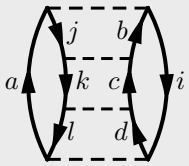
Diag2PS

A text driven diagram drawing tool

based on PSTricks

```
phl lU <;phl rU >
cl R;~
phl lU <;phl rU >
```




```
loop ( DD < @l "$a$"
loop ) DD > ["R0" @r "$j$" "R1" @r "$k$" "R2" @r "$l$" "R3"]
move Rr
loop ( DD < ["L0" @l "$b$" "L1" @l "$c$" "L2" @l "$d$" "L3"]
loop ) DD > @r "$i$"
#define IL(A, B) move A;il B
IL("R0", "L0");IL("R1", "L1");IL("R2", "L2");IL("R3", "L3")
```

Michael Hanrath

Köln 2012–2019

Contents

1 First Steps **1**

1.1 Build . . . 1
1.1.1 Prerequisites . . . 1
1.1.1.1 Compilation . . . 1
1.1.1.2 Usage . . . 1
1.1.2 Compilation . . . 1
1.1.3 Initial test . . . 1
1.2 Running Diag2PS . . . 2
1.2.1 Command line arguments . . . 2
1.2.1.1 Output generation . . . 2
1.2.1.2 Reading custom configuration file . . . 2
1.3 Debugging . . . 2
1.3.1 Pre-processing related errors . . . 2
1.3.2 Debugging LaTeX errors . . . 3
1.4 Temporary files . . . 3

2 `Diag2PS` drawing language reference **5**

2.1 Notation . . . 5
2.2 Positioning . . . 5
2.2.0.1 `Diag2PS`'s coordinate system . . . 5
2.2.1 `move` *pos* . . . 5
2.2.1.1 Relative positioning . . . 6
2.2.1.2 Absolute positioning . . . 6
2.2.2 `~` follow last direction vector . . . 7
2.2.2.1 `storepos` and `clearpos` . . . 8
2.3 Objects . . . 8
2.3.1  `showGrid`: show PSTricks grid . . . 8
2.3.2  Dot objects . . . 9
2.3.3  Interaction/cluster lines . . . 9
2.3.4 Particle/hole lines . . . 10
2.3.4.1 Modifiers (optional) common to all particle/hole lines . . . 10
2.3.4.2  `phl`: straight particle/hole line . . . 10
2.3.4.3  `loop`: loops . . . 12
2.3.4.4  Special bent particle/hole lines . . . 15
2.3.5  Pointing arrows . . . 16
2.3.6 Interface to `PSTricks` . . . 17
2.3.6.1  `rput`: TeX/LaTeX output . . . 17
2.3.6.2  `uput`: TeX/LaTeX output . . . 18
2.3.6.3  `verbatim`: using PSTricks directly . . . 19
2.4 State change . . . 20
2.4.1 `push` and `pop` or `(` and `)` : save current state on stack . . . 20
2.4.2 `color`: set drawing color . . . 20
2.4.3 `path`: changing background of line elements . . . 21

2.4.4 `unitSize`: set unit size (RLDU) to x cm . . . 22
2.4.5 `labelYShift`: change vertical shift for labels (`@` instruction) . . . 22
2.4.6 `labelDist`: change distances for labels (`@` instruction) . . . 23
2.4.7 `pointDist`: change pointing arrow distance from node center . . . 23
2.4.8 `lineWidth`: set linewidth to x pt . . . 23
2.4.9 `bentArrows`: control bending of arrows on arc . . . 23
2.4.10 `reverse`: reverse all arrow directions . . . 24
2.4.11 `arrowFillColor`: set arrow fill color . . . 25
2.4.12 `arrowWidth`: set arrow width . . . 25
2.4.13 `arrowLength2Width`: set relation arrow length/width . . . 25
2.5 Miscellaneous . . . 26
2.5.1 Injecting preamble commands . . . 26
2.5.2 Tuning bounding boxes for alignment . . . 26

3 Examples **27**
3.1 Gallery . . . 27
3.1.1 Selected coupled cluster single projection and reversed arrows . . . 27
3.1.2 Using macros . . . 27
3.1.3 Moving arrows on crossing lines . . . 27
3.1.4 Crossed loops . . . 28
3.1.5 Hollow arrows (for e.g. spin averaged diagrams) . . . 28
3.1.6 $\hat{C}_2$ operator (disconnected) . . . 28
3.1.7 Gluon interaction lines . . . 28
3.1.8 Placing labels and arrows . . . 28
3.1.9 Hugenholtz diagrams . . . 29
3.1.10 4th order PT . . . 29
3.1.11 colored diagrams and using `pstricks` directly . . . 29
3.2 Fixing alignments between LaTeX and `Diag2PS` objects . . . 30
3.2.1 Bottom aligned (default) . . . 30
3.2.2 Center aligned . . . 30
3.2.3 Fixes . . . 31
3.2.3.1 Typeset whole equations within `Diag2PS` . . . 31
3.2.3.2 Center aligned with centered bounding box using `BBYCenter` . . . 31

A Language specification **33**
A.1 Lexer tokens . . . 33
A.2 Yacc (bison) grammar . . . 34

Chapter 1

First Steps

1.1 Build

1.1.1 Prerequisites

There are minor prerequisites for the compilation and translation of `Diag2PS`. In most cases the required packages – all of which are open source – will be installed anyway.

1.1.1.1 Compilation

Compilation of `Diag2PS` requires

- a recent C++ compiler (versions: `g++` $\geq$ 8.x.x, Apple CLANG $\geq$ 10)
  remark: current (June 2019) `g++` on debian stable (stretch) is version 6.3.0 and therefore too old. Use of a custom C++ compiler has to be announced by doing `export CXX=/path/to/C++-compiler` (on bash) before running `cmake`.
- `bison`, parser generator
- `flex`, lexicographical analyzer
- `make`, dependency and rule driven build tool (e.g GNU make)
- `cmake`, portable build system (generates `Makefile`s)

1.1.1.2 Usage

Running of `Diag2PS` requires

- the `Diag2PS` binary and various dynamic system dependent binaries (unless statically linked)
- a LaTeX installation providing packages amssymb, pstricks, pst-node, pst-coil, color, graphicx, pstricks-add, bm, vmargin (on debian based installations this is included by texlive-latex-extra)
- `ps2pdf`, `pdfcrop`, `awk` for creation of PDF files
- `pdftocairo` for creation of SVG files

1.1.2 Compilation

To compile `Diag2PS` from source untar `Diag2PS-1.0.tar.xz` and `cd` into the directory `Diag2PS-1.0/BUILD` and do a `cmake ../cmake;make`. After a few seconds the executable `Diag2PS` will reside in directory `Diag2PS-1.0/BUILD`.

1.1.3 Initial test

As an initial test try the following:

1. `./DiagPS <test.diag`
   reads input from `stdin` and writes LaTeX/PSTricks code to `stdout`.
2. `./DiagPS -pdf test.diag`
   reads input from file `.../test.diag` and writes PDF to `.../test.pdf`.

1.2 Running Diag2PS

By default (no flags given) PSTricks code is generated. If no input filename is given input is read from `stdin` with output written to `stdout`. If an input filename `file.xyz` is given the output is written to `file.tex`, `file.ps`, `file.pdf`, `file.svg` depending on the flags given.

The flags from the following sections may be given.

1.2.1 Command line arguments

1.2.1.1 Output generation

`-tex`: no additional program is run

`-ps`: runs `latex` and `dvips`

`-pdf`: additionally runs `ps2pdf` and `pdfcrop`

`-svg`: additionally runs `pdftocairo`

Options `-ps`, `-pdf`, `-svg` require a filename to be given.

