国立研究開発法人　理化学研究所　御中

2020 年度 InSitu 処理向け三次元可視化フレームワークのプロトタイプ整備

作業報告書

第 1.0 版

富士通株式会社

2021 年 1 月

改版履歴

リリース	版数	備考
2021/01/27	1.0	初版

目次

1. はじめに

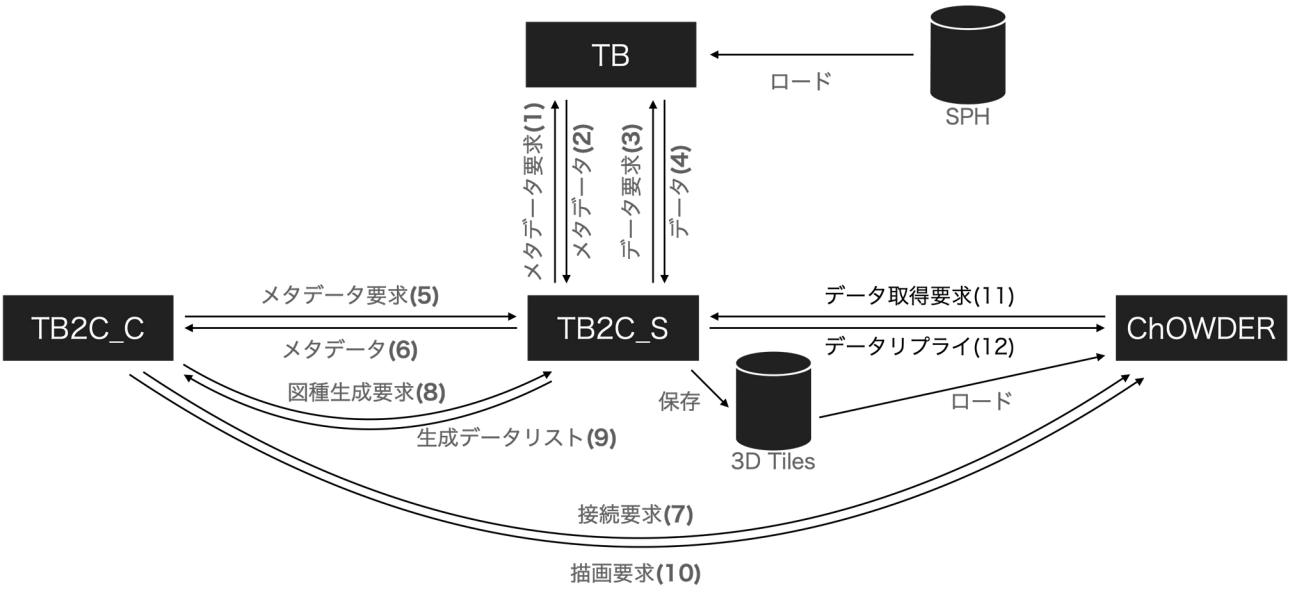
本書は、国立研究開発法人理化学研究所向け「2020 年度 InSitu 処理向け三次元可視化フレームワークのプロトタイプ整備」の作業報告書です。
本作業で実装した TB2C の実装報告および動作検証報告を記述しています。

2. 実装報告

2.1. システム設計

2.1.1. システムの構成

TB2C は、Temporal Buffer (TB)と TB2C_server (TB2C_S)および TB2C client (TB2C_C)から構成されるシステムであり、ChOWDER に接続して動作します(下図参照)。



Temporal Buffer(TB)は、SPH フォーマットで用意された時系列の数値シミュレーション結果データを読み込んでバッファリングし、外部からの要求に応じて必要な時刻スライス・物理量のデータを提供します。

TB2C client(TB2C_C)はユーザーが直接操作する GUI プログラムであり、可視化パラメータの設定や時刻スライスの指定、視点の変更等の操作を行い、ChOWDER に対する表示更新要求を行います。

TB2C server(TB2C_S)は、TB2C_C からの要求に応じて、TB が保持するデータを取得して可視化図種の生成を行い、ChOWDER 表示用に 3D-Tiles 形式のファイルに出力します。

TB および TB2C server は、ChOWDER と同一のマシン上で動作することを前提としています。TB2C client は別のマシン上で動作し、ChOWDER および TB2C server と通信を行います。

2.1.2. プログラム間の通信データ

以下に、各プログラム間で通信されるデータを示します。

(1) TB2C_server から TB へのメタデータ要求
　TB のポートへの HTTP GET
　`path="/",`
　`param:`なし

(2) TB から TB2C_server へ返されるメタデータ
```
JSON[{"id":データ ID,
      "uri":先頭 SPH ファイルの URI,
      "type":"SPH",
      "dims":[I,J,K],
      "datalen":ベクトル長,
      "bbox":[[x0,y0,z0],[x1,y1,z1]],
      "steps":ステップ数,
      "timerange":[開始時刻,終了時刻],
      "vrange":[最小値,最大値]}, …]
```
　TB が保持するデータがリスト化された JSON が返される(今回の実装では 1 個のみ)

(3) TB2C_server から TB へのデータ要求(指定したタイムステップのデータを取得する)

