vecchio metodo

- importazione librerie
- definizione funzioni
- variabili indipendenti
- variabili dipendenti
- glifo
  - variabili locali
  - lunghezza massima manipolatori
  - lunghezza effettiva manipolatori
  - coordinate punti
  - chiamata del glifo
  - disegno
  - rimando del glifo

nuovo metodo

- importazione librerie
- corstruttore della classe
- variabili indipendenti
- variabili dipendenti
- funzione glifo
  - chiamata del glifo
  - variabili locali
  - quarti/poligoni
  - rimando del glifo
- richiamo della funzione glifo

Organizzazione dello script

importazione librerie

```
import robofab.world
[…]
from tools import tools
```

corstruttore della classe

```
## Class constructor
tools = tools.Tools()
```

variabili indipendendi

```
## Variabili indipendenti (valori nominali)
# Altezza x
xht_nom = 500
[…]
#innesto
inn_nom= 0.6
```

variabili dipendendi

```
## Variabili dipendenti (valori nominali)
# Altezza x
xht_eff = xht_nom
[…]
#innesto
inn_eff= inn_nom
```

funzione glifo

```
# Glyph: o
def drawing_o():
```

chiamata del glifo

```
# Calling glyph
font = CurrentFont ()
glyph = font[‘o’]
glyph.clear()
```

variabili locali

```
## Variabili Locali [o]
# Espansione
exp_o = 2 * exp_eff /  (nor_eff + 1)
[…]
# Spessore Curve Orizzontali
tch_o = tcv_eff * mon_o
```

