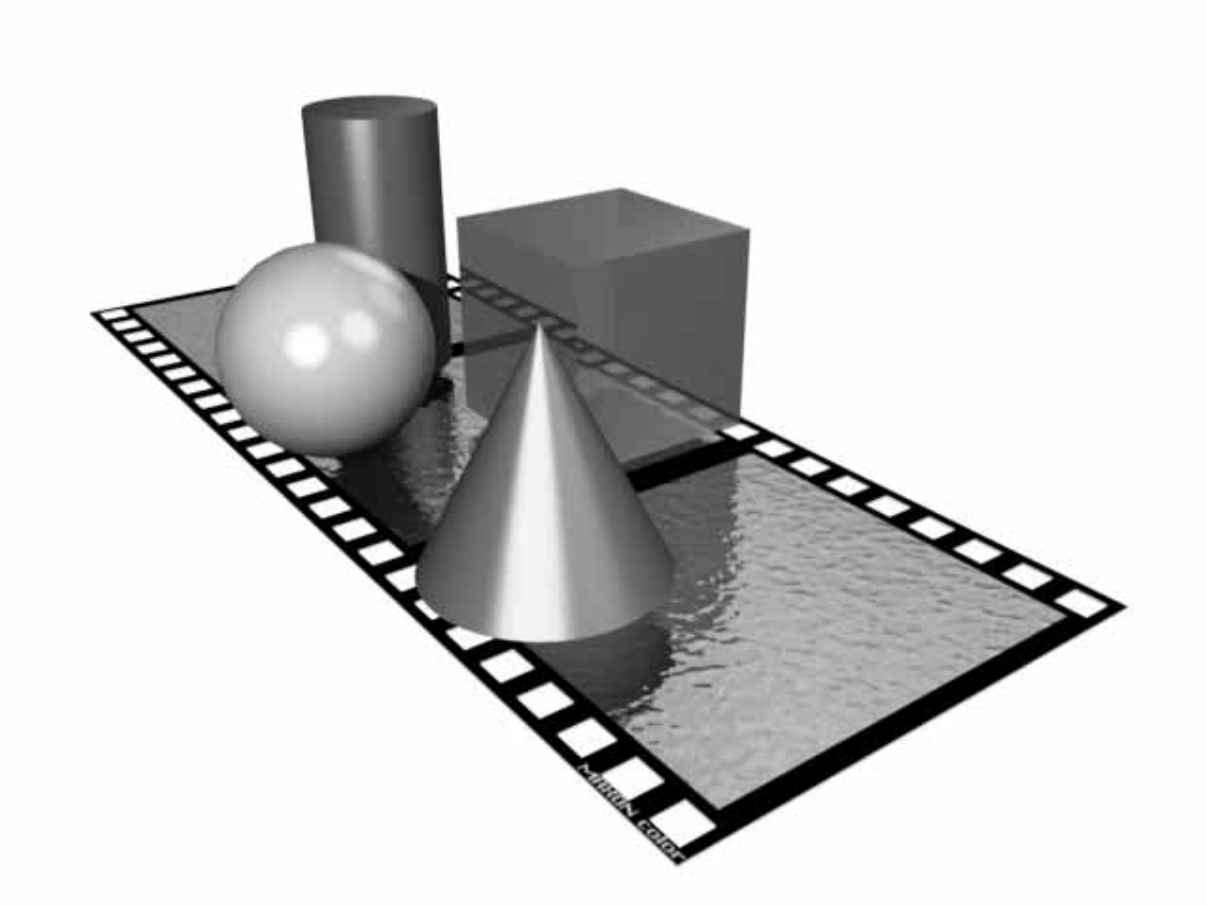
CINEMA 4D®

File Format Description



MAXON computer

Document type	CINEMA 4D® — File Format Description
	Version 4.x
Version	1.0
Date	30.03.1997
Creator	Philip Losch
	p_losch@maxon.de
Contact	MAXON Computer GmbH
	Industriestr. 26
	65760 Eschborn
	Germany
Phone	++ 49 (0) 61 96 – 48 18 11
Fax	++ 49 (0) 61 96 – 4 11 37
Website	www.maxon.de
Basic knowledge	IFF specification
	IEEE floating point specification
	Little & Big endian byte order
Last revised	30.03.1997
Last changes	No changes currently made

If you find any errors in this documentation, please report us, that we can always provide you and the other developers with the latest stuff!

Table of Contents

1 Introduction

This document shows you how you can read and write CINEMA 4D files.

CINEMA 4D doesn't use separate files for material, object and scene information. Instead CINEMA 4D uses one file format which can contain all these elements. This is the user friendly CINEMA 4D concept to offer one integrated solution - that means one editor where animation, modeling, texturing and rendering can be done at the same time.

Most things will in brief be discussed in detail. But you should know that this can be no substitute for reading the manual and working with CINEMA 4D.

CINEMA 4D was created to bring power, speed and flexibility to the users. We spend a lot of time on the quality of our product and hope you will do the same. Don't write a conversion tool which hangs up every 2nd time. Your application should have all CINEMA typical features:

- Stability
- Extreme speed
- Innovative features
- Style guide compliant and intuitive interface.

We wish you good luck!

2 Conventions

- The coordinate system of CINEMA 4D looks like this: the x axis goes from left to right, the y axis from bottom to top and the z axis from front to back. CINEMA 4D uses a left hand coordinate system.

- As CINEMA 4D is a multiplatform development all data is written platform independent. It is the Motorola byte order in which all data is represented. So, on 68k or PowerPC based systems (e.g. Apple Macintosh) you can directly read from the file, whereas on Intel based systems you have to change the byte order first.

