お手軽パースツール in Photoshop

Photoshopでパースの計算を任せるためのプラグイン。

一部クラッシュ対策を行っていないため非公開にしている。

README.mdについて

このドキュメントは本プラグインの使い方、設計を解説する。 対象はユーザー、またはプログラマを想定している。

コンセプト

- フルスクラッチで実装する。
- アーティストが積極的に使えるように設計。
- 機能追加を見越して拡張しやすいようにクラス分けするなどこれからを考える。

技術選定

Photoshopプラグイン開発にはUXP API(Javascript)、CEP(Javascript)、CSDK(C++によるネイティブプラグイン)による方法がある。

CEPはこれから古いものとなりサポートされなくなるおそれがある。

CSDKはGUIの実装に時間を取られてしまう。

しかしUXP APIはAdobeが現在力を入れてサポートを強化しており、6月にはC++との相互運用も可能となる予定のため将来性がある。

またPhotoshopAPIに関しても今年の3月末のアップデートによりCEPとくらべても十分に実用レベルに達していると考え、これらを踏まえUXP APIを採用した。

要求環境

- Adobe Photoshop 23.3 or higher.

  旧バージョンだとAPIが古いため動作しない(おそらくロードはできるが想定した動作をしない)。

- Adobe UXP Developer Tool

諸注意

- 開発途中・また技術的制約のため動作が不安定なおそれがある。
- 壊れてもいいドキュメントでご使用すること。
- プラグインの日本語名はお手軽パースツール(仮)ですが開発では**ExtendYourPerspectivs**にしている。
- 表記ゆれ(保持、格納、管理)はすべて同じ意味である。

技術的理由による制約

- 技術的理由によりインプットボックスが数値以外も入力できるようになっている。 これはAdobe UXP APIのバグ(もしくは仕様)によるもの。
- 一部の操作が複雑になっている(パスの一覧更新ボタンを押さないとドロップダウンメニューを上手く扱えないなど)

インストール

1. ダウンロード&解凍
2. **Adobe UXP Developer Tool**で**manifest.json**を読み込む
3. ExtendYourPerspectivesをload。
4. パネルがメインディスプレイの左上に表示されるので適当な位置に移動＆サイズ変更して使う。
5. パネルが表示されない場合はPhotoshopメニュー -> ウィンドウ -> プラグインでプラグインパネルを表示すること。

使い方

パースライン描画(グリッド描画)

