

Book2Book

Résumé des étapes de traitement:

Architecture des projets:

est un ensemble d'outils pour produire **une mise en page à partir d'un spécimen**. À partir d'un livre numérisé, l'objectif est d'extraire, de conserver et de réutiliser tous les éléments permettant sa **re-production**.

- Plusieurs sources de numérisations
- Extraction des contenus textuels (texte brut) et para-textuels (style, mise en page)
- Re-fabrication des éléments de mise en page (typographie, métriques, maquette)
- Génération de fac-similé à partir du texte et de la mise en page extraite

Le projet consiste en une succession de scripts **Python** qui articule différents outils **libres ou open-sources** en normalisant leurs entrées et leurs sorties dans une architecture de projet claire. La plupart des autres opérations consistent en l'automatisation de manipulations bureautiques (copié, collé, renommer) et de traitement d'image simple (faire une moyenne, un niveau...). Chaque étape est conservée pour permettre de vérifier, corriger ou détourner les différentes formes intermédiaires générées. Un fichier de configuration centralise toutes les variables nécessaires à cette chaîne d'opérations.

1. Récupérer le livre numérisé depuis différentes sources (GoogleBooks ou une numérisation à la main)
2. Décomposer et préparer les pages
3. Extraire le contenu textuel et les informations para-textuelles (mise en page)
4. Extraire chaque image de chaque glyphe de chaque page
5. Récupérer les informations de métriques des fontes
6. Générer le contours moyen de chaque glyphe de chaque fonte
7. Créer les fontes et y importer les contours des glyphes
8. Configurer les métriques et diacritiques des fontes

```
.
├── config.yaml
└── Book
    ├── Font
    └── Glyphs
        ├── auto-extracted
        ├── average
        ├── clean
        ├── levels
        ├── specialChar
        └── vectors
    └── Layout
        └── hocr-charboxes
    └── Pages
    └── Toolbox
        ├── BookScanner
        └── extensionInkscape
    └── WebViewer
```

livre numérisé en PDF
éléments relatifs aux fontes du livre
les images des glyphes

informations de mise en page

images des pages préparées pour le traitement
intégralité des outils de Book2Book

Ceci est une proposition de nomenclature qui peut être changée dans le fichier de configuration **config.yaml**.

Comment fonctionne cette documentation ?

Optical Layout Recognition

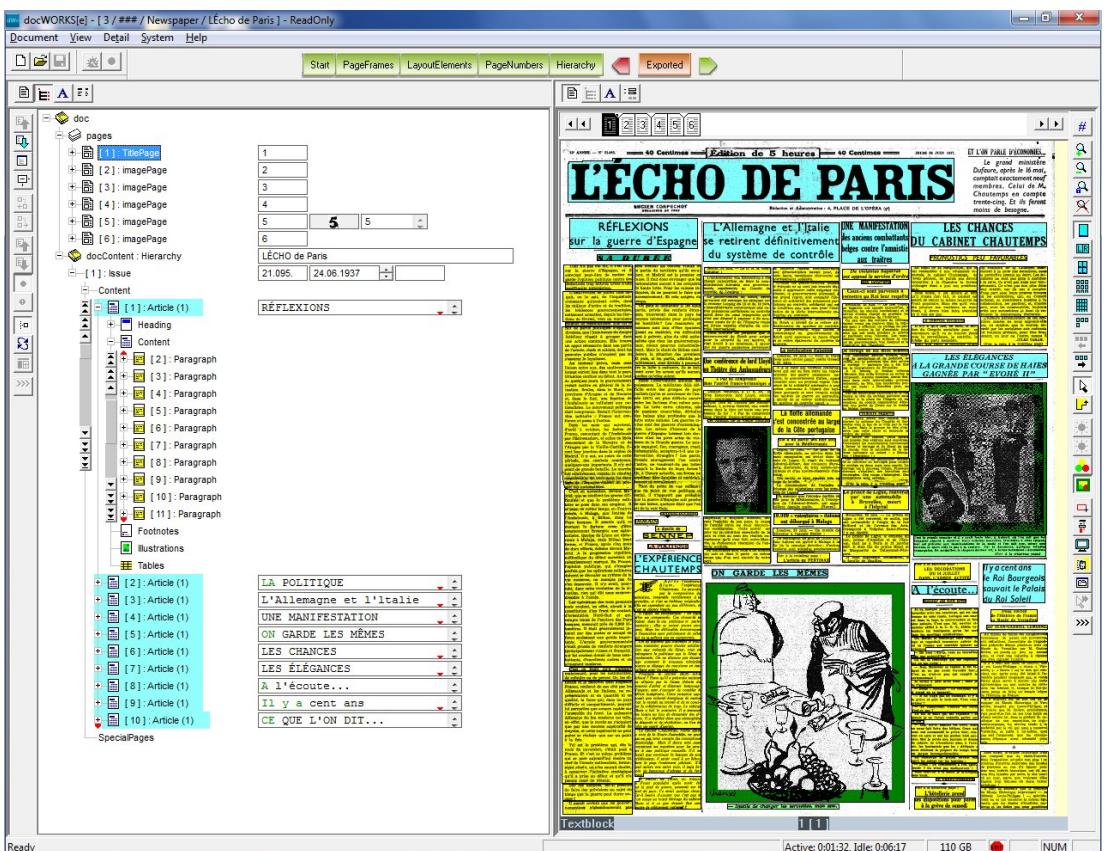
- Un diagramme basé sur les graphes de flux présente les outils principaux, leurs dépendances et leurs interactions [LOOKME.svg](#)
- Les exemples partent toujours du postulat que l'on se trouve à la racine du projet (.) et que l'on utilise la nomenclature par défaut.
- Le détail de l'outil est développé dans des documents séparés [README.md](#) à la racine des différents dossiers pré-existants (voir Architecture des projets).
- Ces documents décrivent les étapes d'un projet en expliquant le fonctionnement des principaux outils (de leur installation à leur usage dans le cadre de ce projet), ainsi que des commentaires sur les choix techniques, des astuces et des éléments auquel il faut être vigilant.
- ***Si une documentation bavarde vous dérange, se référer au diagramme***

Pour la compiler en HTML :

```
>> pandoc -s -c README.css -f markdown -t html5 README.md BOOK.md LAYOUT.md  
GLYPHS.md FONT.md -o README.html
```

OLR pour **Optical Layout Recognition** est un domaine de recherche du **traitement et analyse d'image**. De la même manière que l'**OCR** à pour mission de reconnaître et traduire du texte dans des images, l'**OLR** analyse et encode tout les autres éléments de mise en page d'un document à partir d'une image. L'**OLR** est en réalité la première étape d'une reconnaissance de caractère en permettant de décomposer le document en différentes zones d'intérêt à traiter (paragraphes, lignes, mots...). En plus de connaître la position du texte sur le format, on peut en tirer des informations **sémantiques** (les titres, les chapôs, les paragraphes) et **typographiques** (la police, la graisse, l'interlignage...). En regroupant toutes ces strates d'informations :

