ニューラルソフト有限会社

モデリング	技術資料	検認	照査	作成
				市来 博記

表　　題

3D ボリューム データ処理ライブラリ

OpenVDB Ver 6.0.0 の構築方法

副　　題

How to build OpenVDB Library Ver 6.0.0

キ ー ワ ー ド

OpenVDB,3D,ボリューム,Volume

参照/添付資料

A ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~G~~ ~~H~~ ~~I~~
~~J~~ ~~K~~ ~~L~~ ~~M~~ ~~N~~ ~~O~~ ~~P~~ ~~Q~~ ~~R~~
~~S~~ ~~T~~ ~~U~~ ~~V~~ ~~W~~ ~~X~~ ~~Y~~ ~~Z~~

目次

1. 概要

本資料は Windows 10 64Bit 上で Visual Studio 2017(Toolset 2015)を使用して OpenVDB Ver 6.0.0 を構築する手順を記したものです。

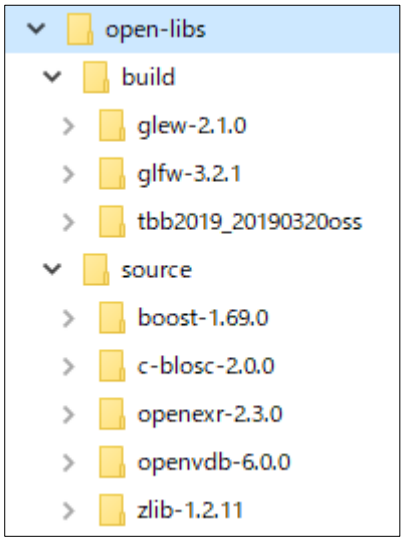
2. 手順

1　OpenVDB ライブラリ構築に必要なライブラリ ファイルとツールをダウンロードします。
下記の表の名称欄が網掛けになっているものはバイナリ配布を使用し、そうでないものはソースをビルドして使用します。

名称	ダウンロード サイト	バージョン
Boost C++ライブラリ	https://www.boost.org/	1.69.0
c-blosc A blocking, shuffling and loss-less compression library	https://github.com/Blosc/c-blosc	2.0.0
GLEW The OpenGL Extension Wrangler Library	http://glew.sourceforge.net/	Windows 32-bit and 64-bit Binaries 2.1.0
GLFW An OpenGL library	https://www.glfw.org	64-bit Windows binaries 3.2.1
OpenEXR(Ilmbase, openexr)	https://github.com/openexr/openexr	2.3.0
OpenVDB	https://www.openvdb.org → github	6.0.0
TBB Threading Building Blocks	https://www.threadingbuildingblocks.org → github	Blocks 2019 Update 5 tbb2019_20190320oss_win.zip
zlib A Massively Spiffy Yet Delicately Unobtrusive Compression Library	https://www.zlib.net/	1.2.11
cmake ビルド自動化ツール	https://cmake.org/	Binary distributions cmake-3.14.1-win64-x64.zip

2　cmake バイナリの圧縮ファイルを展開します。(場所はどこでも構いません。)

3　ダウンロードしたファイルを展開して OpenVDB ライブラリ構築用のフォルダー構造を作成します。



バイナリ配布のファイルは build フォルダーに格納し、ソース配布のファイルは source フォルダーに格納します。

4　boost-1.69.0 をビルドします。

4.1　コマンド プロンプトを起動して、カレント フォルダーを Visual Studio 2015 Tootset の格納フ

ォルダーに移動します。

例 C:¥Program Files (x86)¥Microsoft Visual Studio 14.0¥VC¥bin¥x86_amd64

4.2 vcvarsx86_amd64.bat を起動します。

4.3 コマンド プロンプトのカレント フォルダーを ~¥open-libs¥source¥boost-1.69.0 に移動します。

4.4 bootstrap.bat を起動します。（処理完了まですこし時間が掛かります。）

4.5 b2.exe を起動します。（処理完了までかなり時間が掛かります。）

b2.exe toolset=msvc-14.0 link=static runtime-link=static,shared --build-dir=build/x64 address-model=64 –j7 install --includedir=~¥open-libs¥build¥boost-1.69.0¥include --libdir=~¥open-libs¥build¥boost-1.69.0¥lib

