Display driver memory buffer | API

C4DtoA offers access to the memory buffers while rendering via the display driver which can be useful to efficiently display the render in external viewers.

To be able to access the memory buffers in your plugin first you need to enable the feature by setup the render settings as described in Custom display driver | API.

```
// from c4dtoa_symols.h
#define ARNOLD_RENDER_OVERRIDES 1038579
#define ARNOLD_DRIVER 1030141
#define ARNOLD_AOV 1030369
#define ARNOLD_RENDER_COMMAND 1038578 // 1: start IPR, 2: stop IPR, 3: pause/unpause IPR, 4: update IPR, 5:
render
// from arnold_driver.h
#define C4DAI_DRIVER_TYPE 101
// from NodeIds.h
#define C4DAIN_DRIVER_C4D_DISPLAY 1927516736
// from arnold_renderer.h
#define C4DTOA_RENDEROVERRIDE_DRIVER 1000                 // [Link] The driver object used as the display
driver. The default display driver is used when does not set.
#define C4DTOA_RENDEROVERRIDE_ONLY_BEAUTY 1001            // [Bool] If true then no AOVs of the display driver
are exported. Default is true.
#define C4DTOA_RENDEROVERRIDE_EXPORT_ALL_DRIVERS 1002     // [Bool] If false then only the display driver is
exported. Default is true.
#define C4DTOA_RENDEROVERRIDE_CAMERA 1003                 // [Link] The camera object used for rendering. The
active camera is used when does not set.
#define C4DTOA_RENDEROVERRIDE_XRES 1004                   // [Int32] Resolution width.
#define C4DTOA_RENDEROVERRIDE_YRES 1005                   // [Int32] Resolution height.
#define C4DTOA_RENDEROVERRIDE_USE_REGION 1006             // [Bool] If true then a region is rendered.
#define C4DTOA_RENDEROVERRIDE_REGION_X1 1007              // [Int32] Left border of the render region.
#define C4DTOA_RENDEROVERRIDE_REGION_X2 1008              // [Int32] Right border of the render region.
#define C4DTOA_RENDEROVERRIDE_REGION_Y1 1009              // [Int32] Top border of the render region.
#define C4DTOA_RENDEROVERRIDE_REGION_Y2 1010              // [Int32] Bottom border of the render region.
#define C4DTOA_RENDEROVERRIDE_AA_SAMPLES 1011             // [Int32] Camera AA samples.
#define C4DTOA_RENDEROVERRIDE_IGNORE_MOTION_BLUR 1012     // [Bool] Ignore motion blur flag.
#define C4DTOA_RENDEROVERRIDE_IGNORE_SUBDIV 1013          // [Bool] Ignore subdivision flag.
#define C4DTOA_RENDEROVERRIDE_IGNORE_DISPLACEMENT 1014    // [Bool] Ignore displacement flag.
#define C4DTOA_RENDEROVERRIDE_IGNORE_BUMP 1015            // [Bool] Ignore bump flag.
#define C4DTOA_RENDEROVERRIDE_IGNORE_SSS 1016             // [Bool] Ignore SSS flag.
#define C4DTOA_RENDEROVERRIDE_FRAME_START 1017            // [Int32] Start frame when rendering an animation.
#define C4DTOA_RENDEROVERRIDE_FRAME_END 1018              // [Int32] End frame when rendering an animation.
#define C4DTOA_RENDEROVERRIDE_FRAME_STEP 1019             // [Int32] Frame step when rendering an animation.
#define C4DTOA_RENDEROVERRIDE_PROGRESSIVE_REFINEMENT 1020 // [Bool] Enable / disable progressive refinement in
the IPR.
#define C4DTOA_RENDEROVERRIDE_PROGRESSIVE_SAMPLES 1021    // [Int32] Level of progressive refinement (e.g. -3).
#define C4DTOA_RENDEROVERRIDE_PROGRESSIVE_RENDER 1022     // [Bool] Enable / disable progressive rendering.
