# The euclidean-lattice package

Jérôme Plût ⟨jerome.plut@cyber.gouv.fr⟩

euclidean-lattice Euclidean lattices, dated 2024-07-26

This package provides a simple and efficient way of drawing TikZ pictures of two-dimensional Euclidean lattices.

## 1 Usage

### 1.1 The \lattice command

\lattice `\lattice <overlay specification> [options...] (a,b)(c,d);` This commands draws a part of the two-dimensional lattice generated by the vectors $(a, b)$ and $(c, d)$.

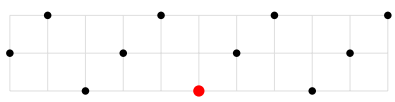
The *overlay specification* is optional and follows beamer syntax. It will only work if the beamer class is loaded.

The *options* enable customizing how the lattice is displayed. They follow the standard TikZ/PGF key-value interface. The following options are available.

x y `x=`$x_1$`:`$x_2$ and `y=`$y_1$`:`$y_2$ (default value `x=-2:2,y=-2:2`).

These two options specify a bounding box which determines which part of the lattice is drawn. As a shortcut, passing the value `x=`$x_1$ is equivalent to `x=`$-x_1$`:`$x_1$, and likewise for `y`.

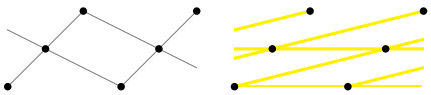
```
\draw[help lines,gray!25] (-5,0) grid (5,2);
\lattice[x=5,y=0:2](3,0)(1,1);
\node[inner sep=1.5pt,circle,fill=red] at (0,0){};
```



grid `grid=`*options...*

This option specifies how to draw the grid generated by the two given lattice vectors. It may take either a single value (generally a color name), or several values grouped in braces. Note that this grid depends on the basis vectors and not only on the lattice; see the example below.

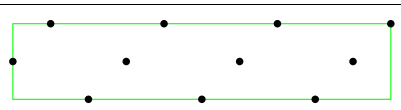
```
\lattice[y=0:2,x=0:5,grid=gray](2,-1)(1,1);
% Same lattice, different basis:
\lattice[y=0:2,x=6:11,grid={yellow,very thick}]
  (3,0)(4,1);
```



bounding box `bounding box=`*options...*

This option specifies how to draw the bounding box given by the `x` and `y` options. It may take either a single value (generally a color name), or several values grouped in braces.

```
\lattice[bounding box=green,x=-5:5,y=0:2](3,0)(1,1);
```



each point `each point/.code n args={5}{` *code...*`}`

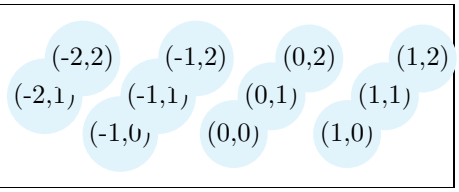
This option contains contains the code to execute for each lattice point. This code is called with the five following parameters:

- (`#1`) the parsed lattice node options,
- (`#2`,`#3`) the $(x, y)$ coordinates of the current lattice point (as TikZ canvas coordinates);

– (#4,#5) the coordinates of the same point relative to the provided lattice basis (as integers).

Note that, due to this parameter being expanded inside a macro, all the # signs must be doubled in its definition, as in the following example.

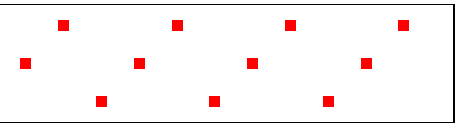
```
\lattice[x=-5:5,y=0:2,each point/.code n args=
  {5}{\node[##1,fill=cyan!10] at (##2,##3){(##4,##5)};}]
  (3,0)(1,1);
```




**Other options**

Any remaining options are directly passed to the \node calls for each lattice point.

```
\lattice[red,rectangle,inner sep=2pt,x=-5:5,y=0:2]
  (3,0)(1,1);
```

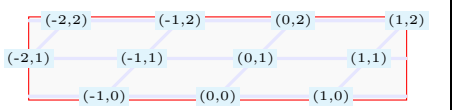


## 1.2 Configuring default behaviour

x
y
grid
bounding box
each point

The /lattice/x, /lattice/y, /lattice/grid, /lattice/bounding box, /lattice/each point PGF keys contain the default parameters for the correspondingly-named options. Note that the # signs must *not* be doubled in /lattice/each point.