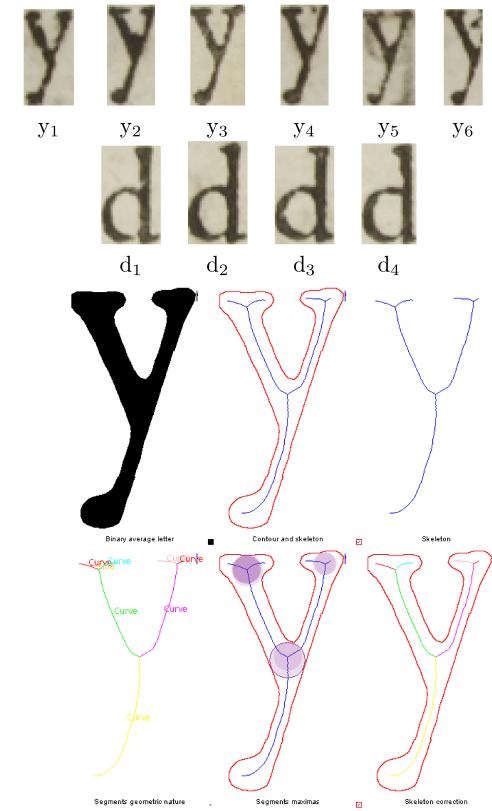
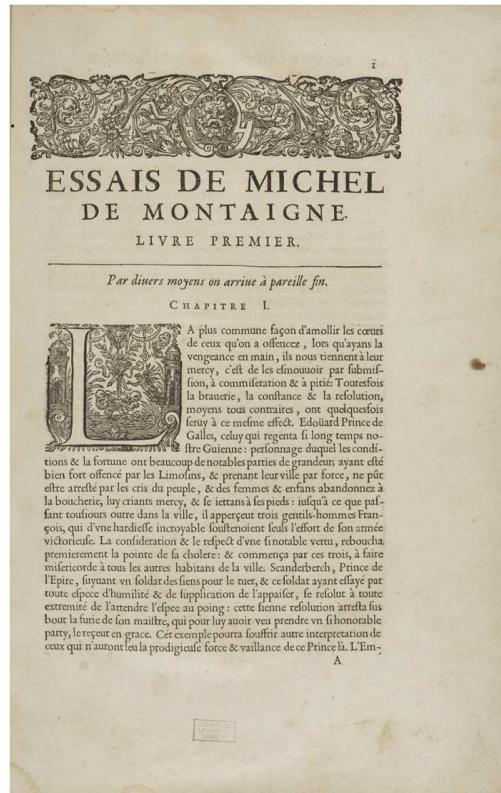
- textuelles (le contenu)
- sémantique (la structure du document)
- graphique (la mise en page et la caractérisation des éléments graphiques)



Imitation typographique

Rendre disponible des dessins typographiques anciens dans des standards numériques actuels est une activité commune dans le monde du design. Exercice de style, hommage, positionnement sur un marché spécifique... Ce travail est celui de typographes qui réinterprètent les formes du passé dans un objectif précis et avec un processus teinté de subjectivité.

Des recherches scientifiques comme [Re-Typograph](#) proposent des processus d'objectivations de la forme pour une plus grande pertinence historique et technique (comme le **squelette générique**). C'est cette recherche de la reproduction parfaite qui les amène, par exemple, à supprimer les imperfections de l'impression au plomb.



Cette démarche est complémentaire aux deux autres. Le traitement formel est basique et ignore les spécificités technique et historique des spécimens, proposant une approche plus photographique que typographique. Toutes les caractéristiques du fac-similé sont intégrées dans la création des nouvelles formes (les défauts d'impression, de conservation, de numérisation), ne cherchant pas à créer un **idéal numérique** d'un document historique mais un outil de **re-production numérique**.

La démarche d'ouverture de ce projet est surtout une tentative de ré-intégration des outils mathématiques et scientifiques dans une pratique de création de forme.

Numérisation des documents

Les plateformes :

La première étape avant de reproduire un document est d'en posséder une version numérique. Pour celà, il existe deux possibilités :

- Numériser le document soi-même.
- Récupérer un fichier déjà existant.

Dans tout les cas, la qualité et la résolution de la numérisation aura un impact dans tout le processus de numérisation.

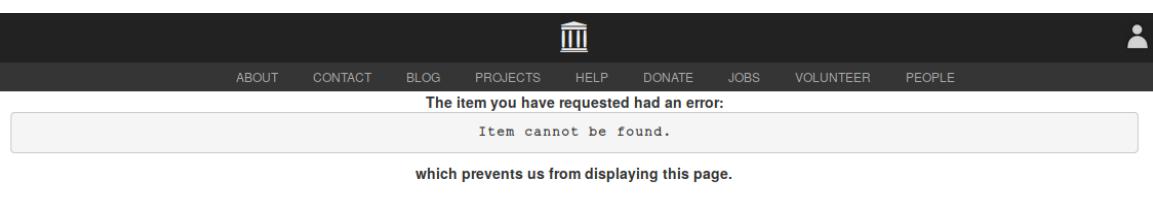
Il existe de nombreuses plateformes qui proposent des documents à télécharger. En effet, ... Dans le cas des œuvres du domaine public, le leader de la numérisation est GoogleBooks...

Par exemple ce livre de 1839 n'est disponible (pour le moment) que sur GoogleBooks :

Industrie française: rapport sur l'exposition de 1839, Volume 1



et (pas encore) sur [Internet Archive](#)



"Peu importe ! Tant qu'il est disponible gratuitement..."

Mais la license de GoogleBooks est, entre autre, imcompatible avec [Wikipedia](#). Car Google interdit les usages commerciaux de ces scans d'œuvres élevé depuis longtemps dans le [domaine public](#). Ce qui amène à des situations paradoxales comme celle-ci :

7. ^ Marcellin JOBARD, "Industrie française: rapport sur l'exposition de 1839 – Volume II, p. 350-351." French industry, report on the 1839 exhibition, Vol 2 pp. 350-351 (French text available on-line)

L'article ne peut pas intégrer directement les passages intéressants. Une citation ne suffisant pas car l'article parle spécifiquement d'éléments graphiques de cet ouvrage. Le sourçage reste indirecte et l'information est plus complexe à transmettre sans illustrations.

Digitized by Google

Ce filigrane est la marque de propriété de l'entreprise américaine sur l'image du livre.



A propos de ce livre

Ceci est une copie numérique d'un ouvrage conservé depuis des générations dans les rayonnages d'une bibliothèque avant d'être numérisé avec précaution par Google dans le cadre d'un projet visant à permettre aux internautes de découvrir l'ensemble du patrimoine littéraire mondial en ligne.

Ce livre étant relativement ancien, il n'est plus protégé par la loi sur les droits d'auteur et appartient à présent au domaine public. L'expression "appartenir au domaine public" signifie que le livre en question n'a jamais été soumis aux droits d'auteur ou que ses droits légaux sont arrivés à expiration. Les conditions requises pour qu'un livre tombe dans le domaine public peuvent varier d'un pays à l'autre. Les livres libres de droit sont autant de liens avec le passé. Ils sont les témoins de la richesse de notre histoire, de notre patrimoine culturel et de la connaissance humaine et sont trop souvent difficilement accessibles au public.

Les notes de bas de page et autres annotations en marge du texte présentes dans le volume original sont reprises dans ce fichier, comme un souvenir du long chemin parcouru par l'ouvrage depuis la maison d'édition en passant par la bibliothèque pour finalement se retrouver entre vos mains.

Consignes d'utilisation

Google est fier de travailler en partenariat avec des bibliothèques à la numérisation des ouvrages appartenant au domaine public et de les rendre ainsi accessibles à tous. Ces livres sont en effet la propriété de tous et de toutes et nous sommes tout simplement les gardiens de ce patrimoine. Il s'agit toutefois d'un projet coûteux. Par conséquent et en vue de poursuivre la diffusion de ces ressources inépuisables, nous avons pris les dispositions nécessaires afin de prévenir les éventuels abus auxquels pourraient se livrer des sites marchands tiers, notamment en instaurant des contraintes techniques relatives aux requêtes automatisées.