1.2.1.2 Reading custom configuration file

To customize `Diag2PS` an optional configuration (user provided) file may be included as prologue. The default name for this configuration file is `$HOME/.Diag2PS`. This default may be changed by the options

`-noconfig`: disables read of any configuration file

`-config` <filename>: change the default file name

1.3 Debugging

A `Diag2PS` run consists of several steps:

1. input is piped through the C-preprocessor
2. preprocessed input is fed into the `Diag2PS` diagram language interpreter
3. LaTeX (PSTricks) output is generated
4. depending on command line arguments additional programs are run

   `-tex`: no additional program is run (default)

   `-ps`: runs `latex` and `dvips`

   `-pdf`: additionally runs `ps2pdf` and `pdfcrop`

   `-svg`: additionally runs `pdftocairo`

Each processing step may generate its own error messages.

1.3.1 Pre-processing related errors

To debug preprocessor related issues use

```
gcc -c -E -P -x c-header YourFile.diag -o YourFile.out
```

In `YourFile.out` you will find the preprocessed source for inspection.

Preprocessor errors typically arise from

- non-existent include files
- typos in preprocessor commands
- errors in macro definitions.

1.3.2 Debugging LaTeX errors

LaTeX errors may appear after the preprocessor and `Diag2PS` diagram interpretation has been carried out successfully. LaTeX errors are debugged best by running LaTeX separately. So use `DiagPS <input.diag >output.tex` and run LaTeX manually.

LaTeX errors typically arise from

- `\verbatim` commands
- strings passed on to LaTeX (e.g. labels, `\uput`, `\rput`)
- too large document coordinates passed on to LaTeX.

1.4 Temporary files

In order to run LaTeX `Diag2PS` needs a temporary directory. It resides in `/tmp` and is named `Diag2PS_TMP.XXXXXX`.

Chapter 2

Diag2PS drawing language reference

2.1 Notation

Syntax and examples are given with pale blue and light gray background, respectively.

Syntax examples use a regular expression like notation:

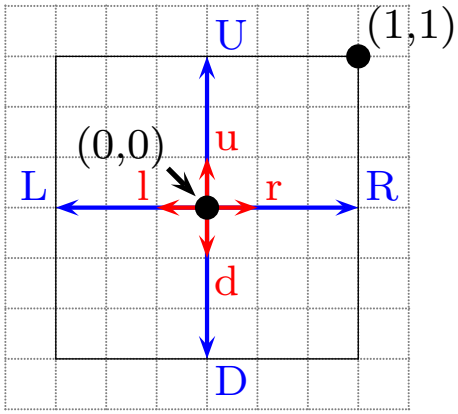
symbol	meaning
`'.'`	any single character
`A|B`	A or B (exclusive)
`?`	optional argument
`+`	one or more arguments
`*`	zero or more arguments
`[...]`	character classes
`(...)`	grouping

If appropriate the notation in this manual may be somewhat simplified to make it more readable. The given examples should clarify the usage. The precise regular expressions and grammar can be found in the appendix A. White space `[ \t]` (except new line) is in most cases insignificant. That is: `moveD` and `move D` and `move  D` are all equivalent. However, `move DD` and `move D D` is not equivalent and will most likely result in a syntax error.

2.2 Positioning

2.2.0.1 `Diag2PS`'s coordinate system

`Diag2PS` uses a grid of thirds as coordinate system:



2.2.1 `move` *pos*

Move the actual position.

2.2.1.1 Relative positioning

- syntax

  (i) `move [RLUDrlud|.]+`

  Shift the current position by given direction code.

  - `RLUD`: moves the current position one unit size right, left, up, down, respectively.
  - `rlud`: moves the current position $\frac{1}{3}$ unit size right, left, up, down, respectively.
  - prefixes
    - `|` scales the following step by $\frac{3}{2}$
    - `.` scales the following step by $\frac{1}{10}$ and executes it once for each given character '.'

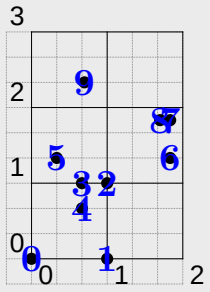
    Prefixes may be given multiple times. so
    - `||R` moves $\left(\frac{3}{2}\right)^2$ to the right (multiplicative)
    - `.......r` moves $7 \cdot \frac{1}{10}\frac{1}{3}$ to the right (additive)

  (ii) `move += (dx, dy)`

  Shift the current position by (dx, dy).

- example

relative positioning




```
#define mark(pos) bullet;(color "blue";rput | pos)
verbatim \Large \bf            // use large bold font
showGrid (-0.33, -0.33) (2, 3)// show coordinate
    ↪ grid
//-------------------------------------------------
mark(0)                    // set marker "1"
move R;mark(1)             // full step right
move U;mark(2)             // full step up
move l;mark(3)             // 1/3 step left
move d;mark(4)             // 1/3 step down
move luu;mark(5)           // 1/3 step right, 2/3 up
move |R;mark(6)            // 3/2 step right
move |u;mark(7)            // 1/2 step up
move ....l;mark(8)         // 4 * 1/10 * 1/3 steps left
move +=(-1, 0.5);mark(9)// relative positioning
```

2.2.1.2 Absolute positioning

- `move "<label>"`

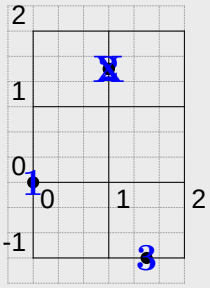
  Move to label `<label>`.

- `move = (x, y)`

  Move to (x, y).

- example

absolute positioning and labels




```
#define mark(pos) bullet;(color "blue";rput | pos)
verbatim \Large \bf            // use large bold font
showGrid (-0.33, -1.33) (2, 2.33)// show coordinate
    ↪ grid
//-------------------------------------------------
mark(1)                    // set marker "1"
move =(1, 1.5);mark(2)     // absolute positioning
storepos "P"               // save current position as
    ↪ "P"
move =(1.5, -1);mark(3)    // move somewhere else
move "P";mark(X)           // go back to "P"
```

`move =` $(x,\ y)$ should be used seldom as it prohibits shifting diagrams easily.

2.2.2 ~ follow last direction vector

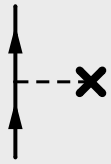
- purpose
  Shift the current position by the last directionial step. "~" may be used as a prefix to other commands.
- syntax

```
~
```

- examples

a)

following last direction



```
phl U >
~(il R;~cross)
phl U >
```

2.2.2.1 storepos and clearpos

- purpose

  - storepos
    Store the actual position in a label. Storing an actual position to an already used label will cause an error. If you want to reuse a label you must first delete it with clearpos. Labels are particularly useful to refer to locations on loops.
  - clearpos
    Remove specific stored positions. Calling clearpos with no arguments will delete all stored labels.

- syntax

```
storepos "<label>"
clearpos '[' "<label1>" ... "<label$n$>" ']'
```

- examples

a)

storing and clearing locations

```
storepos "Harry"
il R;move RU;phl "Harry" >
move "Harry";clearpos "Harry"
move D
storepos "Harry"
il R;move RU;phl "Harry" >
```

For more examples please see 2.3.4.3 example h), 2.3.6.3.

2.3 Objects

2.3.1  showGrid: show PSTricks grid

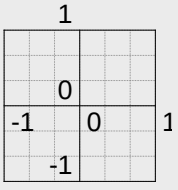
- purpose
  Shows the underlying coordinate grid.