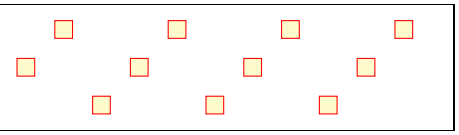
```
\pgfqkeys{/lattice}{x=-5:5,y=-.1:2.1,
  grid={blue!10,very thick},
  bounding box={red,fill=gray!5},
  each point/.code n args={5}{%
  \node[#1,fill=cyan!10,rectangle] at (#2,#3)
    {\tiny (#4,#5)};}}
\lattice[grid,bounding box](3,0)(1,1);
```




node

The /lattice/node key contains the default parameters for the \node call for each point. Equivalently, these are the default options passed to each \lattice call.

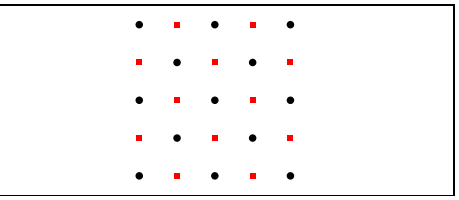
```
\pgfkeys{/lattice/node={fill=yellow!25,draw=red}}
\lattice[x=-5:5,y=0:2](3,0)(1,1);
```



## 1.3 Exactness and limitations

This command only outputs lattices containing the origin. An offset lattice may still be obtained by using the PGF transformation mechanism and correspondingly adjusting the x,y bounding box parameters:

```
\begin{scope}\pgftransformshift{\pgfpoint{1cm}{0}}
\lattice[x=-3:1,red,rectangle](2,0)(1,1);
\end{scope}
\lattice(2,0)(1,1);
```



This command might be executed a large number of times in a single document, for example in the case of a beamer presentation with a large number of overlays. This means that we need to be careful to make it efficient.

Therefore, all computations are performed using TEX registers. Since these registers only implement 32-bit integer arithmetic, we use fixed-point arithmetic with 16-bit offset for numeric computations (which include matrix inversion). This implies some precision loss and enforces some numeric limits. The use of fixed-point arithmetic means that the products appearing in the determinant of the lattice must not exceed $2^{15}$. In particular, this package should work as long as all the coordinates (in absolute value) do not exceed 128, which should cover most typical scales of TikZ pictures.

We compensate for the precision loss by sampling a bit too many lattice points (relative to the required bounding box) and then filtering *a posteriori* to ensure that all points are in the bounding box. This last step should remain exact, at least as soon as the coordinates of the lattice vectors are integers (or integer multiples of $2^{-8}$).

Therefore, we expect that in most cases, the command will exactly output the lattice points inside the bounding box. If this were to fail, however, as a last-resort option, enlarging the bounding box by a small number in all directions should ensure that all lattice points appear.

# 2 Implementation

**Initialization.** We obviously depend on tikz (and more precisely on pgfkeys).

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{euclidean-lattice}[2024-07-26 Euclidean~lattices 1.0]
\RequirePackage{tikz}
```

We use fixed-point arithmetic in the 31-bit registers supported by TEX, using 16 bits for offset.

TODO: we could slightly enhance the precision of computations by correctly rounding the last bit (instead of truncating). Depending on the value's sign, `\advance XXX by .5\la@unit` should do the trick.

This means that we represent a real number $x$ using the integer $f(x) = Bx$, where $B = 2^{16}$. To multiply two such integers, we note that $f(xy) = Bxy = f(x)f(y)/B$, which we compute as $(f(x)/\beta) \cdot (f(y)/\beta)$, where $\beta = \sqrt{B} = 2^8$. We store this value in the register `\la@unit`.