Nous vous demandons également de:

- + *Ne pas utiliser les fichiers à des fins commerciales* Nous avons conçu le programme Google Recherche de Livres à l'usage des particuliers. Nous vous demandons donc d'utiliser uniquement ces fichiers à des fins personnelles. Ils ne sauraient en effet être employés dans un quelconque but commercial.
- + *Ne pas procéder à des requêtes automatisées* N'envoyez aucune requête automatisée quelle qu'elle soit au système Google. Si vous effectuez des recherches concernant les logiciels de traduction, la reconnaissance optique de caractères ou tout autre domaine nécessitant de disposer d'importantes quantités de texte, n'hésitez pas à nous contacter. Nous encourageons pour la réalisation de ce type de travaux l'utilisation des ouvrages et documents appartenant au domaine public et sérieux heureux de vous être utile.
- + *Ne pas supprimer l'attribution* Le filigrane Google contenu dans chaque fichier est indispensable pour informer les internautes de notre projet et leur permettre d'accéder à davantage de documents par l'intermédiaire du Programme Google Recherche de Livres. Ne le supprimez en aucun cas.
- + *Rester dans la légalité* Quelle que soit l'utilisation que vous comptez faire des fichiers, n'oubliez pas qu'il est de votre responsabilité de veiller à respecter la loi. Si un ouvrage appartient au domaine public américain, n'en déduisez pas pour autant qu'il en va de même dans les autres pays. La durée légale des droits d'auteur d'un livre varie d'un pays à l'autre. Nous ne sommes donc pas en mesure de répertorier les ouvrages dont l'utilisation est autorisée et ceux dont elle ne l'est pas. Ne croyez pas que le simple fait d'afficher un livre sur Google Recherche de Livres signifie que celui-ci peut être utilisé de quelque façon que ce soit dans le monde entier. La condamnation à laquelle vous vous exposeriez en cas de violation des droits d'auteur peut être sévère.

À propos du service Google Recherche de Livres

En favorisant la recherche et l'accès à un nombre croissant de livres disponibles dans de nombreuses langues, dont le français, Google souhaite contribuer à promouvoir la diversité culturelle grâce à Google Recherche de Livres. En effet, le Programme Google Recherche de Livres permet aux internautes de découvrir le patrimoine littéraire mondial, tout en aidant les auteurs et les éditeurs à élargir leur public. Vous pouvez effectuer des recherches en ligne dans le texte intégral de cet ouvrage à l'adresse <http://books.google.com>

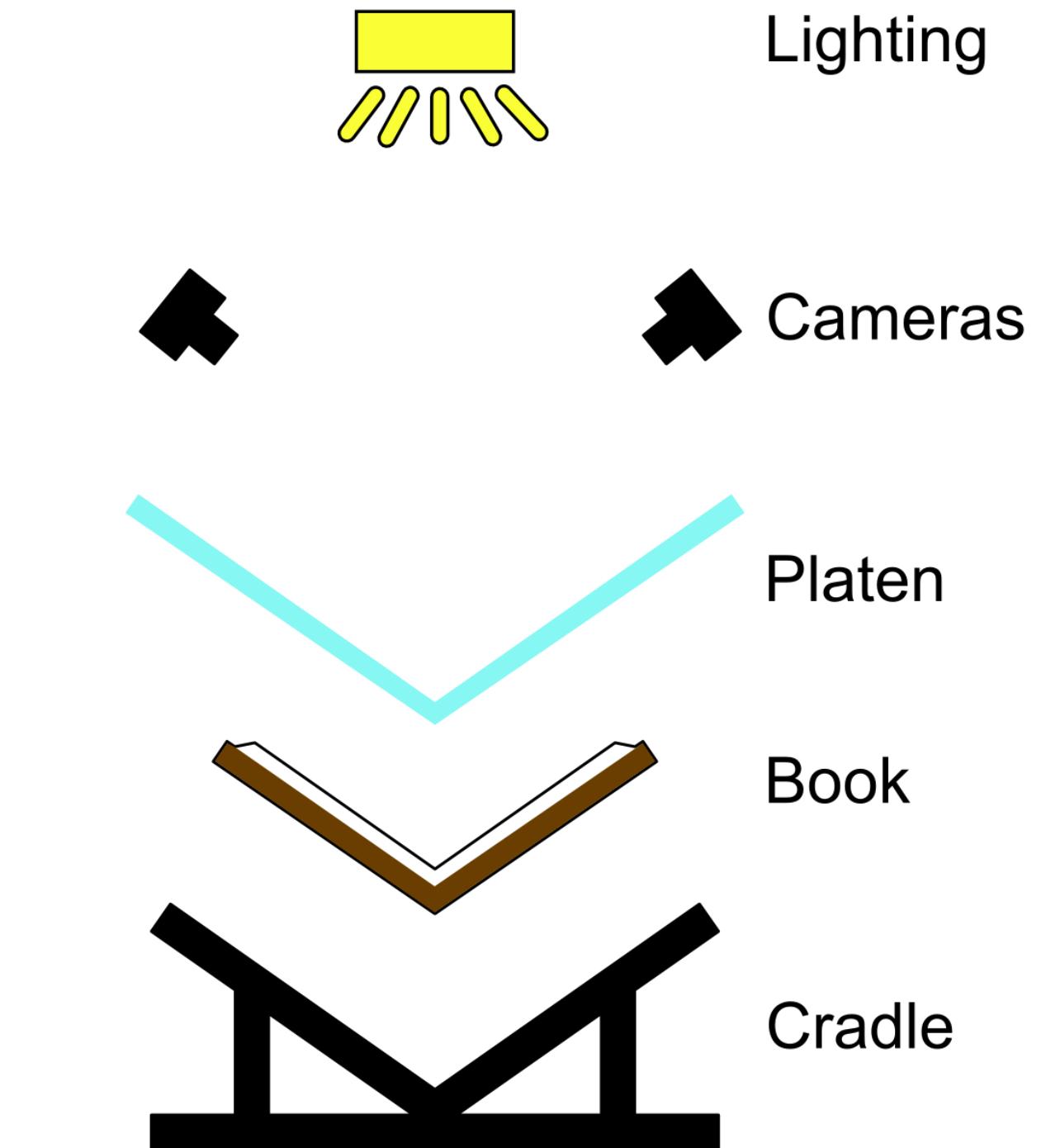
tpb.py : bot de transfert GoogleBooks -> InternetArchive

Automatiser des action sur le web en simulant une activité **humaine** avec la librairie python **selenium**. Ce script utilise le driver de **Firefox** disponible sur le

DIYBookScanner :

La possession des outils de numérisation devient donc importante pour permettre le partage du patrimoine écrit. Si le scanner à plat s'est démocratisé avec l'imprimante de bureau, il ne convient pas aux formes reliées. C'est pour cela que des communautés travaillent sur des outils de numérisation DIY et open source pour rendre accessible à ceux qui veulent partager leur livres comme ils l'entendent.

DIYBookScanner.org regroupe par exemple des dizaines de designs différents de dispositif pour photographier les pages d'un livre dans les meilleures conditions, selon les moyens et les compétences de chacuns.



bookscanner.py

Ce script est basé sur la librairie **gphoto2** qui permet de contrôler les boîters d'appareil photo numérique.

```
>> gphoto2 --auto-detect # lister les APN branché à l'ordinateur  
>> gphoto2 --capture-image # prendre une photo
```

Si on veut automatiser la prise de vue par ce dispositif, il faut choisir des APN compatibles avec la librairie (liste dispo avec **--list-cameras**) et jongler avec les modes de communication **PTP** ou **MTP**

bookscanner.py utilise le wrapper python de la librairie :

```
>> apt install libgphoto* && pip install gphoto2-cffi
```

Le script permet de contrôler le ou les appareils photos en déclenchant la prise de vue et en téléchargeant l'image sur la machine. Dans l'état, il est contrôlé par un signal en **série** pouvant venir d'un **Arduino**.

Le reste du traitement des images se fait avec l'interface graphique de [ScanTailor](#)

Analyse de mise en page

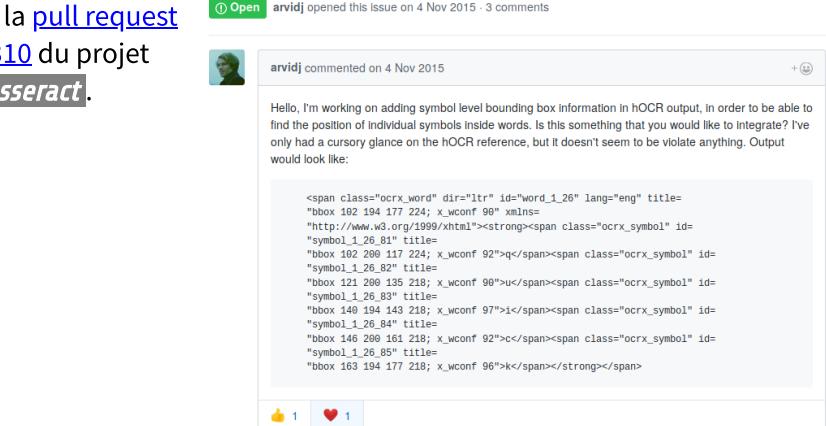
HOOCR ou PAGE ?

