



US Patent No. 6,687,003 • AU Patent No. 763370 • EP Patent No. 1131658

ColorCode RT Encoder™ SDK

Whitepaper

The ColorCode RT Encoder™ SDK is integrated through the use of a DLL file containing code needed to do full ColorCode 3-D encoding. The DLL will try to select the best and fastest way to do the encoding, and will, if needed, automatically fallback to a method that will work. All graphic cards that are able to do the encoding are supported. Your own system will have to take care of the actual user interaction if a user does not have the required hardware. (A dialog box, no stereo, etc.)

Below are 5 steps that you will have to do in your system:

1. Render your scene seen from the LEFT eye to a render target texture.
2. Render your scene seen from the RIGHT eye to a render target texture.
3. Set the LEFT texture in texture slot 0.
4. Set the RIGHT texture in texture slot 1.
5. Use the APPLY method in the ColorCode 3-D plug-in. This method will take the two input textures and composite them into a single image displayed on the currently set render target (typically the screen). Normally you will set the whole target/screen, but if needed, you can output to only a subset.

We have attached the current DX8 based sample cpp file to demonstrate it for you. The file uses a very simple 3D rendering pipeline, but shows exactly what you need to do for a more complicated system too:

Note: the following • 1 to • 15 refers to the code lines in the sample cpp file.

- 1 This is the main entry point, the following lines just setup a rendering window, a 3D device, some test geometry and the two render targets mentioned above.
- 2 This is where the actual plug-in is loaded - in • 3 the creation is made. In this method, the shader code is attached, a rendering quad's geometry is allocated (this quad is used to output the final result) and a volume texture for some special internal 3D colorspace tables.
- 4 Here the scene is rendered, first the LEFT, then the RIGHT (• 5).
- 6 We are ready to apply final ColorCode 3-D compositing. We start out by setting our render target to the screen. This is important since the last render target set was the RIGHT texture. We could choose to use a third off-screen render target if needed.
- 7 We clear the render target colors and z values.

- 8 and • 14. Since we just set the render target we need to call `beginscene/endscene`. See DX documentation for this.
- 9 to • 10. We want to fill the whole screen. In this case we are running in 800x600 and consequently set the size to that.
- 11 to • 12. This is where we set our LEFT and RIGHT texture renderings of our scene to texture slot 0 and 1 in DX. It's important that LEFT is in 0 and RIGHT in 1, since the ColorCode 3-D processing code will assume that.
- 13 We are ready to go! We call the `apply` method in the plug-in. It will return, and a ColorCode 3-D encoded image of the LEFT and RIGHT textures will be shown on the screen/render target.
- 15 The rendering is presented (flipped) on the screen.

The interface and methods are exactly the same for the DirectX 9 and OpenGL versions.

```

/*=====
Main.cpp: SampleDX8 file.
Copyright 1993-2009 Digital Arts.

Revision history:
* Created 28/11/2005 by Poul Hansen & Thomas Rued
=====*/

#include <crtdbg.h> // For run-time memoryleak info
#include <d3d8.h>

#include <daColorCode.h>
#include <daGeometry.h>

#include <math.h>

// Globals
bool exitapp=false;
HWND hwnd=NULL;

LPDIRECT3D8 D3D=NULL;
LPDIRECT3DDEVICE8 D3Ddevice=NULL;
D3DPRESENT_PARAMETERS D3Dpp;

LPDIRECT3DSURFACE8 screentarget=NULL;
LPDIRECT3DSURFACE8 screenzbuffertarget=NULL;
LPDIRECT3DTEXTURE8 left=NULL;
LPDIRECT3DTEXTURE8 right=NULL;
LPDIRECT3DSURFACE8 lefttarget=NULL;
LPDIRECT3DSURFACE8 righttarget=NULL;
LPDIRECT3DSURFACE8 zbuffertarget=NULL;

LPDIRECT3DVERTEXBUFFER8 vertexbuffer=NULL;
LPDIRECT3DINDEXBUFFER8 indexbuffer=NULL;

#define DA_RENDERTARGETWIDTH 1024
#define DA_RENDERTARGETHEIGHT 1024

#define DA_FULLSCREEN false

