Melange Quick-start Guide

Sep 2, 2015

What is Melange?

It is MAXON Computer's file exchange and render connection library. Melange allows to write, read and render CINEMA 4D project files (.c4d) from outside the program.

What do you need?

Please download the latest version of the Melange library from developers.maxon.net. The newest versions of the documentation and this Quick-start guide are included in the Melange Zip archive.
Minimum OS Requirements are Windows 7 (or newer) and OS X 10.8.5 (or newer).

How to setup your project?

Extract the archive and add the the include path and regarding library version to your project.

On Windows there're library versions for Visual Studio 2010, 2012 and 2013 (64-bit / MD & MT / debug & release).
On OS X there're libraries for XCode 4.6.2 or newer (64-bit / debug & release / Intel only).

There are a few functions in Melange which have to be overloaded. For the start you can just include "default_alien_overloads.h" to define them. But if you want to use your own memory functions you can have a look inside this header, copy the functions to your project and change them to your needs.
One important function has to be defined in any case : `GetWriterInfo()`. Because in this one the name of the writer and the unique (!) ID has to be set. This identifier will be saved to the project as well and will be displayed in CINEMA 4D at the "Info" section of the project settings.

```
// overload this function and fill in your own unique data
void GetWriterInfo(Int32 &id, String &appname)
{
    id = 1234567; // register your own plugin ID once for your exporter and enter it here (www.plugincafe.com)
    appname = "My Application Name";
}
```

A namespace is defined to avoid trouble when linking Melange to other APIs which may contain the same definitions. Therefore you should add "using namespace _melange_;" to relevant part of your project or use the prefix "_melange_::" for the Melange types explicitly.

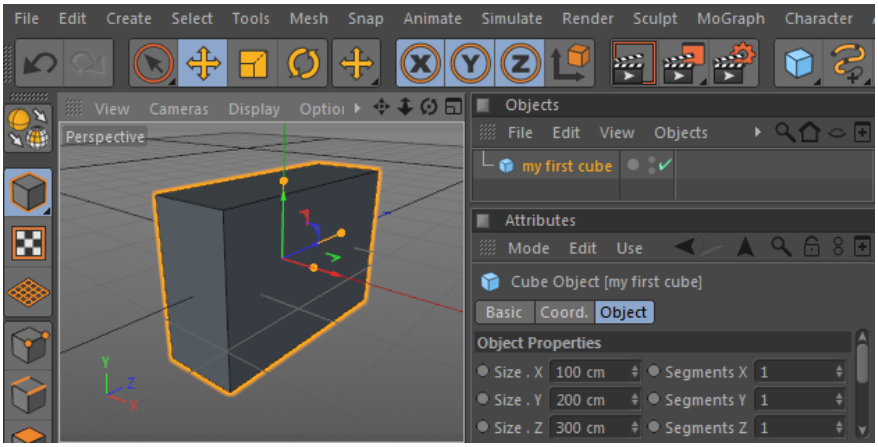
All about writing a CINEMA 4D project file

(The examples do not contain any error checking to keep the code as simple as possible.)

How to create and save a simple c4d project containing only one Cube object?

```
// create a document (includes all the objects, materials and tags)
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create a cube primitive object
BaseObject* myCube = BaseObject::Alloc(Ocube); // other types are Olight, Ocamera, Ospline, ...

// set the name
myCube->SetName("my first cube");

// set length parameter (the IDs can be found in "c4d_parameter_ids.h" or inside the folder "parameter_ids")
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0, 200.0, 300.0));

// insert into the project
c4dDoc->InsertObject(myCube, nullptr);

// write the document (the third parameter is private and must be set to SAVEDOCUMENTFLAGS_0)
SaveDocument(c4dDoc, "simple_project.c4d", SAVEDOCUMENTFLAGS_0);
```