```
\newdimen\la@unit \la@unit=256sp
\def\lattice@mul#1#2{% writes #1*#2 in #1
  \@tempdimc=#2 \divide\@tempdimc\la@unit
  \divide #1\la@unit
  \multiply #1\@tempdimc}%
```

We perform division using $f(x/y) = Bx/y = Bf(x)/f(y)$, which we compute as $\frac{\beta f(x)}{f(y)/\beta}$.

```
\def\lattice@div#1#2{%
  \@tempdimc=#2\divide\@tempdimc\la@unit
  \multiply #1\la@unit \divide#1\@tempdimc}%
\def\lattice@invert#1{% writes 1/#1 in #1
  \@tempdimc=#1 \divide\@tempdimc\la@unit
  #1=256\p@ \divide #1\@tempdimc}%
```

The next macro is our main enumeration routine. It enumerates all the point in the lattice $\Lambda$ generated by the vectors ($a =$ `#1`, $b =$ `#2`) and ($c =$ `#3`, $d =$ `#4`) inside the box $R$ delimited by the registers [`\la@Ax`, `\la@Bx`] × [`\la@Ay`, `\la@By`]. On each such point, it invokes the macro `\lattice@donode`.

Since $\Lambda = B \cdot \mathbb{Z}^2$, instead of enumerating $S = \Lambda \cap R$ we enumerate $B^{-1}S = \mathbb{Z}^2 \cap (B^{-1}R)$. We use $(x, y)$ coordinates to describe elements of $S$ and $(w, z)$ coordinates to describes elements of $B^{-1}S$.

We declare several numeric registers, all prefixed with `\la@`:

- $(a, b)$ and $(c, d)$ are the vectors of the lattice basis $B$;
- $R := [Ax, Bx] \times [Ay, By]$ is the bounding box for the drawing;

- $U := (Ux, Uy)$ and $V := (Vx, Vy)$ are the side vectors of the parallelogram $P := B^{-1} \cdot R$, and $C := (Cx, Cy)$ is its center;
- $sM < sL$ are the slopes of the vectors $U$ and $V$;
- $wA, wB, wM$ are the horizontal parameters for enumerating the parallelogram;
- $zA, zB, zC$ are the vertical parameters for enumerating the parallelogram;
- $D$ plays a double role, first as the (inverse) determinant of the lattice, and then as a vertical enumeration parameter.

```
\newdimen\la@a \newdimen\la@b \newdimen\la@c \newdimen\la@d
\newdimen\la@Ax\newdimen\la@Bx\newdimen\la@Ay\newdimen\la@By
\newdimen\la@D
\newdimen\la@Ux\newdimen\la@Uy \newdimen\la@Vx\newdimen\la@Vy
\newdimen\la@Cx\newdimen\la@Cy

\newdimen\la@sL \newdimen\la@sM
\newdimen\la@zA \newdimen\la@zB \newdimen\la@zC
\newdimen\la@wA \newdimen\la@wB
\newcount\la@wM

\def\lattice@enumerate#1#2#3#4{%
  \la@a=#1\p@ \la@b=#2\p@ \la@c=#3\p@ \la@d=#4\p@
```

First we store the inverse of the determinant in `\la@D`.

```
  \la@D=\la@a \lattice@mul\la@D\la@d
  \@tempdima=\la@b \lattice@mul\@tempdima\la@c
  \advance\la@D -\@tempdima
  \lattice@invert\la@D
```

Compute the inverse image $B^{-1}R = P$. We first compute the sides $U$, $V$ of the parallelogram, multiplying by $-1$ if needed so that both $w$-coordinates are $\geqslant 0$.

```
  \la@Ux=\la@Ax \advance\la@Ux-\la@Bx
  \lattice@mul\la@Ux \la@D
  \la@Uy=-\la@Ux
  \lattice@mul\la@Ux \la@d \ifdim\la@Ux <\z@
    \multiply\la@Ux -1 \multiply\la@Uy -1 \fi
  \lattice@mul\la@Uy\la@b
  \la@Vx=\la@By \advance\la@Vx-\la@Ay \lattice@mul\la@Vx\la@D
  \la@Vy=-\la@Vx
  \lattice@mul\la@Vx\la@c \ifdim\la@Vx<\z@
    \multiply\la@Vx -1 \multiply\la@Vy -1\fi
  \lattice@mul\la@Vy \la@a
```

We ensure that $\det(U, V) > 0$, swapping both vectors if needed. Since we only need the sign of this determinant, we scale down the values to help prevent a numeric overflow.