```

```
// Win32 window setup
LRESULT CALLBACK WndProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    switch (message)
    {
    case WM_DESTROY:
        exitapp=true;
        break;

    case WM_CLOSE:
        exitapp=true;
        break;
    }

    return DefWindowProc(hwnd, message, wParam, lParam);
};
```

```

void SetupWindows(HINSTANCE hThisInstance, int nCmdShow)
{
    RECT rect={0,0,800,600};
    AdjustWindowRectEx(&rect,WS_OVERLAPPED | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX, FALSE, WS_EX_APPWINDOW);

    WNDCLASSEX wndclass;
    wndclass.cbSize      = sizeof(WNDCLASSEX);
    wndclass.style       = CS_DBLCLKS;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance  = hThisInstance;
    wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wndclass.lpszMenuName = NULL;
    wndclass.lpszClassName = "ColorCode3D - DirectX8 sample application";
    wndclass.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClassEx(&wndclass);

    hwnd = CreateWindowEx(WS_EX_APPWINDOW, // window style
        "ColorCode3D - DirectX8 sample application", // window class name
        "ColorCode3D - DirectX8 sample application", // window caption
        WS_OVERLAPPED | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX, // window style
        (GetSystemMetrics(SM_CXFULLSCREEN)-rect.right)/2, // initial x position
        (GetSystemMetrics(SM_CYFULLSCREEN)-rect.bottom)/2, // initial y position
        rect.right-rect.left, // initial x size
        rect.bottom-rect.top, // initial y size
        NULL, // parent window handle
        NULL, // window menu handle
        hThisInstance, // program instance handle
        NULL); // creation parameters

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    SetFocus(hwnd);
};

```

```

// Direct3D setup
void SetupDevice()
{
    D3D = Direct3DCreate8(D3D_SDK_VERSION);

    D3DDISPLAYMODE mode;
    D3D->GetAdapterDisplayMode(0,&mode);

    ZeroMemory(&D3Dpp, sizeof(D3Dpp));
    D3Dpp.BackBufferWidth = 800;
    D3Dpp.BackBufferHeight = 600;
    D3Dpp.BackBufferFormat = mode.Format;
    D3Dpp.BackBufferCount=1;
    D3Dpp.MultiSampleType = D3DMULTISAMPLE_NONE;
    D3Dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    D3Dpp.hDeviceWindow = hwnd;
    D3Dpp.Windowed = (DA_FULLSCREEN!=true);
    D3Dpp.EnableAutoDepthStencil = TRUE;
    D3Dpp.AutoDepthStencilFormat = D3DFMT_D24X8;
    D3Dpp.Flags=0;
    D3Dpp.FullScreen_RefreshRateInHz=D3DPRESENT_RATE_DEFAULT;
    D3Dpp.FullScreen_PresentationInterval=D3DPRESENT_INTERVAL_DEFAULT;

    D3D->CreateDevice(0, D3DDEVTYPE_HAL, hwnd, D3DCREATE_PUREDEVICE | D3DCREATE_HARDWARE_VERTEXPROCESSING, &D3Dpp, &D3Ddevice);
}

void SetupRenderTarget(int width, int height)
{
    // Store original screen and zbuffer targets
    D3Ddevice->GetRenderTarget(&screentarget);
    D3Ddevice->GetDepthStencilSurface(&screenzbuffertarget);

    // Create rendertargets
    D3Ddevice->CreateTexture(width, height, 1, D3DUSAGE_RENDERTARGET, D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT, &left);
    left->GetSurfaceLevel(0,&lefttarget);
    D3Ddevice->CreateTexture(width, height, 1, D3DUSAGE_RENDERTARGET, D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT, &right);
    right->GetSurfaceLevel(0,&righttarget);

    // Create Z-Buffer (it will be shared between the targets)
    D3DSURFACE_DESC desc;
    left->GetLevelDesc(0,&desc);
    D3Ddevice->CreateDepthStencilSurface(desc.Width,desc.Height, D3DFMT_D24X8,D3DMULTISAMPLE_NONE,&zbuffertarget);

    // Setup simple directional light
    D3DCOLORVALUE black;
    ZeroMemory(&black, sizeof(D3DCOLORVALUE));

    D3DCOLORVALUE white;