- syntax

```
showGrid '(' float ',' float ')' '(' float ',' float ')'
```

- example

a sample grid




```
showGrid (-1, -1) (+1, +1)
```

2.3.2 Dot objects

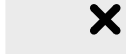
- purpose
  Draw different types of point objects at the actual position.
- syntax

```
cross | [b]?bullet | [b]?square | [b]?circle | dc
```

- examples

a)

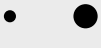
Fock interaction



```
cross
```

b)

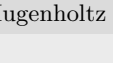
Hugenholtz interaction



```
bullet
move R
bbullet
```

c)

Hugenholtz cluster



```
square
move R
bsquare
```

d)

circle

```
circle
move R
bcircle
```

e)

disconnecting circle


```
dc
```

2.3.3 Interaction/cluster lines

- purpose
  Draw different types of interaction lines from actual position to given position.
- syntax

```
[il|id|pl|sl|cl] position
```

- examples

a)

standard interaction line


```
il R
```

b)

alternative interaction line


```
id R
```

c)

photon line

```
pl R
```

d)

spring line

```
sl R
```

e)

cluster line


```
cl R
```

2.3.4 Particle/hole lines

2.3.4.1 Modifiers (optional) common to all particle/hole lines

(i) `arrowdir` $\in \{$`<`, `>`, `<<`, `>>`, `<<<`, `>>>`$\}$, defaults to no arrow

Arrow directions are given with respect to current position. So `>` and `<` mean creator and annihilator, respectively.

(ii) `arrowshift` $\in$ *float*, given in

- % of line length for straight lines
- degrees for bent lines.

Shifts are given relative to default value which is the center of a straight particle hole line, angle 0 for half loops, and 25 degrees for quarter loops.

Arrow shift arguments must be preceded by a colon ":".

(iii) `linemultiplicity` $\in \{$`-`, `=`$\}$, defaults to single line

(iv) `labelSequence`
Consists of a sequence of labels. A (single) `label` is given by `'@'[BCE]?[lrudc]?[+-]* "label"`

`@` may be followed by two characters and additional shift characters

(a) `BCE`: position label at begin, center, end of particle/hole line, respectively. Defaults to center.

(b) `lrudc`: put label on left, right, up, down side or vertically centered, respectively. If empty defaults to continuation of particle/hole line.

(c) `+-`: shifts by 0.5pt each, may be given multiple times

- up/down if second character is `ud`
- left/right if second character is `lr` or first character is `C`
- along particle hole/line else

`+-` allows for fine tuning of positioning.

If the option `c` (center) was not given all labels are vertically aligned to the base line shifted by `labelYShift`. By default it is `labelYShift` $= -2.15\,\mathrm{pt}$

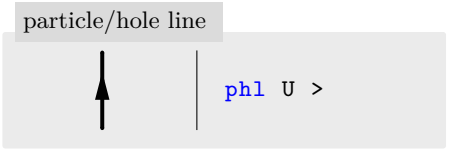
Modifiers must be given in the above order.

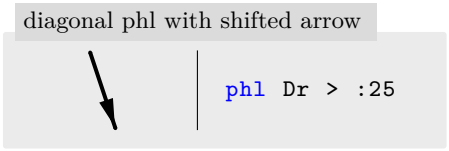
2.3.4.2 `phl`: straight particle/hole line

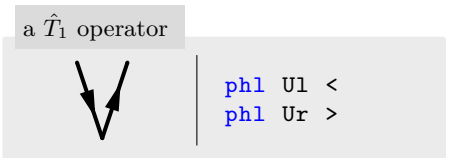
- purpose
  Draws particle/hole line from actual position to given position.
- syntax

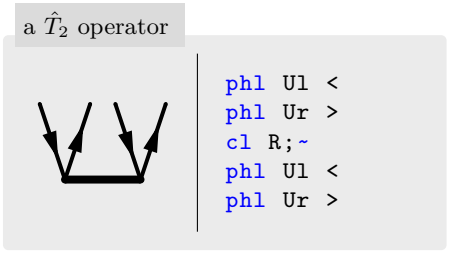
```
phl position arrowdir? (':' arrowshift)? linemultiplicity? labelSequence?
```

- examples

a)


b)


c)


d)


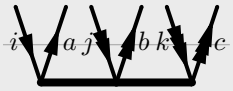
e) double arrows

```
phl D >>
```

f) double lines (experimental)

```
phl D > =
```

g) centering

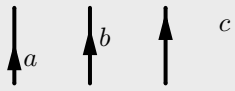



```
verbatim \psline[linecolor=black!30,
    ↪ linewidth=0.5pt](-0.5,0.5)(2.5,0.5)
phl Ul < @l "$i$";phl Ur > @r "$a$"
cl R;~
phl Ul << @l "$j$";phl Ur >> @r "$b$"
cl R;~
phl Ul <<< @l "$k$";phl Ur >>> @r "$c$"
```

h) with labels

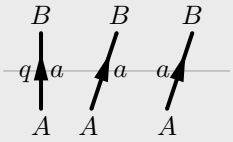

```
phl U > @r "$a$" @l "$b$"
move R
phl U > @r--- "$a$" @l+++ "$b$"
move R
phl U > @B "$A$" @E "$B$"
```

i) with shifted arrows/labels




```
phl U > :-20 @r "$a$"
move R
labelYShift 0
phl U > :0 @r "$b$"
move R
labelDist 20
phl U > :20 @r "$c$"
```

j) center and terminal placements




```
verbatim \psline[linecolor=
    ↪ black!30,linewidth=0.5pt
    ↪ ](-0.5,0.5)(2.5,0.5)
phl U > @ "$a$" @l "$q$" @B "
    ↪ $A$" @E "$B$"
move rr
phl Ur > @C "$a$" @B "$A$" @E "
    ↪ $B$"
move R
phl Ur > @Cl "$a$" @Bd "$A$" @
    ↪ Eu "$B$"
```

2.3.4.3 loop: loops

- purpose
  Draw loop, loop fragment, or oyster from actual position to given position.
- syntax
  - loop

    ```
    loop fragment? position loopWidth? arrowdir? (':' arrowshift)? linemultiplicity?
        ↪   branchSequence?
    ```

  - branchSequence

    ```
    '[' ("pos_i" labelSequence?)+ "pos_N" ']'
    ```

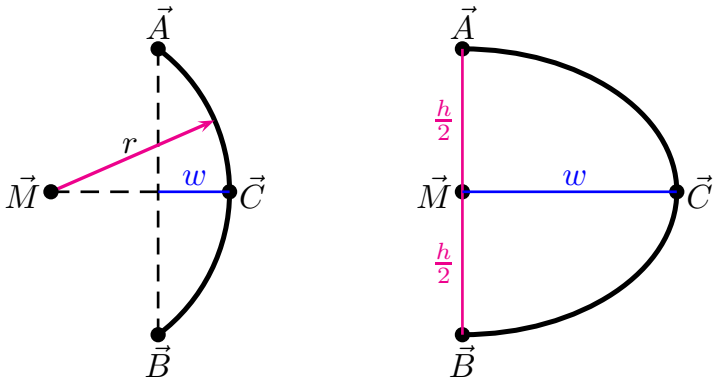
    "pos$_i$" defines a position similar to storepos that may be referenced later.

    Label sequences are allowed for non-full loops only. There is no label after the last position.

