CONTEXT

Pauta Module

Grids for calligraphy practice

Andrés Conrado Montoya Acosta

March 14, 2024

Grids for calligraphy practice

```
\writestatus{loading}{Pauta (ver: 2024.03.14)}

\startmodule [pauta]
\usemodule   [module-catcodes]
\unprotectmodulecatcodes
```

Setups

We define a start/stop pair to configure the macro structure. Each `Pauta` call will have a "section" of sorts.

```
\definestartstop[pauta][
  before={\page\start},
  after={\stop\page},
]
```

We use setups to configure the top / bottom marks for a Pauta page

```
\startsetups pauta:layout:bottommarks
  \setuplayout[top=\zeropoint, bottom=2\bodyfontsize]
  \setupbottomtexts[\PAUTAinfoLeft][\PAUTAinfoRight]
\stopsetups

\startsetups pauta:layout:topmarks
  \setuplayout[top=2\bodyfontsize, bottom=\zeropoint]
  \setuptoptexts[\PAUTAinfoLeft][\PAUTAinfoRight]
\stopsetups

\startsetups pauta:content:leftmark
  Nib:\space\PAUTAnibWidth
  \quad(\PAUTAascenders/\PAUTAxHeight/\PAUTAdescenders)\quad
  \PAUTAnibAngle\textdegree{}
\stopsetups

\startsetups pauta:content:rightmark
  \doifsomething{\PAUTAhand}{\PAUTAhand}
  \doifsomething{\PAUTAhandInfo}{\quad(\PAUTAhandInfo)}
\stopsetups
```

Grids for calligraphy practice

Pauta definition

We define the doPauta macro, that takes up to 16 arguments. All arguments are optional, they come with default values. If you want to disable the top / bottom text, you can use `infoLeft=,` and `infoRight=,` .

Do not leave other variables blank. Just don't define them if you want to accept the defaults.

We use `\getparameters` to, well, get the parameters. Created following the wiki article for Handling Arguments

```
\starttexdefinition nospaces doPauta [#1]
  \getparameters[PAUTA] [
    hand=,
    handInfo=,
    infoPosition=bottom,
    infoLeft={\setup{pauta:content:leftmark}},
    infoRight={\setup{pauta:content:rightmark}},
    displayNibs=false,
    displayAngleMarks=false,
    nibWidth=3mm,
    nibAngle=35,
    ascenders=3,
    xHeight=4,
    descenders=3,
    adjustment=0,
    mainColor={s=.4},
    secondaryColor={s=.6},
    tertiaryColor={s=.8},
    #1,
  ]
```

This creates a macro for each config value, containing the value. We use these values to setup all the variables we need.

Configure the info position:

```
  \doifelse{\PAUTAinfoPosition}{bottom}
    {\setup[pauta:layout:bottommarks]}
    {\setup[pauta:layout:topmarks]}
```

Configure the colors:

```
  \definecolor[tertiaryColor] [\PAUTAtertiaryColor]
  \definecolor[mainColor]     [\PAUTAmainColor]
  \definecolor[secondaryColor][\PAUTAsecondaryColor]
```

Setup MP variables:

```
  \setupMPvariables[pauta][
    displayNibs=\PAUTAdisplayNibs,
    displayAngleMarks=\PAUTAdisplayAngleMarks,
    nibWidth=\PAUTAnibWidth,
    nibAngle=\PAUTAnibAngle,
    ascenders=\PAUTAascenders,
    xHeight=\PAUTAxHeight,
    descenders=\PAUTAdescenders,
    adjustment=\PAUTAadjustment,
  ]
```

Finally, draw the MP graphic *pauta* based on user settings.

```
  \startpauta\useMPgraphic{pauta}\stoppauta
```

```
\stoptexdefinition
```

Empty argument handling

We use the `\dosingleargument` macro to call doPauta, as explained at Handling Arguments. This helps us avoid issues with empty arguments.

```
\starttexdefinition Pauta
  \dosingleargument\doPauta
\stoptexdefinition
```