```
  \@tempdima \la@Ux
  \@tempdimb \la@Vx
  \ifdim\@tempdima<100\p@\else
    \ifdim\@tempdimb<100\p@\else
    \divide\@tempdima 1024
    \divide\@tempdimb 1024\fi\fi
  \lattice@mul\@tempdima\la@Vy
  \lattice@mul\@tempdimb\la@Uy
  \ifdim \@tempdimb>\@tempdima
    \@tempdimc=\la@Ux \la@Ux=\la@Vx \la@Vx=\@tempdimc
    \@tempdimc=\la@Uy \la@Uy=\la@Vy \la@Vy=\@tempdimc
  \fi
```

Compute the center of the parallelogram. Note that we also use `\la@sL`, `\la@sM` (not yet needed) as temporaries here.

```
\la@sL=\la@Ax \advance\la@sL\la@Bx \divide\la@sL 2
\la@sM=\la@Ay \advance\la@sM\la@By \divide\la@sM 2
\la@Cx=\la@sL \lattice@mul\la@Cx\la@d
\@tempdima=\la@sM \lattice@mul\@tempdima\la@c
\advance\la@Cx -\@tempdima
\la@Cy=\la@sM \lattice@mul\la@Cy\la@a
\@tempdima=\la@sL \lattice@mul\@tempdima\la@b
\advance\la@Cy -\@tempdima
\lattice@mul \la@Cx\la@D
\lattice@mul \la@Cy\la@D
```

Compute the bounding-box in $(w, z)$ space. Note that the `\la@wA`,`\la@wB` values will be re-used below for enumerating all lattice points; on the other hand, `\la@zA` and `\la@zB` are used here as temporary values.

```
\@tempdima\la@Ux \advance\@tempdima\la@Vx
\la@wA=\la@Cx \advance\la@wA -.5\@tempdima
\la@wB=\la@wA \advance\la@wB \@tempdima
\@tempdima\la@Uy \ifdim\@tempdima<\z@ \multiply\@tempdima -1\fi
\ifdim\la@Vy<\z@ \advance\@tempdima -\la@Vy
\else\advance\@tempdima\la@Vy\fi
\la@zA=\la@Cy \advance\la@zA -.5\@tempdima
\la@zB=\la@zA \advance\la@zB \@tempdima
```

Draw the grid if the options require it.

```
\ifx\lattice@grid\@empty\else \begin{scope}%
  \clip (\strip@pt\la@Ax,\strip@pt\la@Ay)
    rectangle (\strip@pt\la@Bx,\strip@pt\la@By);
  \pgftransformcm{\strip@pt\la@a}{\strip@pt\la@b}%
    {\strip@pt\la@c}{\strip@pt\la@d}{\pgfpointorigin}%
  \expandafter\draw\expandafter[\lattice@grid]
    (\strip@pt\la@wA,\strip@pt\la@zA) grid (\strip@pt\la@wB,\strip@pt\la@zB);
\end{scope}\fi
```

Compute the last values required to enumerate over $\mathbb{Z}^2 \cap P$. We represent the parallelogram $P$ as the intersection of four half-planes. Let $A_1 = (w_1, z_1)$ and $A_2 = (w_2, z_2)$ be the left-most and right-most vertices of $P$, and $\lambda, \mu$ be the slopes of the sides (the condition $\det(U, V) > 0$ guarantees that $\lambda < \mu$).

The two sides originating from $A_i (i = 1, 2)$ have equations

$$z = z_i + \lambda(w - w_i), \qquad z = z_i + \mu(w - w_i);$$

By defining $z_3 = z_2 - \lambda(w_2 - w_1)$ and $z_4 = z_2 - \mu(w_2 - w_1)$, we may rewrite both equations from $A_2$ as:

$$z = z_3 + \mu(w - w_1), \qquad z = z_4 + \lambda(w - w_1)$$

The parallelogram $P$ is then defined by the following inequations, where we write $t = w - w_1$:

$$\max(z_3 + \mu t, z_1 + \lambda t) \leqslant z \leqslant \min(z_1 + \mu t, z_4 + \lambda t).$$

For any given value of $w$ we store the value of these four affine functions in the variables `\la@zB`, `\la@zA`, `\la@D` (not a typo), `\la@zC`, in this order. When increasing the value of $w$ we only need to increase the values of these four registers respectively by $\mu$, $\lambda$, $\mu$, $\lambda$.

First we compute the values $z_2 = z_C + \frac{1}{2}(z_U + z_V)$ and $w_2 - w_1 = w_U + w_V$.