- Floating point data is always stored in single precision IEEE format.

- CINEMA uses for angles the Heading/Pitch/Bank system. Angles are stored in the 'Rad' system, e.g. 90° as pi / 2.

- All geometric data in CINEMA 4D is stored locally. This means that the objects' points are given relative to their axis system. Also an axis system is relative to its parent object system.

3 Data types

```
#define FALSE   0
#define TRUE    1
#define MAXREAL 10000000.0

#define BOOL  unsigned char    // BOOL  is a 1 byte unsigned character
#define CHAR  char             // CHAR  is a 1 byte integer
#define WORD  word             // WORD  is a 2 byte integer
#define UWORD unsigned word   // UWORD is a 2 byte unsigned integer
#define LONG  long             // LONG  is a 4 byte integer
#define LONG  unsigned long    // ULONG is a 4 byte unsigned integer
#define Real  float            // Real  is a 4 byte float value

typedef struct Vector
{
  Real  x,y,z;
}
```

The data type ‘Vector’ can be used as vector, point and color.

When being used as a color, the x/y/z components are used for the red, green and blue component. The values vary from 0.0 to 1.0.

Points can be set between -MAXREAL and +MAXREAL. Larger values are valid but should not be set.

Vectors always have to be normalized, that means v.x*v.x+v.y*v.y+v.z*v.z = 1.0.

4 General file format structure

CINEMA 4D uses IFF, which means „Interchange File Format“. IFF is a chunk based format. IFF allows you to read only these structures of a file which your application can handle. So, if you don’t know a CINEMA 4D chunk, just skip it!

CINEMA 4D Header:

The header consists of 12 bytes as follows:

Size	Type	Description
4 bytes	4 CHARS	IFF identification ‘FORM’ (= 0x464F524D)
4 bytes	LONG	Size of the file (starting at this position)
4 bytes	4 CHARS	CINEMA 4D identification ‘MC4D’ (= 0x4D433444)

Then some IFF chunks follow:

Size	Type	Description
4 bytes	4 CHARS	Chunk identification
4 bytes	LONG	Size of the chunk (starting at this position)
‘size’ bytes		Chunk data

Here some chunk IDs:

Chunk ID	Description
‘PLTF’ (= 0x504C5446)	Contains information on the creator platform
‘MAT4’ (= 0x4D415434)	Contains information on materials.
‘OBJ5’ (= 0x4F424A35)	Contains information on objects.
‘UMG4’ (= 0x554D4734)	Contains information on the editor camera and the editor view.

All chunks are optional. If material and object chunks are written, the material chunk must be written first. A platform chunk has to be placed at the beginning of a file.

4.1 The Platform Chunk

Size	Type	Description
4 bytes	4 CHARS	Chunk identification ‘PLTF’ (= 0x504C5446)
4 bytes	LONG	Size of the chunk (starting at this position)
4 bytes	LONG	Windows = 1 Macintosh = 2 Unix = 4

If no platform can be found, CINEMA 4D assumes that the file was written on the platform CINEMA 4D runs on.

4.2 The Material Chunk

This chunk is only written if it contains at least one material.

The material chunk is a collection of many small subchunks. After the header of this chunk subchunks are stored until the chunk size is reached.

If you write down information, you have to write only these values that differ from the defaults. The default material is a white surface without highlight, reflectance etc.

Size	Type	Description
4 bytes	4 CHARS	Chunk identification 'MAT4'
4 bytes	LONG	Size of the chunk (starting at this position)
2 bytes	WORD	Subchunk ID
2 bytes	WORD	Subchunk 'size'
subsize bytes		Subchunk data
...	...	...

CINEMA 4D builds materials on different channels which have very similar parameters. Please refer to the manual for further information on the material concept of CINEMA 4D.

The different channels are:

Char	Channel
F	Specular channel
L	Diffuse channel
T	Transparent channel
S	Reflectant channel
U	Environment reflection channel
N	Fog channel
R	Bump channel
G	Genlock channel
H	Highlight channel
Z	Highlight colour channel

Subchunk name	Subchunk ID
MAT_END	0
MAT_FACTIVE	1
MAT_LACTIVE	2
MAT_TACTIVE	3
MAT_SACTIVE	4
MAT_UACTIVE	5
MAT_NACTIVE	6
MAT_RACTIVE	7
MAT_GACTIVE	8
MAT_HACTIVE	9
MAT_ZACTIVE	10
MAT_FCOLOUR	11
MAT_LCOLOUR	12
MAT_TCOLOUR	13
MAT_SCOLOUR	14
MAT_UCOLOUR	15
MAT_NCOLOUR	16
MAT_GCOLOUR	17
MAT_ZCOLOUR	18
MAT_FINTENS	19
MAT_LINTENS	20
MAT_TINTENS	21
MAT_SINTENS	22
MAT_UINTENS	23

MAT_NINTENS	24
MAT_ZINTENS	25
MAT_FTINTENS	26
MAT_LTINTENS	27
MAT_TTINTENS	28
MAT_STINTENS	29
MAT_UTINTENS	30
MAT_ZTINTENS	31
MAT_FTNAME	39
MAT_LTNAME	40
MAT_TTNAME	41
MAT_STNAME	42
MAT_UTNAME	43
MAT_RTNAME	44
MAT_ZTNAME	45
MAT_FFLAG	46
MAT_LFLAG	47
MAT_TFLAG	48
MAT_SFLAG	49
MAT_UFLAG	50
MAT_RFLAG	51
MAT_ZFLAG	52
MAT_HWIDTH	53
MAT_HHEIGHT	54
MAT_REFRACTION	55
MAT_BUMPANGLE	56
MAT_DISTANCE	57
MAT_RANGE	58
MAT_FRESNEL	59
MAT_SHADOW	60
MAT_SHADER	61
MAT_NAME	64
MAT_NEXT	65
MAT_FMCOLOUR	66
MAT_LMCOLOUR	67
MAT_TMCOLOUR	68
MAT_SMCOLOUR	69
MAT_UMCOLOUR	70
MAT_GTNAME	72
MAT_GFLAG	73

If a subchunk ID is unknown by your application you can skip the subchunk by going ‘subchunk size’ bytes forward.