- HOOCR est un format de donnée de "OLR" (Optical Layout Recognition) basé sur le html. L'outil en ligne de commande de [Tesseract](#) avec l'option HOOCR ne s'arrêtait qu'au niveau des mots.
- Le labo de recherche [PRIMa](#) a développé son propre format le PAGE basé sur le xml. Ils proposent des outils open-source pour les visualiser et les éditer mais l'outil permettant de générer ces analyses n'est disponible que sur Windows et les sources sont introuvable. Pourtant TPT utilise [Tesseract](#) et permet de générer un OLR jusqu'au niveau des caractères. (Possibilité de convertir le HOOCR en PAGE avec [PAGEConverterValidator](#))

Mais :

Pour dessiner les contours des glyphes qui composent le document, nous avons besoin de connaître les coordonnées et les dimensions de chacune d'entre elle. Les documents de OLR doivent donc comprendre toute l'arborescence de la mise en page, jusqu'aux caractères.

Symbol level bounding box information in hOCR output
#135
voir : [issue #135](#)
et la [pull request #310](#) du projet [Tesseract](#).



L'option `hocr_char_boxes` n'est pas disponible dans la version 3.04.01 dispo sur Debian Stretch (ni la branche master 4.00.00alpha). Il faut donc compiler un [fork de la 3.05 avec ce tuto](#). Ne pas oublier [le\(s\) fichier\(s\) de langue et d'osd à mettre dans /usr/local/share/tessdata/](#) ou spécifier à la main avec l'option `--tessdate-dir`

```
>> git clone https://github.com/nickjwhite/tesseract.git --branch hocrcharboxes  
--single-branch tesseract-hocrcharboxes
```

Structure d'un document *hOCR*

```
...  
<p class='ocr_par' lang='fra' title="bbox930">  
  <span class='ocr_line' title="bbox 348 797 1482 838; baseline -0.009 -6">  
    <span class='ocrx_word' title='bbox 348 805 402 832; x_wconf 93'>  
      Un  
    </span>  
    <span class='ocrx_word' title='bbox 421 804 697 832; x_wconf 90'>  
      autre  
    </span>  
    <span class='ocrx_word' title='bbox 717 803 755 831; x_wconf 96'>  
      mote  
    </span>  
  </span>  
</p>  
...
```

Analyse

Analyses de mise en page réalisées sur les fichiers dans [/Pages](#).

- [Tesseract](#) -> **HOCR**(html) : zones et mots:
 - code : `tesseract -l fra [input image] [output file] hocr`
 - [/Layout/hocr](#)
- [Tesseract charboxes] -> **HOCR**(html) : zones, lignes, mots et glyphes:
 - code : `tesseract {image file} -c tessedit_create_hocr=1 -c hocr_char_boxes=1 {output name}`
 - [/Layout/hocr-charboxes](#)
- [PRIMA Tesseract OCR to PAGE](#) (disponible que sur windows)
 - -> **PAGE**(xml) : zones:
 - code : `...`
 - [/Layout/T2P-layout](#)
 - -> **PAGE**(xml) : zones, lignes, mots et glyphes:
 - code : `...`
 - [/Layout/T2P-layout-glyphs](#)

Avec le script : `Toolbox/olr-analyse.py`

```
>> python3.5 Toolbox/olr-analysis.py [-c CONFIGFILE] [-t TARGET [TARGET ...]]  
[-o OUTPUT] [-l LANG]  
# exemple:  
>> python3.5 Toolbox/olr-analysis.py -t Pages/ -o Layout/hocr-charboxes/ -l fra
```

Correction

hocr-tools

[hocr](#) est une boite à outil écrite en Python pour valider et corriger les fichiers **HOCR**

```
>> hocr-combine Layout/hocr-charboxes/*.hocr > Layout/p336-p356.html
```

ensuite, pour suivre la [specification du format pour la visionneuse](#) on applique au document le regex suivant :

- rechercher : `id="page_1" title="image "Pages/p(\d+)\.\.png"; bbox (\d+ \d+ \d+ \d+); ppageno \d+"`
- remplacer : `id="page_1" title="image ../Pages/p$1.png; bbox $2; ppageno $1"`

Générer un pdf "recherchable" : `hocr-pdf . > hocr.pdf` dans un dossier où il y a un fichier **HOCR** pour un **JPEG** (ex : Layout/hocr-charboxes/)

Il est possible de corriger les fichiers hOCR à partir de fichiers txt ligne par ligne corrigés à la main. (pas encore testé mais cela ne règle pas le problème des césures)

WebViewer

Visionneuse d'analyses de mise en page (layout) en HTML sur les scans des pages.

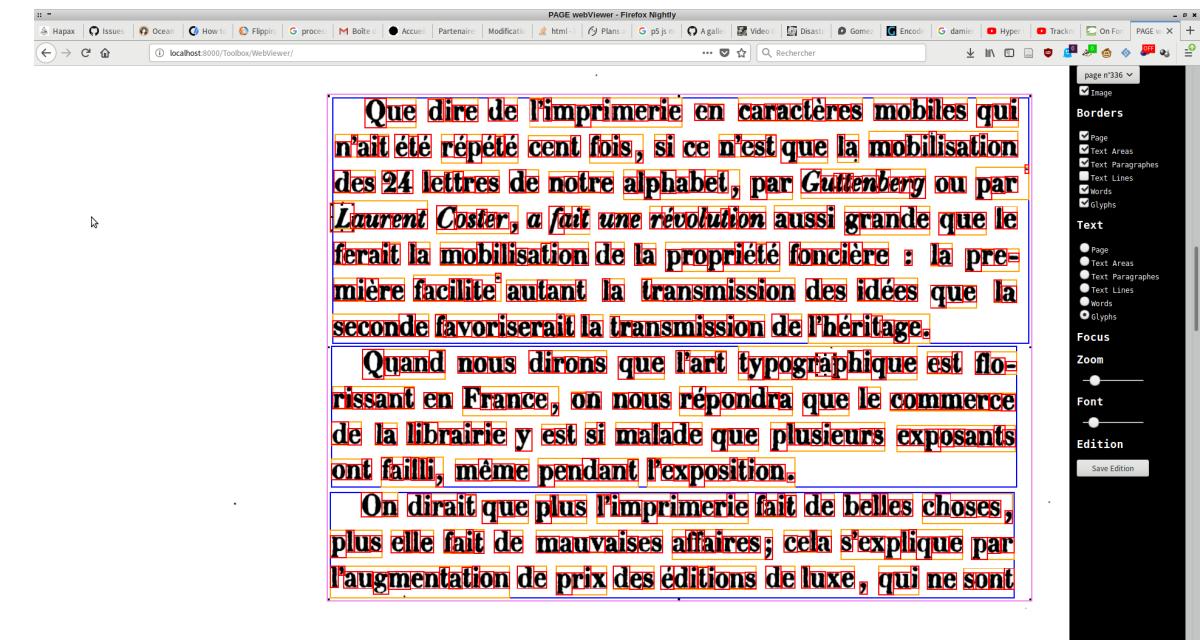
```
>> python3.5 -m http.server 8000
```

ou pour lancer le serveur et configurer l'appli :