1. 視点と注視点、画角を設定する。

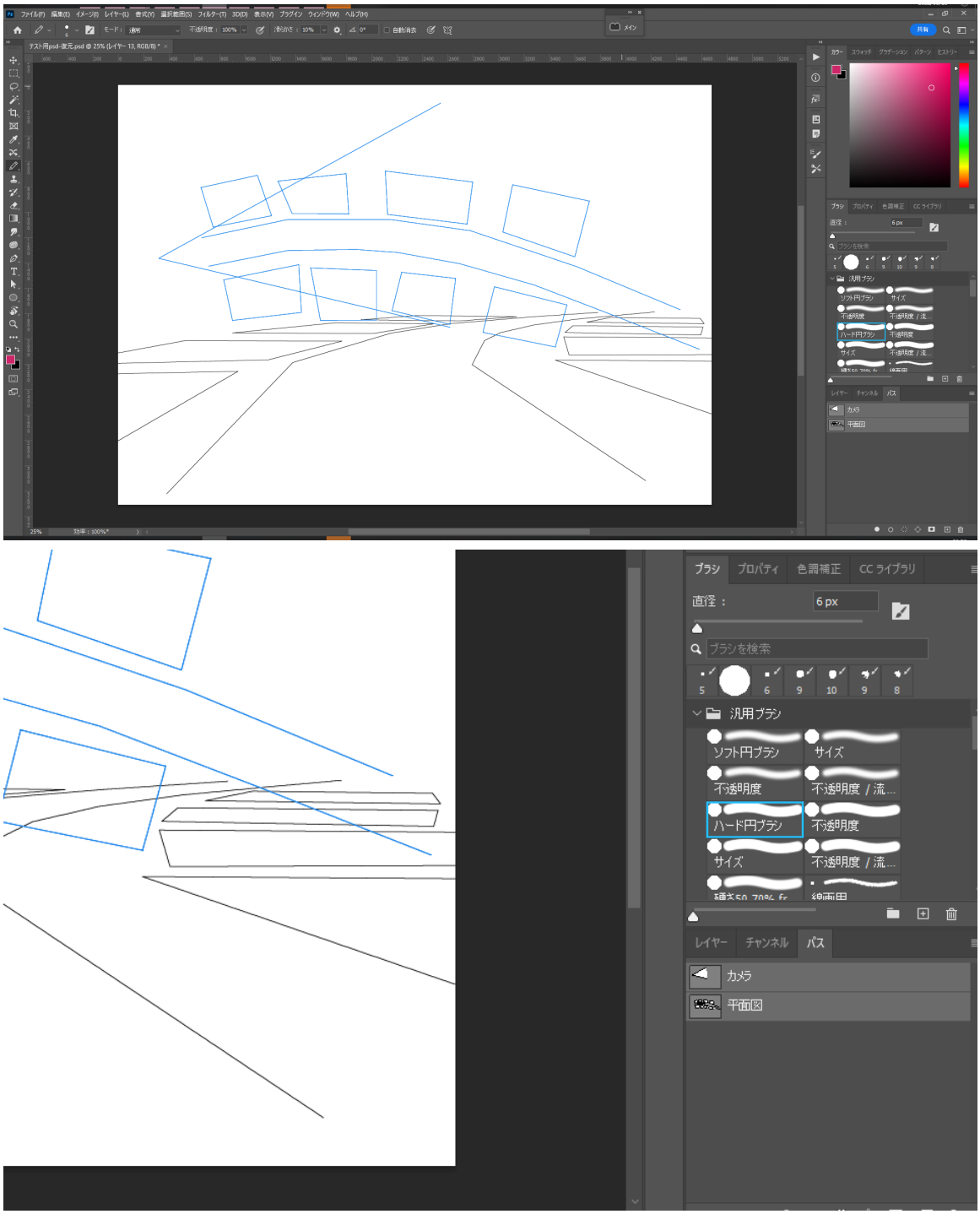
2. **パースライン設定**から引く本数を指定する。

3. パース描画をクリック。

パスの変形

パスを平面図としてとらえ変形する。 言葉でも説明できるが伝わらないため実際に試していただきたい。

1. カメラのパラメータを決めるパス"カメラパス"と平面図となるパス"平面図パス"の2つを用意する。



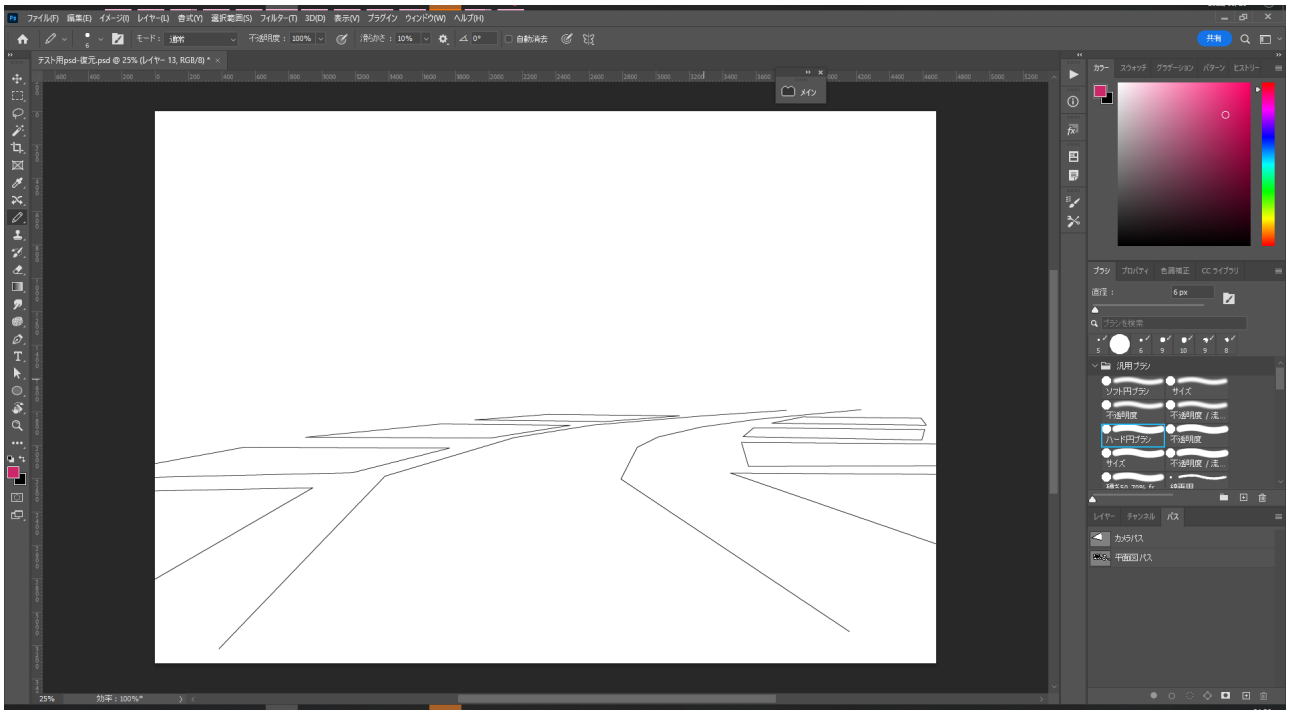
2. ホームタブのカメラパスグループのパスの一覧更新を押すとカメラパスに使えるパスの一覧がドロップダウンメニューに追加される