TB のポートへの HTTP GET
path="/data",
param:id=データ ID,
step=取得するステップ番号

⑷ TB から TB2C_server へ返されるデータ
JSON{"type":"SPH",
"step":ステップ番号,
"data":{データ本体}}
{データ本体}は、SPH クラスをbase64 でエンコードして JSON 化したもの

⑸ TB2C_client から TB2C_server へのメタデータ要求
TB2C_server のポートへの HTTP GET
path="/",
param:なし

⑹ TB2C_server から TB2C_client へ返されるメタデータ
JSON{"id":データ ID,
"uri":先頭 SPH ファイルの URI,
"type":"SPH",
"dims":[I,J,K],
"datalen":ベクトル長,
"bbox":[[x0,y0,z0],[x1,y1,z1]],
"steps":ステップ数,
"timerange":[開始時刻,終了時刻],
"vrange":[最小値,最大値],
"vistype":[“isosurf”]}
vistype には、TB2C_server がサポートする可視化図種のリストが入る⑴今回の実装では“iso-surf”のみ⑴

⑺ TB2C_client から ChOWDER への接続要求
ChOWDER への WebSocket 通信
path="ws://chowder_host/v2",
method:"AddContent"
今回の実装では、chowder_host は“localhost”のみ

⑻ TB2C_client から TB2C_server への図種生成要求
TB2C Server のポートへの HTTP PUT
path="/visualize"
param:step=対象ステップ番号,
vistype="isosurf",
visparam={"value":等値面の値}
SPH がベクトルデータの場合は、データの L2 ノルムで等値面を生成する
TB2C Server による図種生成完了を待って ChOWDER への描画要求⑼を行う

⑼ TB2C_server から TB2C_client へ返される生成データリスト
JSON[{tiledLayer}, …]
TB2C_server で生成された 3D-Tiles データの個数分の tiledLayer のリストが返される

⑽ TB2C_client から ChOWDER への描画要求
ChOWDER への WebSocket 通信
path="ws://chowder_host/v2",
method:"UpdateMetaData"
タイムステップまたは等値面の値が変更された場合は、アップデート ID を変更する
視界が変更された場合は、”cameraWorldMatrix”を更新する

⑾ ChOWDER から TB2C_server へのデータ取得要求
TB2C_server のポートへの HTTP GET
path="/visualized/...",
param:なし

⑿ TB2C_server から ChOWDER へのデータリプライ
要求されたファイル
今回の実装ではファイル渡し

2.2. モジュール設計

2.2.1. Temporal Buffer

(1) TB クラス

```
class TB(builtins.object)
 |  TB() -> None
 |
 |  TB - Temporal Buffer
 |  Temporal Buffer のプロトタイプ実装クラスです。
 |  JSON またはファイルリストで指定された時系列 SPH データを読み込み、保持します。
 |  また、要求されたタイムスライスのデータをエンコードして送信します。
 |
 |  メソッド:
 |
 |  __init__(self) -> None
 |      コンストラクタ
 |
 |  loadFromFilelist(self, fnlist: [], basedir: str = '.') -> bool
 |      SPH ファイルのリストを時系列データとして読み込みます。
 |
 |      Parameters
 |      ----------
 |      fnlist: str[]
 |        SPH ファイルのパスのリスト
 |      basedir: str
 |        SPH ファイルが存在するディレクトリ(省略時は'.')
 |
 |      Returns
 |      -------
 |      bool: True=成功、False=失敗
 |
 |  loadFromJSON(self, json_path: str) -> bool
 |      loadFromJSON
 |      JSON ファイルから時系列 SPH データを読み込みます。
 |      JSON ファイルは、以下の形式であることを想定しています。
 |      {
 |        "basedir": "SPH ファイルが存在するディレクトリ(省略時は'.')"
 |        "filelist": [
 |          {"file": "ファイル名 1", "step": "タイムステップ番号", "time": "時刻"},
 |          ...
 |        ]
 |      }
 |      タイムステップ番号と時刻はオプションで、省略された場合 SPH ファイルに格納されて
 |      いるタイムステップ番号、時刻が採用されます。
 |
 |      Parameters
 |      ----------
 |      json_path: str
 |        JSON ファイルのパス
 |
 |      Returns
 |      -------
 |      bool: True=成功、False=失敗
```

(2) TBReqHandler クラス

```
class TBReqHandler(http.server.SimpleHTTPRequestHandler)
 |  TBReqHandler(*args, directory=None, **kwargs)
 |
 |  TBReqHandler
 |  Temporal Buffer 用の HTTP リクエストハンドラー実装クラスです。
 |
 |  メソッド:
 |
 |  do_GET(self)
 |      GET メソッド用のリクエストハンドラー
```

要求されたパスが'/'の場合はメタデータを返し、'/quit'の場合は終了します。
要求パスが'/data'の場合は、指定された step のデータを返します。

(3) TSDataSPH クラス

class TSDataSPH(TSData.TSData)
TSDataSPH() -> None

TSDataSPH
時系列 SPH ファイル群を扱うクラスです。

メソッド:

__init__(self) -> None
コンストラクタ

reset(self) -> None
初期化

setupFiles(self, fnlist: Iterable, basedir: str = '.') -> bool
SPH ファイルエントリーのリストからクラスパラメータを設定します。

Parameters
---
fnlist: [{}]
SPH ファイルエントリーのリスト
basedir: str
SPH ファイルが存在するディレクトリ(省略時は'.')