Grids for calligraphy practice

METAPOST macros

First, we include the hatching.mp macro definitions to create a hatched pattern for the nib angle guides. After that, we include all our vardefs that won't change between runs.

```
\startMPinclusions
  % ---------------------------------------------------------------------
  % hatching.mp
  % ---------------------------------------------------------------------
  % Made in BOP, Gdánsk, Poland
  % E-mail contact: B.Jackowski@gust.org.pl
  % Public domain software (no copyrights, copylefts, copyups, copydowns, etc.)
  % Current version: 21.09.2000 -- ver 0.11 (ending semicolon
  %    added in |extra_beginfig| ; |hatchfill_| introduced in order
  %    to make possible something like |def fill = hatchfill enddef|
  def hatchfill_ expr c = addto currentpicture contour c _op_ enddef ;

  vardef hatchfill text p =
    save c_, p_ ; path p_ ; color c_[\\] ; c_.num := 0 ;
    save withcolor_ ; let withcolor_ := withcolor ;
    def withcolor = ; c_[incr c_.num] := enddef ;
    p_ := p ; let withcolor := withcolor_ ;
    for i_ := c_.num downto 1: % find the least ``true'' fill
      c_.num' := i_ ; exitif bluepart(c_[i_])>0 ;
    endfor
    if c_.num>0:
      for i_ := c_.num' upto c_.num:
        if bluepart(c_[i_])<0: draw hatched(p_)c_[i_] ;
        else: hatchfill_ p_ withcolor c_[i_] ; fi
      endfor
    else: hatchfill_ p_ ; fi
  enddef ;

  vardef hatched(expr o) primary c =
    save a_, b_, d_, l_, i_, r_, za_, zb_, zc_, zd_ ;
    path b_ ; picture r_ ; pair za_, zb_, zc_, zd_ ;
    r_ := image(
      a_ := redpart(c) mod 180 ; l_ := greenpart(c) ; d_ := -bluepart(c) ;
      b_ := o rotated -a_ ;
      b_ := if a_>=90: (lrcorner b_--llcorner b_--ulcorner b_--urcorner
b_--cycle)
      else: (llcorner b_--lrcorner b_--urcorner b_--ulcorner b_--cycle) fi
      rotated a_ ;
      za_ := point 0 of b_ ; zb_ := point 1 of b_ ;
      zc_ := point 2 of b_ ; zd_ := point 3 of b_ ;
      if hatch_match>0:
        n_ := round(length(zd_-za_)/l_) ; if n_<2: n_ := 2 ; fi ; l_ :=
length(zd_-za_)/n_ ;
      else: n_ := length(zd_-za_)/l_ ; fi
      for i_ := if hatch_match>0: 1 else: 0 fi upto ceiling n_-1:
        draw_hatched_band((i_/n_)[zd_, za_], (i_/n_)[zc_, zb_], a_, l_, d_) ;
      endfor
    ) ;
    clip r_ to o ; r_
  enddef ;

  def draw_hatched_band(expr za, zb, a, l, d) = % normally, |a| and |l| are
ignored
    draw za--zb withpen pencircle scaled d _hop_ ;
  enddef ;

  def hatchoptions(text t) = def _hop_ = t enddef enddef ;

  newinternal hatch_match ; hatch_match := 1 ;
  hatchoptions() ; extra_beginfig := extra_beginfig & " ;hatchoptions() ;" ;

  % -----------------------------------------------------------------
  % Vardefs
  % -----------------------------------------------------------------
  % Draw a section (ascendant, x-height or descendant)
  vardef Section(expr lines, startPosition) =
    % Draw section lines
    for i = 0 upto lines :
      save endPos ; endPos := i*nibWidth ;
      save distance ; distance := endPos + startPosition ;
      pair a ; a := (0, distance) ;
      pair b ; b := (TextWidth, distance) ;
      draw a -- b withpen pencircle scaled thinLine
        withcolor secondaryColor ;
    endfor ;

    % Draw section separators
    draw (0, startPosition) -- (TextWidth, startPosition)
      withpen pencircle scaled thickLine
      withcolor mainColor ;

    draw (0, distance) -- (TextWidth, distance)
      withpen pencircle scaled thickLine
      withcolor mainColor ;

    % Return the distance
    distance
  enddef ;

  % Draw a line with three sections
  vardef TextLine(expr startPosition, ascendant, xHeight, descendant) =
    if displayNibs = true :
      % Calculate nib-width marks
      numeric lines ; lines := descendant + ascendant + xHeight ;
      numeric nibs ; nibs := lines - 1 ;
      % Display nib-width marks
      for i = 0 upto nibs :
        numeric nib ; nib := i * nibWidth + startPosition ;
        fill unitsquare scaled nibWidth shifted
          (if (i mod 2 = 0) :
            (0, nib)
          else:
            (nibWidth, nib)
          fi) withcolor tertiaryColor ;
      endfor ;
    fi ;

    % Draw the three sections
    numeric descendants, xHeights, ascendants ;
    descendants := Section(descendant, startPosition) ;
    xHeights := Section(xHeight, descendants) ;
    ascendants := Section(ascendant, xHeights) ;

    % Draw a rectangle to contain dotted angle guides
    numeric space ;

    if displayAngleMarks = true :
      if displayNibs :
        space := nibWidth * 2 ;
      else :
        space := 0 ;
      fi ;

      path angleContainer ; angleContainer :=
        (space, startPosition) -- (space, ascendants) --
        (TextWidth, ascendants) -- (TextWidth, startPosition) --
        cycle ;

      % We use hatching.mp to fill the box with lines
      % with the right angle, gap and pen
      hatchoptions (withcolor tertiaryColor dashed evenly) ;
      hatchfill angleContainer withcolor (nibAngle, nibWidth*3, -thinLine) ;
    fi ;

    % Return final position, adding interline space
    ascendants + nibWidth * 2
  enddef ;

  % Line thickness that won't change
  numeric thinLine ; thinLine = 0.2mm ;
  numeric thickLine ; thickLine = 0.4mm ;
\stopMPinclusions
```