3. カメラパスにしたいパスを選択した状態でカメラ座標逆算ボタンを押すと カメラパスからカメラの位置が計算され、カメラ設定の値が変化する。

4. ワールドタブに移動

5. 平面図パスを選択し変形ボタンを押す。

6. 以下のように変形される。



設計・コードについて

index.htmlからindex.jsを読み込んでいる。

index.htmlはプラグインのGUI部分となるインターフェースとなる。

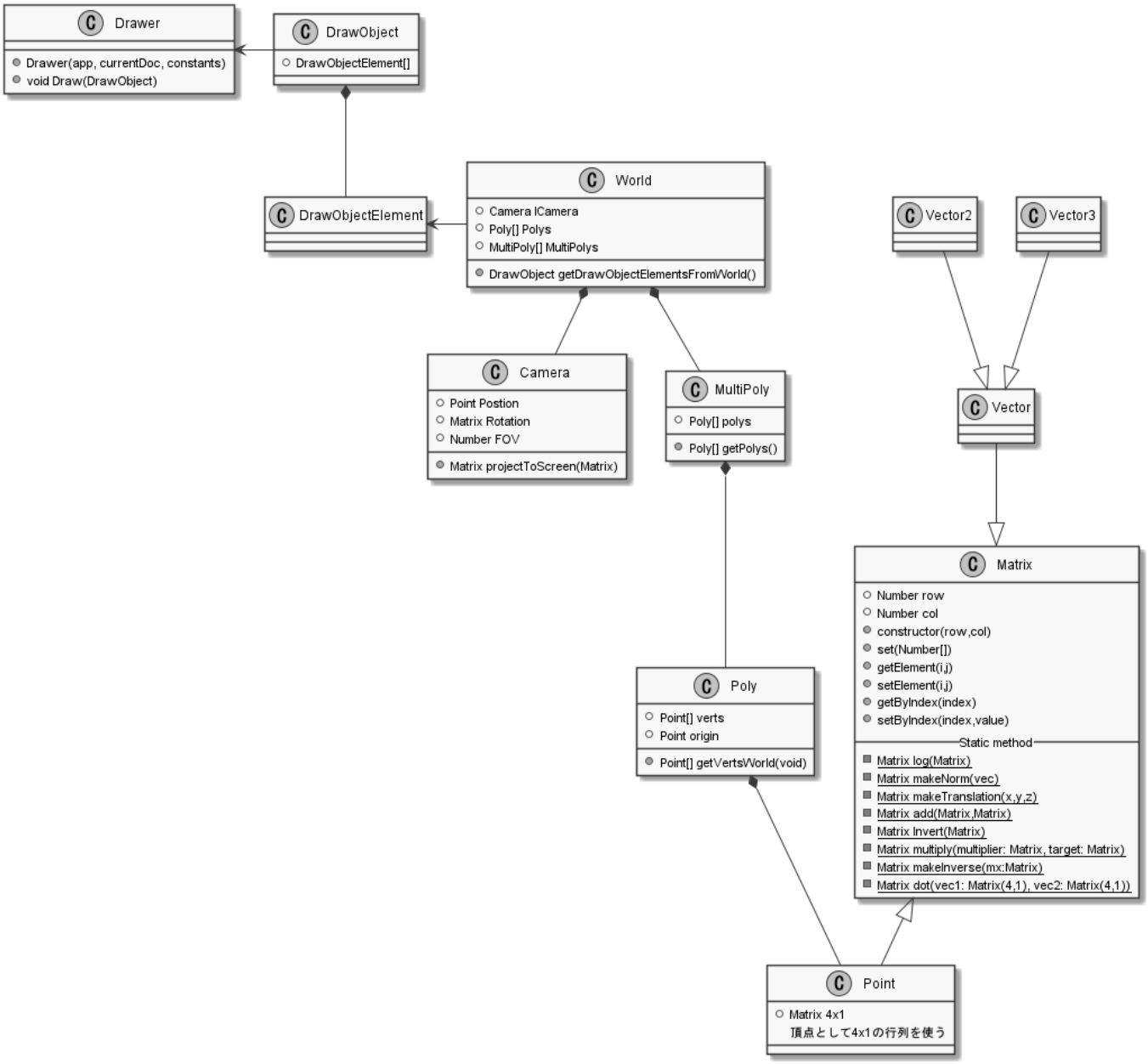
クラスは機能ごとにクラス分けし、機能拡張しやすいようできるだけ結合が弱くなるように設計した。