Returns
---
bool: True=成功、False=失敗

(4) TSData クラス

class TSData(builtins.object)
TSData() -> None

TSData
時系列データファイル群を扱うクラスです。

メソッド:

__init__(self) -> None
コンストラクタ

convStepToIdx(self, stp)
時系列データのタイムステップ番号からタイムステップインデックス番号に変換します。

Parameters
---
stp: int
タイムステップ番号

Returns
---
int: タイムステップインデックス番号

getDataIdx(self, stpIdx)
stpIdx で指定されたタイムステップインデックス番号のデータを返します。

Parameters
---
stpIdx: int
タイムステップインデックス番号

Returns
---
object: データ

getDataStp(self, stp)
stpIdx で指定されたタイムステップ番号のデータを返します。

Parameters
----
stp: int
  タイムステップ番号

Returns
----
object: データ

reset(self) -> None
  初期化

2.2.2. TB2C server

(1) TB2C_server クラス

class TB2C_server(builtins.object)
  TB2C server のプロトタイプ実装クラスです。

  メソッド:

  __init__(self)
    コンストラクタ

  connectTB(self, uri: str)
    URI で指定されたデータソースに接続し、メタデータを読み込み、
    'vistype'を付加して保持します。

    Parameters
    ----
    uri: str
      データソースの URI

  generateIsosurf(self, value: float) -> bool
    現在保持している SPH データに対し、value で指定された値で等値面を生成し、
    3D-Tiles 形式のファイルに出力します。

    Parameters
    ----
    value: float
      等値面を生成する値

    Returns
    ----
    bool: True=成功、False=失敗

  getSPHdata(self, id: int, stp: int) -> [<class 'pySPH.SPH.SPH'>]
    TB より、id と step を指定して SPH データを取得します。
    実際にアクセスする URL は'{uri}/data?id={id}&step={stp}'

    Parameters
    ----
    id: int
      取得する SPH データの ID
    step: int
      取得する SPH データのタイムステップインデックス番号

    Returns

    [SPH.SPH]: 取得したデータ(を分割したリスト)

(2) TB2C_server_ReqHandler クラス

class TB2C_server_ReqHandler(http.server.SimpleHTTPRequestHandler)
  TB2C_server_ReqHandler(*args, directory=None, **kwargs)

  TB2C server 用の HTTP リクエストハンドラー実装クラスです。

  メソッド:

do_GET(self)
GET メソッド用のリクエストハンドラー
要求されたパスが'/'の場合はメタデータを返し、'/quit'の場合は終了します。

do_POST(self)
POST メソッド用のリクエストハンドラー
要求されたパスが'/visualize'の場合はパラメータに従い可視化を行います。

sendMsgRes(self, code: int, msg: str)
HTTP アクセスに対する text/plain 形式のレスポンスを返す。

Parameters
---
code: int
レスポンスコード
msg: str
レスポンスメッセージ

translate_path(self, path)
SimpleHTTPRequestHandler.translate_path のオーバーロードメソッド。
SimpleHTTPRequestHandler.do_GET が呼ばれた際の、リクエストされたパスを
ファイルシステムのパスに変換する。
TB2C_server_ReqHandler クラスでは、do_GET においてリクエストパスが
'/visualized/'で始まる場合のみ(super().do_GET()から)呼ばれ、
'/visualized/'までのパスを g_app.out_dir に置き換えて返す。

Parameters
---
path: str
リクエストパス

Returns
---
str: 変換されたパス

(3) TB2C_visualize クラス

class TB2C_visualize(builtins.object)
TB2C_visualize(outdir: str = '.', bbox=[[0, 0, 0], [1, 1, 1]])

TB2C サーバ用の、可視化機能実装クラスです。
SPH データに対する可視化(等値面生成)結果をジオメトリ(OBJ)ファイルに出力し、
obj23dtiles コマンドを使用して 3D-Tiles に変換します。

メソッド:

__init__(self, outdir: str = '.', bbox=[[0, 0, 0], [1, 1, 1]])
コンストラクタ

bbox2Box(self, bbox: [[<class 'float'>], [<class 'float'>]]) -> [<class 'float'>]
バウンディングボックスデータを、3D-Tile の"Box"形式に変換します。

Parameters
---
bbox: [[float],[float]]
バウンディングボックスデータ

Returns

[float]: 3D-Tile の"Box"形式データ

checkB3dmDir(self) -> bool
出力先ディレクトリ(self._outDir)配下に"b3dm"ディレクトリを
(存在すれば削除してから)作成します。

