D3D11toCLANGlib Manual – Windows 3D graphics for C programmers

Pekka Upo 13.7.2024, Finland
Draft: 13.7.2024
Last updated: 5.8.2024

* * *

D3D11toCLANGlib.lib is a 64-bit Windows 11 library, which enables C based 3D graphics application creation under Windows. Library is to be taken into compillation with all application C-files.
The library itself is written mainly with C++ and C and provides access to Direct3D11 graphical services under the hood. Since D3D11 is totally C++ oriented with hierarchies, I decided that much easier and direct approach is needed especially for those who want to continue programming with plain C (me myself) and forget implicit limitations, almost secresies and limitations concerning ways of program D3D.
The good old OpenGL/GLSL was an example of well designed and well implemented piece of logically working interface that had many ways for programmer to express himself/herself. D3D does not. D3D has implicit rules and limitations, which are too often poorly documented or not documented at all as if “everybody should just guess these and be glad”.

1.1 Generic vision

CLANG library is strongly based on **handles**. E.g. when loading texture you get a handle to that texture, which can be used with many texture consumers like materials with dedicated bind functions. Again, when you create either preset mesh or design your own mesh you get a handle to that mesh. There is a possibility to have different materials on different portions of the mesh using material handles. Objects can be created with handle to a mesh that is suited best and many objects can share same hmesh.

```
Texture0------|
Texture1------|--> Material ---> Mesh ---> Object
BumpTexture---|
```

See chapter 3 as an example.

1.2 Compiling with library

Compiling with library is done simply with just appending library in to the command line, e.g.

```
CL.exe main.c file1.c file2.c /link user32.lib gdi32.lib d3d11toCLANG.lib
```

or if you prefer easily maintaining make file:

```
# MAKE.MAK
#
CFLAGS      = /Ehcs /O2
LINKFLAGS   = /MACHINE:X64
OBJFILES    = main.obj file1.obj file1.obj
LIBS        = user32.lib gdi32.lib d3d11toCLANG.lib

main.obj:   main.c commonhdr.h
      CL.exe /c $(CFLAGS) main.c

file1.obj:  file1.c
      CL.exe /c $(CFLAGS) file1.c

file2.obj:  file2.c
      CL.exe /c $(CFLAGS) file2.c

clean:
      del main.obj
      del file1.obj
      del file2.obj
```

If you prefer using VisualStudio(Community) IDE, there is a huge amount of options to set and that is beyond this manual. VS options tend to switch places and are renamed between major releases.

---

2. Interface functions

Interface functions are declared in **cinterface.h**. This manual contains small descriptions of those functions. Some functions are there to speeden up development process such as SetObjectPosition() and SetObjectRotation(), while the rest of the object's variables has to change with function dealing with all available object's variables like SetObjectPhysicals() (with which Position and Rotation can also be set).
For creating meshes (user defined or preset meshes) you need to understand **primitives.h**.
Note that after this research and development time functions and data structs are subject to change.