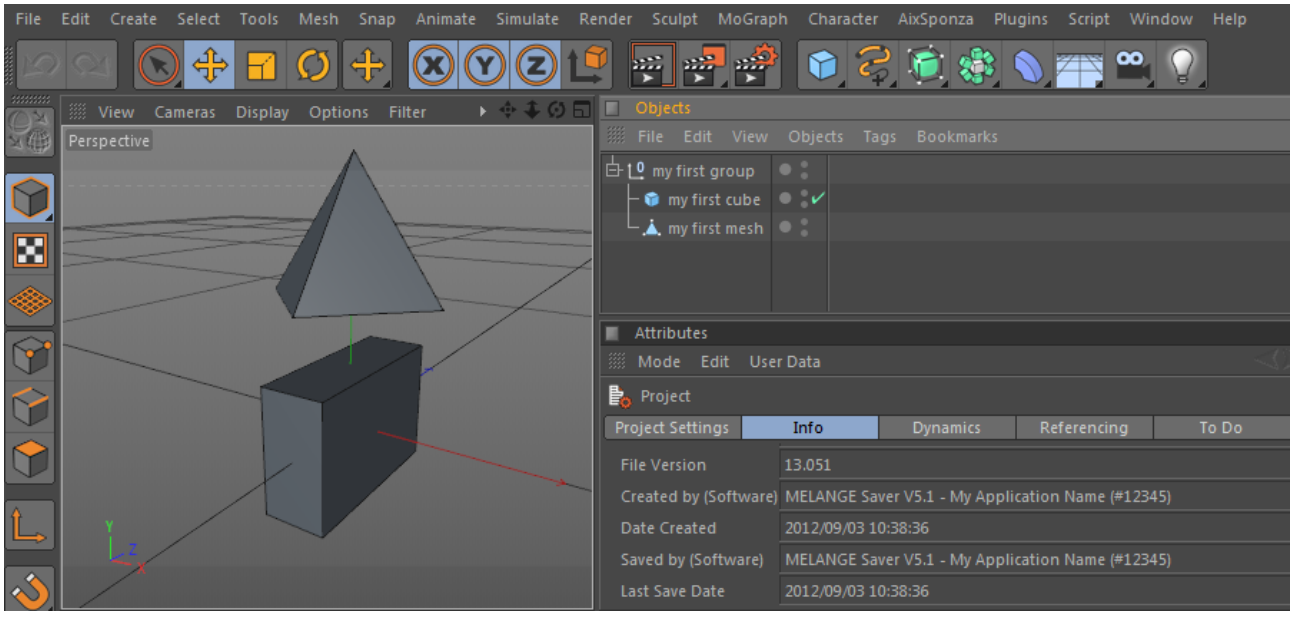
The resulting scene loaded into CINEMA 4D:



How to create and save a more complex c4d project containing also a null and a polygon object?

```
// create a document
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create a null object
BaseObject* myNull = BaseObject::Alloc(Onull);
myNull->SetName("my first null");
c4dDoc->InsertObject(myNull, nullptr);

// create a cube object
BaseObject* myCube = BaseObject::Alloc(Ocube);
myCube->SetName("my first cube");
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0, 200.0, 300.0));
c4dDoc->InsertObject(myCube, myNull); // insert into the project as children of the null object

// create a polygon object
BaseObject* myMesh = BaseObject::Alloc(Opolygon);
myMesh->SetName("my first mesh");
// set position/rotation/scale
myMesh->SetAbsPos(Vector(0.0, 200.0, 0.0));
myMesh->SetAbsRot(Vector(Rad(45.0), 0.0, 0.0));
myMesh->SetAbsScale(Vector(100.0, 100.0, 100.0));
// insert into the project as children of the null object and after the cube object
c4dDoc->InsertObject(myMesh, myNull, myCube);

// resize the object, setup points and polygons
((PolygonObject*)myMesh)->ResizeObject(5, 5, 0); //  ResizeObject(Point Count, Polygon Count, Ngon Count)
// get the pointer to the vertices for writing
Vector *vAdr = ((PolygonObject*)myMesh)->GetPointW();
// get the pointer of the polygons for writing
CPolygon *pAdr = ((PolygonObject*)myMesh)->GetPolygonW();

// set point data Vector(x, y ,z)
vAdr[0] = Vector( 1.0, 0.0, 1.0);
vAdr[1] = Vector(-1.0, 0.0, 1.0);
vAdr[2] = Vector(-1.0, 0.0,-1.0);
vAdr[3] = Vector( 1.0, 0.0,-1.0);
vAdr[4] = Vector( 0.0, 2.0, 0.0);

// set polygon data (point indices CPolygon(a, b, c, d) for quads, CPolygon(a,b,c) for triangles)
pAdr[0] = CPolygon(0, 1, 2, 3);
pAdr[1] = CPolygon(4, 1, 0);
pAdr[2] = CPolygon(4, 2, 1);
pAdr[3] = CPolygon(4, 3, 2);
pAdr[4] = CPolygon(4, 0, 3);

// write the document
SaveDocument(c4dDoc, "complex_project.c4d", SAVEDOCUMENTFLAGS_0);
```