一部設計不良と思われるところがあるが、よい解決策を見つけられなかったのでご理解いただきたい。

index.js

このクラスでパネル上エレメントにイベントを登録したり関数を呼び出したりしている。

クラス図



使用しているコードはすべて **src/** にある。

基本的にクラスごとでファイル分けしている。

例　:CameraクラスはCamera.js、WorldクラスはWorld.jsに記述されている。

各クラス解説

- World

  このクラスにカメラやPolyなど登録する。

  右手中指を水平線に並行、人差し指を鉛直上向きにしたとき 親指、人差し指、中指がそれぞれX、Y、Z軸に対応している。 **右手座標系。**

  Z軸を本来は鉛直上向きにしなければいけないかもしれないが一般ユーザー使う場合奥行きがZ座標にしたほうがわかりやすいかと考えこのようにした。

- Camera

頂点の座標変換(World -> Screen)を行う。 カメラは**左手座標系**

- Poly

  Pointインスタンスを格納する。Pointインスタンスが格納された順番で線を引くイメージ。

  つまり一つの多角形を表すのがPolyクラス。

- MultiPoly

  Polyを複数格納するクラス。多角形を複数持つので3Dモデルのようなものの表現が(技術的には)可能。

- Point

  空間の座標(頂点)を表現するクラス。

  4x1のMatrix。Matrixクラスを継承してPointクラスとしている。

- Matrixクラス

  自作行列計算ライブラリ。

  インスタンス化せず呼びたい場合のほうが多いため計算に関する機能はstaticで実装した。

  namespaceを使うべきか静的メソッドを使うべきかわからなかったため暫定的に静的メソッドで実装。

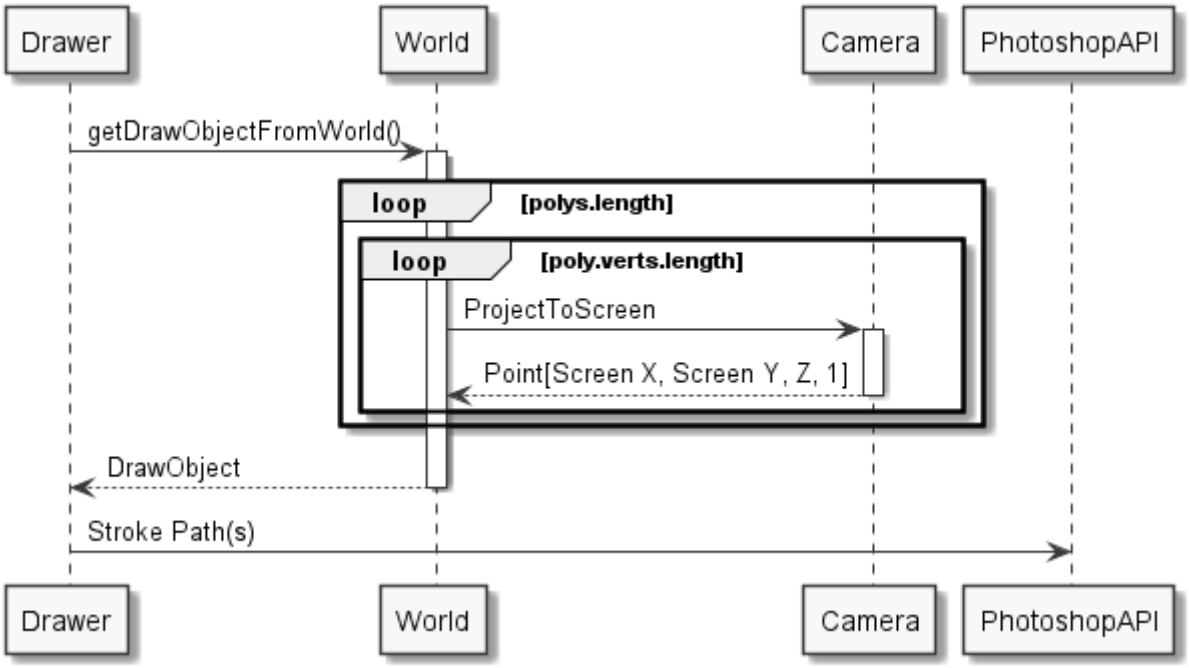
  Matrixを扱うことはわかっているため静的メソッドのほうがよいのではないかと考えた。