#define C4DTOA_RENDEROVERRIDE_RENDER_TO_BUFFER 1023       // [Bool] Render to a memory buffer instead of the
UI. Works only with a c4dtoa_display_driver.
#define C4DTOA_RENDEROVERRIDE_RENDER_TO_BUFFER_DATA 1024  // [BaseContainer] Output data of the render to
buffer feature.


class MyRenderer
{
private:
    BaseObject* m_driver;
    BaseObject* m_diffuseDirectAOV;
    BaseObject* m_specularDirectAOV;

private:
    void SetupRenderOverrides()
    {
        // create a display driver
        // NOTE you can use a display driver from the scene if it exists
        m_driver = BaseObject::Alloc(ARNOLD_DRIVER);
        m_driver->GetDataInstance()->SetInt32(C4DAI_DRIVER_TYPE, C4DAIN_DRIVER_C4D_DISPLAY);
        m_driver->SetName(String("My Driver"));
```

```
        // create some AOVs
        m_diffuseDirectAOV = BaseObject::Alloc(ARNOLD_AOV);
        m_diffuseDirectAOV->SetName(String("diffuse_direct"));
        m_diffuseDirectAOV->InsertUnder(m_driver);
        m_specularDirectAOV = BaseObject::Alloc(ARNOLD_AOV);
        m_specularDirectAOV->SetName(String("specular_direct"));
        m_specularDirectAOV->InsertUnder(m_driver);

        // setup render settings
        BaseContainer settings;
        settings.SetLink(C4DTOA_RENDEROVERRIDE_DRIVER, m_driver); // set the custom display driver
        settings.SetBool(C4DTOA_RENDEROVERRIDE_ONLY_BEAUTY, FALSE); // render with AOVs
        settings.SetBool(C4DTOA_RENDEROVERRIDE_EXPORT_ALL_DRIVERS, FALSE); // only the display driver
        settings.SetInt32(C4DTOA_RENDEROVERRIDE_XRES, 800); // x res
        settings.SetInt32(C4DTOA_RENDEROVERRIDE_YRES, 600); // y res
        settings.SetBool(C4DTOA_RENDEROVERRIDE_PROGRESSIVE_REFINEMENT, FALSE); // no progressive refinement in
the IPR
        settings.SetInt32(C4DTOA_RENDEROVERRIDE_FRAME_START, 0);
        settings.SetInt32(C4DTOA_RENDEROVERRIDE_FRAME_END, 0);
        settings.SetInt32(C4DTOA_RENDEROVERRIDE_FRAME_STEP, 1);
        // enable 'render to buffer' feature
        settings.SetBool(C4DTOA_RENDEROVERRIDE_RENDER_TO_BUFFER, TRUE);
        // add the settings to the document
        BaseDocument* doc = GetActiveDocument();
        doc->GetSettingsInstance((Int32)DOCUMENTSETTINGS_DOCUMENT)->SetContainer(ARNOLD_RENDER_OVERRIDES,
settings);
    }

    void CleanupRenderOverrides()
    {
        if (m_diffuseDirectAOV)
        {
            m_diffuseDirectAOV->Remove();
            BaseObject::Free(m_diffuseDirectAOV);
        }
        m_diffuseDirectAOV = nullptr;
        if (m_specularDirectAOV)
        {
            m_specularDirectAOV->Remove();
            BaseObject::Free(m_specularDirectAOV);
        }
        m_specularDirectAOV = nullptr;
        if (m_driver)
        {
            m_driver->Remove();
            BaseObject::Free(m_driver);
        }
        m_driver = nullptr;
    }

public:
    Renderer()
    {
       m_driver = nullptr;
       m_diffuseDirectAOV = nullptr;
       m_specularDirectAOV = nullptr;
    }

    ~Renderer()
    {
        CleanupRenderOverrides();
    }

    void StartIPR()
    {
        SetupRenderOverrides();
        CallCommand(ARNOLD_RENDER_COMMAND, 1);
    }

    void StopIPR()
    {
```

```
        CallCommand(ARNOLD_RENDER_COMMAND, 2);
        CleanupRenderOverrides();
    }
};
```