```
\@tempdima=\la@Uy \advance\@tempdima\la@Vy
\@tempdimb=\la@Cy \advance\@tempdimb .5\@tempdima % z2
\@tempdima=\la@Ux \advance\@tempdima\la@Vx % w_2-w_1 = w(U+V)
```

Then we compute the slopes $\lambda =$ `\la@sL` and $\mu =$ `\la@sM`. A Special case happens when either $w_U$ or $w_V$ is zero. In the first case, $\lambda = -\infty$. This means that the comparisons with `\la@zA` and `\la@zC` must be ignored.

```
\ifdim\la@Ux=\z@
  \la@sL=\z@ \la@zC=\maxdimen \la@zA=-\maxdimen
\else
  \la@sL=\la@Uy \lattice@div\la@sL\la@Ux
  \la@zC=-\@tempdima \lattice@mul\la@zC\la@sL \advance\la@zC \@tempdimb
  \la@zA=\@tempdimb \advance\la@zA -\la@Uy \advance\la@zA -\la@Vy
\fi
```

Likewise, if $w_V = 0$ then $\mu = +\infty$ and the comparisons with `\la@zB` and `\la@D` are ignored.

```
\ifdim\la@Vx=\z@
  \la@sM=\z@ \la@zB=-\maxdimen \la@D=\maxdimen
\else
  \la@sM=\la@Vy \lattice@div\la@sM\la@Vx
  \la@zB=-\@tempdima \lattice@mul\la@zB\la@sM \advance\la@zB \@tempdimb
  \la@D=\@tempdimb \advance\la@D -\la@Uy \advance\la@D -\la@Vy
\fi
```

At this point the enumerator for $B^{-1}S$ is fully computed; it uses the registers `\la@sL`, `\la@sM` for the slopes, `\la@wA`, `\la@wB` for the boundaries, `\la@zA`, `\la@zB`, `\la@zC`, `\la@D` for the four affine functions, as well as the original matrix $B$ in `\la@a`, `\la@b`, `\la@c`, `\la@d` (to recompute the points of $\Lambda$).

We re-use the freed registers in the following way:

- (`\la@Vx`, `\la@Vy`) hold the $(w, z)$ coordinates of the current lattice point (stored as integers);
- (`\la@Ux`, `\la@Uy`) hold the corresponding point in $(x, y)$ space (stored as dimens);
- (`\la@Cx`, `\la@Cy`) hold the $(x, y)$ coordinates of the $(w, 0)$ point, which is used for each column.

Because the fixed-point arithmetic is imprecise, we might miss some lattice points on the boundaries. We compensate for this by enlarging the bounds ($-1$ for lower bounds, $+1$ for upper bounds) and *a posteriori* checking that all the computed points are in the bounding box.

Store in `\la@Vx` our starting $(w, z)$ coordinate, which is the integer $\lceil$`\la@wA`$\rceil - 1$. Since TEX knows only *signed* division (groumpf), computing the ceiling is sign-dependent.

```
\la@Vx=\la@wA \advance\la@Vx\expandafter\ifdim\la@Vx>\z@ -1sp\else-\p@\fi
\divide\la@Vx \p@
```

Compute the upper boundary `\la@wB` $= \lfloor$`\la@wB`$\rfloor + 1$.