    The default (empty) label sequence is '[' "" "" ']' drawing a loop with a single arrow and no positions or labels defined.
- additional optional parameters
  1. fragment may be empty (full loop) ore one of
     - (): full loop, default
     - (: left half loop
     -): right half loop
     - (,: left quarter loop
     -),: right quarter loop
  2. loopWidth
     Loop widths are given as the distance perpendicular to the connecting line between the current position and the loop directional length. It is given in units of $\frac{1}{8}$ and $\frac{1}{3}$ of a full grid step for half (full) and quarters loops, respectively. Defaults to 1.0.
  3. branchSequence
     The branch sequence allows to name points on loops for further reference and place labels. The first and last label refer to the start and end points of a loop. If more than two labels are given the loop is divided into segments along the distance $\vec{A}$–$\vec{B}$. Any additional label increases the number of segments and arrows of a loop.

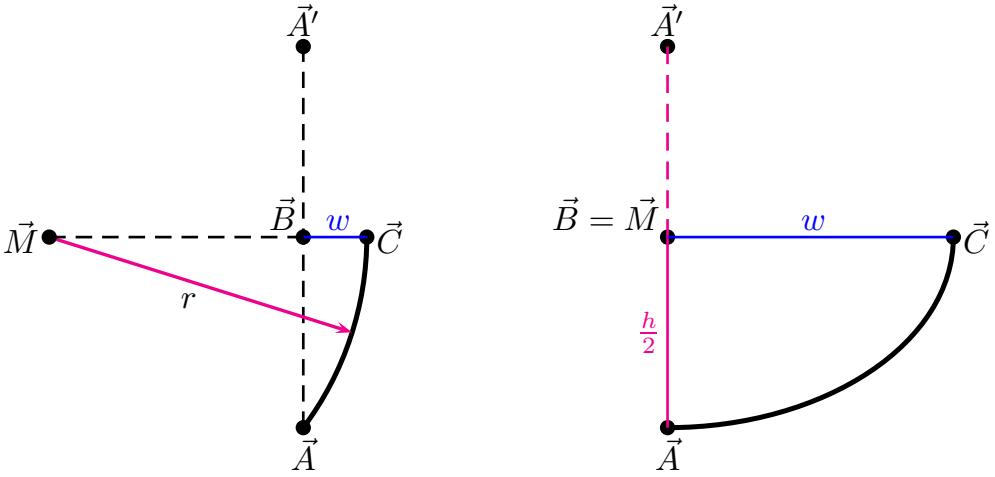
- loop geometry
  - half (and full) loops:
    - $w = \frac{1}{8}\|\vec{A} - \vec{B}\| \cdot$ `loopWidth`
    - $\vec{A}$: current position
    - $\vec{B} = \vec{A} +$ `direction`
    - $\vec{C}$: perpendicular on half line segment $\vec{A}$–$\vec{B}$ at distance w

`loop D`	`loop D 6`
circle arc, center at $\vec{M}$	elliptic arc, center at $\vec{M}$
$w \le \frac{1}{2}\|\vec{A} - \vec{B}\|$	$w > \frac{1}{2}\|\vec{A} - \vec{B}\|$



  - quarter loops:
    - $w = \frac{1}{3} \cdot$ `loopWidth`
    - $\vec{A}$: current position
    - $\vec{B} = \vec{A} +$ `direction`
    - $\vec{C}$: perpendicular on end of line segment $\vec{A}$–$\vec{B}$ at distance w

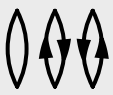
`loop ), UU`	`loop ), UU 9`
circle arc, center at $\vec{M}$	elliptic arc, center at $\vec{M}$
$w \le \|\vec{A} - \vec{M}\|$	$w > \|\vec{A} - \vec{M}\|$



- examples

a)

plain loops



```
loop D
move |r
loop D >
move |r
loop D <
```

b)

loop with double arrows

```
loop Dd >>
```

c) diagonal loop

```
loop DR >
```

d) left half loop

```
loop ( D
```

e) wide half loops with arrows

```
loop ) D 8 >
loop ) D 4 >
```

f) double lines

```
loop ) D > =
```

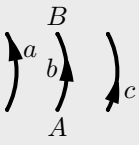
g) quarter loop upwards, arrow shifted

```
loop (, U 1
    ↪ < :-10
```

h) wide quarter loop downwards

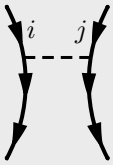
```
loop ), D 2
    ↪ >>
```

h) labeled loops



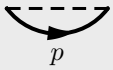

```
loop ) U > :15 @r "$a$"
move rr
loop ) U > :0 @l "$b$" @B "$A$" @E "$B$"
move rr
loop ) U > :-15 @r "$c$"
move rr
```

i) arcs with branches and labels



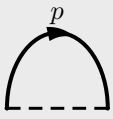

```
loop ) DD > [ "" @r "$i$" "L1" "" "" ]
move Rr
loop ( DD > [ "" @l "$j$" "R1" "" "" ]
move "L1";il "R1"
```

j) lower oyster




```
loop ) Rr 2 > @d "$p$"
il Rr
```

k) upper wide oyster




```
loop ( Rr 6 > @u+++ "$p$"
il Rr
```

l) lower oyster, double arrows

```
loop ) Rr 2 <<
il Rr
```

2.3.4.4 Special bent particle/hole lines

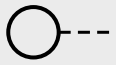
- purpose
  Draw particle/hole lines of special horizontal bent type.
- syntax

```
bubble width arrowdir? linemultiplicity? labelSequence?
snake [()] position floatNumber arrowdir? arrowshift? linemultiplicity?
```

  `width` is a floating point number.

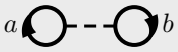
- examples

a) plain bubble



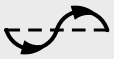
```
bubble -2
il rr
```

b) double bubble




```
bubble -1.5
    ↪ < @l "
    ↪ $a$"
il rr;~
bubble +1.5
    ↪ < @r "
    ↪ $b$"
```

c) standard snake

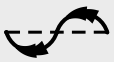


```
snake ) Rr 3 >
il Rr
```

d) ∞ snake

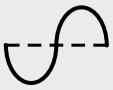
```
snake ) Rr 2
snake ( Rr 2
il Rr
```

e) snake with double arrows



```
snake ) Rr 3 <<
il Rr
```

f) wide snake



```
snake ) Rr 6
il Rr
```

2.3.5 Pointing arrows

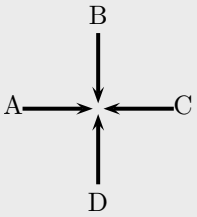
- purpose
  Point at some object in diagram.
- syntax

```
pointer position labelSequence
```

  `position` refers to the tail of the arrow. This may at first seem counter intuitive. It is, however, more convenient in every day usage.

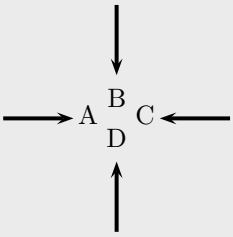
- example

a) end placements



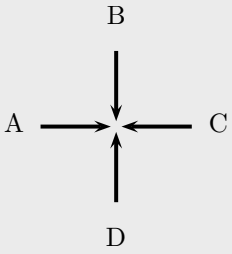

```
pointer L @E "A"
pointer U @E "B"
pointer R @E "C"
pointer D @E "D"
```

b) begin placements, increase point distance



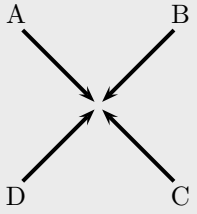

```
(move |l;pointer L @B "A")
(move |u;pointer U @B "B")
(move |r;pointer R @B "C")
(move |d;pointer D @B "D")
```

c) end placements, increased label distance



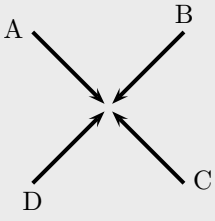

```
labelDist 10
pointer L @E "A"
pointer U @E "B"
pointer R @E "C"
pointer D @E "D"
```

d) end placements, diagonal arrows




```
pointer LU @E "A"
pointer RU @E "B"
pointer RD @E "C"
pointer LD @E "D"
```

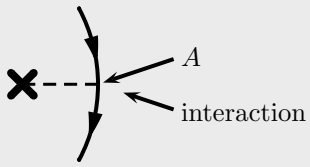
e) end placements, lrud orientation




```
pointer LU @El "A"
pointer RU @Eu "B"
pointer RD @Er "C"
pointer LD @Ed "D"
```

pointing to loop, varying label and point distances

f)




```
loop ) DD > [ "A" "B" "C" ]
move "B";(il L;~cross)
labelDist 3
pointer Ru @Ec "$A$"
(pointDist 10;pointer Rd @Ec "interaction")
```

2.3.6 Interface to PSTricks

2.3.6.1 rput: TEX/LATEX output

- purpose
  Draw TEX/LATEX output at aligned actual position.
  The command is terminated by a new line "\n", or closing brace ")", semicolon ";". If you need to typeset ")" or ";" escape it as "\)" or "\;".