Returns
---
bool: True=成功、False=失敗

checkObj23dtiles(self) -> bool
"obj23dtiles --version"を実行し、正常に動作するか確認します。

Returns
----
bool: True=成功、False=失敗

isosurf(self, sph_lst: [<class 'pySPH.SPH.SPH'>], value: float, fnbase: str = 'isosurf') -> bool
sph_lst で渡された SPH データ群に対し、value で指定された値で等値面を生成し、
OBJ ファイルに出力した後、obj23dtiles コマンドを使用して 3D-Tiles に変換します。
self._out_dir 配下に、以下のファイルが作成されます。
b3dm/Batchedfnbase_nnn/fnbase_nnn.b3dm
b3dm/Batchedfnbase_nnn/tileset.json

Parameters
----
sph_lst: [SPH.SPH]
等値面を生成する SPH データのリスト
value: float
等値面を生成する値
fnbase: str
等値面ファイルのベース名(省略時:"isosurf")

Returns
----
bool: True=成功、False=失敗

(4) SPH_filter クラス

class SPH_filter(builtins.object)
スタティックメソッド:

divideShareEdge(d: pySPH.SPH.SPH, div: []) -> []
SPH データについて、隣接格子点を共有した分割を行う(static method)
格子サイズが５の次元を２分割する場合、分割されたデータの格子サイズは(3, 3)になる.
分割されたデータの格子サイズは 1 以上でならなければならない.

Parameters
----
d: SPH.SPH
分割する SPH データ
div: int[3]
各軸方向の分割数(>0)

Returns

SPH.SPH[]: 分割された SPH データのリスト、空のリスト=失敗

extractScalar(d: pySPH.SPH.SPH, dataIdx: int) -> pySPH.SPH.SPH
ベクトルデータを持つ SPH からスカラーの SPH を生成する(static method)

Parameters
----
d: SPH.SPH
vector データを持つ SPH データ
dataIdx: int
抽出するスカラーデータのインデックス番号

Returns

SPH.SPH: 抽出したスカラーの SPH データ、None: 失敗

fromJSON(sd: str) -> pySPH.SPH.SPH
base64 エンコードで JSON 化された SPH データを復元する(static method)

Parameters
----
sd: str
文字列化した JSON データ

Returns

SPH.SPH: 復元された SPH データ

toJSON(d: pySPH.SPH.SPH) -> str
SPH データについて、base64 でエンコードして JSON 化する(static method)

Parameters
---
d: SPH.SPH
JSON 化する SPH データ

Returns
---
str: 文字列化した JSON データ

vectorMag(d: pySPH.SPH.SPH) -> pySPH.SPH.SPH
ベクトルデータを持つ SPH からベクトルのノルムをスカラーとして持つ SPH を生成する。
(static method)

Parameters
---
d: SPH.SPH
vector データを持つ SPH データ

Returns

SPH.SPH: ベクトルノルムのスカラー SPH データ、None: 失敗

2.2.3. TB2C client

(1) TB2C_App クラス

class TB2C_App(wx.core.App)
TB2C_App(redirect=False, filename=None, useBestVisual=False, clearSigInt=True)

TB2C client のプロトタイプ App クラスです。
wxPython の App クラスを継承しています。

メソッド:

Message(self, msg: str, err: bool = False)
msg を MessageDialog に表示します。

Parameters
---
msg: str
メッセージ文字列
err: bool
MessageDialog のアイコンをエラーアイコンにするかどうか

OnClose(self, evt)
ウインドウクローズ時のイベントハンドラーです。アプリケーションを終了します。

Parameters
---
evt: wx.Event
ウインドウクローズイベント

OnConnectChOWDER(self, evt)
'Connect to ChOWDDER'メニューのイベントハンドラーです。
URL およびパスワード入力用のダイアログを表示し、ChOWDER に接続します。

Parameters
---
evt: wx.Event
メニューイベント

OnConnectTB2CSrv(self, evt)
'Connect to TB2C server'メニューのイベントハンドラー。
URL 入力用のダイアログを表示し、TB2C server に接続します。

Parameters
---

evt: wx.Event
メニューイベント

OnInit(self)
App 初期化時のイベントハンドラー。
トップレベルの Frame を作成し、OpenGL canvas と UI panel を横方向に配置します。

OnQuit(self, evt)
'Quit'メニューのイベントハンドラーです。アプリケーションを終了します。

Parameters
---
evt: wx.Event
メニューイベント

connectChOWDER(self, hostnm: str, pswd: str) -> bool
hostnm で指定されたホスト上の ChOWDER server に接続します。

Parameters
---
hostnm: str
ChOWDER server が動作するホスト(ホスト名または IP アドレス)
接続先は'ws://{hostnm}/v2'となる。
pswd: str
ChOWDER server に接続する際の APIUser のパスワード

Returns
---
bool: True=成功、False=失敗(self._lastErr にエラーメッセージを登録)

connectTB2CSrv(self, url: str) -> bool
url で指定された TB2C server へ接続し、メタデータを取得します。
取得したメタデータのチェック後、App の UI に反映します。

Parameters
---
url: str
TB2C server の URL

Returns
---
bool: True=成功、False=失敗(self._lastErr にエラーメッセージを登録)

updateRequest(self, flag) -> bool
updateRequest
データ更新処理を行います。
flag の TB2C_App.REQ_UPDDATA ビットが ON の場合は、TB2C サーバに等値面の再作成を
依頼し、ChOWDER に表示更新を依頼します。これは、タイムステップまたは等値面の値が
変更された場合にコールされます。
flag の TB2C_App.REQ_UPDVIEW ビットが ON の場合は ChOWDER への表示更新依頼のみを
行います。これは視界が変更された場合にコールされます。