Now we'll go on with details.

4.2.1 Subchunk MAT_NAME

Size	Type	Description
1 byte	CHAR	String length 'len'
'len' bytes	CHAR	Material name

4.2.2 Subchunk MAT_xACTIVE

Size	Type	Description
1 byte	BOOL	If set, this channel is active

Default:
All channels are inactive except for the specular colour channel.

4.2.3 Subchunk MAT_xCOLOUR

Size	Type	Description
12 bytes	Vector	Channel colour

Default:
All colours set to white.

4.2.4 Subchunk MAT_xINTENS

Size	Type	Description
4 bytes	Real	Channel colour intensity

Default:
All intensities set to 1.0.

4.2.5 Subchunk MAT_xTNAME

Size	Type	Description
1 byte	CHAR	String length 'len'
len bytes	CHAR	Channel texture name

Filenames are stored in OS specific ASCII code. Filenames must not contain path information!

Default:
No textures used.

4.2.6 Subchunk MAT_xTINT

Size	Type	Description
4 bytes	Real	Channel texture intensity

Default:
All intensities set to 1.0.

4.2.7 Subchunk MAT_xTFLAG

Size	Type	Description
1 byte	CHAR	Channel texture interpolation None = 0 Circle = 1 Square = 2 Anti1 = 3 Anti2 = 4 Anti3 = 5

Default:
All interpolations set to ‘Square’.

4.2.8 Subchunk MAT_xMCOLOUR

Size	Type	Description
12 bytes	Vector	Middle colour of channel texture

Middle colours are used in CINEMA 4D for colouring the object’s wireframes.

Default:
All middle colours set to black.

4.2.9 Subchunk MAT_HWIDTH

Size	Type	Description
4 bytes	Real	Highlight width (from 0.0 to 1.0)

Default:
Highlight width set to 0.3

4.2.10 Subchunk MAT_HHEIGHT

Size	Type	Description
4 bytes	Real	Highlight height (from 0.0 to 1.0)

Default:
Hightlight height set to 1.0

4.2.11 Subchunk MAT_REFRACTION

Size	Type	Description
4 bytes	Real	Refraction index (from 0.25 to 4.0)

Default:
Refraction index set to 1.0

4.2.12 Subchunk MAT_BUMPANGLE

Size	Type	Description
4 bytes	Real	Bump angle (from -100.0 to 100.0)

Default:
Bumpangle set to 20.0

4.2.13 Subchunk MAT_DISTANCE

Size	Type	Description
4 bytes	Real	Fog distance (>0.0)

Default:
Distance set to 1000.0

4.2.14 Subchunk MAT_RANGE

Size	Type	Description
4 bytes	Real	Genlock range (from 0.0 to 1.0)

Default:
Range set to 0.1

4.2.15 Subchunk MAT_FRESNEL

Size	Type	Description
1 byte	BOOL	If set, Fresnel’s equations are used

Default:
Fresnel set on.

4.2.16 Subchunk MAT_SHADOW

Size	Type	Description
1 byte	BOOL	If set, material receives shadow

Default:
Shadows set on.

4.2.17 Subchunk MAT_SHADER

Size	Type	Description
4 byte	LONG	Material Shader Phong = 0 Metal = 1

Default:
Shader set to Phong.

4.2.18 Subchunk MAT_NEXT

This chunk contains no data. It indicates that a new material follows.

4.2.19 Subchunk MAT_END

This chunk contains no data. It indicates that the end of the material chunk has been reached.
All subchunks are optional, but a MAT_END subchunk has to be written at the end of your material information.

4.3 The Object Chunk

This chunk is only written if it contains at least one object.

The object chunk is a collection of many small subchunks. After the header of this chunk subchunks are stored until the chunk size is reached.

Size	Type	Description
4 bytes	4 CHARS	Chunk identification 'OBJ5'
4 bytes	LONG	Size of the chunk (starting at this position)
4 bytes	ULONG	Subchunk ID and size subid = (val >> 24) subsize = val & 16777215
subsize bytes		Subchunk data
...	...	...

Subchunk name	Subchunk ID	Remarks
T_END	0	Terminates the object description (other objects can follow) Non-optional
T_OBJECT	1	Introduces the object description Non-optional
T_POINTS	16	
T_TRIANGLES	17	
T_QUADRANGLES	18	
T_EDGES	19	
T_POLYGON	20	
T_LIGHT	21	
T_CAMERA	22	
T_TEXTURE	23	Optional; multiple textures allowed
T_PHONG	24	Optional
T_GROUND	25	
T_SKY	26	
T_SPHERE	27	
T_DISPLAY	28	Optional
T_ENVIRONMENT	29	
T_HERMITE	30	
T_KINEMATIC	31	Optional
T_ROOT	32	Optional

Every object descriptions begins with the T_OBJECT subchunk and ends with T_END. Some subchunks like T_TEXTURE and T_PHONG are optional - they can be used with any type of object.

T_TEXTURE is the only subchunk which can be used several times for an object.