2.1 Program initialization and deinitialization

```
typedef struct {
      UINT32                  iWidth;      // should be divisible by 16
      UINT32                  iHeight;
      HWND                          hWndAssociated;
      UINT32                  iRefreshRate;
      COLOR4                  BackgroundColor;
} DIRECT3D11HARDWAREINIT;

int   Direct3D11HardwareInit( DIRECT3D11HARDWAREINIT *pInit );


typedef struct {
```

```
      BOOL                      bKeepTextInfoWindowHidden;
      BOOL                      bShowViewportTestInWindowCorner;
      UINT16                    iDefaultEnvironmentCubeMapSideSize;
      UINT16                    iDefaultShadowCubeMapSideSize;
} DIRECT3D11SOFTWAREINIT;

      bShowViewportTestInWindowCorner draws viewport up look in the parent window.

int   Direct3D11SoftwareInit( DIRECT3D11SOFTWAREINIT *pInit );


Example initialization of d3dtoCLANG.lib:

      ZeroMemory( &HardwareInit, sizeof( HardwareInit ) );
      HardwareInit.iWidth             = (rcDX.right  - rcDX.left) / 32 * 32;
      HardwareInit.iHeight            = (rcDX.bottom - rcDX.top)  / 32 * 32;
      HardwareInit.hWndAssociated     = hWndDirectXCanvas;
      HardwareInit.iRefreshRate       = 60;
      if((i = Direct3D11HardwareInit( &HardwareInit )) < 0) {
            Error( "#failed to initialize DirectX11 hardware: %08x\n", i );
            PrintfMessageBox( "Error", "#failed to initialize DirectX11 \ hardware:
%08x", i );
            goto ErrI;
      }

      Puts( "(hardware part of DirectX11 initialized)\n" );

      InfoTextWindowOpen( TRUE, TRUE, FALSE );      // show info window when something
emerges to print

      ZeroMemory( &SoftwareInit, sizeof( SoftwareInit ) );
      SoftwareInit.bKeepTextInfoWindowHidden              = FALSE;
                  // open when e.g. error occurs
      SoftwareInit.bShowViewportTestInWindowCorner        = TRUE;
      SoftwareInit.iDefaultEnvironmentCubeMapSideSize     = 256;
      SoftwareInit.iDefaultShadowCubeMapSideSize          = 512;
      if((i = Direct3D11SoftwareInit( &SoftwareInit )) < 0) {
            Error( "#failed to initialize DirectX11 software: %08x\n", i );
            PrintfMessageBox( "Error", "#failed to initialize DirectX11 software: %08x",
i );
            goto ErrI;
      }

      if(GetVideoMemoryAmount( &iVideoMemoryAmount ) == 0) Printf( "VideoMemoryAmount =
%u Mbytes\n", iVideoMemoryAmount / 1000U );
```

2.2 System functions

For practical examples of using functions below try to struggle with files createobjects.c and main.c.

```
int   SetDepthClearColor( const COLOR4 *pColor4 );
      // Background color for rendering.

int   SetEnvironmentCubeMapSideSize( UINT16 iSideSize );
      // The more detailed cube map you want the bigger iSideSize you provide. That of
cource consumes more graphics memory. Acceptable sizes are power of two.

int   SetShadowCubeMapSideSize( UINT16 iSideSize );
      // Acceptable sizes are power of two.
```

```
int    DeinitializeDirect3D11( void );

int    GetVideoMemoryAmount( UINT32 *piVideoMemory );

/*-------------------------- --------------------------*/

int    SetProjection( float fFovDegrees, float fZNear, float fZFar );
int    GetEyeTargetAndUp( VECTOR *pEye, VECTOR *pTarget, VECTOR *pUp );
int    SetEyeTargetAndUp( const VECTOR *pEye, const VECTOR *pTarget, const VECTOR *pUp );
int    Handle3DView( float fRotX, float fRotY, float fForward, float fSide, float fUp );
int    GetFrustumPoints( VECTOR P[ 8 ] );

/*-------------------------- --------------------------*/

int    ResizeGraphicsWindow( RECT *rcWin );
       Keep in mind that D3D requires the width is divisible by 16.

/*-------------------------- --------------------------*/

int    RenderScene( void );

/*-------------------------- --------------------------*/

int    OpenInfoTextWindow( BOOL bOpen );
              // otherwise openes automatically if print/warning/error occurs

int    DoNotOpenInfoTextWindowAtAll();
              // stay close even if error occurs

/*-------------------------- --------------------------*/

typedef INT32                   HTEXTURE;    // handle to texture
typedef INT32                   HMATERIAL;   // handle to material
typedef INT32                   HMESH;       // handle to mesh
typedef INT32                   HOBJECT;     // handle to object
typedef INT32                   HLIGHT;      // handle to light

/*-------------------------- --------------------------*/
```