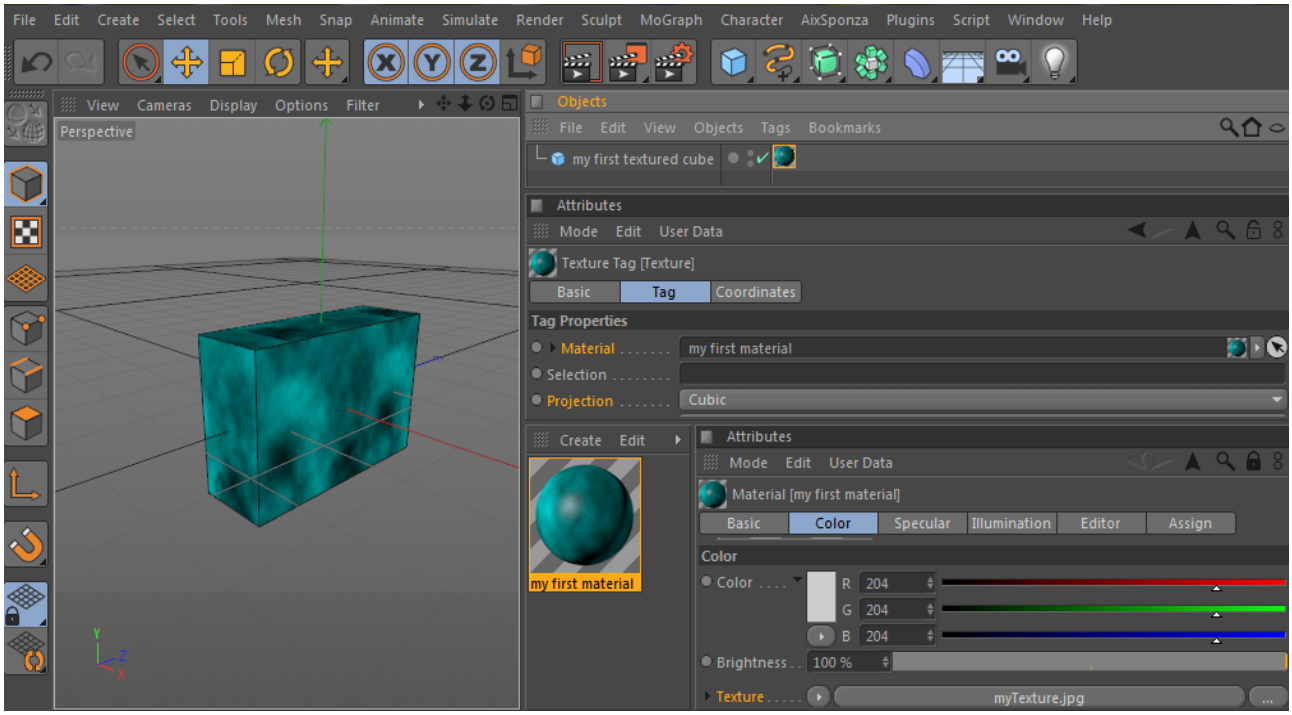
The resulting project loaded into CINEMA 4D:



How to apply materials and textures to the objects?

```
// create a document
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create and add a cube
BaseObject* myCube = BaseObject::Alloc(Ocube);
myCube->SetName("my first textured cube");
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0, 200.0, 300.0));
c4dDoc->InsertObject(myCube, nullptr);

// create a material with a color texture
Material* myMat = (Material*)BaseMaterial::Alloc(Mmaterial);
myMat->SetName("my first material");
myMat->MakeBitmapShader(MATERIAL_COLOR_SHADER, "myTexture.jpg");

// insert the material into the project
c4dDoc->InsertMaterial(myMat);

// create a tag which maps the material to the object (the tag is inserted into the object/document)
BaseTag* myTextureTag = myCube->MakeTag(Ttexture);

// create a link for the new material
BaseLink* myMatLink = BaseLink::Alloc();
myMatLink->SetLink(myMat);

// set material link of the texture tag
myTextureTag->SetParameter(TEXTURETAG_MATERIAL, *myMatLink);
BaseLink::Free(myMatLink);

// set texture projection to cubic
myTextureTag->SetParameter(TEXTURETAG_PROJECTION, TEXTURETAG_PROJECTION_CUBIC);
SaveDocument(c4dDoc, "simple_material_project.c4d", SAVEDOCUMENTFLAGS_0);
```

The resulting project loaded into CINEMA 4D:

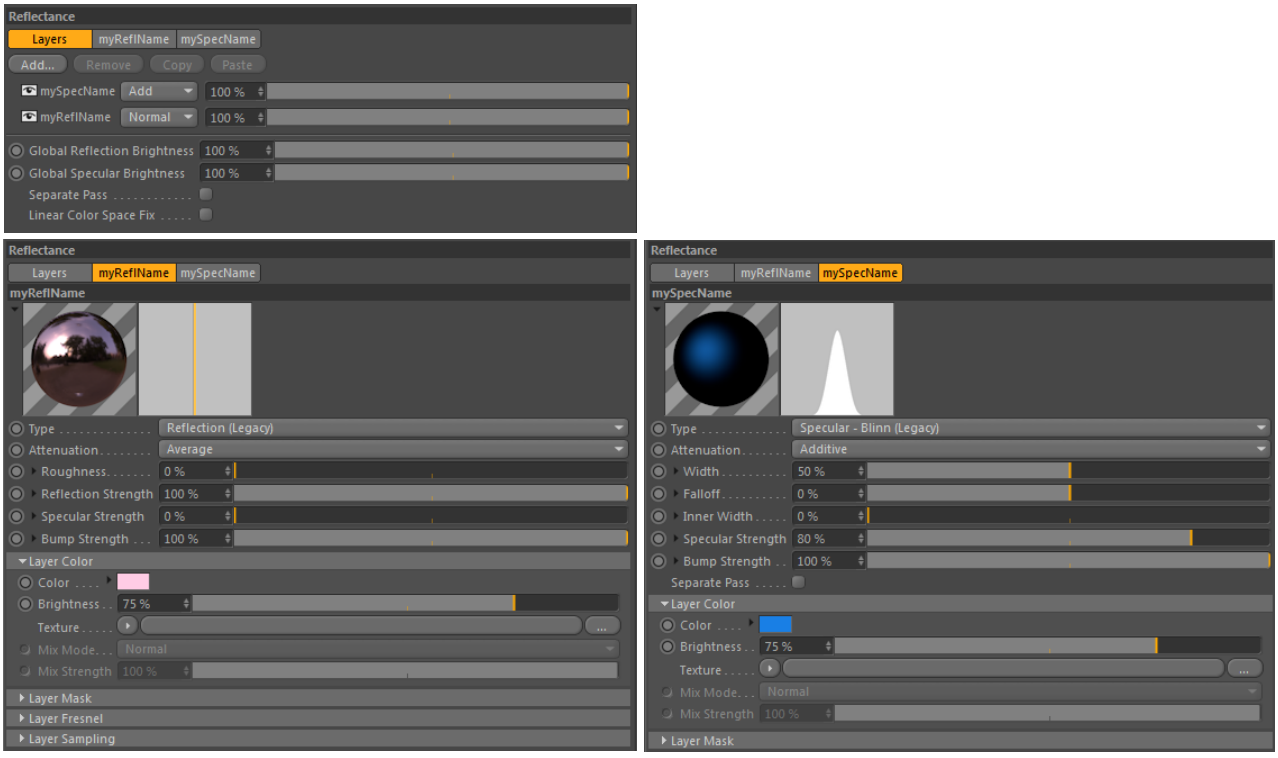


How to add reflection layers to materials?

(Using the previous scene after the material was created.)

```
...
    // turn on the reflection channel
    myMat->SetChannelState(CHANNEL_REFLECTION, true);

    // remove existing reflection layers first (there's a default specular layer which could be used as well)
    myMat->RemoveReflectionAllLayers();

    // get the data reference of the material
    BaseContainer* reflData = myMat->GetDataInstance();

    // add a reflection layer for REFLECTION and SPECULAR
    ReflectionLayer* reflLayer = myMat->AddReflectionLayer();

    // set layer name
    reflLayer->name = String("myReflName");

    // the layer data ID is used as offset for all parameter Ids
    Int32 rDataID = reflLayer->GetDataID();
    reflData->SetInt32 (rDataID + REFLECTION_LAYER_MAIN_BLEND_MODE, MATERIAL_TEXTUREMIXING_NORMAL);
    reflData->SetFloat (rDataID + REFLECTION_LAYER_MAIN_DISTRIBUTION, REFLECTION_DISTRIBUTION_SIMPLE);
    reflData->SetBool  (rDataID + REFLECTION_LAYER_MAIN_ADDITIVE, REFLECTION_ADDITIVE_MODE_AVG);
    reflData->SetVector(rDataID + REFLECTION_LAYER_COLOR_COLOR, Vector(1.0, 0.8, 0.9));
    reflData->SetFloat (rDataID + REFLECTION_LAYER_COLOR_BRIGHTNESS, 0.75);
    reflData->SetFloat (rDataID + REFLECTION_LAYER_MAIN_VALUE_REFLECTION, 1.0);
    reflData->SetFloat (rDataID + REFLECTION_LAYER_MAIN_VALUE_ROUGHNESS, 0);
    reflData->SetFloat (rDataID + REFLECTION_LAYER_MAIN_VALUE_SPECULAR, 0);
    reflData->SetFloat (rDataID + REFLECTION_LAYER_MAIN_VALUE_BUMP, 1.0);

    // and an additional layer for SPECULAR
    ReflectionLayer* specLayer = myMat->AddReflectionLayer();

    // set layer name
    specLayer->name = String("mySpecName");

    // set parameters using the data ID
    Int32 sDataID = specLayer->GetDataID();
    reflData->SetInt32 (sDataID + REFLECTION_LAYER_MAIN_BLEND_MODE, MATERIAL_TEXTUREMIXING_ADD);
    reflData->SetFloat (sDataID + REFLECTION_LAYER_MAIN_DISTRIBUTION, REFLECTION_DISTRIBUTION_SPECULAR_BLINN);
    reflData->SetInt32 (sDataID + REFLECTION_LAYER_MAIN_ADDITIVE, REFLECTION_ADDITIVE_MODE_ADD);
    reflData->SetVector(sDataID + REFLECTION_LAYER_COLOR_COLOR, Vector(0.1, 0.5, 0.9));
    reflData->SetFloat (sDataID + REFLECTION_LAYER_COLOR_BRIGHTNESS, 0.75);
    reflData->SetFloat (sDataID + REFLECTION_LAYER_MAIN_VALUE_REFLECTION, 0.0);
    reflData->SetFloat (sDataID + REFLECTION_LAYER_MAIN_VALUE_ROUGHNESS, 0.5);
    reflData->SetFloat (sDataID + REFLECTION_LAYER_MAIN_VALUE_SPECULAR, 0.8);
    reflData->SetFloat (sDataID + REFLECTION_LAYER_MAIN_VALUE_BUMP, 1.0);
...
```

The resulting layers look like this in CINEMA 4D:



Using the render connection

(The examples does not contain any error checking to keep the code as simple as possible.)