Other subchunks can be used only in several combinations:

1. T_POINTS + T_TRIANGLES (optional) + T_QUADRANGLES (optional) +
   T_EDGES (required when T_TRIANGLES or T_QUADRANGLES is used)
2. T_POLYGON + T_POINTS (optional) +
   T_HERMITE (required when T_POINTS is used and polygon interpolation is set to hermite)
3. T_LIGHT
4. T_CAMERA
5. T_GROUND
6. T_SKY
7. T_SPHERE + T_POINTS (exactly one point)
8. T_ENVIRONMENT
9. None of these

These combinations cannot be mixed. For example T_ENVIRONMENT cannot be used with T_POINTS. Each combination defines the object's type.

4.3.1 Subchunk T_OBJECT

Size	Type	Description
12 bytes	Vector	Axis position
12 bytes	Vector	Axis scale
12 bytes	Vector	Axis rotation (Euler angles; Heading-, Pitch-Bank System)
1 byte	CHAR	'len' of object name
len bytes	CHAR	Object name
1 byte	CHAR	Object 'flag' (see below)

4.3.2 Subchunk T_POINTS

Size	Type	Description
subsize bytes	Vector	Point array

4.3.3 Subchunk T_EDGES

Size	Type	Description
subsize bytes	UWORD	Edge array All edge indices start with zero. Two indices build up a edge. The first index of an edge has to be smaller or equal than the second index.

4.3.4 Subchunk T_TRIANGLES

Size	Type	Description
subsize bytes	UWORD	Triangle array All triangle indices start with zero. Three indices build up a triangle. The index order defines the surface normal.

4.3.5 Subchunk T_QUADRANGLES

Size	Type	Description
subsize bytes	UWORD	Quadrangle array All quadrangle indices start with zero. Four indices build up a quadrangle. The index order defines the surface normal.

4.3.6 Subchunk T_HERMITE

Size	Type	Description
subsize bytes	Vector	Hermite vectors Two vectors belong to one point of T_POINTS.

4.3.7 Subchunk T_KINEMATIC

Size	Type	Description
4 bytes	BOOL	If set, Heading is restricted
4 bytes	BOOL	If set, Pitch is restricted
4 bytes	BOOL	If set, Bank is restricted
4 bytes	Real	Heading Min (<pi and >-pi)
4 bytes	Real	Pitch Min (<pi and >-pi)
4 bytes	Real	Bank Min (<pi and >-pi)
4 bytes	Real	Heading Max (<pi and >heading_min)
4 bytes	Real	Pitch Max (<pi and >pitch_min)
4 bytes	Real	Bank Max (<pi and >bank_min)
4 bytes	Real	Dampening (from 0.0 to 1.0)

4.3.8 Subchunk T_POLY

Size	Type	Description
4 bytes	LONG	Polygon Type Linear = 0 Cubic = 1 Akima = 2 B-Spline = 3 Hermite = 4 (Default: Linear)
4 bytes	BOOL	If set, polygon is closed (Default: FALSE)
4 bytes	LONG	Polygon subdivision None = 0 Natural = 1 Uniform = 2 Adaptive = 3 (Default: Adaptive)
4 bytes	LONG	Number for natural/uniform subdivision (Default: 20)
4 bytes	Real	Angle for adaptive subdivision (Default: 5.0/180.0*pi)

4.3.9 Subchunk T_LIGHT

Size	Type	Description
12 bytes	Vector	Light colour
12 bytes	Vector	Visible light decay (>0.0) (Default: 100.0)
4 bytes	Real	Falloff distance (>0.0) (Default: 1000.0)
4 bytes	Real	Spot angle (>0.0 and <pi) (Default: 30.0/180.0*pi)
4 bytes	Real	Cylinder radius (>0.0) (Default: 100.0)
4 bytes	Real	Lensglow intensity (>=0.0) (Default: 1.0)
4 bytes	Real	Lens reflections intensity (>=0.0) (Default: 1.0)
4 bytes	Real	Visible light intensity (>=0.0) (Default: 1.0)
4 bytes	Real	Visible light decrease (>=0.0) (Default: 10.0)
4 bytes	LONG	Visible light function None = 0 XYZ = 1 XY = 2 Z = 3 Constant = 4 (Default: None)
4 bytes	LONG	Shadow type None = 0 Soft = 1 Hard = 2 (Default: None)
4 bytes	LONG	Shadow map size 128x128 = 0 256x256 = 1 512x512 = 2 768x768 = 3 (Default: 256x256)
4 bytes	BOOL	If set, lensglows are enabled (Default: FALSE)
4 bytes	BOOL	If set, lens reflections are enabled (Default: FALSE)
4 bytes	BOOL	If set, lens flares fade off screen (Default: TRUE)
4 bytes	BOOL	If set, lens streaks are randomly distributed (Default: FALSE)
4 bytes	LONG	If set, spot light is enabled (Default: FALSE)
4 bytes	LONG	If set, light falloff is enabled (Default: FALSE)
4 bytes	LONG	If set, parallel light is enabled (Default: FALSE)
4 bytes	LONG	If set, soft spot cones are enabled (Default: TRUE)
4 bytes	LONG	If set, lightsource does not illuminate (Default: FALSE)

Now follows 20 times this block:

Size	Type	Description
4 bytes	BOOL	If set, lens element is active
4 bytes	LONG	Number of streaks (from 3 to 100)
4 bytes	LONG	Lens type Exp1 = 0 Exp2 = 1 Exp3 = 2 Linear = 3

		Inv. Exp = 4
4 bytes	Real	Lens position (from -2.5 to 2.5)
4 bytes	Real	Lens scale x (>0.0 and <1.0)
4 bytes	Real	Lens scale y (>0.0 and <1.0)
4 bytes	Real	Lens rotation
12 bytes	Vector	Lens colour