Once the feature is enabled C4DtoA sends the following plugin messages which you can handle in your custom plugin.

Message id	Parameters
MSG_C4DTOA_RENDER_TO_BUFFER_INIT (1041303)	
MSG_C4DTOA_RENDER_TO_BUFFER_START (1041295)	• C4DTOA_RENDERTOBUFFER_NUM_BUFFERS (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_FIRST + [AOV index] (BaseContainer) • C4DTOA_RENDERTOBUFFER_BUFFER_NAME (String) • C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE (Int32) • C4DTOA_RENDERTOBUFFER_XRES (Int32) • C4DTOA_RENDERTOBUFFER_YRES (Int32)
MSG_C4DTOA_RENDER_TO_BUFFER_PREPARE_BUCKET (1041302)	• C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_XO (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_YO (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_X (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_Y (Int32)
MSG_C4DTOA_RENDER_TO_BUFFER_WRITE_BUCKET (1041297)	• C4DTOA_RENDERTOBUFFER_NUM_BUFFERS (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_FIRST + [AOV index] (BaseContainer) • C4DTOA_RENDERTOBUFFER_BUFFER_NAME (String) • C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_DATA (Void*) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_XO (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_YO (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_X (Int32) • C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_Y (Int32)
MSG_C4DTOA_RENDER_TO_BUFFER_END (1041296)	
MSG_C4DTOA_RENDER_TO_BUFFER_FINISH (1041304)	

The parameters used in the following messages are:

Parameter	ID	Type	Description
C4DTOA_RENDERTOBUFFER_NUM_BUFFERS	100	Int32	Number of render output buffers (AOVs).
C4DTOA_RENDERTOBUFFER_XRES	101	Int32	Resolution width.
C4DTOA_RENDERTOBUFFER_YRES	102	Int32	Resolution height.
C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_XO	103	Int32	The top-left x coordinate of the bucket.
C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_YO	104	Int32	The top-left y coordinate of the bucket.
C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_X	105	Int32	The width of the bucket.
C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_Y	106	Int32	The height of the bucket.
C4DTOA_RENDERTOBUFFER_BUFFER_FIRST	1000	BaseContainer	Each buffer is stored in a BaseContainer started from this ID.

Parameters in the buffer containers:

Parameter	ID	Type	Description
C4DTOA_RENDERTOBUFFER_BUFFER_NAME	100	String	Name of the AOV (e.g. RGBA, diffuse_direct, etc.)

C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE	101	Int32	Data type of the AOV. (e.g. AI_TYPE_FLOAT, etc.)
C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA	102	Void*	Memory buffer of the AOV. The pixels are stored line-by-line from the top left corner. The data type depends on the C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE parameter.

The ids of the parameters above defined in the ***res/description/arnold_renderer.h*** header file.