In your Application starting routine call the following initialization functions:

```
// initialize the main thread pointer and timer
GeInitThreads();

// initialize Melange network code
GeIpInitNetwork();

// start the renderer with the path to the executable and the custom port (optional)
// (the current PID is passed to the renderer as well and will shutdown the server if the process ends)
StartRenderServer(PathToCINEMA4D, 1234);
```

For rendering, define your own class derived from IpCommunicationThread:

```
// one thread used to manage the rendering
class RenderThread : public melange::IpCommunicationThread
{
public:
   virtual void Main ()
   {
      // open communication to the server
      // (local host + port used when starting, UID of your application, initial timeout, connection timeout)
      Open("127.0.0.1:1234", UNIQUE_APPLICATION_ID, 10, 60);

      // send the scene to the renderer
      // (pass path of CINEMA 4D file or an optional BaseDocument reference as second parameter)
      SendScenefile(SceneFilePath);

      // start rendering
      // (X resolution, Y resolution, bitmap color mode, allow saving the render image, high priority)
      StartRender(640, 480, COLORRESOLUTION_FLOAT, false, false);

      // get the render image and progress while rendering
      Bool running = true;
      while (!TestBreak() && running)
      {
         // calls your callback "UpdateImage" to get the image update
         GetImageUpdate(MyUpdateImage, nullptr, &running);

         // check for error and how much rendering is done
         GetRenderStatus(nullptr, &rendererror, &percent_done, &renderphase);
      }

      // frees the session on the server (old scenes will be freed automatically if new ones are uploaded)
      FreeSession();

      // ends the communication with the server and shuts it down
      // (you can pass "true" to shutdown the server at this point if no further rendering is needed)
      Close(false);
   }
};

// only needed if you did not shutdown the server in the render thread (Close(false) used)
class ShutDownThread : public melange::IpCommunicationThread
{
public:
   virtual void Main ()
   {
      Open("127.0.0.1:1234", 1, 10, 60);
      Close(true);
   }
};
```

To receive the rendered image define an *UpdateImage* callback:

```
// here the rendered parts of the image are received
void MyUpdateImage(void *userdata, Int32 xpos, Int32 ypos, Int32 xcnt, ColorResolution bpp, UChar* rgba_data)
{
   // xpos & ypos defines the starting position and xcnt the amount of pixel of the RGBA data
   // bpp informs about the color depth (8, 16 or 32 bit)
   // rgba_data contains the image data (4 bytes per pixel)
   // fill your bitmap structure with the given data
   // ...
}
```

To start rendering, create a new render thread from your class and start it:

```
RenderThread*   rThread  = nullptr; // (global defined)
ShutDownThread* sdThread = nullptr; // (global defined)

// if no render thread is active start a new one
if (!rThread) rThread = NewObj(RenderThread);
if (!rThread || rThread->IsRunning()) return;
rThread->Start(true);
```

In your application ending routine shutdown the server, delete the threads and call the following functions:

```
// shutdown the renderer
if (!sdThread) sdThread = NewObj(ShutDownThread);
if (!sdThread || sdThread->IsRunning()) return;
sdThread->Start(true);
sdThread->Wait(); // make sure that the command will be send to the server before execution continue
// delete the threads
DeleteObj(rThread);
DeleteObj(sdThread);

// free Melange network
GeIpCloseNetwork();

// free Melange thread code
GeEndThreads();
```

All about reading a CINEMA 4D project file

(The examples does not contain any error checking to keep the code as simple as possible.)