The defaults are:

Lens element	Default Parameters
0	TRUE, 4, 0, 0.0, 0.2, 0.2, 0.0, (0.15/0.15/0.15)
1	TRUE, 4, 0, 0.0, 0.05, 0.05, 0.0, (1.0/0.6/0.6)
2	TRUE, 4, 0, 0.0, 0.1, 0.1, 0.0, (0.25/0.0/0.0)
3	TRUE, 4, 0, 0.0, 0.17, 0.17, 0.0, (1.0/1.0/1.0)
4	TRUE, 12, 0, 0.0, 0.14, 0.14, 0.0, (1.0/1.0/1.0)
5	TRUE, 4, 1, -0.2, 0.03, 0.03, 0.0, (0.4/0.4/0.4)
6	TRUE, 4, 2, 0.13, 0.018, 0.018, 0.0, (0.0/0.0/0.2)
7	TRUE, 4, 3, 0.15, 0.05, 0.05, 0.0, (0.0/0.0/0.11)
8	TRUE, 4, 3, 0.18, 0.027, 0.027, 0.0, (0.0/0.0/0.15)
9	TRUE, 4, 3, 0.25, 0.022, 0.022, 0.0, (0.3/0.2/0.0)
10	TRUE, 4, 4, 0.32, 0.005, 0.005, 0.0, (1.0/1.0/1.0)
11	TRUE, 4, 4, 0.44, 0.007, 0.007, 0.0, (1.0/1.0/1.0)
12	TRUE, 4, 2, 0.46, 0.052, 0.052, 0.0, (0.2/0.1/0.0)
13	TRUE, 4, 2, 0.54, 0.025, 0.025, 0.0, (0.3/0.2/0.0)
14	TRUE, 4, 4, 0.6, 0.033, 0.033, 0.0, (0.0/0.45/0.8)
15	TRUE, 4, 0, 0.8, 0.1, 0.1, 0.0, (0.2/0.3/0.0)
16	TRUE, 4, 0, 1.0, 0.2, 0.2, 0.0, (0.18/0.18/0.25)
17	FALSE, 4, 0, 1.0, 0.2, 0.2, 0.0, (0.18/0.18/0.25)
18	FALSE, 4, 0, 1.0, 0.2, 0.2, 0.0, (0.18/0.18/0.25)
19	FALSE, 4, 0, 1.0, 0.2, 0.2, 0.0, (0.18/0.18/0.25)

Now follows - eventually - this block (check this with the subchunk size):

Size	Type	Description
4 bytes	Real	Lightsource bias (>0.0 and <0.5) (Default: 0.05)

4.3.10 Subchunk T_CAMERA

Size	Type	Description
4 bytes	Real	Focal length (from 0.1 to MAXREAL) (Default: 50.0)
4 bytes	Real	Front (>0.0) (Default: 0.0)
4 bytes	Real	Middle (>Front) (Default: 1000.0)
4 bytes	Real	Back (>Middle) (Default: 2000.0)
4 bytes	LONG	Depth of field type None = 0 Front & Back = 1 Front = 2 Back = 3 (Default: None)

4.3.11 Subchunk T_TEXTURE

Size	Type	Description
12 bytes	Vector	Texture axis position
12 bytes	Vector	Texture axis scale
12 bytes	Vector	Texture axis angle
4 bytes	Real	Texture offset x
4 bytes	Real	Texture offset y
4 bytes	Real	Texture scale x
4 bytes	Real	Texture scale y
4 bytes	LONG	Texture projection Spherical = 0 Cylindrical = 1 Flat = 2
4 bytes	BOOL	If set, texture is tiled
4 bytes	LONG	Material number (starting with zero) Can be -1, if no material is refered If != -1 material chunk must be there!

4.3.12 Subchunk T_PHONG

This subchunk can be empty or can contain a 4 byte Real value (maximum phong angle, default:0.0).
Use the subchunk size for decision.

4.3.13 Subchunk T_GROUND

This subchunk is empty and indicates, that the object is of type ‘ground’.

4.3.14 Subchunk T_SKY

This subchunk is empty and indicates, that the object is of type ‘sky’.

4.3.15 Subchunk T_ROOT

This subchunk is empty and indicates, that the object is a root object when inverse kinematics is applied.

4.3.16 Subchunk T_SPHERE

Size	Type	Description
4 bytes	Real	Sphere radius

4.3.17 Subchunk T_DISPLAY

Size	Type	Description
4 bytes	LONG	Display flag Gouraud = 0 Flat = 1 Wireframe = 2 Bounding box = 3 Invisible = 4

4.3.18 Subchunk T_ENVIRONMENT

Size	Type	Description
12 bytes	Vector	Ambient colour (Default: 1.0)
12 bytes	Vector	Fog colour (Default: 1.0)
4 bytes	Real	Ambient intensity (from 0.0 to 1.0) (Default: 0.1)
4 bytes	Real	Fog intensity (from 0.0 to 1.0) (Default: 1.0)
4 bytes	Real	Fog distance (>0.0) (Default: 10000.0)
4 bytes	Real	Foreground picture intensity (from 0.0 to 1.0) (Default: 1.0)
4 bytes	Real	Background picture intensity (from 0.0 to 1.0) (Default: 1.0)
4 bytes	Real	Unused (set to 0.0)
4 bytes	Real	Unused (set to 0.0)
4 bytes	LONG	Foreground picture interpolation None = 0 Circle = 1 Square = 2 Anti1 = 3 Anti2 = 4 Anti3 = 5 (Default: Square)
4 bytes	LONG	Background picture interpolation None = 0 Circle = 1 Square = 2 Anti1 = 3 Anti2 = 4 Anti3 = 5 (Default: Square)
4 bytes	BOOL	If set, fog is active (Default: FALSE)
4 bytes	BOOL	If set, sun is active (Default: FALSE)
1 byte	CHAR	‘len’ of Foreground picture name
len bytes	CHAR	Foreground picture name
1 byte	CHAR	‘len’ of Background picture name
len bytes	CHAR	Background picture name
4 bytes	Real	Geographical Latitude (Default: 51.0/180.0*pi)
4 bytes	LONG	Hour (Default: 12)
4 bytes	LONG	Day (Default: 21)
4 bytes	LONG	Month (Default: 9)
4 bytes	LONG	Minutes (Default: 0)
4 bytes	BOOL	If set, shadows are active (Default: TRUE)