2.3 Material functions

```
#define TEXTURE_COMBINE_ADDITIVE 0      // Texture 1 and 2 are summed
#define TEXTURE_COMBINE_MULTIPLY 1      // Texture 1 and 2 are multiplied

//#pragma pack( push, 1 )
typedef struct tMATERIALPROPERTIES {
       COLOR3                   AmbientColor;
       float                    fAmbientAmount;

       COLOR3                   EmissiveColor;
       float                    fEmissiveAmount;

       COLOR3                   MaterialColor;
                                // used in diffuse, emissive and specular components
       float                    fSpecularPower;

       float                    fSpecularAmount;          // [0, 1]
       float                    fDiffuseAmount;           // [0, 1]
       float                    fEnvironmentReflections;  // [0, 1]
       float                    fSkyboxReflectionAmount;  // [0, 1]

       float                    fTexture1Amount;
       float                    fTexture2Amount;
       float                    fBumpMappingAmount;
```

```
      float                     fTransparency;

      float                     fReflectionAmount;        // no operation
      float                     fRefractionAmount;
      float                     fRefractionFactor;
      UINT32                    iCombineModeEmissive;     // TEXTURE_COMBINE_xxx

      HTEXTURE                  hPrimaryTexture;
      HTEXTURE                  hSecondaryTexture;
      HTEXTURE                  hBumpTexture;
      UINT32                    iCombineModeTexture;      // TEXTURE_COMBINE_xxx

      UINT32                    bAlphaMask;
      UINT32                    iTest1;
      UINT32                    iUnused1;
      UINT32                    iUnused2;

} MATERIALPROPERTIES;
//#pragma pack( pop )


/*-------------------------- --------------------------*/
```

2.4 Texture functions

```
typedef struct {
      int                       iWidth;
      int                       iHeight;
      int                       nBitsPerPixel;
      int                       iArraySize;
      UINT32                    iDxgiFormat;
      int                       iResourceType;
} TEXTUREINFO;

typedef struct {
      float                     fUShift0, fVShift0;
                                // for texture1 and normal (bump) mapping if present
      float                     fUScale0, fVScale0;
      float                     fUShift1, fVShift1; // for texture2
      float                     fUScale1, fVScale1;
} TEXTUREPARAMETERS;

int   LoadTextureFromFile( const char *pszFilename, HTEXTURE *phTexture, TEXTUREINFO
*pOptTextureInfo );
      // Loaded texture can be bmp, png, jpg, tga, gif and ddi

int   LoadTextureFromMemory( const BYTE *pDataSrc, size_t iDataSrcSize, HTEXTURE
*phTexture, TEXTUREINFO *pOptTextureInfo );

int   CreateEmptyTexture( TEXTUREINFO *pTexInfo, HTEXTURE *phTexture );
      // usable with manual UpdateTextureData()

int   GetTextureInfo( HTEXTURE hTexture, TEXTUREINFO *pTextureInfo );

int   UpdateTextureData( HTEXTURE hTexture, const BYTE *pbtData );

int   ReadTextureData( HTEXTURE hTexture, TEXTUREINFO *pTextureInfo, BYTE
*pbtTextureData );

int   ReadCubeSideTextureDataFromObject( HOBJECT hObject, TEXTUREINFO *pTextureInfo,
UINT16 iCubeSideId, BYTE *pbtTextureData );

int   ResizeImageRGBA8( UINT32 iSrcWidth, UINT32 iSrcHeight, const BYTE *pbtDataSrc,
UINT32 iDstWidth, UINT32 iDstHeight, BYTE *pbtDataDst, UINT32 iFormat );
```

```
int    SaveData32AsBitmap24or32BitsPerPixel( const char *pszFileName, UINT32 iWidth,
UINT32 iHeight, BOOL bDataDst32, const BYTE *pbtData );

int    GetTextureParameters( HTEXTURE hTexture, TEXTUREPARAMETERS *pParams ); //@REVISE

int    SetTextureParameters( HTEXTURE hTexture, TEXTUREPARAMETERS *pParams ); //@REVISE

int    DeleteTexture( HTEXTURE hTexture );

int    _DeleteAllTextures( void );      //@REVISE

/*--------------------------- ---------------------------*/

typedef struct {
      UINT32                   iSwapWidth;
      UINT32                   iSwapHeight;
      float                    fSwapRefreshRate;
      UINT32                   iSwapFormat;
      UINT32                   nBufferCount;
      UINT32                   iTexWidth;
      UINT32                   iTexHeight;
      UINT32                   iTexFormat;
} SWAPBUFFERINFO;

int    ReadSwapBufferPixels( SWAPBUFFERINFO *pSwapBufferInfo, BYTE *pbtPixelData );

/*--------------------------- ---------------------------*/
```