Parameters
---
flag: int
更新要求フラグ

Returns
---
bool: True=成功、False=失敗(self._lastErr にエラーメッセージを登録)

(2) TB2C_Canvas クラス

class TB2C_Canvas(wx._glcanvas.GLCanvas)
TB2C_Canvas(parent, app)

TB2C client の OpenGL 表示用キャンバスクラスです。
wxPython の GLCanvas クラスを継承しており、マウスイベント用のハンドラが実装されています。

Draw(self)
描画処理を行います。

GetFitMatrix(self)
　オブジェクトを視界にフィットさせる変換行列を返します。

　Returns

　Mat4: 変換行列

GetMatrix(self)
　カメラの変換行列を返します。

　Returns

　Mat4: カメラ変換行列

OnDoubleClick(self, evt)
　ダブルクリックに対するイベントハンドラーです。視界のリセットを行います。

　Parameters
　----------
　evt: wx.Event
　　マウスイベント

OnMouseDown(self, evt)
　マウスボタン押下に対するイベントハンドラーです。

　Parameters
　----------
　evt: wx.Event
　　マウスイベント

OnMouseMotion(self, evt)
　マウス移動に対するイベントハンドラーです。

　Parameters
　----------
　evt: wx.Event
　　マウスイベント

OnMouseUp(self, evt)
　マウスボタンリリースに対するイベントハンドラーです。

　Parameters
　----------
　evt: wx.Event
　　マウスイベント

OnMouseWheel(self, evt)
　マウスホイール回転に対するイベントハンドラーです。

　Parameters
　----------
　evt: wx.Event
　　マウスイベント

OnPaint(self, event)
　再描画に対するイベントハンドラーです。描画処理を行います。

　Parameters
　----------
　evt: wx.Event
　　再描画イベント

OnSize(self, event)
　キャンバスサイズの変更に対するイベントハンドラーです。ビューポート変更を行います。

　Parameters
　----------
　evt: wx.Event
　　サイズイベント

setBoxSize(self, minpos, maxpos)
　表示オブジェクトのバウンディングボックスサイズを設定します。

Parameters
----
evtminpos: [float]
  最小座標値
evtmaxpos: [float]
  最大座標値

(3) TB2C_UIPanel クラス

class TB2C_UIPanel(wx._core.Panel)
TB2C_UIPanel(parent, app, size=wx.Size(250, -1))

メソッド:

__init__(self, parent, app, size=wx.Size(250, -1))
  コンストラクタ

OnIsovalSlider(self, evt)
  等値面の値スライダー操作に対するイベントハンドラーです。

  Parameters
  ----
  evt: wx.Event
    スライダーイベント

OnIsovalTxt(self, evt)
  等値面の値テキストボックス入力に対するイベントハンドラーです。

  Parameters
  ----
  evt: wx.Event
    テキストボックス入力イベント

OnTsSlider(self, evt)
  タイムステップスライダー操作に対するイベントハンドラーです。

  Parameters
  ----
  evt: wx.Event
    スライダーイベント

OnTsTxt(self, evt)
  タイムステップテキストボックス入力に対するイベントハンドラーです。

  Parameters
  ----
  evt: wx.Event
    テキストボックス入力イベント

setInformation(self, info: str)
  情報表示欄の表示内容設定。

  Parameters
  ----
  info: str
    表示内容

setTimeStepRange(self, steps: int) -> bool
  タイムステップ数の設定。

  Parameters
  ----
  steps: int
    タイムステップ数

setValueRange(self, vrange: []) -> bool
  データ値域の設定。

  Parameters
  ----
  vrange: [float]

```
|       データ値域
```

(4) ConnectChOWDERDlg クラス

```
class ConnectChOWDERDlg(wx._core.Dialog)
 |  ConnectChOWDERDlg(hostname: str = None)
 |
 |  ChOWDER 接続用のダイアログクラスです。
 |  ChOWDER ホスト名とパスワードの入力テキストボックスを配置しています。
```

(5) ConnectTB2CSrvDlg クラス

```
class ConnectTB2CSrvDlg(wx._core.Dialog)
 |  ConnectTB2CSrvDlg(url: str = None)
 |
 |  TB2C server 接続用のダイアログクラスです。
 |  TB2C server の接続 URL の入力テキストボックスを配置しています。
```