Finally, we use the graphic, redefining the variables we need for each run.

```
\startuseMPgraphic{pauta}
  % These variables will be recalculated every time we call the MPgraphic
  % and that's why I don't put them in the MPinclusions

  % Display square nib-width marks at line start?
  boolean displayNibs ;
  if known \MPvar{displayNibs} :
    displayNibs = \MPvar{displayNibs} ;
  else :
    displayNibs = false ;
  fi ;

  % Color settings
  color mainColor ; mainColor = \MPcolor{mainColor} ;
  color secondaryColor ; secondaryColor = \MPcolor{secondaryColor} ;
  color tertiaryColor ; tertiaryColor = \MPcolor{tertiaryColor} ;

  % Text height (without footer or header)
  numeric SimpleTextHeight ; SimpleTextHeight = TextHeight - (HeaderHeight +
FooterHeight) ;

  % Distance between lines (nib width)
  numeric nibWidth ; nibWidth = \MPvar{nibWidth} ;

  % Ascenders
  numeric ascenders ; ascenders = \MPvar{ascenders} ;

  % X-Height
  numeric xHeight ; xHeight = \MPvar{xHeight} ;

  % Descenders
  numeric descenders ; descenders = \MPvar{descenders} ;

  % Adjustment value for layout
  numeric adjustment ; adjustment = \MPvar{adjustment} ;

  % Full line height
  numeric lineHeight ; lineHeight = (ascenders + xHeight + descenders +
adjustment) * nibWidth ;

  % Available lines
  numeric availableLines ; availableLines = floor(SimpleTextHeight / lineHeight)
;

  % Start position (zero)
  numeric startPosition ; startPosition = 0 ;

  % Nib-width angle
  boolean displayAngleMarks ;
  if known \MPvar{displayAngleMarks} :
    displayAngleMarks := \MPvar{displayAngleMarks} ;
  else :
    displayAngleMarks := false ;
  fi ;

  numeric nibAngle ; nibAngle = \MPvar{nibAngle} ;

  % Draw a page
  for i=1 upto availableLines :
    startPosition := TextLine(startPosition, ascenders, xHeight, descenders) ;
  endfor ;
\stopuseMPgraphic

\stopmodule
```