How to load and browse through a c4d project?

```
// load the project with objects
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS);

// start browsing at the first top level object
BrowseHierarchy(c4dDoc->GetFirstObject());

// delete the project after usage
BaseDocument::Free(c4dDoc);


// define this function to browse through the hierarchy
void BrowseHierarchy(BaseObject* object)
{
   while (object)
   {
      // get basic data
      String name     = object->GetName();
      Int32 type      = object->GetType();      // Ocube, Ospline, Opolygon, Olight, Ocamera, ...
      Matrix ml       = object->GetMl();        // local matrix
      Matrix mg       = object->GetMg();        // global matrix

      // depending on the type, get specific data for the current object
      GeData data; // universal data type, can handle all data formats (Float, Int32, Bool, Vector, ...)
      if (type == Ocamera)
      {
         object->GetParameter(CAMERAOBJECT_FOV, data);
         Float fov = data.GetFloat();
      }
      // some objects have their own type definition
      else if (type == Opolygon)
      {
         PolygonObject* polyObject = (PolygonObject*)object;
         Int32 pointCount = polyObject->GetPointCount();
         Int32 polygonCount = polyObject->GetPolygonCount();
         const Vector* points = polyObject->GetPointR();
         const CPolygon* polys = polyObject->GetPolygonR();

         // access the polygons and points
         for (Int32 i = 0; i < polygonCount; i++)
         {
            // get the 3 / 4 points of the triangle / quadrangle
            Vector pointA = points[polys[i].a];
            Vector pointB = points[polys[i].b];
            Vector pointC = points[polys[i].c];

            // only if the index of 3rd and 4th point is different we have a quad, if not we have a triangle
            if (polys[i].c != polys[i].d)
            {
               Vector pointD = points[polys[i].d];
            }
         }
      }

      // do the children
      BrowseHierarchy(object->GetDown());

      // get the next object
      object = object->GetNext();
   }
}
```

How to access the materials of a c4d project?

```
// load the project with materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_MATERIALS);

// get the first material from the document and go through the whole list
BaseMaterial* mat = c4dDoc->GetFirstMaterial();
while (mat)
{
    // basic data
    String name = mat->GetName();

    // check if the material is a standard material
    if (mat->GetType() == Mmaterial)
    {
        // check if the given channel is used in the material
        if (((Material*)mat)->GetChannelState(CHANNEL_COLOR)) // CHANNEL_LUMINANCE, CHANNEL_BUMP, ...
        {
            // channel color is turned on

            // get the color and brightness
            GeData data;
            mat->GetParameter(MATERIAL_COLOR_COLOR, data);
            Vector color = data.GetVector();
            mat->GetParameter(MATERIAL_COLOR_BRIGHTNESS, data);
            Float brightness = data.GetFloat();
        }

        // special reflectance channel since release 16
        if (((Material*)mat)->GetChannelState(CHANNEL_REFLECTION))
        {
            BaseContainer* reflectanceData = ((Material*)mat)->GetDataInstance();

            for (Int32 refLayIdx = 0; refLayIdx < ((Material*)mat)->GetReflectionLayerCount(); refLayIdx++)
            {
                ReflectionLayer* refLay = ((Material*)mat)->GetReflectionLayerIndex(refLayIdx);
                if (!refLay)
                    continue;

                Int32 dataId     = refLay->GetDataID();
                Int32 refType    = reflectanceData->GetInt32(dataId + REFLECTION_LAYER_MAIN_DISTRIBUTION);
                Float brightness = reflectanceData->GetFloat(dataId + REFLECTION_LAYER_COLOR_BRIGHTNESS);
                Float reflection = reflectanceData->GetFloat(dataId + REFLECTION_LAYER_MAIN_VALUE_REFLECTION);
                Float specular   = reflectanceData->GetFloat(dataId + REFLECTION_LAYER_MAIN_VALUE_SPECULAR);
            }
        }
    }
    else
    {
        // plugin material
        // parameters depend on the type, get with GetParameter()
        // ...
    }

    mat = mat->GetNext();
}

// delete the project after usage
BaseDocument::Free(c4dDoc);
```