2.5. Binding Texture to Material

```
int    CreateMaterial( const MATERIALPROPERTIES *pMaterial, HMATERIAL *phMaterial );

int    GetMaterialProperties( HMATERIAL hMaterial, MATERIALPROPERTIES *pMaterial );

int    SetMaterialProperties( HMATERIAL hMaterial, const MATERIALPROPERTIES *pMaterial );

int    BindPrimaryTextureToMaterial( HTEXTURE hTexturePrimary, float fTextureAmount,
HMATERIAL hMaterial );

int    BindSecondaryTextureToMaterial( HTEXTURE hTexturePrimary, float fTextureAmount,
HMATERIAL hMaterial );

int    BindBumpTextureToMaterial( HTEXTURE hTextureBump, float fTextureAmount, HMATERIAL
hMaterial );

/*--------------------------- ---------------------------*/
```

2.6. Mesh functions

```
typedef struct {
      UINT16                   nSections;
      UINT32                   nTriangles;
      BOUNDINGBOX              BBox;
      BOUNDINGSPHERE           BSphere;
} MESHINFO;

#define MESHCONTAINFL_TRANSPARENCY           0x00020000
#define PRIMITIVEFL_VERTEXNORMALS            0x00000004
#define PRIMITIVEFL_FLATNORMALS              0x00000002

int    CreatePrimitiveMesh( MESHGEOMETRY *pGeometry, HMATERIAL hMaterial, DWORD dwFlags,
HMESH *phMesh );
```

```
int    CreateMultiMaterialMesh( UINT32 nTriangles, const VERTEX_PNT *pVerticesArray,
HMATERIAL *pTriangleMaterialArray, DWORD dwFlags, HMESH *phMesh );

int    GetMeshInfo( HMESH hMesh, MESHINFO *pMeshInfo );

int    GetPrimitiveMeshParameters( HMESH hMesh, MESHGEOMETRY *pMeshGeom );

int    SetPrimitiveMeshParameters( HMESH hMesh, MESHGEOMETRY *pMeshGeom );

int    UpdatePrimitiveMesh( HMESH hMesh );

int    DeleteMesh( HMESH hMesh );

int    _DeleteAllMeshes( void ); //@REVISE

/*-------------------------- --------------------------*/
```