これで boost-1.69.0 のビルドは完了です。（~¥open-libs¥build¥ boost-1.69.0 下に include、lib フォルダーと boost ライブラリを利用する際に必要になるファイルが生成されます。）

5 zlib-1.2.11 をビルドします。

5.1 cmake バイナリを展開したフォルダー¥bin¥cmake-gui.exe を起動します。

5.2 Where is the source code: に~¥open-libs¥source¥zlib-1.2.11 を設定します。

5.3 Where to build the binaries:に~¥open-libs¥build¥zlib-1.2.11 を設定します。

5.4 File/Delete Cache メニューを選択します。（確認ダイアログは yes を選択します。）

5.5 Add Entry ボタンを押下します。

5.5.1 name に CMAKE_INSTALL_PREFIX を設定します。

5.5.2 type に path を設定します。

5.5.3 value に~¥open-libs¥build¥zlib-1.2.11 を設定します。

5.6 Configure ボタンを押下します。（フォルダー生成確認ダイアログが表示された場合は yes を選択します。）

5.6.1 Specify the generator for this project に Visual Studio 15 2017 を設定します。

5.6.2 Optional platform for generator に x64 を設定します。

5.6.3 Optional toolset to use に v140 を設定します。

5.6.4 finish ボタンを押下します。

5.7 Generate ボタンを押下します。

5.8 Visual Studio 2017 で~¥open-libs¥build¥zlib-1.2.11¥zlib.sln を開いて、ソリューション構成（Release/Debug）を設定後、ALL_BUILD プロジェクトをビルドしてから INSTALL プロジェクトをビルドします。

これで zlib-1.2.11 のビルドは完了です。（~¥open-libs¥build¥zlib-1.2.11 下に zlib ライブラリを利用する際に必要になるファイルを格納したフォルダーが生成されます。）

6 openexr-2.3.0 をビルドします。（Python 用ライブラリは生成しません。）

6.1 cmake バイナリを展開したフォルダー¥bin¥cmake-gui.exe を起動します。

6.2 Where is the source code: に~¥open-libs¥source¥openexr-2.3.0 を設定します。

6.3 Where to build the binaries:に~¥open-libs¥build¥openexr-2.3.0 を設定します。

6.4　File/Delete Cache メニューを選択します。（確認ダイアログは yes を選択します。）

6.5　Add Entry ボタンを押下します。

6.5.1 name に CMAKE_INSTALL_PREFIX を設定します。

6.5.2 type に path を設定します。

6.5.3 value に~¥open-libs¥build¥openexr-2.3.0 を設定します。

6.6　Add Entry ボタンを押下します。

6.6.1 name に CMAKE_PREFIX_PATH を設定します。

6.6.2 type に path を設定します。

6.6.3 value に~¥open-libs¥build¥zlib-1.2.11 を設定します。

6.7　Configure ボタンを押下します。（フォルダー生成確認ダイアログが表示された場合は yes を選択します。）

6.7.1 Specify the generator for this project に Visual Studio 15 2017 を設定します。

6.7.2 Optional platform for generator に x64 を設定します。

6.7.3 Optional toolset to use に v140 を設定します。

6.7.4 finish ボタンを押下します。

6.8　OPENEXR_BUILD_PYTHON_LIBS の Value を OFF に設定します。

6.9　OPENEXR_BUILD_STATIC の Value を ON に設定します。

6.10　Configure ボタンを押下します。

6.11　Generate ボタンを押下します。

6.12　Visual Studio 2017 で~¥open-libs¥build¥openexr-2.3.0¥OpenEXR.sln を開いて、ソリューション構成（Release/Debug）を設定後、ALL_BUILD プロジェクトをビルドしてから INSTALL プロジェクトをビルドします。