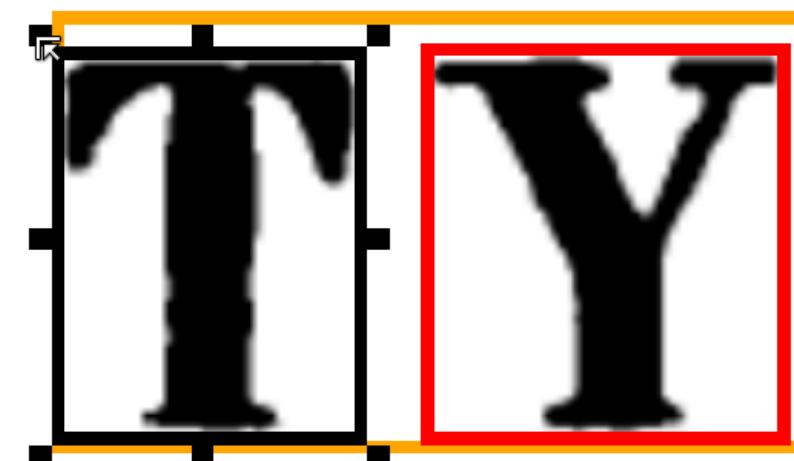
```
>> python3.5 Toolbox/run-web-viewer.py
```

- lance un serveur local
- édite le fichier de configuration `Toolbox/WebViewer/config.json`

Ouvrir le navigateur et entrer l'url : `localhost:8000/Toolbox/WebViewer`



Cette interface permet aussi de modifier les fichiers `hOCR`s en redimensionnant les contours des zones et en cliquant sur `Save Edition`.



Dessin de caractère à partir des scans des glyphs :

Extraction

Tri

depuis des fichiers HOCR

Les images des glyphs sont extraits avec le script `/Toolbox/extract-images-from-HOCR.py`:

```
>> python3.5 Toolbox/extract-images-from-HOCR.py [-c CONFIGFILE] [-H HOCR [HOCR  
...]][-i IMAGE [IMAGE ...]] [-m MARGIN] [-o OUTPUT]  
# exemple :  
>> python3.5 Toolbox/extract-images-from-HOCR.py -H Layout/hocr-charboxes/ -i  
Pages/ -o Glyphes/auto-extracted/ -m 2
```

elles sont enregistrées sous la forme : `glyph-confiance-page-id_mot-numéro.png`

- `glyph`: caractère ou nom du glyphe
- `confiance`: score de confiance du logiciel OCR en %
- `page`: numéro de la page
- `id_mot`: attribut "id" du mot (parent direct) du glyphe dans le fichier HOCR
`(#word_1_numéroDuMot)`
- `numéro`: numéro du glyphe dans la page

exemple : `a-85-339-word_1_230-4591.png`

Les glyphs extraits sont directement placés dans des dossiers portant leur nom, pour gagner du temps sur l'étape de tri

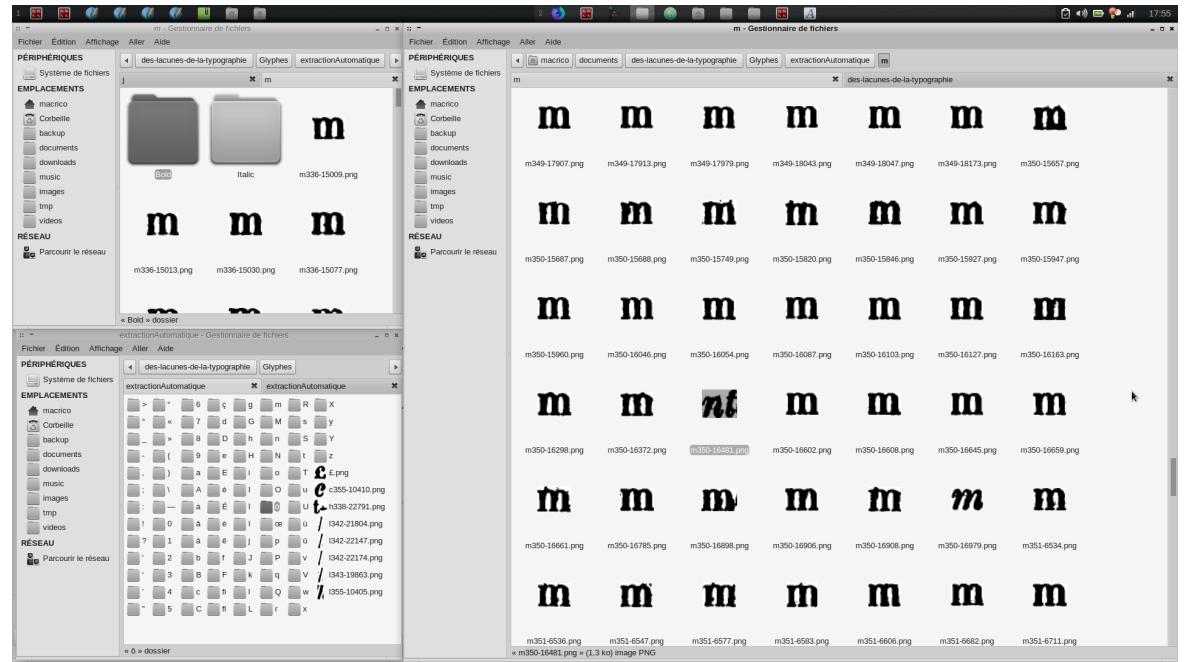
Le script `/Toolbox/sort-image-of-char.py`

```
>> python3.5 Toolbox/sort-image-of-char.py [-c CONFIGFILE] [-r ROOTFOLDER] [-o  
OUTPUT] [-n] [--hocrI HOCR] [--hocrO HOCR]  
# exemple :  
>> python3.5 Toolbox/sort-image-of-char.py -r Glyphes/auto-extracted/ --hocrI  
Layout/hocr-originaux --hocrO Layout/hocr-modifié
```

le script va :

- trier les images à la racine, celles qui ne sont pas triées. Il identifie le caractère de l'image par le premier élément de son titre. Il crée un dossier pour ce caractère si il n'existe pas encore. Il déplace l'image dans le dossier correspondant.
- retrier les images en parcourant les fichiers de tri pour repérer les images dont le nom ne correspond pas au fichier dans lequel elle est. Renommer ces images pour qu'elles correspondent au dossier dans lequel elles sont.
- modifier le nom du glyphe dans le fichier HOCR pour sauvegarder le tri

Il faut vérifier à la main si une image ne correspond pas au dossier où elle est, la déplacer dans le fichier où elle devrait être ou la supprimer. Lancer le script pour mettre à jour leur nom et le fichier HOCR. Il faut aussi séparer les glyphs d'un autre style de caractère (italique, gras...).



préciser un fichier `hocr0` pour ne pas sauvegarder dans le fichier original

TODO : vérifier que ces sous-dossiers de style de caractère ne pose pas de problème au script

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

mieux automatiser la création des gifs et générer un fichier markdown avec tous les noms des gif créés pour rapidement vérifier les tris