quarti/poligoni

```
### quarto/poligono
## punti di ancoraggio
# disegno
```

rimando del glifo

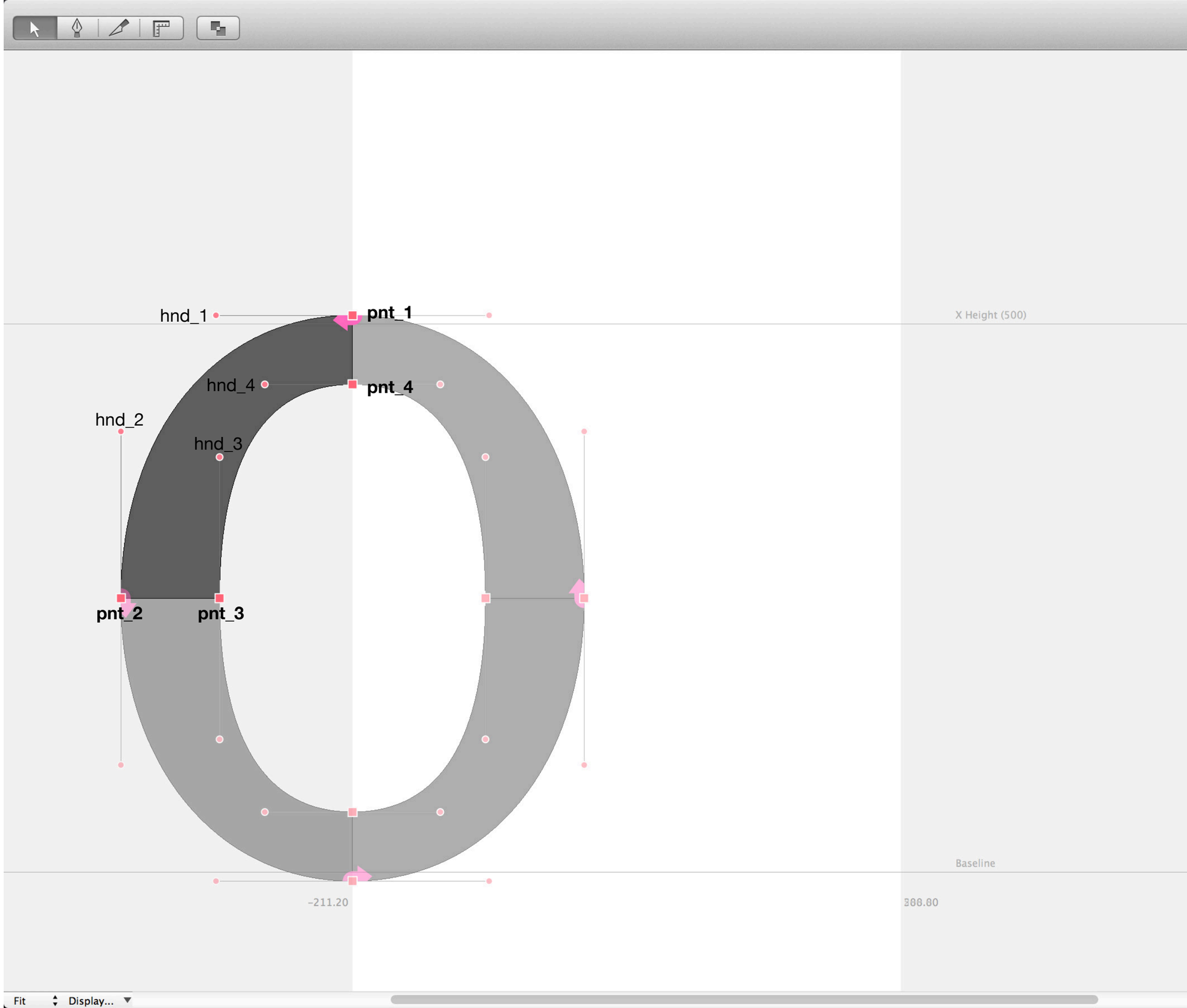
```
return glyph
```

richiamo della funzione glifo

```
# Glyph draw
o = drawing_o()
```

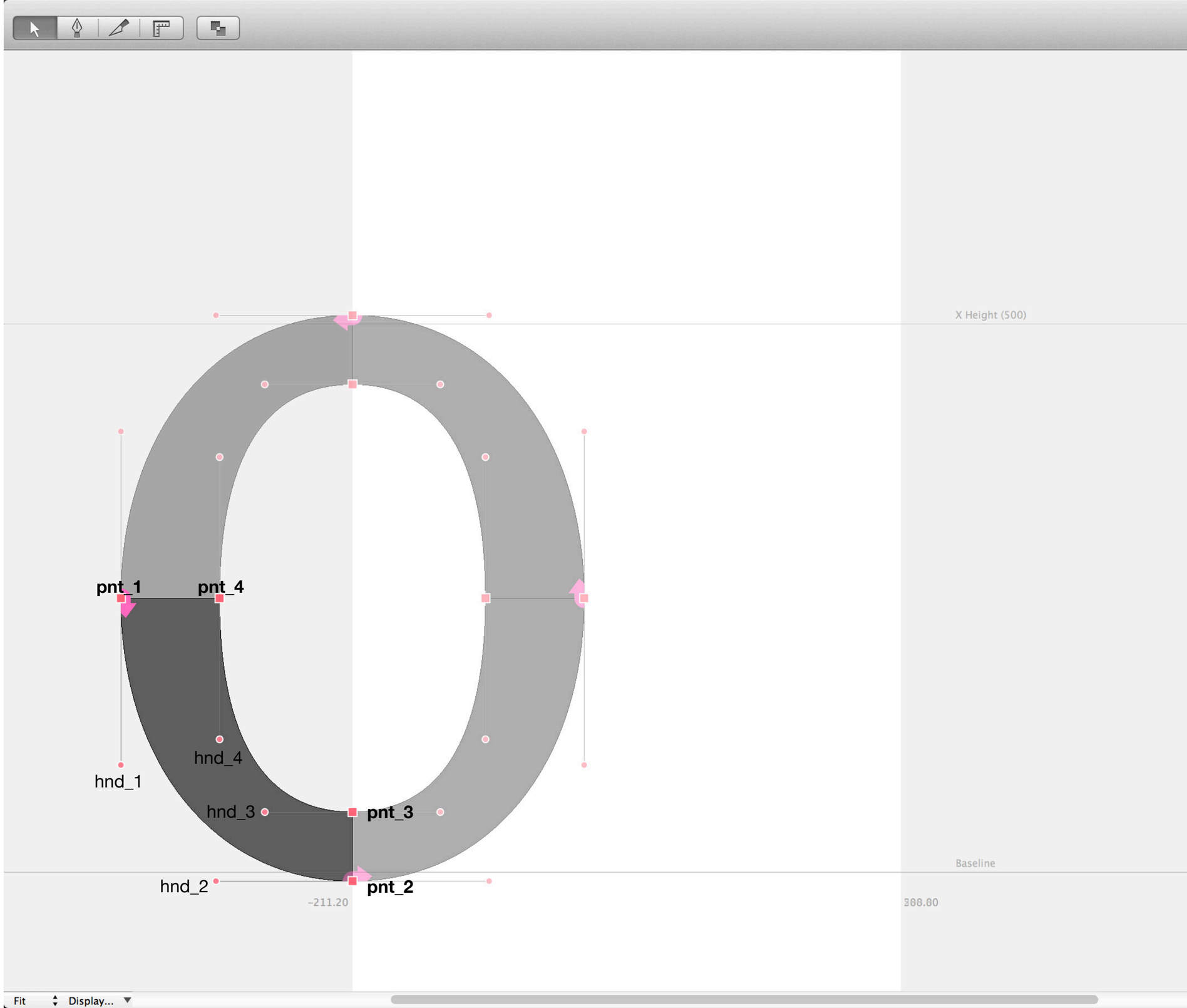
GenericQuarter
scrip di un quarto

```
### UpperLeft
## punti di ancoraggio

#ext
pnt_1 = (dis_hor) , (xht_eff + ovs_eff)
hnd_1 = (dis_hor + ((-wdt_o/2 - tcv_eff/2) * sqe_eff)) , (xht_eff + ovs_eff)
hnd_2 = (-wdt_o/2 - tcv_eff/2) , ((xht_eff/2-dis_ver) + ((+xht_eff/2+ovs_eff) * sqe_eff))
pnt_2 = (-wdt_o/2 - tcv_eff/2) , (xht_eff/2-dis_ver)

#int
pnt_3 = (-wdt_o/2 + tcv_eff/2) , (xht_eff/2 + dis_ver)
hnd_3 = (-wdt_o/2 + tcv_eff/2) , ((xht_eff/2) + ((xht_eff/2 + ovs_eff - tch_o) * sqi_eff))
hnd_4 = ((-wdt_o/2 + tcv_eff/2) * sqi_eff) , (xht_eff - tch_o + ovs_eff)
pnt_4 = (-dis_hor) , (xht_eff - tch_o + ovs_eff)

# Disegno
tools.genericQuarter(glyph,
              (pnt_1) ,
              (hnd_1) , (hnd_2) , (pnt_2) ,
              (pnt_3) ,
              (hnd_3) , (hnd_4) , (pnt_4))
```