4.3.19 T_NULL

This subchunk is empty and indicates that this object description is complete.
Now let's have a closer look on the one byte 'flag' value in the T_OBJECT subchunk.

Bit	Decription
(flag & 1)	This hierarchy level contains more objects ('brother objects' follow)
(flag & 2)	The sub-hierarchy contains objects ('child object' follow)
(flag & 4)	This is the active object (this can be set only once in the whole scene)
(flag & 8)	Child objects are also active (can be only set for the active object)
(flag & 16)	This is the active camera (this can be set only once in the whole scene)

'Child objects' are read before 'brother objects'.

4.4 The Environment chunk

This optional chunk contains information about the placement of the editor views.

Size	Type	Description
4 bytes	4 CHARS	Chunk identification ‘UMG4’
4 bytes	LONG	Size of the chunk (starting at this position)
4 bytes	LONG	View Front = 0 Side = 1 Top = 2 Perspective = 3 4-Side-View = 4
12 bytes	Vector	Offset of planar views
4 bytes	Real	Zoom factor of planar views
12 bytes	Vector	Editor camera position
12 bytes	Vector	Editor camera angle
4 bytes	Vector	Editor camera zoom

5 Sample code

Here we show you how to implement a loading module in C++.

```
#include "c4dtypes.h"

class Loader
{
  private:
    File   *file;

    File   *FOpen(CHAR *name);     // Open a file
    void    FSkip(LONG relskip);   // Skip 'relskip' bytes
    void    FClose(File *file);    // Close a file
    LONG    FPos();                // Return absolute position in file

    CHAR    ReadCHAR();            // Read  1 byte
    WORD    ReadWORD();            // Read  2 bytes
    LONG    ReadLONG();            // Read  4 bytes
    Real    ReadREAL();            // Read  4 bytes
    Vector  ReadVECTOR();          // Read 12 bytes
    void   *ReadNBytes(LONG n);    // Read  n bytes

    void LoadMaterials(void);
    void LoadObjects(void);
    void LoadEnvironment(void);

  public:
    Loader(void);
    ~Loader(void);

    Bool LoadC4D(CHAR *name);
}

Loader::Loader(void)
{
  file = 0;
  platform = GetCurrentOS(); // On which platform runs this application?
}

Loader::~Loader(void)
{
  if (file)
  {
    FClose(file);
    file = 0;
  }
}

void Loader::LoadMaterials()
{
  CHAR     strlen;
  WORD     id=MAT_NEXT,len;
  Material *mat;

  while (id!=MAT_END)
  {
    mat = AllocMaterial();

    do
    {
      id  = ReadWORD();
      len = ReadWORD();

      switch (id)
      {
        case MAT_NAME:
```

```
  strlen = ReadCHAR();
  mat->name = ReadNBYTES(strlen);
  break;

case MAT_FACTIVE:
  mat->bf = ReadCHAR();
  break;

case MAT_LACTIVE:
  mat->bl = ReadCHAR();
  break;

case MAT_TACTIVE:
  mat->bt = ReadCHAR();
  break;

case MAT_SACTIVE:
  mat->bs = ReadCHAR();
  break;

case MAT_UACTIVE:
  mat->bu = ReadCHAR();
  break;

case MAT_NACTIVE:
  mat->bn = ReadCHAR();
  break;

case MAT_RACTIVE:
  mat->br = ReadCHAR();
  break;

case MAT_GACTIVE:
  mat->bg = ReadCHAR();
  break;

case MAT_HACTIVE:
  mat->bh = ReadCHAR();
  break;

case MAT_ZACTIVE:
  mat->bz = ReadCHAR();
  break;

case MAT_FCOLOUR:
  mat->f = ReadVECTOR();
  break;

case MAT_LCOLOUR:
  mat->l = ReadVECTOR();
  break;

case MAT_TCOLOUR:
  mat->t = ReadVECTOR();
  break;

case MAT_SCOLOR:
  mat->f = ReadVECTOR();
  ReadREAL(u,&mat->s,3);
  break;

case MAT_UCOLOUR:
  mat->u = ReadVECTOR();
  break;

case MAT_NCOLOUR:
  mat->n = ReadVECTOR();
  break;

case MAT_GCOLOUR:
  mat->g = ReadVECTOR();
```

```
  break;

case MAT_ZCOLOUR:
  mat->z = ReadVECTOR();
  break;

case MAT_FMCOLOUR:
  mat->mf = ReadVECTOR();
  break;

case MAT_LMCOLOUR:
  mat->ml = ReadVECTOR();
  break;

case MAT_TMCOLOUR:
  mat->mt = ReadVECTOR();
  break;

case MAT_SMCOLOUR:
  mat->ms = ReadVECTOR();
  break;

case MAT_UMCOLOUR:
  mat->mu = ReadVECTOR();
  break;

case MAT_FINTENS:
  mat->fint = ReadREAL();
  break;

case MAT_LINTENSL:
  mat->lint = ReadREAL();
  break;

case MAT_TINTENS:
  mat->tint = ReadREAL();
  break;

case MAT_SINTENS:
  mat->sint = ReadREAL();
  break;

case MAT_UINTENS:
  mat->uint = ReadREAL();
  break;

case MAT_NINTENS:
  mat->nint = ReadREAL();
  break;

case MAT_ZINTENS:
  mat->zint = ReadREAL();
  break;

case MAT_FTINTENS:
  mat->ftint = ReadREAL();
  break;

case MAT_LTINTENS:
  mat->ltint = ReadREAL();
  break;

case MAT_TTINTENS:
  mat->ttint = ReadREAL();
  break;

case MAT_STINTENS
  mat->stint = ReadREAL();
  break;

case MAT_UTINTENS:
```

```
  mat->utint = ReadREAL();
  break;

case MAT_ZTINTENS:
  mat->ztint = ReadREAL();
  break;

case MAT_HWIDTH:
  mat->hwidth = ReadREAL();
  break;

case MAT_HHEIGHT:
  mat->hheight = ReadREAL();
  break;

case MAT_REFRACTION:
  mat->refr = ReadREAL();
  break;

case MAT_BUMPANGLE:
  mat->bumpangle = ReadREAL();
  break;

case MAT_DISTANCE:
  mat->dist = ReadREAL();
  break;

case MAT_RANGE:
  mat->range = ReadREAL();
  break;

case MAT_FTNAME:
  strlen      = ReadCHAR();
  mat->ftname = ReadNBYTES(strlen);
  break;

case MAT_LTNAME:
  strlen      = ReadCHAR();
  mat->ltname = ReadNBYTES(strlen);
  break;

case MAT_TTNAME:
  strlen      = ReadCHAR();
  mat->ttname = ReadNBYTES(strlen);
  break;

case MAT_STNAME:
  strlen      = ReadCHAR();
  mat->stname = ReadNBYTES(strlen);
  break;

case MAT_UTNAME:
  strlen      = ReadCHAR();
  mat->utname = ReadNBYTES(strlen);
  break;

case MAT_RTNAME:
  strlen      = ReadCHAR();
  mat->rtname = ReadNBYTES(strlen);
  break;

case MAT_ZTNAME:
  strlen      = ReadCHAR();
  mat->ztname = ReadNBYTES(strlen);
  break;

case MAT_GTNAME:
  strlen      = ReadCHAR();
  mat->gtname = ReadNBYTES(strlen);
  break;
```

```
        case MAT_FFLAG:
          mat->fflag = ReadCHAR();
          break;

        case MAT_LFLAG:
          mat->lflag = ReadCHAR();
          break;

        case MAT_TFLAG:
          mat->tflag = ReadCHAR();
          break;

        case MAT_SFLAG:
          mat->sflag = ReadCHAR();
          break;

        case MAT_UFLAG:
          mat->uflag = ReadCHAR();
          break;

        case MAT_RFLAG:
          mat->rflag = ReadCHAR();
          break;

        case MAT_ZFLAG:
          mat->zflag = ReadCHAR();
          break;

        case MAT_GFLAG:
          mat->gflag = ReadCHAR();
          break;

        case MAT_FRESNEL:
          mat->fresnel = ReadCHAR();
          break;

        case MAT_SHADOW:
          mat->shadow = ReadCHAR();
          break;

        case MAT_SHADER:
          mat->shader = ReadLONG();
          break;

        default:
          FSkip(len);
          break;
      }
    }
    while (id!=MAT_END && id!=MAT_NEXT);
  }
}

void Loader::LoadObjects(void)
{
  Object  *op;
  CHAR    flag,len;
  LONG    val,size,type,i;

  do
  {
    op   = AllocObjekt();
    flag = 0;

    do
    {
      val  = ReadLONG();
      typ  = TagTyp(tag>>24);
      size = tag & 16777215;

      switch (typ)
```

```
{
  case T_OBJECT:
    op->pos   = ReadVECTOR();
    op->scale = ReadVECTOR();
    op->rot   = ReadVECTOR();

    len       = ReadCHAR();
    op->name  = ReadNBYTES(len);

    flag      = ReadCHAR();
    break;

  case T_POINTS:
    for (i=0; i<size/12; i++)
      point[i] = ReadVECTOR();
    break;

  case T_HERMITE:
    for (i=0; i<size/24; i++)
    {
      left_tangent [i] = ReadVECTOR();
      right_tangent[i] = ReadVECTOR();
    }
    break;

  case T_EDGES:
    for (i=0; i<size/4; i++)
    {
      index_a[i] = GetUWORD();
      index_b[i] = GetUWORD();
    }
    break;

  case T_TRIANGLES:
    for (i=0; i<size/6; i++)
    {
      index_a[i] = GetUWORD();
      index_b[i] = GetUWORD();
      index_c[i] = GetUWORD();
    }
    break;

  case T_QUADRANGLES:
    for (i=0; i<size/8; i++)
    {
      index_a[i] = GetUWORD();
      index_b[i] = GetUWORD();
      index_c[i] = GetUWORD();
      index_d[i] = GetUWORD();
    }
    break;

  case T_KINEMATIC:
    op->kinematic.resx = ReadLONG();
    op->kinematic.resy = ReadLONG();
    op->kinematic.resz = ReadLONG();
    op->kinematic.minx = ReadREAL();
    op->kinematic.miny = ReadREAL();
    op->kinematic.minz = ReadREAL();
    op->kinematic.maxx = ReadREAL();
    op->kinematic.maxy = ReadREAL();
    op->kinematic.maxx = ReadREAL();
    op->kinematic.damp = ReadREAL();
    break;

  case T_POLY:
    op->poly.type   = ReadLONG();
    op->poly.closed = ReadLONG();
    op->poly.inter  = ReadLONG();
    op->poly.num    = ReadLONG();
    op->poly.angle  = ReadREAL();
```

```
  break;

case T_LIGHT:
  op->light.col      = ReadVECTOR();
  op->light.drad     = ReadVECTOR();
  op->light.dist     = ReadREAL();
  op->light.angle    = ReadREAL();
  op->light.intens   = ReadREAL();