Ouvrir dans la visionneuse

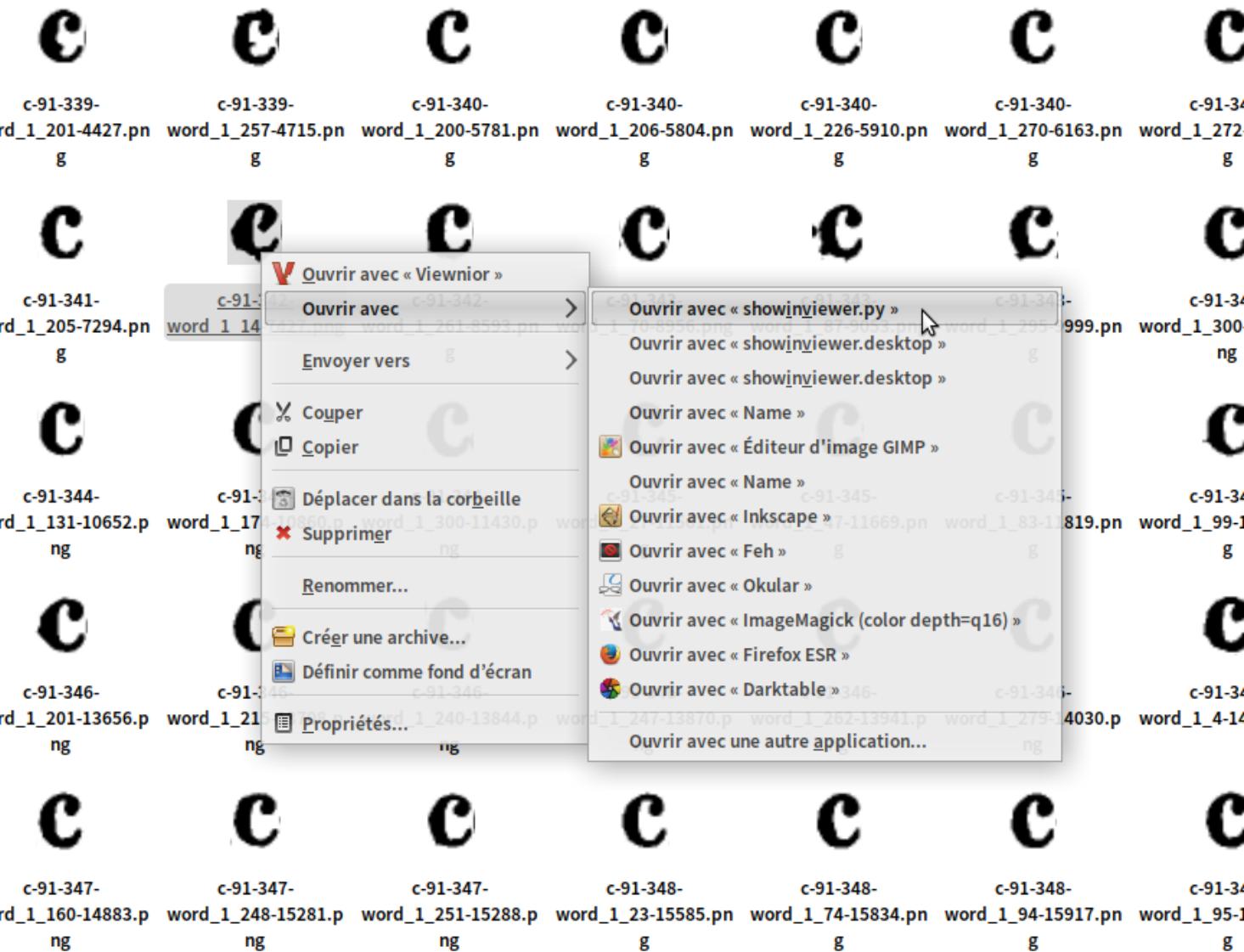
Pour accélérer le tri, on peut vérifier sur l'image de la page, quelle caractère a été réellement scanné. Pour cela on peut mettre en place un raccourci pour ouvrir le **WebViewer** à la bonne page et centrer la vue sur la glyphe en question. Un script ouvre un navigateur web à l'adresse où la visionneuse est lancée avec comme arguments la page et l'id du mot parent du glyphe.

```
>> python3.5 Toolbox/show_in_viewer.py [page-id_mot]
```

Par exemple : `python3.5 Toolbox/show_in_viewer.py 343-word_1_203` ouvrira Firefox sur `localhost:8000/Toolbox/WebViewer/#343-word_1_203`, ainsi la bonne page sera chargée et le focus se fera sur le bon nœud du document HTML. Pour rendre accessible cette manipulation depuis un navigateur de fichier, sur GNU/linux, on rend le script exécutables `chmod +x Toolbox/show_in_viewer.py`, puis on configure son gestionnaire de fichier pour pouvoir accéder au script depuis le menu `ouvrir avec...` pour que tous les .png puisse être ouvert avec notre programme.

Moyenne

click droit sur l'image du glyphe, ouvrir avec... -> showinviewer.py



! cette fonction lancera toujours le WebViewer du projet qui a été configuré. Si plusieurs projets sont en cours ou que le dossier du projet a été déplacé, il faudra éditer le script pour bien configurer les chemins à la main.

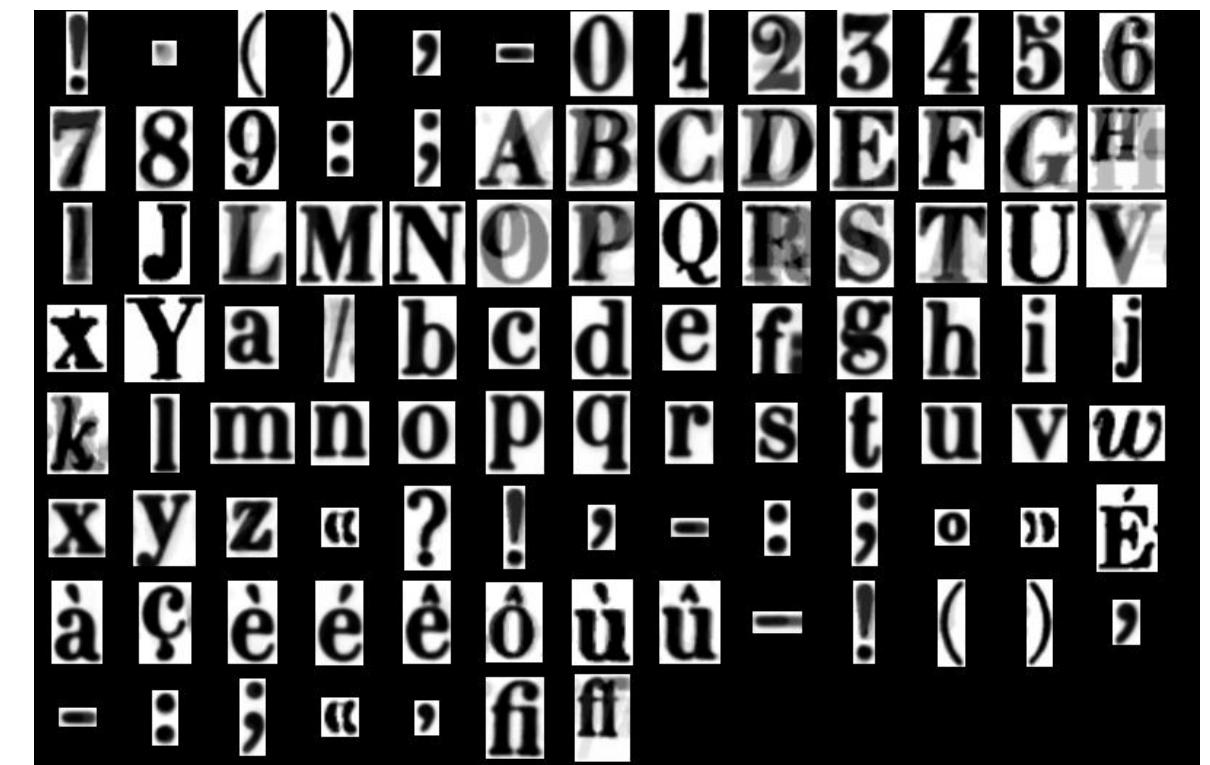
Créer un caractère qui soit représentatif de l'ensemble des glyphes de l'extrait pour gommer et intégrer les erreurs d'impression. Faire la "moyenne" de tous les glyphes, c'est-à-dire superposer toutes les images du caractère en réduisant leurs opacités. On utilise pour cela la fonction `convert` de [ImageMagick](#)

```
>> convert [images...] -average average.png
```

a Automatisé dans le script : [Toolbox/average.py](#)