```
\ifdim\la@wB<\z@ \advance\la@wB 1sp\else\advance\la@wB\p@\fi
\divide\la@wB\p@
```

Compute the difference $t = \lceil w_1 \rceil - w_1$ and correspondingly advance the four affine functions.

```
\@tempdima=\la@Vx \multiply\@tempdima\p@
\advance\@tempdima -\la@wA
\@tempdimb=\@tempdima \lattice@mul\@tempdimb\la@sL
\advance\la@zA\@tempdimb \advance\la@zC\@tempdimb
\@tempdimb=\@tempdima \lattice@mul\@tempdimb\la@sM
\advance\la@zB\@tempdimb \advance\la@D\@tempdimb
```

Store in `\la@Cx`, `\la@Cy` the $(x, y)$ coordinates of the lattice vector $w \cdot (a, b)$:

```
\la@Cx=\la@a \multiply\la@Cx\la@Vx
\la@Cy=\la@b \multiply\la@Cy\la@Vx
```

The main $w$-loop starts here.

```
\loop
```

Compute the range of $z =$ `\la@Vy` for this value of $w =$ `\la@Vx`. The minimum value is $\max($`\la@zA`$,$ `\la@zB`$)-1$:

```
    \la@Vy=\la@zA \ifdim\la@Vy<\la@zB \la@Vy=\la@zB\fi
    \advance\la@Vy\expandafter\ifdim\la@Vy>\z@ -1sp\else-\p@\fi
    \divide\la@Vy\p@
```

The maximum value is `\la@wM` $= \min($`\la@zC`$,$ `\la@D`$)$:

```
    \la@wM=\la@zC \ifnum\la@wM>\la@D \la@wM=\la@D\fi
    \advance\la@wM\expandafter\ifnum\la@wM<\z@\@ne\else\p@\fi
    \divide\la@wM\p@
```

Compute the ($\backslash$`la@Ux`$,$ `\la@Uy`) point in $(x, y)$ space:

```
    \la@Ux=\la@c \multiply\la@Ux\la@Vy \advance\la@Ux\la@Cx
    \la@Uy=\la@d \multiply\la@Uy\la@Vy \advance\la@Uy\la@Cy
```

Inner ($z$) loop. Nested loops require grouping.

```
    {\loop
```

If the point is inside the bounding box, we invoke `\lattice@donode`. Note the use of `\expandafter` to ensure that the `\node` call inside the PGF key `/lattice/each node` gets the expanded options list.

```
      \ifdim\la@Ux<\la@Ax\else\ifdim\la@Ux>\la@Bx\else
      \ifdim\la@Uy<\la@Ay\else\ifdim\la@Uy>\la@By\else
        \expandafter\lattice@donode\expandafter{\lattice@node}%
      \fi\fi\fi\fi
```

End of the $z$ loop. We increase $z$ and correspondingly advance the vector.

```
     \ifnum\la@Vy<\la@wM
        \advance\la@Vy 1sp
        \advance\la@Ux\la@c \advance\la@Uy\la@d
      \repeat}
```

End of the $w$ loop. We increase $w$ and correspondingly advance the $w \cdot (a, b)$ vector as well as the four affine functions.

```
    \ifnum\la@Vx<\la@wB
    \advance\la@Vx 1sp
    \advance\la@Cx\la@a \advance\la@Cy\la@b
    \advance\la@zA \la@sL \advance\la@zC \la@sL
    \advance\la@zB \la@sM \advance\la@D \la@sM
  \repeat
}
```

This macro gets called for each found lattice point. Its main job is to ensure that the node options (here `#1`) are correctly expanded.

```
\def\lattice@donode#1{%
  \pgfkeys{/lattice/arg/each point={#1}{\strip@pt\la@Ux}{\strip@pt\la@Uy}%
    {\number\la@Vx}{\number\la@Vy}}}

```

The PGF keys used for parsing the arguments of the `\lattice` command; they correspond to the options passed in square brackets.