これで openexr-2.3.0 のビルドは完了です。
~¥open-libs¥build¥openexr-2.3.0 下に openEXR ライブラリを利用する際に必要になるファイルを格納したフォルダーが生成されます。

7　c-blosc-2.0.0 をビルドします。

7.1　cmake バイナリを展開したフォルダー¥bin¥cmake-gui.exe を起動します。

7.2　Where is the source code: に~¥open-libs¥source¥c-blosc-2.0.0 を設定します。

7.3　Where to build the binaries:に~¥open-libs¥build¥c-blosc-2.0.0 を設定します。

7.4　File/Delete Cache メニューを選択します。（確認ダイアログは yes を選択します。）

7.5　Add Entry ボタンを押下します。

7.5.1 name に CMAKE_INSTALL_PREFIX を設定します。

7.5.2 type に path を設定します。

7.5.3 value に~¥open-libs¥build¥c-blosc-2.0.0 を設定します。

7.6　Configure ボタンを押下します。（フォルダー生成確認ダイアログが表示された場合は yes を選択します。）

7.6.1 Specify the generator for this project に Visual Studio 15 2017 を設定します。

7.6.2 Optional platform for generator に x64 を設定します。

7.6.3 Optional toolset to use に v140 を設定します。

7.6.4 finish ボタンを押下します。

7.7 Generate ボタンを押下します。

7.8 Visual Studio 2017 で~¥open-libs¥build¥c-blosc-2.0.0¥blosc.sln を開いて、ソリューション構成（Release/Debug）を設定後、ALL_BUILD プロジェクトをビルドしてから INSTALL プロジェクトをビルドします。

これで c-blosc-2.0.0 のビルドは完了です。
~¥open-libs¥build¥ c-blosc-2.0.0 下に c-blosc ライブラリを利用する際に必要になるファイルを格納したフォルダーが生成されます。

8 openvdb-6.0.0 をビルドします。（Python 用モジュールは生成しません。）

8.1 ~¥open-libs¥source¥openvdb-6.0.0 フォルダーに cmake バイナリを展開したフォルダー¥bin¥cmake-gui.exe のショートカットを作成します。

8.2 ~¥open-libs¥source¥openvdb-6.0.0 フォルダーに以下のコマンド行を含むバッチ ファイルを作成します。

```
set BOOST_ROOT=～¥open-libs¥build¥boost-1.69.0
set GLEW_ROOT=～¥open-libs¥build¥glew-2.1.0
set GLFW3_ROOT=～¥open-libs¥build¥glfw-3.2.1
set ILMBASE_ROOT=～¥open-libs¥build¥openexr-2.3.0
set OPENEXR_ROOT=～¥open-libs¥build¥openexr-2.3.0
set TBB_ROOT=～¥open-libs¥build¥tbb2019_20190320oss
set BLOSC_ROOT=～¥open-libs¥build¥c-blosc-2.0.0
```