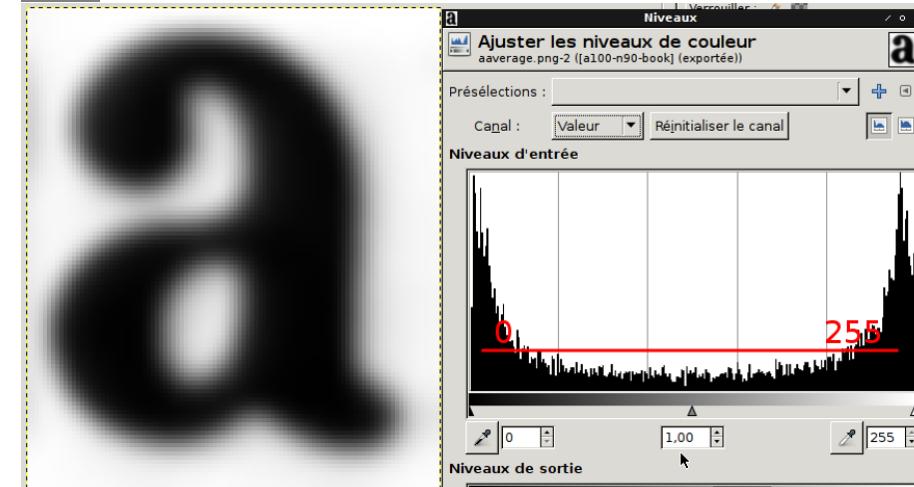
```
>> python3.5 average.py [-c CONFIGFILE] [-t TARGETFOLDER [TARGETFOLDER ...]] [-o OUTPUT]
# exemple :
>> python3.5 Toolbox/average.py -t Glyphes/extractionAutomatique-sorted/a/
Glyphes/extractionAutomatique-sorted/b/ Glyphes/extractionAutomatique-sorted/c
-o Glyphes/average/
```

Contraste de l'image et graisse du glyphe

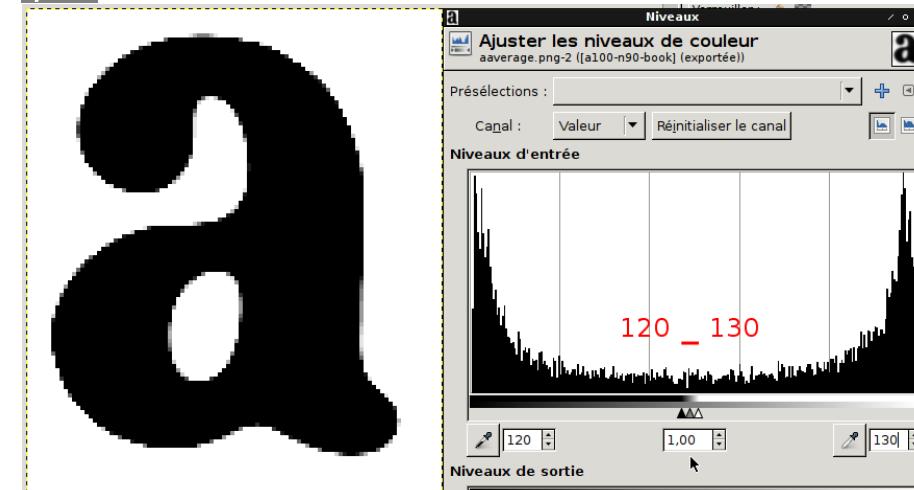


A partir de cette image moyenne, avant de la vectoriser, il faut augmenter ses contrastes. Les contours flous de cette forme permettent déjà de choisir la graisse des glyphes. Pour illustrer l'opération, avec [gimp](#), on augmente la résolution de l'image puis on utilise l'outil **Niveaux**.

avant:



après:



`120 + 130 / 2 = 125` <- numéro du niveau

[Un point sur les épaisseurs de typographie](#)

la préparation des bitmap peut-être réalisée aussi avec [mkbitmap](#)

ou alors avec [ImageMagick](#) :

```
>> convert {imgSource} -level {mini},{max} {imgOutput}  
#exemple :  
>> convert Glyphs/clean/a100.png -level 45%,55% Glyphs/clean/a100-50pc.png #45%  
+ 55% / 2 = 50pc <- pourcentage du niveau.
```

avec le script [Toolbox/level.py](#)

```
>> python3.5 Toolbox/level.py [-c CONFIGFILE] [-m METRICS] [-t TARGET [TARGET  
...]] [-o OUTPUT] [-l LEVEL [LEVEL ...]] [-d DELTA]  
# exemple :  
>> python3.5 Toolbox/level.py -t Glyphs/average/ -o Glyphs/levels/ -l 15 30 60
```

- agrandit la taille des images
- génère les différents contrastes



Vectorisation

avec Inkscape en mode GUI ou ligne de commande, ou directement avec [Potrace](#) puis retouche et simplification du tracé avec Inkscape.

```
>> inkscape -f clean/a100.png --select image10 --verb SelectionTrace  
# ouvre l'interface graphique d'inkscape directement avec l'outil de vectorisation  
>> potrace clean/a100.bmp -s -o test.svg  
# vectorisation en ligne de commande (voir man potrace)
```

```
potrace clean/a100.bmp -s -o -a test2.svg potrace a100.bmp -s --opttolerance 1 -o  
test4.svg potrace a100.bmp -s --opttolerance 2 -o test5.svg
```

Qualité du SVG.

Avec Potrace la noeud "path" est enfant d'une balise "g" avec une grosse transformation. Les coordonnées des points du path sont arrondies après la virgule ! Avec inkscape aussi les tracés sont "optimisés" (mélange de coordonnées relatives et absolues). Pour les passer en absolues :

'Edit> preferences > SVG Output > Path Data' to always use absolute coordinates (i.e. do not allow relative coordinates). This will only affect newly created paths, or existing objects for which a rewrite of the path data is triggered. For existing paths, use 'Edit > Select All in All Layers', and nudge the selection with the arrow keys (e.g. one step up and one back down again). This will trigger a rewrite of the path data in 'd' which will follow the changed preferences for optimized path data. resave.

Visionner les tracés :

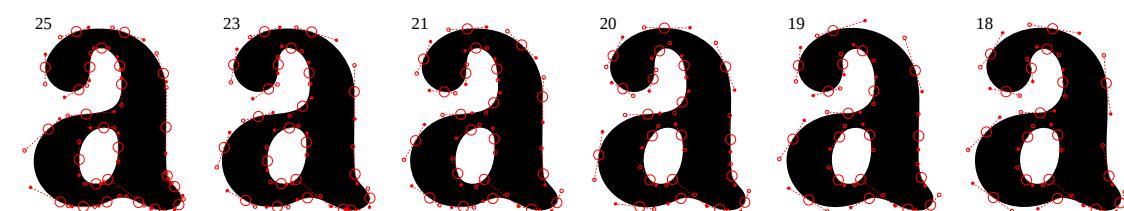
Visionneuse de tracés, points et poignées : [/Glyphes/vectors/index.html](#)

(to do: corriger le décalage entre les handles d'un point à un autre/OK-25-01-17/ mais il reste des problèmes pour les derniers points, après un segClosePath)

Simplification :

pour le moment la simplification des tracés n'est pas implémentée dans un script

simplification réalisée avec inkscape :



(to do : faire un script qui simplifie et sauvegarde le tracé une 30ene de fois et qui génère le html pour la visionneuse.)