  op->light.rad      = ReadREAL();
  op->light.glowint  = ReadREAL();
  op->light.reflint  = ReadREAL();
  op->light.dint     = ReadREAL();
  op->light.ddec     = ReadREAL();
  op->light.vl       = ReadLONG();
  op->light.shadow   = ReadLONG();
  op->light.mapsize  = ReadLONG();
  op->light.lensglow = ReadLONG();
  op->light.lensrefl = ReadLONG();
  op->light.fadeoff  = ReadLONG();
  op->light.random   = ReadLONG();
  op->light.spot     = ReadLONG();
  op->light.decay    = ReadLONG();
  op->light.parallel = ReadLONG();
  op->light.soft     = ReadLONG();
  op->light.noemiss  = ReadLONG();

  for (i=0; i<20; i++)
  {
    op->light.lens[i].active  = ReadLONG();
    op->light.lens[i].streaks = ReadLONG();
    op->light.lens[i].type    = ReadLONG();
    op->light.lens[i].pos     = ReadREAL();
    op->light.lens[i].sizex   = ReadREAL();
    op->light.lens[i].sizey   = ReadREAL();
    op->light.lens[i].rot     = ReadREAL();
    op->light.lens[i].col     = ReadVECTOR();
  }

  if (size>904)
    op->light.bias = ReadREAL();
  break;

case T_CAMERA:
  op->camera.zoom   = ReadREAL();
  op->camera.front  = ReadREAL();
  op->camera.middle = ReadREAL();
  op->camera.back   = ReadREAL();
  op->camera.type   = ReadLONG();
  break;

case T_TEXTURE:
{
  Texture tex;
  tex.pos   = ReadVECTOR();
  tex.scale = ReadVECTOR();
  tex.rot   = ReadVECTOR();
  tex.ox    = ReadREAL();
  tex.oy    = ReadREAL();
  tex.lenx  = ReadREAL();
  tex.leny  = ReadREAL();
  tex.proj  = ReadLONG();
  tex.tile  = ReadLONG();
  tex.mat   = ReadLONG();

  op->AppendTexture(tex);
  break;
}

case T_PHONG:
  op->phong = TRUE;
  if (size)
```

```
            op->phong_angle = ReadREAL();
          break;

        case T_GROUND:
          op->ground = TRUE;
          break;

        case T_SKY:
          op->sky = TRUE;
          break;

        case T_ROOT:
          op->root = TRUE;
          break;

        case T_SPHERE:
          op->sphere_rad = ReadREAL();
          break;

        case T_DISPLAY:
          op->display = ReadLONG();
          break;

        case T_ENVIRONMENT:
          op->env.ambient_col = ReadVECTOR();
          op->env.fog_col     = ReadVECTOR();
          op->env.ambient_int = ReadREAL();
          op->env.fog_int     = ReadREAL();
          op->env.fog_dist    = ReadREAL();
          op->env.fg_int      = ReadREAL();
          op->env.bg_int      = ReadREAL();
          FSkip(8);
          op->env.fg_inter    = ReadLONG();
          op->env.bg_inter    = ReadLONG();
          op->env.fog         = ReadLONG();
          op->env.sun         = ReadLONG();

          len                 = ReadCHAR();
          op->env.fg_name     = ReadNBYTES(len);

          len                 = ReadCHAR();
          op->env.bg_name     = ReadNBYTES(len);

          op->env.latitude    = ReadREAL();
          op->env.hour        = ReadLONG();
          op->env.day         = ReadLONG();
          op->env.month       = ReadLONG();
          op->env.minute      = ReadLONG();
          op->env.shadow      = ReadLONG();
          break;
      }
    }
    while (val);

    if (flag & 2)
      LoadObjects(); // Load Child Objects
  }
  while (flag & 1);
}

void Loader::LoadEnvironment(void)
{
  LONG    view;
  Real    zoom,kamzoom;
  Vector  offset,kampos,kamdir;

  view    = ReadLONG();
  offset  = ReadVECTOR();
  zoom    = ReadREAL();

  campos  = ReadVECTOR();
```

```
  camdir  = ReadVECTOR();
  camzoom = ReadREAL();
}

Bool Loader::LoadC4D(CHAR *name)
{
  file = FOpen(name);
  if (!file) return FALSE;

  if (ReadLONG()!=FORM) return FALSE;
  Seek(file,4); // Skip file length value
  if (ReadLONG()!=MC4D) return FALSE;

  while (!FileEnd())
  {
    chunkname = ReadLONG();
    chunksize = ReadLONG();

    if (chunkname==PLTF)
      platform = ReadLONG();        // Read platform chunk
    else if (chunkname==OBJ5)
      LoadObjects();                // Read object chunk
    else if (chunkname==MAT4)
      LoadMaterials();              // Read material chunk
    else if (chunkname==UMG4)
      LoadEnvironment();            // Read environment chunk
    else
      FSkip(chunksize);             // Skip unknown chunk

    if (Odd(FPos()))                // IFF specification: pad byte at
      FSkip(1);                     // the end of odd chunk length
}

  return TRUE;
}
```