- syntax

```
rput (<PSTricks::rput positioning string>)? '|' <string>
```

  Note the | separation between the rput arguments and the actual string. For the documentation of positioning string please refer to the PSTricks documentation.

- examples

labels on $\hat{T}_1$ operator, bottom aligned

a)


```
phl Ul <
phl Ur >
move |u
(move l..l;rput [b] | $i$) // note |..| positioning
(move r..r;rput [b] | $a$) // note r..r positioning
```

labels on a loop, top aligned

b)


```
loop U >
move |u
(move l;rput [t] | $i$)
(move r;rput [t] | $a$)
```

2.3.6.2 uput: TEX/LATEX output

- purpose
  Draw TEX/LATEXoutput at actual position with displacement.
  The command is terminated by a new line (\n), closing brace ()), semicolon ";". If you need to typeset ")" or ";" escape it as "\)" or "\;".

- syntax

```
uput (<PSTricks::uput positioning string>)? '|' <string>
```

  Note the | separation between the uput arguments and the actual string.

- examples

labels on $\hat{T}_1$ operator

a)


```
phl Ul <
phl Ur >
move |u
(move l.R;uput [l] | $i$) // note l.R positioning
(move r.L;uput [r] | $a$) // note r.L positioning
```

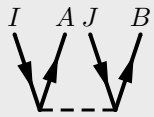
labels on a loop

b)

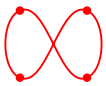

```
loop U >
move |u
uput {10pt}[l] | $i$
uput {10pt}[r] | $a$
```

labels on external lines

c)




```
(phl Ul <;~uput {3pt}[u] | $I$)
(phl Ur >;~uput {3pt}[u] | $A$)
il R;~
(phl Ul <;~uput {3pt}[u] | $J$)
(phl Ur >;~uput {3pt}[u] | $B$)
```

2.3.6.3  `verbatim`: using PSTricks directly

- purpose
  send commands to PSTricks directly. The command is terminated by a new line (`\n`) or "`mitabrev;`. All labels are exported to PSTricks.

- syntax

```
verbatim <PSTricks-command>
```

- examples

using `psccurve`

a) 

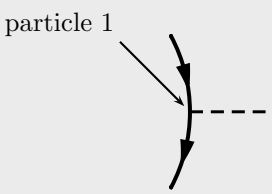
```
phl Ul <
phl Ur >
storepos "P1"
move Ur;storepos "P2"
move ll;storepos "P3"
verbatim \psccurve[linecolor=red,showpoints=
    ↪ true](P1)(P2)(P3)
```

using `nccurve`

b) 

```
phl Ul <
phl Ur >
move Ur;storepos "P2"
move ll;storepos "P3"
verbatim \nccurve[angleA=70,angleB=110,
    ↪ linecolor=blue,linestyle=dashed]{P2}{P3}
```

labeling with pointing arrows using PSTricks directly

c) 


```
loop ) DD > [ "P0" "P1" "P2" ]
move "P1";il R
move LU;storepos "Q"
verbatim \ncline[nodesep=3pt]{->}{Q
    ↪ }{P1}
move "Q";uput {0pt}[ul] | particle
    ↪ 1
```

2.4 State change

2.4.1 push and pop or (and) : save current state on stack

- purpose
  Push and pop current state onto stack. The state consists of

  (i) currentLocation

  (ii) labels

  (iii) colors: color, arrowFillcolor, pathcolor

  (iv) lengths and sizes: unitSize, lineWidth, pathWidth, arrowSize, arrowLength2Width, labelDist, pointDist, labelYShift

  (v) reverse

  (vi) bentArrows

  These instructions are especially useful for positioning of labels and macro programming.

- syntax

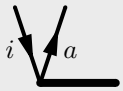
```
push | (
pop | )
```

- examples

a) using push and pop

```
push
color "red"
lineWidth 3
il R;~cross
pop
cl L
```

b) using (and)




```
phl Ul <
phl Ur >
(move ul..l; rput [b] | $i$)
(move ur..r; rput [b] | $a$)
cl R
```

2.4.2 color: set drawing color

- purpose
  Changes the color of the next objects.

- syntax

```
color "<name>"
```

- examples

a) using color and defining new ones

```
verbatim \newgray{mygray}{0.8}
color "blue";phl Ul <
color "red";phl Ur >
color "mygray";cl R
```

2.4.3 path: changing background of line elements

- purpose
  Change background color of line objects.
- syntax

```
path ("<color>" width | off)
```

  `width` is given in points. Setting `width` to 0 or setting `path off` will switch of paths.

- examples

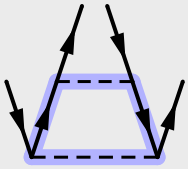
a)

highlighting graph elements by background

```
#include "colors.diaginc"
path "mygray" 6
il R
path "myred" 6
phl lU <;phl rU >
path "myblue" 6
move R;phl lU <;phl rU >
path "mygreen" 6
move R;loop U
```

b)

highlighting a loop through interaction and ph lines



```
#include "colors.diaginc"
phl lU <
path "myblue" 6
il Rrr;phl rU >;~il R
path off
phl rU >
move rrU;phl Dr >;~
path "myblue" 6
phl Dr >
path off
~phl rU >
```

with colors defined in

colors.diaginc

```
verbatim \newgray{mygray}{0.8}
verbatim \newrgbcolor{myred}{1 0.7 0.7}
verbatim \newrgbcolor{myblue}{0.7 0.7 1}
verbatim \newrgbcolor{mygreen}{0.5 0.9 0.5}
```

2.4.4 unitSize: set unit size (RLDU) to x cm

- purpose
  The unit size affects size of all graphical elements of a diagram except the font size. So rescaling it offers a convenient way to change the font size.
- default: 1 cm
- syntax

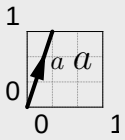
```
unitSize <float>
```

  Size is given in cm.

- examples

unitSize=1 (default)

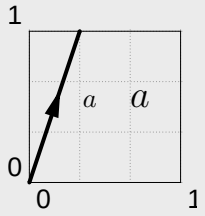
a)




```
showGrid (0, 0) (+1, +1)
phl Ur >
(move |ur..r;rput [b] | $a$)
(move |urr..r;rput [b] | \Large $a$)
```

unitSize=2

b)




```
unitSize 2
showGrid (0, 0) (+1, +1)
phl Ur >
(move |ur..r;rput [b] | $a$)
(move |urr..r;rput [b] | \Large $a$)
```

2.4.5 labelYShift: change vertical shift for labels (@ instruction)

- default: $-\frac{1}{2}$ height of a ($= -\frac{1}{2}$ 4.30554 pt)
- syntax

```
labelYShift <float>
```

  The shift is given in pt with respect to the baseline of a character. labelYShift has no effect on uput and rput instructions.