```

```

white.a=1.0f;
white.r=1.0f;
white.g=1.0f;
white.b=1.0f;

D3DMATERIAL8 material;
material.Ambient=white;
material.Diffuse=white;
material.Emissive=black;
material.Power=0.0f;
material.Specular=black;
D3DDevice->SetMaterial(&material);

D3DLIGHT8 light;
ZeroMemory(&light, sizeof(light));

light.Type = D3DLIGHT_DIRECTIONAL;
light.Ambient.r = 0.4f;
light.Ambient.g = 0.4f;
light.Ambient.b = 0.4f;
light.Diffuse.r = 0.7f;
light.Diffuse.g = 0.7f;
light.Diffuse.b = 0.7f;

D3DVECTOR vector;
vector.x=0.0f;
vector.y=0.0f;
vector.z=1.0f;
light.Direction = vector;
D3DDevice->SetLight(0, &light);
D3DDevice->LightEnable(0, TRUE);

D3DDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
}

```

```

void SetupGeometry()
{
    struct Vertex
    {
        float x,y,z;
        float nx,ny,nz;
    };

    D3DDevice->CreateVertexBuffer(vertexcount*sizeof(Vertex),0, D3DFVF_XYZ | D3DFVF_NORMAL, D3DPOOL_MANAGED, &vertexbuffer);
    D3DDevice->CreateIndexBuffer(indexcount*2,0, D3DFMT_INDEX16, D3DPOOL_MANAGED, &indexbuffer);

    unsigned char* vb;
    vertexbuffer->Lock(0,0, &vb,0);
    memcpy(vb,vertices,vertexcount*sizeof(Vertex));
    vertexbuffer->Unlock();

    unsigned char* ib;
    indexbuffer->Lock(0,0, &ib,0);
    memcpy(ib,indices,indexcount*sizeof(unsigned short));
    indexbuffer->Unlock();
}

```

```

// Geometry pipeline setup
void SetWorld(float ztranslate, float yaw)
{
    D3DMATRIX world;
    ZeroMemory(&world, sizeof(world));
    world._11 = float(cos(yaw));
    world._22 = 1.0f;
    world._33 = float(cos(yaw));
    world._44 = 1.0f;
    world._31 = float(sin(yaw));
    world._13 = float(-sin(yaw));
    world._43 = ztranslate;
    D3DDevice->SetTransform(D3DTS_WORLD,&world);
}

void SetView(float basis)
{
    D3DMATRIX view;
    ZeroMemory(&view, sizeof(view));
    view._11 = 1;
    view._22 = 1;
    view._33 = 1;
    view._44 = 1;
    view._41 = -basis; // "Inverse"
    D3DDevice->SetTransform(D3DTS_VIEW,&view);
}

void SetProjection(float nearclip, float farclip, float hfov, float vfov, float stereowindow)
{
    hfov*=0.017453292519943295769236907684886f;
    vfov*=0.017453292519943295769236907684886f;
    float w = (float)1/float(tan(hfov*0.5f)); // 1/tan(x) == cot(x)
    float h = (float)1/float(tan(vfov*0.5f)); // 1/tan(x) == cot(x)
    float Q = farclip/(farclip - nearclip);
    D3DMATRIX projection;

    ZeroMemory(&projection, sizeof(projection));
    projection._11 = w;
    projection._22 = h;
    projection._33 = Q;
    projection._41 = stereowindow;
    projection._43 = -Q*nearclip;
    projection._34 = 1;
    D3DDevice->SetTransform(D3DTS_PROJECTION,&projection);
}