(6) Frustum クラス

```
class Frustum(builtins.object)
 |  視垂台クラス
 |  視垂台はモデル空間の中の視界を表わす垂台の領域です。
 |   eye: 視点座標
 |   view: 視線方向ベクトル
 |   up: 上方向ベクトル
 |   dist: 視点から注視点までの距離
 |   halfW/halfH: 注視点における視界の幅および高さの半分の値
 |   near/far: 視点から前後のクリップ面までの距離
 |
 |  メソッド:
 |
 |  __init__(self)
 |      コンストラクタ
 |
 |  ApplyModelview(self)
 |      モデルビュー行列の OpenGL 適用
 |        以下の変換を行う行列を生成し、モデルビュー行列とする
 |        - ステレオ表示の場合、視点位置をオフセットする
 |        - 視線方向の回転
 |        - 視点位置への平行移動
 |
 |  ApplyProjection(self, ortho=False, asp=1.0)
 |      プロジェクション行列の OpenGL 適用
 |
 |      Parameters
 |      ----------
 |      ortho: bool
 |        平行投影モード
 |      asp: float
 |        視界のアスペクト比率(横/縦)
 |
 |  GetChOWDERMatrix(self)
 |      ChOWDER 用のカメラ変換行列を返す
 |
 |      Returns
 |      -------
 |      Mat4: ChOWDER 用のカメラ変換行列
 |
 |  GetMVM(self)
 |      モデルビュー行列を返す
 |
 |      Returns
 |      -------
 |      Mat4: モデルビュー行列
 |
 |  GetPM(self, ortho=False, asp=1.0)
 |      プロジェクション行列を返す
 |
 |      Parameters
 |      ----------
 |      ortho: bool
```

平行投影モード
asp: float
視界のアスペクト比率(横/縦)

resetEye(self)
視界をリセットする。

rotHead(self, a)
up 周りに view を a(deg)回転させる

Parameters
---
a: float
回転角度

rotPan(self, a)
Right(=up x (-view))周りに view, up を a(deg)回転させる

Parameters
---
a: float
回転角度

trans(self, x, y, z)
eye を Right(=up x (-view))方向に x, up 方向に y, view 方向に z 移動させる

Parameters
---
x: float
Right 方向移動量
y: float
Up 方向移動量
z: float
view 方向移動量

3. 動作検証報告

3.1. 概要

「富岳」を想定したHPCI計算資源を用いての動作検証として、九州大学ITOシステムの占有フロントエンドノード(ITOフロントエンド)にTB2Cシステムおよび ChOWDERシステムをインストールし、ここで動作するサーバー群とローカルPC上で動作するTB2Cクライアントを接続して動作検証を行います。
この際に、ChOWDERディスプレイとして、理化学研究所R-CCSのタイルドディスプレイ装置に表示するWebブラウザを使用し、ChOWDERのレンダリング時間計測機能を使用してレンダリング性能を測定します。

なお、ITOフロントエンドに割り当てられるIPアドレスはプライベートアドレスであり、ITOログインノードを介したssh接続しか行うことができないため、動作検証作業に際してはITOログインノードへの通信をsshのポートフォワーディング機能を使用してITOフロントエンドに転送する設定を行います。

また、ITOフロントエンドではroot権限が与えられないため、TB2Cシステムおよび ChOWDERシステムのインストールに関わる全てのソフトウエアのインストールはユーザー環境に行います。
ChOWDERが使用する通信ポートは、デフォルトでは80番(および443番)ですが、今回の作業ではroot権限を必要としない1024番以上のポート(8080番)を使用します。

3.2. ITOフロントエンドでの環境構築

3.2.1. ノード構成

- ノードテンプレート：BLGI(ベアメタル)
- OS：Red Hat Enterprise Linux Server release 7.3
- CPU：36コア、主記憶：384GiB

3.2.2. Spackのインストールおよび設定

ITOフロントエンドでPython3、Node.jsおよびこれらの関連モジュールをユーザー環境にインストールするため、パッケージ管理システムSpackを使用します。
以下のコマンドをITOフロントエンド上で実行することで、Spackがインストールされます。

```
cd $HOME
git clone https://github.com/spack/spack.git
```

これにより、ホームディレクトリ配下にspackというディレクトリが作成され、ここにSpackの実行環境がダウンロードされます。ここで、以下のコマンドを実行すると`spack`コマンドが利用可能になります。

```
source $HOME/spack/share/spack/setup-env.sh
```

ITOフロントエンドでは、environment-moduleシステムで複数のコンパイラが使用可能となっていますが、今回の作業ではgcc-9.2.0を使用します。
以下のコマンドをITOフロントエンド上で実行することで、Spackがコンパイラとしてgcc-9.2.0を使用可能になります。

```
module load gcc/9.2.0
spack compiler find
```

上記のコマンドを実行することにより、`$HOME/.spack/linux/compiler.yaml`ファイルが作成され、Spackのコンパイラーリストに`gcc@9.2.0`が追加されます。

最後に、`$HOME/.spack/packages.yaml`ファイルを作成し、以下の内容を記述します。

```
packages:
 perl:
  externals:
  - spec: "perl@5.32.0"
    prefix: /usr
  buildable: False
 all:
  compiler: ['gcc@9.2.0', 'gcc@4.8.5', 'gcc@4.4.7']
  target: [x86_64]
```

これにより、Spackは`/usr`にインストールされているPerlを使用し、パッケージとしてインストールす

ることはなくなります。

3.2.3. Node.js および npm のインストール

Spack を使用して Node.js および npm のインストールを行うには、以下のコマンドを実行します。

```
spack install node-js%gcc@9.2.0
spack install npm%gcc@9.2.0
```