  You can obtain the height of a LaTeX character using

```
\newlength{\myHeight}
\settoheight{\myHeight}{$a$}
\typeout{\the\myHeight}
```

- examples

effect of labelYShift


```
verbatim \psline[linecolor=black!30,linewidth=0.5pt
    ↪ ](-0.333,0.5)(1.666,0.5)
phl U > @r "$a$"
move rr;labelYShift 0
phl U > @r "$a$"
move rr;labelYShift 4
phl U > @r "$a$"
```

2.4.6 labelDist: change distances for labels (@ instruction)

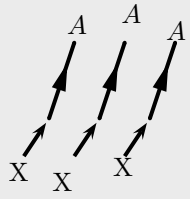
- default: 4 pt
- syntax

```
labelDist <float>
```

The distance is given in pt.

- examples

effect of labelDist




```
phl Ur > @E "$A$"
pointer l|d @E "X"
move rr
labelDist 8
phl Ur > @E "$A$"
pointer l|d @E "X"
move rr
labelDist 1
phl Ur > @E "$A$"
pointer l|d @E "X"
```

2.4.7 pointDist: change pointing arrow distance from node center

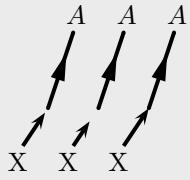
- default: 2 pt
- syntax

```
pointDist <float>
```

The distance is given in pt.

- examples

effect of pointDist




```
phl Ur > @E "$A$"
pointer l|d @E "X"
move rr
pointDist 6
phl Ur > @E "$A$"
pointer l|d @E "X"
move rr
pointDist 0
phl Ur > @E "$A$"
pointer l|d @E "X"
```

2.4.8 lineWidth: set linewidth to x pt

The line width applies to particle/hole lines only. Given in pt. Defaults to 1.5 pt.

2.4.9 bentArrows: control bending of arrows on arc

- purpose
  Switches on or off bending of arrows on arc. Does not affect arrows on straight lines.
  Default: `on`
- syntax

```
bentArrows (on|off)
```

- examples

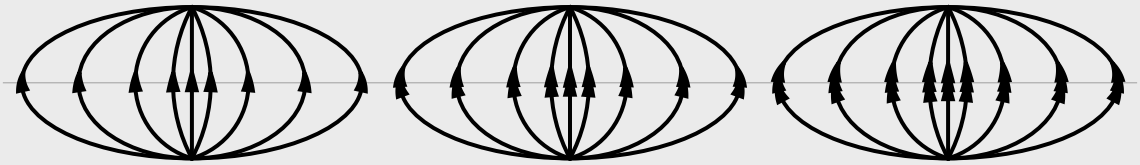
a) bent vs. straight arrows, narrow loops

```
loop D >
bentArrows off
move rr
loop D >
```

b) bent vs. straight arrows: wide loops

```
loop ) D 8 >
bentArrows off
move rr
loop ) D 8 >
```

c)



```
verbatim \psline[linecolor=black!30,linewidth=0.5pt](-2.5,1)(12.5,1)
#define CB (
#define OB )
#define L(d, x, y) loop d x 1 y;loop d x 3 y;loop d x 6 y;loop d x 9 y
#define X(x, y) phl x y;L(OB, x, y);L(CB, x, y)
move =(0, +0.0);X(UU, >)
move =(5, +0.0);X(UU, >>)
move =(10, +0.0);X(UU, >>>)
```

2.4.10 `reverse`: reverse all arrow directions

- purpose
  Swaps arrow directions given by `'<'` and `'>'`
- syntax

```
reverse
```

- examples

a) reversing arrows

```
phl lU <;phl rU >
move R
reverse
phl lU <;phl rU >
```

2.4.11 `arrowFillColor`: set arrow fill color

- purpose
  Specifies the fill color of arrows. Setting it to `white` will draw a hollow arrow.
- syntax

```
arrowFillColor "<color>"
```

- examples

white inset

a)

```
arrowFillColor "white"
phl lU <
phl rU >
```

2.4.12 `arrowWidth`: set arrow width

- purpose
  Specifies the size of the particle/hole arrows. Defaults to 5 pt.
- syntax

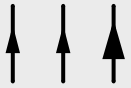
```
arrowWidth <float>
```

  Size is given in points.

- examples

wider arrows

a)



```
phl U >
move rr;arrowWidth 5;phl U >
move rr;arrowWidth 8;phl U >
```

2.4.13 `arrowLength2Width`: set relation arrow length/width

- purpose
  Specifies the relation between length and width of the particle/hole arrows.
- default: 2
- syntax

```
arrowLength2Width <float>
```

- examples

different ratios

a)

```
arrowWidth 6
phl U >
move rr;arrowLength2Width 4;phl U >
move rr;arrowLength2Width 1;phl U >
```

2.5 Miscellaneous

2.5.1 Injecting preamble commands

- purpose
  Inject commands into LaTeX code before `\begin{document}`.
- syntax

```
preamble ...
```

  `preamble` may be given multiple times. The arguments are passed on to LaTeX line by line.
- example

including an additional package

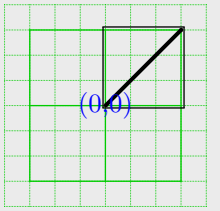
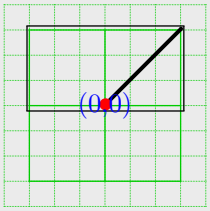
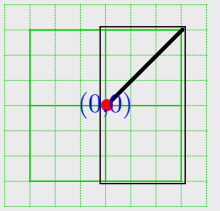
```
preamble \usepackage{phystex}
phl U > @B "$a$" @E "$b$" @ "$\wick{1-2}{}{\hat a^\dagger, \hat b}$"
```

2.5.2 Tuning bounding boxes for alignment

By default `Diag2PS` generates a bounding box enclosing the generated graph. However, with respect to alignment this "natural" bounding box may not be optimal (see also 3.2). To tune the geometry of the bounding box `Diag2PS` offers the keywords `BBXCenter` and `BBYCenter`. Giving one or both of these keywords gets the current position as the center of the generated bounding box with respect to x or y.

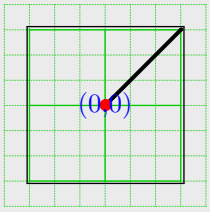
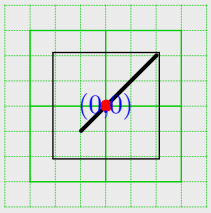
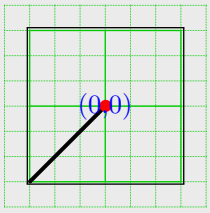
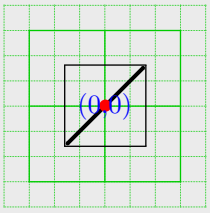
This is illustrated by the following examples:

bounding box

natural	x-centered on origin	y-centered on origin
		

`phl UR`	`phl UR`
	`BBXCenter`
	`color "red"`
	`bullet`
		`phl UR`
		`BBYCenter`
		`color "red"`
		`bullet`

bounding box x, y-centered on

diagram origin	diagram origin `ur`	diagram origin `UR`	center of phl
			

`phl UR`
`BBYCenter`
`BBXCenter`
`color "red"`
`bullet`
	`phl UR`
	`move ur`
	`BBYCenter`
	`BBXCenter`
	`color "red"`
	`bullet`
		`phl UR`
		`move UR`
		`BBYCenter`
		`BBXCenter`
		`color "red"`
		`bullet`
			`phl UR`
			`move |u|r`
			`BBYCenter`
			`BBXCenter`
			`color "red"`
			`bullet`