処理解説

描画フロー

1. **index.js** の**Init()** で初期化。現在のドキュメントを取得
2. **Camera**インスタンスを生成
3. **World**のコンストラクタに**Camera**インスタンスを渡す
4. **Point**インスタンスを生成
5. **Poly**インスタンスを生成
6. **Poly**インスタンスにPointインスタンスを追加
7. **World**に**Poly**インスタンスを追加
8. 描画処理(下記参照)

描画処理詳細



1. DrawerからWorldにDrawObjectをリクエスト
2. World.getDrawObjectFromWorld()が呼び出される
    1. DrawObjectを生成
    2. World.Polysの要素にアクセス。要素の型はPolyインスタンス Polyインスタンスに格納されている頂点をCameraにわたす
    3. WorldはPolyの頂点座標をCameraに渡し、スクリーン座標変換をリクエスト
    4. Cameraはスクリーン座標変換を行い値を返す
    5. 戻り値(スクリーン座標)からDrawObjectElementを生成
    6. DrawObjectにDrawObjectElementを格納
    7. World.Polysの要素数だけ繰り返し
    8. World.MultiPolysの要素にアクセス
    9. 同様にDrawObjectElementを生成しDrawObjectに格納
    10. World.MultiPolysの要素数だけ繰り返し
3. DrawObjectをDrawerに返す
4. DrawObjectからスクリーン座標を取得
5. Drawerがスクリーン座標にパスを生成するようPhotoshopAPIにリクエスト
6. Drawerがスクリーン座標にパスのストロークを描画するようリクエスト
7. DrawerがPhotoshopAPIに作成したパス消去するようリクエスト

命名規則

クラスには命名規則を設けているがそれ以外はまだ設計が固まっていないため規則がない。ただ関数名から意味がわかるようにはなっている。

特に**index.js**。

- Prefix : ***は接頭辞につける文字列を表す

HTML

- id

- Prefix : 小文字(lower case)でなんのエレメントかを表すアクロニム(頭字語)。
- 以降の単語は頭文字を大文字、それ以外は小文字でつける。
- Prefixの次の単語はボタンなら動詞をつける。それを押すと何が起こるのかを表す動詞を適切に選ぶ。

例　カメラパスの**パスの一覧更新**ボタンのid

id="btnUpdateCameraPath"

index.html

```
<sp-button id="btnUpdateCameraPath" variant="primary">
</sp-button>
```

クラス

- 変数
  - Prefix : 型を最初につける。先頭は必ず小文字。
  - インスタンスは接頭辞大文字の**I**を使う。
  - boolの場合は接頭辞にb。

    例 : Poly.bClosed

    src/Poly.js

    ```
    class Poly{
        constructor() {
            ...
            this.bClosed=false
            ...
        }
    }
    ```

- 関数

  基本的に以下に習う。ぱっと見で何をする関数かわかるような名前にする

  - 基本的に最初は小文字。
  - 値を返す場合はgetをつける。
  - 代入する場合はsetをつける。
  - boolを返す場合はIs***()の形式で書く。

    例

    - Poly.IsClosed() Polyが閉じているか開いているかを返す。

        - DrawObjectElements.IsClosed() DrawObjectElementsが閉じているか開いているかを返す。
    - なにかの計算(操作)が発生する場合(行列の計算など)はその計算(操作)を表す動詞をつける。

      例

        - Matrix.multiply() 行列の掛け算(translate)を行うためmultiplyをつける。
        - Matrix.translate() 行列の平行移動(translate)を行うためtranslate
    - インスタンスのメンバ変数を直接書き換える場合は接尾時にOverrideを使う。