2.7 Object functions

```
#define OBJECTFL_SKYBOX                0x00010000

#define FALSE_COLOR_MODE_NONE          0            // modes for iFalseColorsMode
#define FALSE_COLOR_MODE_CUBENORMAL    1
#define FALSE_COLOR_MODE_INVCUBENORMAL 2
#define FALSE_COLOR_MODE_COLORS8       3
#define FALSE_COLOR_MODE_INVCOLORS8    4

#define OBJFL_NOTORIGINCENTERED        0x00000100
                    // set if mesh contains absolute coordinates not round origin

Usually objects are created around origin and therefore they are easy to intantiate into
multiple locations by SetObjectPosition().


typedef struct {
      DWORD                    dwFlags;
      VECTOR                   Position;
      ROTATOR                        Rotation;
      VECTOR                   Scale;
      VECTOR                   Pivot;

      HMESH                    hMesh;

      BOOL                     bReceiveShadows;
      BOOL                     bPassLightThrough;

      BOOL                     bUseReplacementLitColor;
      COLOR3                   ColorReplacementLit;

      UINT32                   iFalseColorsMode;
      float                    fFalseColorAmount;
      COLOR4                   ArFalseColors[ 8 ];

      void                     *pvUserData;

} OBJECTDEF;

typedef struct {
      VECTOR                   Position;
      ROTATOR                        Rotation;
      VECTOR                   Scale;
      VECTOR                   Pivot;
} OBJECTPHYSICALS;

int    CreateObject( OBJECTDEF *pObjectDef, const char *pszName, HOBJECT *phObject );
```

```
int    BindMeshToObject( HMESH hMesh, HOBJECT hObject );

int    GetObjectPhysicals( HOBJECT hObject, OBJECTPHYSICALS *pPhys );

int    SetObjectPhysicals( HOBJECT hObject, OBJECTPHYSICALS *pPhys );

int    GetObjectPosition( HOBJECT hObject, VECTOR *pPosition );

int    SetObjectPosition( HOBJECT hObject, const VECTOR *pPosition );

int    GetObjectRotation( HOBJECT hObject, ROTATOR *pRotator );

int    SetObjectRotation( HOBJECT hObject, const ROTATOR *pRotator );

int    GetObjectMatrix( HOBJECT hObject, MATRIX *pMM );

int    SetObjectMatrix( HOBJECT hObject, const MATRIX *pMM );

int    SetObjectReplacementLitColor( HOBJECT hObject, BOOL bUse, const COLOR3 *pColor );

//-int ResetObjectAxes( HOBJECT hObject );           //@NOT USABLE

int    DeleteHObject( HOBJECT hObject );

int    DeleteAllObjects( void );        //@REVISE

/*--------------------------- ---------------------------*/
```

2.8 Skybox functions

```
int    CreateSkyboxObject( UINT32 iTextureCubeMapSideSize, UINT32 iPhysicalSideSize );

int    CopyToSkyboxSideImage( UINT16 iCubeSideId, HTEXTURE hTexture );

int    SetSkyboxColorTint( const COLOR3 *pColor );

/*--------------------------- ---------------------------*/
```

2.9 Light source functions

```
#define LIGHTFL_ENABLED                0x00000002
#define LIGHTFL_TYPEMASK               0x000000F0
#define LIGHTFL_POINTLIGHT             0x00000020
#define LIGHTFL_SPOTLIGHT              0x00000030

typedef struct {
      VECTOR             Position;
      float              fRadius;

      COLOR3             Color;
      float              fRadiusSquared;

      VECTOR             SpotLightDirection;
      UINT32             iLightFlags;

      float              fSpotLightInnerAngle;
      float              fSpotLightOuterAngle;
      UINT32             bCastsShadows;
      UINT32             bLightInsideObject;

} LIGHTPROPERTIES;

int    CreateLight( LIGHTPROPERTIES *pInit, HLIGHT *phLight );
```

```
int    DeleteLight( HLIGHT hLight );

int    GetLightPosition( HLIGHT hLight, VECTOR *pLightPosition );

int    SetLightPosition( HLIGHT hLight, const VECTOR *pLightPosition );

int    GetSpotLightDirection( HLIGHT hLight, VECTOR *pLightDirection );

int    SetSpotLightDirection( HLIGHT hLight, const VECTOR *pLightDirection );

int    ObtainLight( HLIGHT hLight, LIGHTPROPERTIES *pInit );

int    ModifyLight( HLIGHT hLight, LIGHTPROPERTIES *pInit );

int    _DeleteAllLights( void );

/*--------------------------- ---------------------------*/
```

2.10 Auxilliary functions

```
int    SetFarDistanceForObjectsToBeSeen( float fDist );

int    SetShadowBias( float fShadowBias );    //@TODO SHOULD BE AUTOMAGICAL

int    SetLightThresholdOffset( float fLightThresholdOffset );
                                              //@TODO SHOULD BE AUTOMAGICAL

/*--------------------------- ---------------------------*/
```

2.11 Debug Line Functions

```
int    AddDebugLine( const VECTOR *P0, const VECTOR *P1, const COLOR3 *pColor3 );
              // accumulates debug line

int    CommitDebugLinesForDrawing( void );
              // realizes accumulated debug lines and make them visible

int    DeleteAllDebugLines( void );
              // clear line buffer

/*--------------------------- ---------------------------*/
```