Chapter 3

Examples

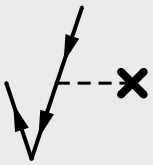
3.1 Gallery

3.1.1 Selected coupled cluster single projection and reversed arrows

a)

```
phl Ul <
phl Ur >
~phl Ur >
(il R;~cross)
```

b)



```
reverse
phl Ul <
phl Ur >
~phl Ur >
(il R;~cross)
```

3.1.2 Using macros

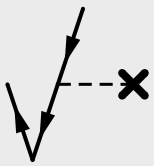
- first define the macro in file "A1FNT1.diaginc":

A1FNT1.diaginc

```
#define A1_FN_T1(dir) \
push;phl Dr dir;move Dr;phl Ur dir;move Ur
    ↪ ;phl Ur dir;il R;move R;cross;pop
```

- then use it as:

```
#include "A1FNT1.diaginc"
A1_FN_T1(>)
```



```
#include "A1FNT1.diaginc"
A1_FN_T1(<)
```

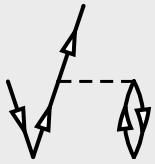
3.1.3 Moving arrows on crossing lines

```
loop ( D >          move R
il R                loop ) D <
(move D;il R)       phl LD > :-20
phl RD > :-20
```

3.1.4 Crossed loops

```
loop DR      ~loop DL
il R         ~cl R
```

3.1.5 Hollow arrows (for e.g. spin averaged diagrams)

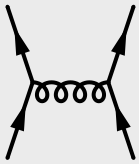


```
arrowFillColor "white"      ~phl Ur >
arrowWidth 6                il R
phl Ul <                    ~loop D >
phl Ur >
```

3.1.6 $\hat{C}_2$ operator (disconnected)

```
cl R                        move R
phl Ul <;phl Ur >;dc        phl Ul <;phl Ur >;dc
```

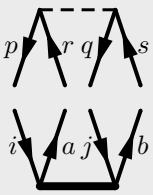
3.1.7 Gluon interaction lines



```
phl Ul >
phl Dl <
sl R;~
phl Ur >
phl Dr <
```

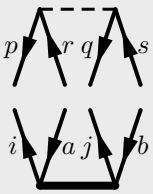
3.1.8 Placing labels and arrows

a)




```
phl Dl > @l "$p$";phl Dr < @r "$r$"
il R;~
phl Dl > @l "$q$";phl Dr < @r "$s$"
move DDdL
phl Ul < @l "$i$";phl Ur > @r "$a$"
cl R;~
phl Ul < @l "$j$";phl Ur > @r "$b$"
```

b)



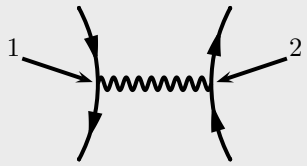

```
// same as before but using macros:
#include "VNT2macro.diaginc"
(V(D,p,r);il R;~;V(D,q,s))
move DDd
(V(U,i,a);cl R;~;V(U,j,b))
```

with

VNT2macro.diaginc

```
#define X(x) #x              // use stringification
#define XX(x) X($##x##$)     // embed the "$"
#define V(_1,_2,_3) phl _1##l > @l XX(_2);phl _1##r < @r XX(_3)
```

c)




```
loop ) DD > [ "" "P" "" ]
move RR
loop ( DD < [ "" "Q" ""]
move "P";pl "Q"
move "P";pointer Lu @E "1"
move "Q";pointer Ru @E "2"
```

3.1.9 Hugenholtz diagrams

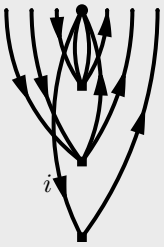
a)

```
bullet
loop D 1 >
loop D 2.5 >
~square
```

b)

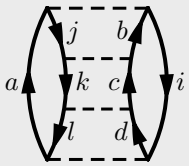
```
bullet
loop D >
~bullet
loop D  >
~bullet
loop UU 1.5 >
```

c)




```
bullet                          move D;square
loop ) DD <                     loop (, UU 2 <
loop ( DDD > :15 @l "           loop ), UU 2 >
    ↪ $i$"                      loop (, UU 3 <
loop D 0.75;~square             move D;square
loop ), U > :10                 loop ), UUU 3 >
loop (, U < :-10
```

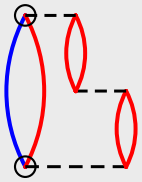
3.1.10 4th order PT




```
loop ( DD < @l "$a$"
loop ) DD > ["R0" @r "$j$" "R1" @r "$k$" "R2" @r "$l$" "R3"]
move Rr
loop ( DD < ["L0" @l "$b$" "L1" @l "$c$" "L2" @l "$d$" "L3"]
loop ) DD > @r "$i$"
#define IL(A, B) move A;il B
IL("R0", "L0");IL("R1", "L1");IL("R2", "L2");IL("R3", "L3")
```

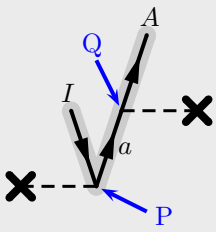
3.1.11 colored diagrams and using pstricks directly

a)



```
storepos "pos1"
color "blue";loop ( UU
color "red";loop ) UU
color "black";il Rr
color "red";move Rr;loop U
color "black";move U;il ll
color "red";move ll;loop U
color "black";move U;il ll
move ll;storepos "pos2"
verbatim \pscircle(pos1){0.15}
verbatim \pscircle(pos2){0.15}
```

b)




```
(il L;~cross)
(color "blue";pointer drr @Ec "P")
path "black!20" 7
phl Ul < @E "$I$";phl Ur > @ "$a$"
~phl Ur > @E "$A$"
path off
(il R;~cross)
(color "blue";pointer uul @E "Q")
```

3.2 Fixing alignments between LaTeX and Diag2PS objects

Alignment with LaTeX equations requires special attention.

3.2.1 Bottom aligned (default)

Consider the following source files:

- Diag2PS:

A0FNT1.diag

```
loop D
il R
~cross
```

A0VNT1T1.diag

```
loop D
il R
~loop D
```

A0VNT2.diag

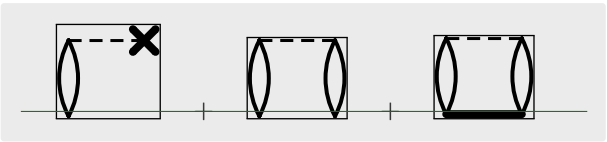
```
loop D
il R
~loop D
~cl L
```

- included as pdf by LaTeX:

YourFile.tex

```
\def\IG#1{\includegraphics{#1.pdf}}
$$ \IG{A0FNT1}  \quad + \quad  \IG{A0VNT1T1}  \quad + \quad  \IG{A0VNT2} $$
```

- results in



  with the enclosing boxes showing the bounding boxes of the PDF files and the green line showing the LaTeX equation center.

- **problem**: diagrams are (approximately) aligned among themselves but not aligned to the LaTeX equation center.

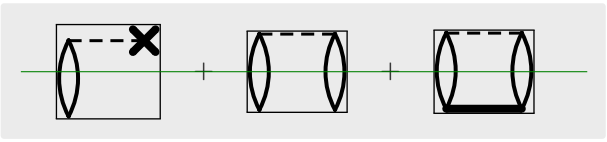
3.2.2 Center aligned

- Diag2PS source as before
- change LaTeX source to

YourFile.tex

```
\def\IG#1{\vcenter{\hbox{\includegraphics{#1.pdf}}}}
$$ \IG{A0FNT1}  \quad + \quad  \IG{A0VNT1T1}  \quad + \quad  \IG{A0VNT2} $$
```

  item results in



- **problem**: vertical centering of diagrams aligns the centers of the bounding boxes on the LaTeX equation center. Due to the asymmetry of the diagrams this breaks alignment with respect to the diagram centers.