8.3 ~¥open-libs¥source¥ openvdb-6.0.0¥CMakeLists.txt の 41 行目に以下の行を挿入します。

```
SET ( DOXYGEN_SKIP_DOT ON )
SET ( Blosc_USE_STATIC_LIBS OFF )
SET ( USE_GLFW3 ON )
SET ( GLFW3_USE_STATIC_LIBS ON )
SET ( Boost_USE_STATIC_LIBS ON )
SET ( BOOST_ROOT "～/open-libs/build/boost-1.69.0" )
SET ( BOOST_INCLUDEDIR "～/open-libs/build/boost-1.69.0/include" )
SET ( BOOST_LIBRARYDIR "～/open-libs/build/boost-1.69.0/lib" )
SET ( ZLIB_INCLUDE_DIR "～/open-libs/build/zlib-1.2.11/include" )
SET ( ZLIB_LIBRARY "～/open-libs/build/zlib-1.2.11/lib/zlibstatic.lib" )
SET ( TBB_LIBRARY_DIR "～/open-libs/build/tbb2019_20190320oss/lib/intel64/vc14" )
SET ( TBB_LIBRARY_PATH "～/open-libs/build/tbb2019_20190320oss/lib/intel64/vc14" )
SET ( Tbb_TBB_LIBRARY "～/open-libs/build/tbb2019_20190320oss/lib/intel64/vc14/tbb.lib" )
SET ( Tbb_TBBMALLOC_LIBRARY "～/open-
libs/build/tbb2019_20190320oss/lib/intel64/vc14/tbbmalloc.lib" )
SET ( Tbb_TBB_PREVIEW_LIBRARY "～/open-
libs/build/tbb2019_20190320oss/lib/intel64/vc14/tbb_preview.lib" )
SET ( CMAKE_INSTALL_PREFIX "～/open-libs/build/openvdb-6.0.0" )
```

8.4 コマンド プロンプトを起動して、カレント フォルダーを Visual Studio 2015 Tootset の格納フォルダーに移動します。
例 C:¥Program Files (x86)¥Microsoft Visual Studio 14.0¥VC¥bin¥x86_amd64

8.5 vcvarsx86_amd64.bat を起動します。

8.6 コマンド プロンプトのカレント フォルダーを ~¥open-libs¥source¥openvdb-6.0.0 に移動します。

8.7 コマンド プロンプトから 8.2 で作成したバッチ ファイルを起動します。

8.8 コマンド プロンプトから 8.1 で作成したショートカットで cmake-gui.exe を起動します。

8.9 Where is the source code: に~¥open-libs¥source¥openvdb-6.0.0 を設定します。

8.10 Where to build the binaries:に~¥open-libs¥build¥openvdb-6.0.0 を設定します。

8.11 File/Delete Cache メニューを選択します。（確認ダイアログは yes を選択します。）

8.12 Configure ボタンを押下します。（フォルダー生成確認ダイアログが表示された場合は yes を選択します。）

8.12.1 Specify the generator for this project に Visual Studio 15 2017 を設定します。

8.12.2 Optional platform for generator に x64 を設定します。

8.12.3 Optional toolset to use に v140 を設定します。

8.12.4 finish ボタンを押下します。

8.13 IlmBase_HALF_LIBRARY の Value を~/open-libs/build/openexr-2.3.0/lib/Half-2_3.lib に設定します。

8.14 OPENVDB_BUILD_PYTHON_MODULE の Value を OFF に設定します。

8.15 OPENVDB_BUILD_UNITTESTS の Value を OFF に設定します。

8.16 Configure ボタンを押下します。

8.17 Generate ボタンを押下します。

8.18 Visual Studio 2017 で~¥open-libs¥build¥openvdb-6.0.0¥OpenVDB.sln を開いて、ソリューション構成（Release/Debug）を設定します。

8.19 openvdb_shared プロジェクトのプロパティーを開いて、C/C++/プリプロセッサ/プリプロセッサの定義から HALF_EXPORTS を削除します。

8.20 vdb_print プロジェクトのプロパティーを開いて、C/C++/プリプロセッサ/プリプロセッサの定義から HALF_EXPORTS を削除し、リンカー/入力/追加の依存ファイルから m.lib と stdc++.lib を削除します。

8.21 vdb_render プロジェクトのプロパティーを開いて、C/C++/プリプロセッサ/プリプロセッサの定義から HALF_EXPORTS を削除し、OPENEXR_DLL を追加し、リンカー/入力/追加の依存ファイルから m.lib と stdc++.lib を削除します。

8.22 ALL_BUILD プロジェクトをビルドしてから INSTALL プロジェクトをビルドします。

これで openvdb-6.0.0 のビルドは完了です。
~¥open-libs¥build¥openvdb-6.0.0 下に openVDB ライブラリを利用する際に必要になるファイルを格納したフォルダーが生成されます。