How to get the assigned material of an object?

```
// load the project with objects and materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS | SCENEFILTER_MATERIALS);

// get the first object
BaseObject* object = c4dDoc->GetFirstObject();

// get the texture tag holding the link to the assigned material
BaseTag* tag = object->GetTag(Ttexture);

// get the assigned material from the tag
BaseMaterial* assignedMat = ((TextureTag*)tag)->GetMaterial();

// delete the project after usage
BaseDocument::Free(c4dDoc);
```

How to use the extended loading functionality to get a geometric description for all objects?

Important note: This is only possible for projects saved with the "Save Polygons for Melange" option turned on in the CINEMA 4D preferences!

```
// load the project with objects and materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS | SCENEFILTER_MATERIALS);

// call all the Execute() of the document
c4dDoc->CreateSceneFromC4D();

// delete the project after usage
BaseDocument::Free(c4dDoc);
```

The following "alien" definitions are important. Add these classes for the objects you can transfer to your own internal data structures. Of course `AlienPolygonObjectData` has to be implemented in any case.

```
// self defined PolygonObjectData with own members and definition of the Execute() method
class AlienPolygonObjectData : public PolygonObjectData
{
    INSTANCEOF(AlienPolygonObjectData, PolygonObjectData) // important to get Execute() called!
public:
    // useful to store your own material and layer ID
    Int32 layID;
    Int32 matID;

    virtual Bool Execute()
    {
        // get object pointer
        PolygonObject* op = (PolygonObject*)GetNode();

        // access the geometry data as described above
        // ...

        return true;
    }
};
```

```
// this self defined alien primitive object data will "understand" a Cube as parametric Cube
// all other objects are unknown and we will get a polygonal description for them
class AlienPrimitiveObjectData : public NodeData
{
    INSTANCEOF(AlienPrimitiveObjectData, NodeData)

    Int32 type_id;
public:

    AlienPrimitiveObjectData(Int32 id) : type_id(id) {}

    virtual Bool Execute()
    {
        // decide the object type
        if (type_id == Ocube)
        {
            // get the size of the cube
            GeData cubeData;
            GetNode()->GetParameter(PRIM_CUBE_LEN, cubeData);
            Vector xyzLength = cubeData.GetVector();

            // create your own parametric cube with length X, Y and Z
            // ...

            // returning "true" we will NOT get another call of PolygonObjectData Execute()
            return true;
        }

        // return "false" to get a simpler geometric description of objects like Osphere, Ocone, ...
        // the Execute() of AlienPolygonObjectData will be called
        return false;
    }
};
```

Now change the function `AllocAlienObjectData` to create the custom alien object data at allocation:

```
// this function has to be overloaded in any case (predefined in default_alien_overloads.h)
// it allocates the scene objects (primitives, camera, light, etc.)
NodeData *AllocAlienObjectData(Int32 id, Bool &known)
{
   NodeData *m_data = nullptr;
   known = true;

   switch (id)
   {
      // supported element types
      case Opolygon: m_data = NewObj(AlienPolygonObjectData);       break;
      case Osphere:  m_data = NewObj(AlienPrimitiveObjectData, id); break;
      case Ocube:    m_data = NewObj(AlienPrimitiveObjectData, id); break;
      case Oplane:   m_data = NewObj(AlienPrimitiveObjectData, id); break;
      case Ocone:    m_data = NewObj(AlienPrimitiveObjectData, id); break;
      case Otorus:   m_data = NewObj(AlienPrimitiveObjectData, id); break;
      case Ocylinder:m_data = NewObj(AlienPrimitiveObjectData, id); break;
      ...

      default:       known = false; break;
   }

   return m_data;
};
```

Note: The example “commandline” which is included in the SDK archive uses this method. Please have a look at it for further information.