インストール終了後は、以下のコマンドを実行すると `node` および `npm` コマンドが利用可能になります。

```
spack load node-js%gcc@9.2.0
spack load npm%gcc@9.2.0
```

次に、インストールした npm を使用して obj23dtiles をインストールします。以下のコマンドを実行します。

```
npm install -g obj23dtiles
```

3.2.4. Python3 および Python モジュール群のインストール

Spack を使用して Python3 のインストールを行うには、以下のコマンドを実行します。

```
spack install python%gcc@9.2.0
```

インストール終了後は、以下のコマンドを実行すると `python3` および `pip3` コマンドが利用可能になります。

```
spack load python%gcc@9.2.0
```

次に、インストールした Python3(`pip3` コマンド)を使用して、TB2C の実行に必要な Python モジュール群をインストールします。以下のコマンドを実行します。

```
pip3 install scikit-image
pip3 install wxPython
pip3 install pyOpenGL
pip3 install websocket-client
```

3.2.5. Spack 環境の bash 設定

Spack を使用してインストールした環境を、次回ログイン時にも再現できるように、
`$HOME/.bash_profile` に以下の記述を追加しておきます。

```
if [ -d $HOME/spack ]; then
     source $HOME/spack/share/spack/setup-env.sh
     spack load python%gcc@9.2.0
     spack load node-js%gcc@9.2.0
     spack load npm%gcc@9.2.0
fi
```

3.2.6. ChOWDER のインストール

ChOWDER のインストールは、ChOWDER の github リポジトリよりソースをダウンロード(clone)し、インストールスクリプトを実行することで行います。

まず、ソースのダウンロードを行います。以下のコマンドを実行します。

```
cd $HOME
git clone -b 202009 http://github.com/digirea/ChOWDER.git
```

これにより、ホームディレクトリ配下の `ChOWDER` ディレクトリに、ChOWDER のソース一式(`202009` ブランチ)がダウンロードされます。

次に、インストールスクリプトを実行します。以下のコマンドを実行します。

```
cd $HOME/ChOWDER/bin
sh ./install.sh
```

通常、インストールスクリプトの実行で ChOWDER のインストールは完了しますが、今回使用している ChOWDER のブランチ(`202009` ブランチ)では、CentOS 7 用の Redis サーバーがインストールされません。そこで、以下のように Redis のソースコードをダウンロードし、Redis サーバーのインストールを行います。

```
wget http://download.redis.io/releases/redis-5.0.5.tar.gz
tar xvfz redis-5.0.5.tar.gz
```

```
cd redis-5.0.5
make
```

コンパイルが終了したら、Redis サーバーの実行ファイルを ChOWDER 環境下にコピーします。
```
cp src/redis-server $HOME/ChOWDER/redis/
```

3.2.7. TB2C のインストール

TB2C のインストールは、TB2C の提供ファイル(TB2C-1.x.tar.gz)を sftp で ITO フロントエンドにコピーし、任意のディレクトリで展開することで行えます。
ここでは、ホームディレクトリ配下の TB2C ディレクトリ以下に展開します。
```
cd $HOME
tar xvfz TB2C-1.x.tar.gz
```

3.3. 実施方法

3.3.1. ITO フロントエンドでのサーバー実行

ITO フロントエンド上で、Tenporal Buffer、TB2C server および ChOWDER を動作させておく必要があります。

尚、以下の説明では TB2C 展開ディレクトリは$HOME/TB2C、ChOWDER 展開ディレクトリは$HOME/ChOWDER としています。

(1) Temporal Buffer
ITO フロントエンド上の TB2C 展開ディレクトリに移動し、以下のコマンドを実行します。
```
cd $HOME/TB2C
python3 python/TB.py -j data/concat_input_p.json
```

(2) TB2C server
ITO フロントエンド上の TB2C 展開ディレクトリに移動し、以下のコマンドを実行します。
```
cd $HOME/TB2C
python3 python/TB2C_server.py --odir $HOME/ChOWDER/public/data ¥
    --dx 2 --dy 2
```

(3) ChOWDER
ITO フロントエンド上の ChOWDER 展開ディレクトリ/bin に移動し、以下のコマンドを実行します。
```
cd $HOME/ChOWDER/bin
sh ./run.sh
```

3.3.2. ITO ログインノードでのポートフォワーディング設定

ITO ログインノード(ito.cc.kyushu-u.ac.jp)上では、外部からの 8080 番ポートへの接続を ITO フロントエンドの 8080 番ポートに、また外部からの 4000 番ポートへの接続を ITO フロントエンドの 4000 番ポートに、それぞれポートフォワーディングする必要があります。
これには、ITO ログインノード上で以下の ssh コマンドを実行します。ここで、xx.xx.xx.xx には ITO フロントエンドノードの IP アドレスを指定します。