```
\pgfqkeys{/lattice/arg}{
  x/.store in=\lattice@x,
  y/.store in=\lattice@y,
  grid/.store in=\lattice@grid,
  each point/.code n args={5}{\pgfkeys{/lattice/each point={#1}{#2}{#3}{#4}{#5}}},
```

```
  bounding box/.store in=\lattice@bbox,
  .unknown/.code={%
    \expandafter\lattice@setnode\pgfkeyscurrentkey=#1\lattice@eov
  }}
```

The PGF keys used for user configuration of the default values.

```
\pgfqkeys{/lattice}{
  x/.initial=-2:2,y/.initial=-2:2,
  node/.initial={circle,inner sep=1pt,draw=none,fill=black},
  grid/.style={/lattice/arg/grid/.default={#1}},
  grid={gray,very thin},
  bounding box/.style={/lattice/arg/bounding box/.default={#1}},
  bounding box={cyan,thin},
  each point/.code n args={5}{\node[#1] at (#2,#3){};},
}
```

This handles the passing of any unknown `\lattice` keys down to the `\node` calls.

```
\def\lattice@setnode/lattice/arg/#1\lattice@eov{%
  \edef\lattice@node{\expandafter\noexpand\lattice@node,#1}}
```

This macro parses the bounding-box `x` and `y` coordinates, replacing `x=10` by `x=-10:10`. The `detokenize` trick was nicely provided by David Carlisle: `https://tex.stackexchange.com/questions/724989/`.

```
\def\lattice@getxy#1#2#3{%
  \def\m@gic##1:##2:##3\end{\ifx ##3: #2=##1\p@ #3=##2\p@\else
    #2=-##1\p@ #3=##1\p@\fi}%
  \expandafter\m@gic\detokenize{#1:}:\end
}
```

The following macros parse the possible combinations of overlays and square-bracket options. They are similar to the way `\tikz@command@path` is defined in `tikz.code.tex`. Just as in that case, the overlay is handled by a `\alt` call, which will fail if `beamer` is not loaded.

```
\def\lattice{\@ifnextchar<\lattice@I{\@ifnextchar[\lattice@II{\lattice@@[]}}}
\def\lattice@I{\ifnum\the\catcode`\;=\active\relax
  \let\lattice@next\lattice@Iactive\else
  \let\lattice@next\lattice@Inormal\fi
  \lattice@next}
\long\def\lattice@Inormal<#1>#2;{\alt<#1>{\lattice@I@#2;}{}}
{\catcode`\;=\active
  \long\gdef\lattice@Iactive<#1>#2;{\alt<#1>{\lattice@I@#2;}{}}
}
\def\lattice@I@{\@ifnextchar[\lattice@@{\lattice@@[]}}
\def\lattice@II[#1]{\@ifnextchar<{\lattice@IIi[#1]}{\lattice@@[#1]}}
\def\lattice@IIi[#1]<#2>{\lattice@I<#2>[#1]}
```

This macros parses the basis vectors and calls the main loop. `#4` and `#7` are dummy parameters swallowing a possible space between the parentheses and the final semicolon.

```
\def\lattice@@[#1](#2,#3)#4(#5,#6)#7{
  \let\lattice@grid\@empty%
  \let\lattice@bbox\@empty%
  \edef\lattice@x{\pgfkeysvalueof{/lattice/x}}%
  \edef\lattice@y{\pgfkeysvalueof{/lattice/y}}%
  \edef\lattice@node{\pgfkeysvalueof{/lattice/node}}%
  \pgfqkeys{/lattice/arg}{#1}%
```

Read the bounding-box coordinates from the `x` and `y` keys:

```
  \expandafter\lattice@getxy\expandafter{\lattice@x}\la@Ax\la@Bx
  \expandafter\lattice@getxy\expandafter{\lattice@y}\la@Ay\la@By
```

Draw the bounding-box if it was required by the options.

```
  \ifx\lattice@bbox\@empty\else
    \expandafter\draw\expandafter[\lattice@bbox]
    (\strip@pt\la@Ax,\strip@pt\la@Ay)
    rectangle (\strip@pt\la@Bx,\strip@pt\la@By);
  \fi
```

Call the main loop.

```
  \lattice@enumerate{#2}{#3}{#5}{#6}%
}
```