2.12 Random Generator Functions

```
       //--- 16 separate random generators ---
int    RandR( UINT16 iChannel );

int    RandRSeed( UINT16 iChannel, unsigned iSeed );

int    RandRFloatRanged( UINT16 iChannel, float fMinValue, float fMaxValue, float
*pfValue );

/*--------------------------- ---------------------------*/

int    DeleteAllResources( void );                  //@TODO INCOMPLETE

/*--------------------------- ---------------------------*/
```

3 Programming example

```
TEXTUREINFO                TextureInfo;
HTEXTURE                   hTexture512;
LoadTextureFromFile( "TESTPICTURE_512.png", &hTexture512, &TextureInfo );

HTEXTURE                   hTexture256;
LoadTextureFromFile( "TESTPICTURE2_256.png", &hTexture256, &TextureInfo );

HMATERIAL                  hMaterial0;
ZeroMemory( &Material, sizeof( Material ) );
SetColor3( &Material.AmbientColor,  0.0f, 0.0f, 0.0f );
SetColor3( &Material.MaterialColor, 1.0f, 0.8f, 0.0f );
Material.fAmbientAmount          = 0.0f;
Material.fSpecularPower          = 50.0f;
Material.fSpecularAmount         = 0.8f;
Material.fDiffuseAmount          = 1.0f;
SetColor3( &Material.EmissiveColor, 0.0f, 0.1f, 1.0f );
Material.fEmissiveAmount         = 0.0f;
Material.fEnvironmentReflections= 0.9f;
Material.fSkyboxReflectionAmount= 0.0f;
Material.fTexture1Amount         = 0.5f;
Material.fTexture2Amount         = 0.5f;
Material.fBumpMappingAmount      = 0.0f;
Material.fTransparency           = 0.0f;
Material.fRefractionAmount       = 0.0f;      // poor temporary solution!
Material.fRefractionFactor       = 0.75f;
CreateMaterial( &Material, &hMaterial0 );

BindPrimaryTextureToMaterial( hTexture512,   1.0f, hMaterial0 );

BindSecondaryTextureToMaterial( hTexture256, 1.0f, hMaterial0 );

MESHGEOMETRY               MeshGeom;
HMESH                      hMesh0;
ZeroMemory( &MeshGeom, sizeof( MeshGeom ) );
MeshGeom.ePrimitiveType          = ePrimitiveType_Sphere;
MeshGeom.Sphere.fRadius          = 8.0f;
MeshGeom.Sphere.nSegments        = 32;
MeshGeom.Sphere.nDivisions       = 32;
CreatePrimitiveMesh( &MeshGeom, hMaterial0, PRIMITIVEFL_VERTEXNORMALS, &hMesh0 );

OBJECTDEF                  ObjectDef;
HOBJECT                    hObject;
ZeroMemory( &ObjectDef, sizeof( ObjectDef ) );
SetVector( &ObjectDef.Position, 20.0f, -8.0f, 0.0f );
SetVector( &ObjectDef.Scale,    1.0f, 1.0f, 1.0f );
ObjectDef.bReceiveShadows = TRUE;
CreateObject( &ObjectDef, "Sphere0", &hObject0 );

BindMeshToObject( hMesh0, hObject0 );  // now object is ready to rendering
```

3.1 Using multiple materials in object mesh

```
CreateMultiMaterialMesh( nTriangles, pVertexData, phArMaterialHandles,
       PRIMITIVEFL_VERTEXNORMALS, &hMeshMulti )) < 0) return i;
       // NOTE: One triangle is 3 pVertexData[] items.
       // NOTE: One material handle is for one triangle.
       // NOTE: library optimizes the rendering order of triangles
       // phArMaterialHandles lists material for each triangle (nTriangles long)
       // NOTE: Vertex data is consist of PNT vertices of triangle lists, not
              strips.
```