3.2.3 Fixes

3.2.3.1 Typeset whole equations within Diag2PS

One way to approach the issue is to typeset everything within Diag2PS:

```
loop D                  loop D                  loop D
il R                    il R                    il R
~cross                  ~loop D                 ~loop D
//——————————————        //——————————————        ~cl L
move rr                 move rr
(move |d;rput | $+$)    (move |d;rput | $+$)
move rr                 move rr
//——————————————        //——————————————
```

This however becomes very inconvenient for larger objects particularly when intermixing diagrammatic and non diagrammatic content (e.g. algebraic formulas) as it inhibits the type setting power of LaTeX.

3.2.3.2 Center aligned with centered bounding box using BBYCenter

- Change Diag2PS source to:

A0FNT1.diag

```
(move |d;BBYCenter)
loop D
il R
~cross
```

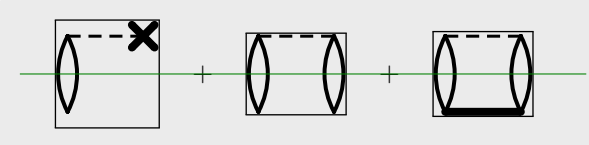
A0VNT1T1.diag

```
(move |d;BBYCenter)
loop D
il R
~loop D
```

A0VNT2.diag

```
(move |d;BBYCenter)
loop D
il R
~loop D
~cl L
```

- YourFile.tex source as before
- results in



- ⇒ **precise vertical alignment**

Appendix A

Language specification

A.1 Lexer tokens

```
blank       [ \t]
alpha       [_A-Za-z]
dig         [0-9]
direction   [lLrRuUdD.|]+
name        \"[^"]*\"
float_num   [-+]?{dig}+\.?{dig}*([eE][-+]?{dig}+)?
operators   [<>=(),+-[\]:]
label       @[BEC]?[udlrc]?[-+]*

%%
{blank}+            {...}
;+                  {...}
\n                  {...}
~                   {...}

\/\/.*              {...}




yes                 {...}
no                  {...}

on                  {...}
off                 {...}

move                {...}
storepos            {...}
clearpos            {...}
push                {...}
pop                 {...}

cross               {...}
bullet              {...}
bbullet             {...}
circle              {...}
bcircle             {...}
dc                  {...}
square              {...}
bsquare             {...}

cl                  {...}
clo                 {...}
il                  {...}
id                  {...}
pl                  {...}
sl                  {...}
```

```
48
49  phl                 {...}
50  loop                {...}
51
52  reverse             {...}
53  snake               {...}
54  bubble              {...}
55
56  pointer             {...}
57
58  preamble            {...}
59
60  verbatim            {...}
61
62  rput                {...}
63
64  uput                {...}
65
66  {label}             {...}
67  showGrid            {...}
68
69  color               {...}
70  unitSize            {...}
71  lineWidth           {...}
72  path                {...}
73  arrowWidth          {...}
74  arrowLength2Width   {...}
75  arrowFillColor      {...}
76  bentArrows          {...}
77  BBXCenter           {...}
78  BBYCenter           {...}
79  labelYShift         {...}
80  labelDist           {...}
81  pointDist           {...}
82
83  {direction}         {...}
84
85  {float_num}         {...}
86
87  {name}              {...}
88
89  {operators}         {...}
90
91  .                   { throw; }
92
93  %%
```

A.2 Yacc (bison) grammar

```
1   input:
2       input _line |
3
4       ;
5
6   _line:
7       line |
8       '(' |
9       MOVELASTDIRECTION |
10      SEPARATOR
11      ;
12
13  line:
14      command SEPARATOR |
15      command ')' SEPARATOR
16      ;
17
```

```
18 command:
19     KEYWORD_cross |
20     KEYWORD_bullet |
21     KEYWORD_bbullet |
22     KEYWORD_circle |
23     KEYWORD_bcircle |
24     KEYWORD_dc |
25     KEYWORD_square |
26     KEYWORD_bsquare |
27     KEYWORD_cl vector |
28     KEYWORD_clo vector |
29     KEYWORD_il vector |
30     KEYWORD_id vector |
31     KEYWORD_pl vector |
32     KEYWORD_sl vector |
33     KEYWORD_showGrid '(' floatNumber ',' floatNumber ')' '(' floatNumber ',' floatNumber ')' |
34     KEYWORD_reverse |
35     KEYWORD_phl vector arrowdir arrowshift linemultiplicity labelSequence |
36     KEYWORD_loop loopfragment vector loopWidth arrowdir arrowshift linemultiplicity branchSequenceBl
37     KEYWORD_bubble floatNumber arrowdir linemultiplicity labelSequence |
38     KEYWORD_snake loopRLfragment vector floatNumber arrowdir arrowshift linemultiplicity |
39     RPUT |
40     UPUT |
41     VERBATIM |
42     KEYWORD_pointer vector labelSequence |
43     KEYWORD_move vector |
44     KEYWORD_storepos name |
45     KEYWORD_clearpos vstring |
46     KEYWORD_push |
47     KEYWORD_pop |
48     PREAMBLE |
49     KEYWORD_path name floatNumber |
50     KEYWORD_path KEYWORD_no |
51     KEYWORD_color name |
52     KEYWORD_unitSize floatNumber |
53     KEYWORD_lineWidth floatNumber |
54     KEYWORD_arrowWidth floatNumber |
55     KEYWORD_arrowLength2Width floatNumber |
56     KEYWORD_arrowFillColor name |
57     KEYWORD_bentArrows yesno |
58     KEYWORD_BBXCenter |
59     KEYWORD_BBYCenter |
60     KEYWORD_labelDist floatNumber |
61     KEYWORD_pointDist floatNumber |
62     KEYWORD_labelYShift floatNumber loopWidth:  floatNumber |
63
64     ;
65
66 branchSequenceBlock:
67     '[' branchSequence ']' |
68     labelSequence
69     ;
70
71 branchSequence:
72     branchSequence NAME labelSequence |
73
74     ;
75
76 label:
77     KEYWORD_label name
78     ;
79
80 labelSequence:
81     labelSequence label |
82
83     ;
```

```
84
85  yesno:
86      KEYWORD_yes |
87      KEYWORD_no
88      ;
89
90  vstring:
91      vstring name |
92
93      ;
94
95  linemultiplicity:
96      '-' |
97      '=' |
98
99      ;
100
101 arrowdir:
102     '>' |
103     '>' '>' |
104     '>' '>' '>' |
105     '<' |
106     '<' '<' |
107     '<' '<' '<' |
108
109     ;
110
111 arrowshift:
112     ':' floatNumber |
113
114     ;
115
116 loopfragment:
117     '(' ')' |
118     '(' |
119     ')' |
120     '(' ',' |
121     ')' ',' |
122
123     ;
124
125 loopRLfragment:
126     '(' |
127     ')'
128     ;
129
130 direction:
131     DIRECTION
132     ;
133
134 vector:
135     direction |
136     NAME |
137     '=' '(' floatNumber ',' floatNumber ')' |
138     '+' '=' '(' floatNumber ',' floatNumber ')'
139     ;
140
141 floatNumber:
142     FLOAT_NUM
143     ;
144
145 name:
146     NAME
147     ;
```