This is an example of writing the render output to bitmaps:

```
// from c4dtoa_symols.h
#define MSG_C4DTOA_RENDER_TO_BUFFER_INIT 1041303
#define MSG_C4DTOA_RENDER_TO_BUFFER_START 1041295
#define MSG_C4DTOA_RENDER_TO_BUFFER_PREPARE_BUCKET 1041302
#define MSG_C4DTOA_RENDER_TO_BUFFER_WRITE_BUCKET 1041297
#define MSG_C4DTOA_RENDER_TO_BUFFER_END 1041296
#define MSG_C4DTOA_RENDER_TO_BUFFER_FINISH 1041304
// from arnold_renderer.h
#define C4DTOA_RENDERTOBUFFER_NUM_BUFFERS 100            // [Int32] Number of render output buffers (AOVs).
#define C4DTOA_RENDERTOBUFFER_XRES 101                   // [Int32] Resolution width.
#define C4DTOA_RENDERTOBUFFER_YRES 102                   // [Int32] Resolution height.
#define C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_XO 103       // [Int32] The top-left x coordinate of the bucket.
#define C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_YO 104       // [Int32] The top-left y coordinate of the bucket.
#define C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_X 105   // [Int32] The width of the bucket.
#define C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_Y 106   // [Int32] The height of the bucket.
#define C4DTOA_RENDERTOBUFFER_BUFFER_FIRST 1000          // [BaseContainer] Each buffer is stored in a
BaseContainer started from this ID.
#define C4DTOA_RENDERTOBUFFER_BUFFER_NAME 100            // [String] Name of the AOV (e.g. RGBA,
diffuse_direct, etc.)
#define C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE 101       // [Int32] Data type of the AOV. (e.g. AI_TYPE_FLOAT,
etc.)
#define C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA 102     // [Void*] Memory buffer of the AOV. The pixels are
stored line-by-line from the top left corner.
// from ai_params.h
#define AI_TYPE_INT          0x01  // Int32
#define AI_TYPE_UINT         0x02  // UInt32
#define AI_TYPE_FLOAT        0x04  // Float32
#define AI_TYPE_RGB          0x05  // 3 * Float32
#define AI_TYPE_RGBA         0x06  // 4 * Float32
#define AI_TYPE_VECTOR       0x07  // 3 * Float32
#define AI_TYPE_VECTOR2      0x09  // 2 * Float32
#define AI_TYPE_UNDEFINED    0xFF

class BitmapSaver
{
private:
    std::map<std::string, BaseBitmap*> m_bitmaps;

public:
    ~BitmapSaver()
    {
        Free();
    }

    BitmapSaver* GetBitmap(const String& aovNameStr)
    {
        char aovName[256];
        aovNameStr.GetCString(aovName, 256);
        std::map<std::string, BaseBitmap*>::iterator it = m_bitmaps.find(aovName);
        if (it != m_bitmaps.end())
            return it->second;
        return nullptr;
    }

    std::map<std::string, BaseBitmap*> GetBitmaps()
    {
        return m_bitmaps;
    }
```

```
    ///
    /// Allocates the bitmaps.
    ///
    void Init(BaseContainer* data)
    {
        Int32 xres = data->GetInt32(C4DTOA_RENDERTOBUFFER_XRES);
        Int32 yres = data->GetInt32(C4DTOA_RENDERTOBUFFER_YRES);

        // initialize the bitmaps
        Int32 numBuffers = data->GetInt32(C4DTOA_RENDERTOBUFFER_NUM_BUFFERS);
        for (int i = 0; i < numBuffers; i++)
        {
            BaseContainer* aovData = data->GetContainerInstance(C4DTOA_RENDERTOBUFFER_BUFFER_FIRST + i);
            const String& aovNameStr = aovData->GetString(C4DTOA_RENDERTOBUFFER_BUFFER_NAME);
            char aovName[256];
            aovNameStr.GetCString(aovName, 256);

            // allocate the bitmap
            GePrintF("Allocate bitmap of %s: %d x %d", aovName, xres, yres);
            bitmap = BaseBitmap::Alloc();
            bitmap->Init(xres, yres);
            m_bitmaps[aovName] = bitmap;
        }
    }

    ///
    /// Writes the buckets to the bitmaps.
    ///
    void WriteBucket(BaseContainer* data)
    {
        int bucket_xo = data->GetInt32(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_XO);
        int bucket_yo = data->GetInt32(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_YO);
        int bucket_size_x = data->GetInt32(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_X);
        int bucket_size_y = data->GetInt32(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_SIZE_Y);

        //GePrintF("Write bucket: %d %d | %d %d", bucket_xo, bucket_yo, bucket_size_x, bucket_size_y);
        int bitmapBufferSize = COLORBYTES_RGBf * bucket_size_x;
        UChar* bitmapBuffer = NewMemClear(UChar, bitmapBufferSize);


        // copy the bucket data to the bitmap
        Int32 numBuffers = data->GetInt32(C4DTOA_RENDERTOBUFFER_NUM_BUFFERS);
        for (Int32 i = 0; i < numBuffers; i++)
        {
            BaseBitmap* bitmap = GetBitmap(aovNameStr);
            if (!bitmap)
                continue;

            BaseContainer* aovData = data->GetContainerInstance(C4DTOA_RENDERTOBUFFER_BUFFER_FIRST + i);
            const String& aovNameStr = aovData->GetString(C4DTOA_RENDERTOBUFFER_BUFFER_NAME);