~¥open-libs¥build¥openvdb-6.0.0¥bin 下に動作確認用実行ファイルが生成されていますので、
https://www.openvdb.org サイトから VDB ファイルをダウンロードして情報表示とレンダリング テストを行うことができます。

vdb_print.exe：VDB ファイル内のボクセル情報を表示します。
vdb_render.exe：VDB ファイルをレンダリングして画像ファイルを生成します。

実行ファイルをパラメータなし、又は-help オプション指定で起動すると HELP 情報が表示されます。

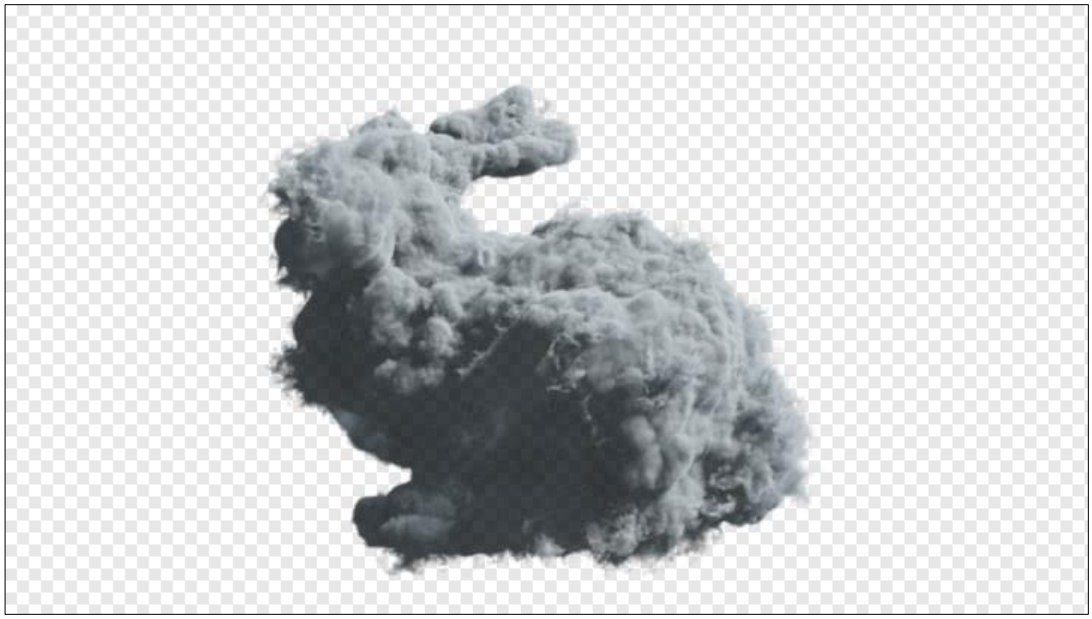
本手順で生成される動作確認用実行ファイルは依存するライブラリの DLL と実行時にリンクされますので、openVDB が依存するライブラリの実行フォルダーを実行ファイル検索パスに追加する必要があります。

実行ファイル検索パスに追加するフォルダー パスは以下の通りです。

～¥open-libs¥build¥openexr-2.3.0¥bin
～¥open-libs¥build¥tbb2019_20190320oss¥bin¥intel64¥vc14
～¥open-libs¥build¥c-blosc-2.0.0¥lib
～¥open-libs¥build¥zlib-1.2.11¥bin

＜レンダリング テスト結果＞

(https://www.openvdb.org の bunny_cloud.vdb を EXR 形式でレンダリング)



3. 特記事項

資料の内容に間違いがないように努めていますが、完全に内容を保証することはできません。間違いにお気付きの場合は、admin@robobiox.com までメール頂ければ幸いです。

ニューラルソフト有限会社

改定履歴	改　　定　　内　　容	検認	照査	作成
初期作成 19/4/5		―	―	市来 博記