■ 8080 番ポートのポートフォワーディング
```
ssh -g -L 8080:xx.xx.xx.xx:8080  xx.xx.xx.xx
```

■ 4000 番ポートのポートフォワーディング
```
ssh -g -L 4000:xx.xx.xx.xx:4000  xx.xx.xx.xx
```

これらは、ITO ログインノードに別個にログインした別のターミナルで実行する必要があります。
また、ITO ログインノードは ito-1 と ito-2 の 2 台のマシンが負荷分散のために切り替えられて使用されているため(本書執筆時点)、ito-1 と ito-2 の両方のマシン上で上記のコマンドを実行する必要があります。

3.3.3. ローカル PC でのポートフォワーディング設定

ローカル PC(TB2C client を実行するマシン)上では、80 番ポートへの接続を ITO ログインノードの 8080 番ポートに、また 4000 番ポートへの接続を ITO グインノードの 4000 番ポートに、それぞれポートフォワーディングする必要があります。
これには、ローカル PC 上で以下の ssh コマンドを実行します。

■ 80 番ポートのポートフォワーディング
```
ssh -i 秘密鍵 -L 80:ito.cc.kyushu-u.ac.jp:8080 ユーザー名@ito.cc.kyushu-u.ac.jp
```

■ 4000 番ポートのポートフォワーディング
```
ssh -i 秘密鍵 -L 4000:ito.cc.kyushu-u.ac.jp:4000 ユーザー名@ito.cc.kyushu-u.ac.jp
```

これらは、ローカル PC 上の別個のターミナルで実行する必要があります。

3.3.4. ChOWDER コントローラーの実行

ローカル PC 上で Web ブラウザを起動し、以下の URL に接続することで ChOWDER コントローラーが表示されます。
```
http://localhost:80/
```

80 番ポートのポートフォワーディングがされているので、ITO フロントエンド上の ChOWDER サーバーに接続しています。

3.3.5. TB2C client の実行

ローカル PC 上で TB2C client を実行し、ITO フロントエンド上の TB2C server および ChOWDER サーバーに接続させます。ローカル PC 上の TB2C 展開ディレクトリに移動し、以下のコマンドを実行します。
```
python3 python/TB2C_client.py -s http://localhost:4000/ -c localhost
```

上記コマンドを実行すると、TB2C client のウインドウが表示され、ChOWDER の APIUser のパスワード入力が求められます。パスワードを入力すると、ChOWDER コントローラーには Content ID が `tb2c_3dtile` であるコンテンツが登録されます。
ここで、TB2C client の GUI で timestep index を 40 に、isosurf value を 0.0 に設定すると、下図のような表示になります。

3.3.6. ChOWDER ディスプレイのレンダリング時間測定

ChOWDER ディスプレイの接続を行い、ローカル PC 上の TB2C 展開ディレクトリに移動して、以下のコマンドを実行すると、ChOWDER ディスプレイのレンダリング時間の測定が行われます。
```
python3 python/chowder_measure.py
```

3.4. 実施結果

3.4.1. ITO フロントエンド・R-CCS タイルドディスプレイでの実行結果

ITO フロントエンド上で Tenporal Buffer、TB2C server および ChOWDER サーバーを動作させ、ChOWDER ディスプレイを理化学研究所 R-CCS のタイルドディスプレイ装置で動作させた場合のレンダ

リング時間測定結果を以下に示します。

```
Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,
 'triangleCount': 6722, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}
Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,
 'triangleCount': 0, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}
Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,
 'triangleCount': 30650, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 1}
Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,
 'triangleCount': 0, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}
```

3.4.2. 課題

今回の作業において明らかになった課題および対応策について、以下に記述します。

⑴ SSH ポートフォワーディング設定の煩雑さ

今回の作業で動作検証を行った環境は、サーバーソフトウエア群を動作させるマシンの IP アドレスがプライベートアドレスであるため、SSH による多段ポートフォワーディングを行う必要がありました。
本システムではソフトウエア間の通信を行うポートが複数あり、ポートフォワーディングの設定も各ポート毎に行う必要があるため、その設定作業は非常に煩雑となりました。
この課題に対しては、ポートフォワーディングの設定を自動または半自動で行うスクリプトの作成が有効であると考えられます。

⑵ システムソフトウエアインストールの手間の大きさ

今回の作業で動作検証を行った環境では、サーバーソフトウエア群を動作させるマシンの root 権限が得られないため、システムを動作させるために必要なシステムソフトウエアを全てユーザー環境にインストールする必要がありました。
今回の作業では、この課題への対応として Spack の利用を行い、有効な対応策であることが分かりましたが、この他の解決策としてソフトウエアー式がインストールされた仮想環境(Docker, Singularity)を作成し、配布することが考えられます。

⑶ TB2C server の並列化未対応

今回の作業で使用したデータは、テスト用に用意された比較的小規模なものであるため、TB2C システムの動作におけるパフォーマンスの問題は、それほど深刻なものではありませんでした。
しかし、実際の運用シーンにおいて大規模なデータを処理する場合、TB2C がシリアル実行(非並列実行)で動作していることは、深刻なパフォーマンス低下を惹起することが容易に想像されます。
この問題に対する解決策は、TB2C システムの並列化です。特に、TB2C server の処理は並列化が有効であると考えられます。一方、Temporal Buffer の処理の並列化は、外部インターフェースをどのように構成するかを併せて考える必要があり、検討が必要だと考えられます。

以上