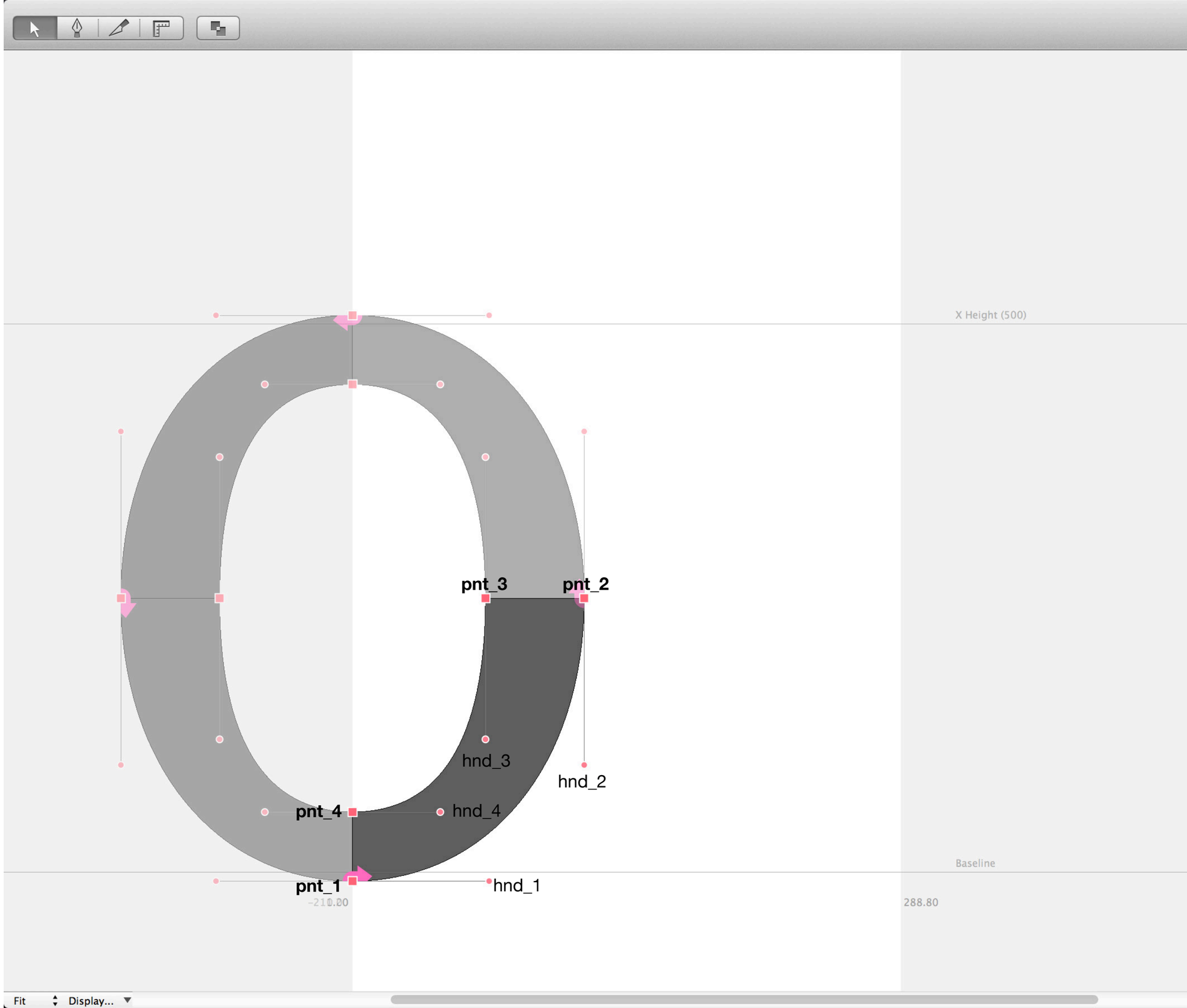


UpperLeft
quarto in alto a sinistra

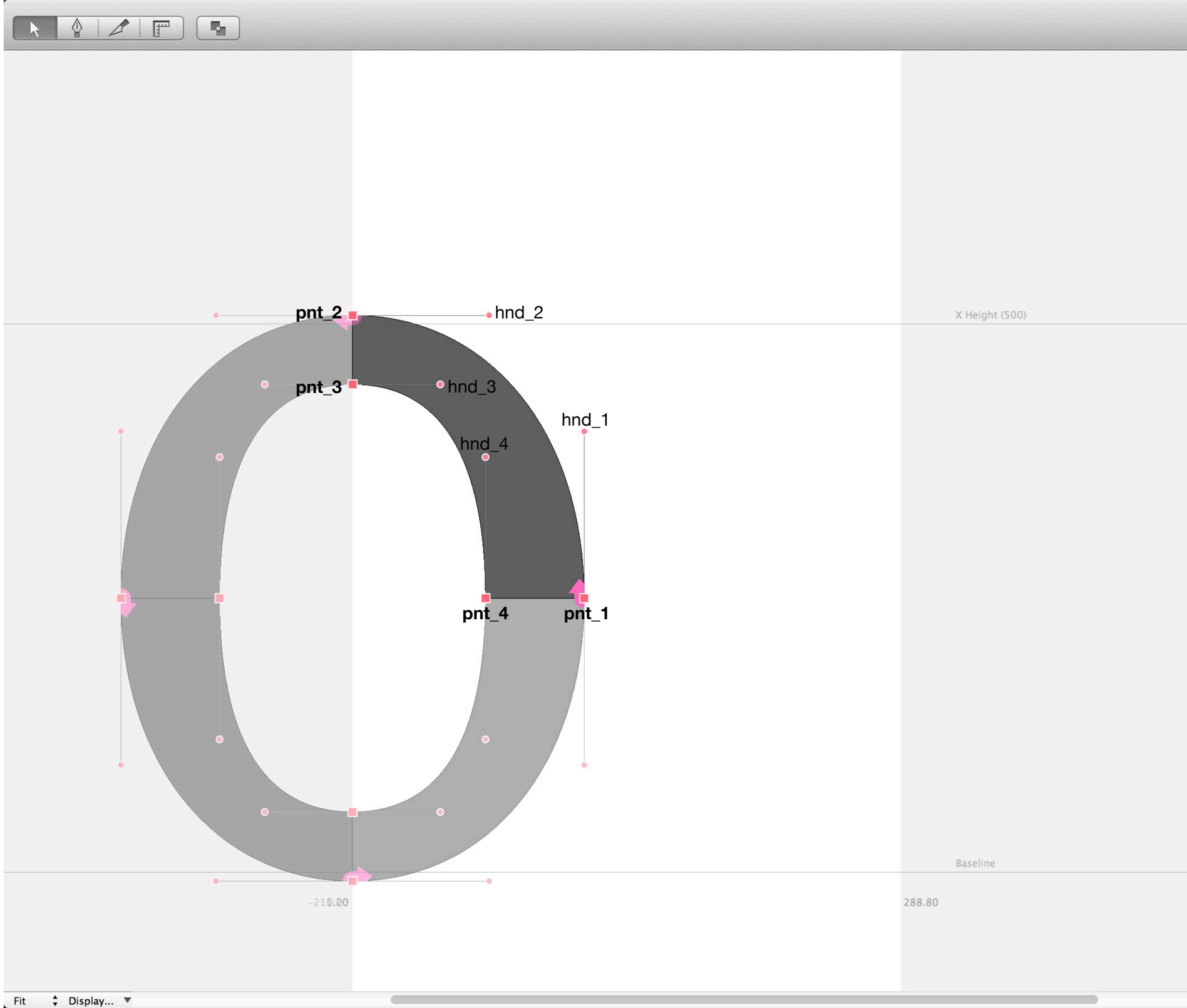
BottomLeft
quarto in basso a sinistra



BottomRight
quarto in basso a destra



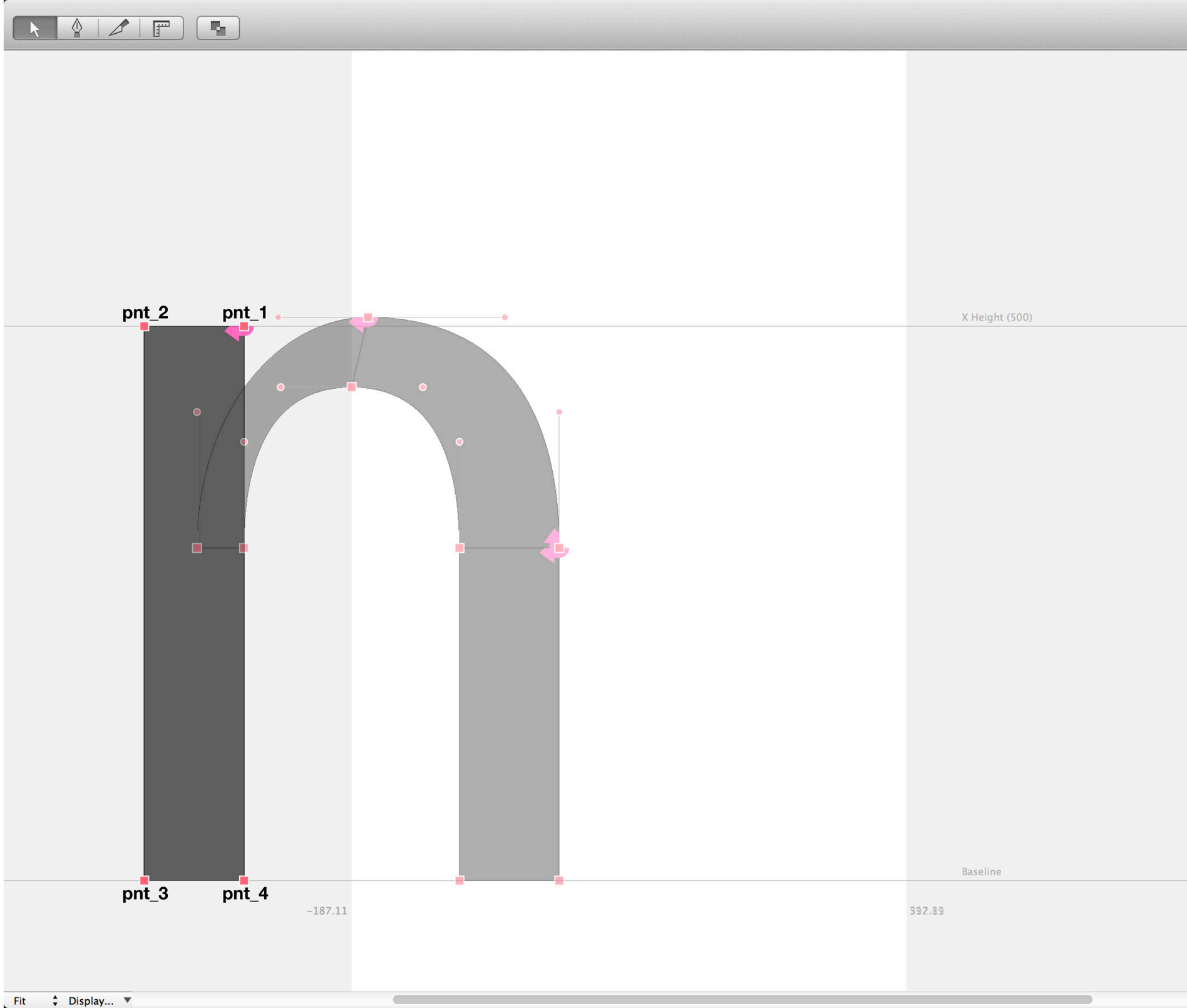
UpperRight
quarto in alto a destra



Polygon
scrip di un poligono

```
### StemLeft
## punti di ancoraggio

pnt_1 = (-wdt_n/2 + tsv_eff/2) , (xht_eff)