Métriques : extraction depuis l'HOOCR

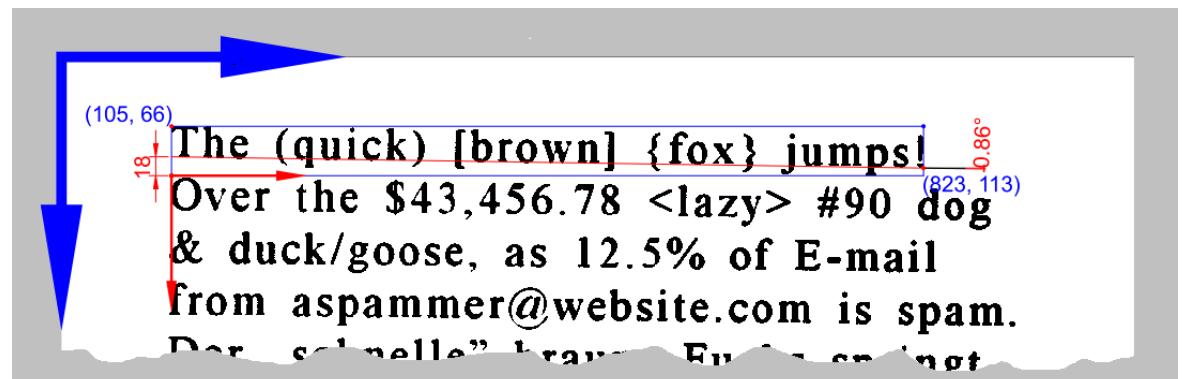
Script `vectorize.py`

```
>> python3.5 Toolbox/vectorize.py [-c CONFIGFILE] [-t TARGET [TARGET ...]] [-o  
OUTPUT] [--pnm PNM] [-s SIMPLE] [--html] [-l LEVEL]  
# exemple  
>> python3.5 Toolbox/vectorize.py -t Glyphes/levels/a30pc10d.png -o Glyphes/vectors/
```

- convertit en ppm et les enregistre dans `Glyphes/clean`
- vectorise dans `{outputFolder}`
- simplifie sous la forme `{char}{level}-simpl{nb simplification}.svg`
- crée un fichier html pour afficher les tracés et leurs poignées

On peut essayer de générer les métriques de notre police de caractère de la même manière que nous avons fait pour les contours : en faisant la moyenne de tout les résultats individuels récupéré par `Tesseract` dans les fichiers `hOCR`. Plusieurs éléments peuvent nous intéresser :

- `x_size`
- `baseline_angle`
- `baseline_offset`
- `x_descenders`
- `x_assenders`



- La taille de l'`espace` devrait être la distance entre les mots. (`ocrx_word`)
- Le `crenage` peut aussi être déduit de la position et de la taille des glyphs (`ocrx_cinfo`) deux à deux.

script `extract-metrics.py`

Ce script va justement extraire toutes ces informations des `hOCR`, en faire la moyenne et les enregistrer dans le fichier `Font/metrics.json`.

```
>> python3.5 extract-metrics.py [-c CONFIGFILE] [-H HOOCR [HOOCR ...]] [-i IMAGE  
[IMAGE ...]] [-m METRICS] [--capheight CAPHEIGHT]  
#exemple  
>> python3.5 extract-metrics.py -h Layout/hocr-charboxes -i Page/ --capheight 80
```

Les metrics dans FontForge

Import des contours Font SVG avec inkscape :

- puisque le cadratin est égale à 1000, la chasse de base l'est aussi. Les approches correspondent aussi à la position du tracé dans le fichier SVG source.
- dans FontForge on peut faire des métriques automatiques (autoWidth) à partir de valeurs (separation, min, max). Est-ce qu'il ne faut pas plutôt utiliser les valeurs sorties en hOCR ?
- ce type de données sont formatées dans des fichiers comme **AFM** (Adobe Font Metrics) regroupant les réglages par groupe de caractère ou pour chaque paire de caractères. (**Crénage** ou **Kerning**)
- on peut importer ces paramètres dans fontforge avec **mergeFeatures** (et/ou LookUp ?) ou les rentrer avec **addPosSub**
- on peut faire le crenage dans inkscape mais comment les importer dans FontForge ?

structure d'une fonte SVG :

```
...
<font>
    horiz-adv-x="1024"
    id="font74"
    inkscape:label="fonte 1">
        <font-face>
            units-per-em="1024"
            id="font-face76"
            font-family="SVGFont 1" />
        <missing-glyph>
            d="M0,0h1000v1024h-1000z"
            id="missing-glyph78" />
        <glyph>
            glyph-name="glyphe 1"
            id="glyph80"
            unicode="a"
            d="M ...
        </glyph>
    ...
</font>
...
```

X-path avec **xml.etree** :

```
svg.findall('.//{http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd}namedview/{http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd}guide')#trouve les guides
svg.find('.//{http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd}guide[@{http://www.inkscape.org/namespaces/inkscape}|label="baseline"]')#trouve le guide qui a comme nom baseline
```

en graphique :

- créer un document de type "**canevas typographique**" `Glyphes/font.svg`
- importer chaque glyphs (`Glyphes/vectors/*.svg`)
- /!\ une matix se crée à l'import (de type `matrix(1.3333333,0,0,1.3333333,277.26123,389.97951)`) et en copiant le tracé de la glyphe depuis inkscape vers le canevas typographique)*
- appliquer les transformations aux tracés (et non au groupe qui l'entoure) **extension->Modifier le chemin->Apply Transform**
- placer le tracé dans un calque typographique **extension->Typographie->Ajouter un calque de glyphe ...**
- une fois chaque glyphe dans un calque typographique, l'extension va générer la font SVG automatiquement avec : **extension->Typographie->convertir les calques de glyphes en police SVG**

en script :

`importSVGinSVGFont.py` pour faire tout ça.

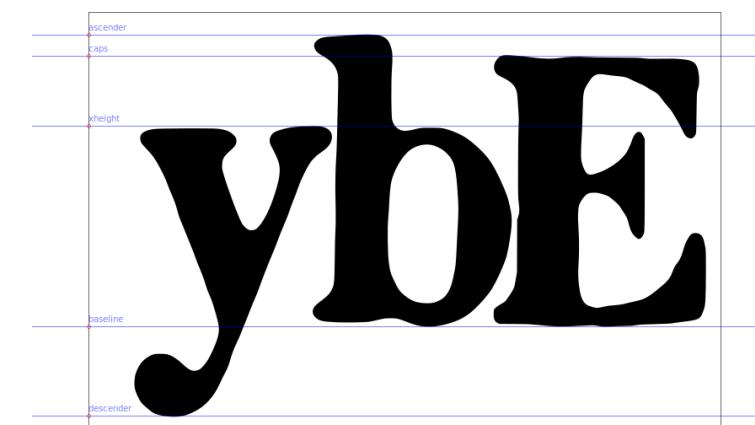
```
>> python3.5 Toolbox/importSVGinSVGFont.py {vectorFiles} > {outputFile}
>> python3.5 Toolbox/importSVGinSVGFont.py Glyphes/vectors2/*30pcl0d.svg > Font/font.svg
```

script

l'extension `adjustSVG2Font` permet de placer et redimensionner les glyphs par rapport à la grille de la typo SVG

- on récupère les glyphs et leur BBOX avec `svgpathools`. -> `(x0,y0,x1,y1)` a = `(253.8495, 649.43808, 271.0452283962141, 753.5576795047747)` (**pozXobj=Gauche,maxXobj=Droit,EM-maxZobj=haut,EM-pozYobj=bas**)

Chaque caractère va venir se placer à un endroit spécifique du canvas. Certain contours se poserons sur la **baseline**, d'autre en dessous de la **capheight** etc.



Font SVG to FontForge :

python-fontforge :

Bonnes pratiques :

extrait de [DesignWithFontForge](#) :

- garder la taille du cadratin à 1000 (donc 1000px dans la svgfont)
- configurer la baseline dans "Element > Font Info > General" :
- descendante fontforge = **y** baseline inkscape
- assendante fontforge = 1000 - **y**
- chaque attribut **d** des **path** doit terminer par un "**Z**" pour que les normales (intérieur/extérieur soit bonnes dans FontForge)

la lib fontforge pour python : [ref ici](#)

ne fonction qu'avec python 2.7

petit guide rapide :

- `font = fontforge.open("Font/font.sfd")` # ouvre un fichier fontforge depuis python
- `newGlyph = font.createMappedChar("a")` # crée un caractère
- système de "selection" comme sur l'interface pour faire une action sur plusieurs glyphs (comme `autoWidth`) : `font.selection.all()` puis `font.autoWidth(200)`
- `font.save(newfont.sfd)`

script

Le script `svgfont2fontforge` permet d'importer une `fontsvg` dans `FontForge`. Il exporte chaque calque en un fichier `SVG` puis les importe pour le caractère correspondant dans FontForge.