```

```

// Draw spinning logo to a rendertarget
void DrawScene(float stereowindow, int rtwidth, int rtheight, LPDIRECT3DSURFACE8 rendertarget, LPDIRECT3DSURFACE8 zbuffer)
{
    struct Vertex
    {
        float x,y,z;
        float nx,ny,nz;
    };

    D3Ddevice->SetRenderTarget(rendertarget,zbuffer);
    D3Ddevice->Clear(0,NULL,D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(45,45,45),1.0L,0L);
    D3Ddevice->BeginScene();
    D3DVIEWPORT8 viewport = {0,0,rtwidth,rtheight, 0.0f, 1.0f};
    D3Ddevice->SetViewport(&viewport);
    SetProjection(1.0f,10000.0f,54.432f,42.184f,stereowindow); // HFOV & VFOV according to 35 mm lens
    D3Ddevice->SetStreamSource(0, vertexbuffer, sizeof(Vertex));
    D3Ddevice->SetIndices(indexbuffer,0);
    D3Ddevice->SetVertexShader(D3DFVF_XYZ | D3DFVF_NORMAL);
    D3Ddevice->SetTexture(0,NULL);
    D3Ddevice->SetTexture(1,NULL);
    D3Ddevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST,0,vertexcount,0,indexcount/3);
    D3Ddevice->EndScene();
}

```

```

// WinMain //
•1 int WINAPI WinMain (HINSTANCE hThisInstance, HINSTANCE, LPSTR, int nCmdShow)
{
    #ifndef NDEBUG
        #define new new(_NORMAL_BLOCK, __FILE__, __LINE__) // Report new'ed memory block
        int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG); // Get current flag
        flag |= _CRTDBG_LEAK_CHECK_DF; // Turn on leak-checking bit
        _CrtSetDbgFlag(flag); // Set flag to the new value
    #endif

    // Create Win32 window
    SetupWindows(hThisInstance, nCmdShow);

    // Create Direct3D using version 8 interface
    SetupDevice();

    // Create rendertarget
    SetupRenderTargets(DA_RENDERTARGETWIDTH, DA_RENDERTARGETHEIGHT);

    // Create geometry for spinning logo
    SetupGeometry();

    // ColorCode plugin //
    •2 HINSTANCE CCplugin = LoadLibrary(".\\daColorCodeDX8.dll"); // Load DLL library
    DACREATECOLORCODEPROC daCreateColorCode; // Declare ColorCode creation procedure/function
    daCreateColorCode = (DACREATECOLORCODEPROC)GetProcAddress(CCplugin, "daCreateColorCode"); // Bind function
    daColorCode* colorcode = daCreateColorCode((daHandle)D3DDevice); // Create CC object using DX8 (since the DX8 dll was used)
    •3 colorcode->Create(DA_RENDERTARGETWIDTH, DA_RENDERTARGETHEIGHT); // Create plugin and its internal codepaths
    const char* description = colorcode->GetDescription(); // Get description if needed

    // Stereo parameters //
    colorcode->SetNearpoint(600);
    colorcode->SetBasisfactor(3);
    float basis=colorcode->GetBasis();
    float stereowindow=colorcode->GetStereoWindow();

    MSG msg;
    ZeroMemory(&msg, sizeof(MSG));
}