            int dataType = aovData->GetInt32(C4DTOA_RENDERTOBUFFER_BUFFER_DATA_TYPE);

            unsigned int pixelIndex = 0;
            for (int y = 0; y < bucket_size_y; y++)
            {
                PIX_F* bi = (PIX_F*)bitmapBuffer;
                for (int x = 0; x < bucket_size_x; x++)
                {
                    switch (dataType)
                    {
                        case AI_TYPE_RGBA:
                        {
                            Float32* bucketData = (Float32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = bucketData[4 * pixelIndex + 0];
                            bi[1] = bucketData[4 * pixelIndex + 1];
                            bi[2] = bucketData[4 * pixelIndex + 2];
                            break;
                        }
                        case AI_TYPE_RGB:
```

```
                        case AI_TYPE_VECTOR:
                        {
                            Float32* bucketData = (Float32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = bucketData[3 * pixelIndex + 0];
                            bi[1] = bucketData[3 * pixelIndex + 1];
                            bi[2] = bucketData[3 * pixelIndex + 2];
                            break;
                        }
                        case AI_TYPE_VECTOR2:
                        {
                            Float32* bucketData = (Float32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = bucketData[2 * pixelIndex + 0];
                            bi[1] = bucketData[2 * pixelIndex + 1];
                            bi[2] = 0.0f;
                            break;
                        }
                        case AI_TYPE_FLOAT:
                        {
                            Float32* bucketData = (Float32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = bucketData[pixelIndex];
                            bi[1] = bucketData[pixelIndex];
                            bi[2] = bucketData[pixelIndex];
                            break;
                        }
                        case AI_TYPE_INT:
                        {
                            Int32* bucketData = (Int32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = (Float32)bucketData[pixelIndex];
                            bi[1] = (Float32)bucketData[pixelIndex];
                            bi[2] = (Float32)bucketData[pixelIndex];
                            break;
                        }
                        case AI_TYPE_UINT:
                        {
                            UInt32* bucketData = (UInt32*)aovData->GetVoid
(C4DTOA_RENDERTOBUFFER_BUFFER_BUCKET_DATA);
                            bi[0] = (Float32)bucketData[pixelIndex];
                            bi[1] = (Float32)bucketData[pixelIndex];
                            bi[2] = (Float32)bucketData[pixelIndex];
                            break;
                        }
                    }

                    bitmap->SetPixelCnt(bucket_xo, bucket_yo + y, bucket_size_x, bitmapBuffer, COLORBYTES_RGBf,
COLORMODE_RGBf, PIXELCNT_0);
                    bi += 3;
                    pixelIndex++;
                }
            }
        }
        DeleteMem(bitmapBuffer);
    }

    void Free()
    {
        for (std::map<std::string, BaseBitmap*>::iterator it = m_bitmaps.begin(); it != m_bitmaps.end(); it++)
        {
            BaseBitmap* bitmap = it->second;
            BaseBitmap::Free(bitmap);
        }
        m_bitmaps.clear();
    }
};


BitmapSaver bitmapSaver;
```

```
Bool PluginMessage(Int32 id, void *data)
{
    switch (id)
    {
        ///////////
        // render to buffer callbacks
        ///////////

        // render init
        case MSG_C4DTOA_RENDER_TO_BUFFER_INIT:
        {
            GePrintF("Render to Buffer init message received");
            break;
        }

        // render iteration start
        case MSG_C4DTOA_RENDER_TO_BUFFER_START:
        {
            GePrintF("Render to Buffer start message received");
            bitmapSaver.Init((BaseContainer*)data);
            break;
        }

        // write bucket
        case MSG_C4DTOA_RENDER_TO_BUFFER_WRITE_BUCKET:
        {
            GePrintF("Render to Buffer write bucket message received");
            bitmapSaver.WriteBucket((BaseContainer*)data);
            break;
        }

        // render iteration end
        case MSG_C4DTOA_RENDER_TO_BUFFER_END:
        {
            GePrintF("Render to Buffer end message received");
            bitmapSaver.Free((BaseContainer*)data);
            break;
        }

        // render finish
        case MSG_C4DTOA_RENDER_TO_BUFFER_FINISH:
        {
            GePrintF("Render to Buffer finish message received");
            break;
        }
    }

    return FALSE;
}
```