pnt_2 = (-wdt_n/2 - tsv_eff/2) , (xht_eff)
pnt_3 = (-wdt_n/2 - tsv_eff/2) , 0
pnt_4 = (-wdt_n/2 + tsv_eff/2) , 0

points_list = [(pnt_1) , (pnt_2) , (pnt_3) , (pnt_4)]

#disegno
tools.drawPolyByPoints(glyph, points_list)
```

Stem
poligoni



Suggerimenti di sintassi

usare uno spazio prima e dopo:

+ - (quando non riferito all'inverso di una variabile)

```
xht_eff + ovs_eff
xht_eff + ovs_eff

–wdt
– wdt
```

*** /** (quando riferiti ad operazioni fra variabili)

```
xht_eff/2
xht_eff / 2

tsv_eff = wgt_eff * xht_eff
tsv_eff = wgt_eff*xht_eff
```

= , (sempre)

```
xht_nom = 500
xht_nom=500

tools.lin (mon_eff , 0 , 0.15 , 1 , 0)
tools.lin (mon_eff,0,0.15,1,0)
```

racchiudere sempre operazioni fra variabili tra parentesi tonde

```
pnt_1 = (dis_hor) , (xht_eff + ovs_eff)
pnt_1 = dis_hor , xht_eff + ovs_eff
```