```

```

// Mainloop
while(!GetAsyncKeyState(VK_ESCAPE) && !exitapp)
{
    // Messaging at operation system level
    if(PeekMessage(&msg, NULL,0,0,PM_NOREMOVE))
    {
        if(!GetMessage (&msg, NULL, 0, 0)) return msg.wParam;
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    } else
    {
        // Check for lost device (only need for fullscreen applications)
        if (D3DERR_DEVICENOTRESET==D3Ddevice->TestCooperativeLevel())
        {
            // Device needs to be reset - The plugin must be destroyed for later re-creation
            colorcode->Destroy();

            // Device needs to be reset - All rendertargets must be released _before_ the device can be reset
            if (left) {left->Release(); left=NULL;}
            if (right) {right->Release(); right=NULL;}
            if (screentarget) {screentarget->Release(); screentarget=NULL;}
            if (lefttarget) {lefttarget->Release(); lefttarget=NULL;}
            if (righttarget) {righttarget->Release(); righttarget=NULL;}
            if (screenzbuffertarget) {screenzbuffertarget->Release(); screenzbuffertarget=NULL;}
            if (zbuffertarget) {zbuffertarget->Release(); zbuffertarget=NULL;}
            if (SUCCEEDED(D3Ddevice->Reset(&D3Dpp)))
            {
                // Device now reset - All rendertargets can be re-created
                SetupRenderTarget(DA_RENDERTARGETWIDTH,DA_RENDERTARGETHEIGHT);

                // Device now reset - The ColorCode plugin must be re-created
                colorcode->Create(DA_RENDERTARGETWIDTH,DA_RENDERTARGETHEIGHT);
            }
        }
    } else
}

```

```

if (SUCCEEDED(D3Ddevice->TestCooperativeLevel()))
{
    // Update scene
    SetWorld(525,float(GetTickCount())/1000.0f);

    if (GetAsyncKeyState(VK_F1)) basis+=0.1f;
    if (GetAsyncKeyState(VK_F2)) basis-=0.1f;
    if (GetAsyncKeyState(VK_F5)) stereowindow+=0.1f;
    if (GetAsyncKeyState(VK_F6)) stereowindow-=0.1f;

    // Set Gamma (by selecting a different gammacorrected LUT - this is not hw gamma)
    if (GetAsyncKeyState(VK_F8)) colorcode->SetGamma(0);
    if (GetAsyncKeyState(VK_F9)) colorcode->SetGamma(1);
    if (GetAsyncKeyState(VK_F10)) colorcode->SetGamma(2);
    if (GetAsyncKeyState(VK_F11)) colorcode->SetGamma(3);
    if (GetAsyncKeyState(VK_F12)) colorcode->SetGamma(4);

    // Draw left
    SetView(-basis/2);
    • 4 DrawScene(-stereowindow/2,DA_RENDERTARGETWIDTH,DA_RENDERTARGETHEIGHT,lefttarget,zbuffertarget);

    // Draw right
    SetView(basis/2);
    • 5 DrawScene(stereowindow/2,DA_RENDERTARGETWIDTH,DA_RENDERTARGETHEIGHT,righttarget,zbuffertarget);

    // ColorCode encode and draw result to screen
    {
    • 6 D3Ddevice->SetRenderTarget(screentarget,screenzbuffertarget);
    • 7 D3Ddevice->Clear(0,NULL,D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,0),1.0L,0L);
    • 8 D3Ddevice->BeginScene();
    • 9 D3DVIEWPORT8 viewport = {0,0,800,600, 0.0f, 1.0f};
    • 10 D3Ddevice->SetViewport(&viewport);
    • 11 D3Ddevice->SetTexture(0,left); // Important! The LEFT rendertarget must be located in texture slot 0
    • 12 D3Ddevice->SetTexture(1,right); // Important! The RIGHT rendertarget must be located in texture slot 1
    • 13 colorcode->Apply(viewport.X,viewport.Y,viewport.Width,viewport.Height);
    • 14 D3Ddevice->EndScene();
    }

    // Present the result
    • 15 D3Ddevice->Present(NULL,NULL, NULL, NULL);
}
}
}

```

```
// Deallocate

//Destroy CC plugin
colorcode->Destroy();
delete colorcode;
FreeLibrary(CCplugin);

// Geometry
indexbuffer->Release();
vertexbuffer->Release();

// Release rendertarget surfaces
left->Release();
right->Release();
screentarget->Release();
lefttarget->Release();
righttarget->Release();
screenzbuffertarget->Release();
zbuffertarget->Release();

// Release/deallocate Device and D3D
D3Ddevice->Release();
D3D->Release();

return msg.wParam;
}
```