optics — une bibliothèque tikz pour les schémas d'optique

Michel Fruchart
michel (dot) fruchart [at] ens-lyon (dot) org

2020-01-25 – version 0.2.5-alpha
https://github.com/fruchart/tikz-optics

1 Introduction

Le but de cette bibliothèque est de permettre de facilement faire des schémas optiques avec tikz, c'est-à-dire de dessiner lentilles, miroirs, etc. Le tracé géométriquement correct (ou non) des rayons est laissé à l'utilisateur.

1.1 Legal matters

1.2 Installation de la bibliothèque

Si vous avez une installation Texlive à jour, la librairie `tikz-optics` devrait déjà être installée. Pour le vérifier, essayez d'utiliser la commande `\usetikzlibrary{optics}` dans un fichier TEX (après avoir chargé tikz). Si cela ne marche pas, il vous faut mettre à jour votre installation TEX, ou bien installer la librairie manuellement (comme expliqué ci-dessous).

Cette bibliothèque est une « bibliothèque tikz ». Il y a deux façons de l'utiliser :

— en ajoutant le fichier `tikzlibraryoptics.code.tex` dans un dossier où TEX peut le trouver, par exemple dans votre dossier `TEXMFHOME` [5], puis en utilisant la commande `\usetikzlibrary{optics}` ;

— en incluant directement le fichier `tikzlibraryoptics.code.tex` contenant la bibliothèque grâce à la commande `\input`.

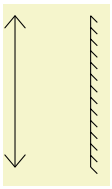
Si la bibliothèque est placée dans un dossier `TEXMF`, disons `/home/agamemnon/texmf/`, il faut respecter la structure TDS de ces répertoires [6]. En pratique, le fichier `tikzlibraryoptics.code.tex` doit être placé dans le dossier `home/agamemnon/texmf/tex/latex/`, ou dans un sous-dossier comme `home/agamemnon/texmf/tex/latex/tikzoptics`. Le plus simple est d'utiliser le contenu de l'archive `tikzoptics.tds.zip`, qui est structuré comme il se doit et peut directement être ajouté dans le dossier `TEXMF` (comme `/home/agamemnon/texmf/`).

1.3 Utilisation de la bibliothèque

`/tikz/use optics` (no value)

1. http://latex-project.org/lppl/lppl-1-3c.txt
2. http://www.gnu.org/licenses/gpl-2.0.en.html
3. http://latex-project.org/lppl/lppl-1-3c.txt
4. https://www.gnu.org/licenses/fdl-1.3.en.html
5. Pour savoir où il se trouve, on peut utiliser la commande `kpsewhich -var-value TEXMFHOME`.
6. Voir par exemple https://www.ctan.org/TDS-guidelines

Après avoir installé la bibliothèque, il suffit de la charger grace à la commande `\usetikzlibrary{optics}`, puis d'utiliser l'option `use optics` sur une `tikzpicture` pour pouvoir utiliser les commandes [7]



```
\begin{tikzpicture}[use optics]
  \node[lens] at (0,0) {};
  \node[mirror] at (1cm,0) {};
\end{tikzpicture}
```

Si l'option `use optics` n'est pas utilisée, le code plantera lamentablement (dans les cas favorables) ou aura un comportement inattendu (dans les cas moins favorables).

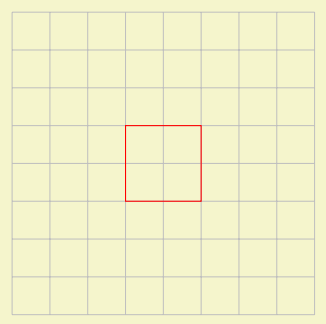
1.4 Élements sur les `node` de tikz et leur placement

Pour plus de détails, lire le manuel de tikz.

En deux mots, une `node` est en gros un objet, qui peut avoir une certaine forme (`shape`). À différents endroits de cette `node` sont définies des ancres (`anchor`). Au contraire d'une `node` qui est en quelque sorte étendue spatialement, ces ancres représentent des points précis sur le canevas de dessin. La `node` a un nom (facultatif) qui permet d'y faire référence dans la suite du dessin, par exemple pour accéder aux coordonnées d'une ancre.

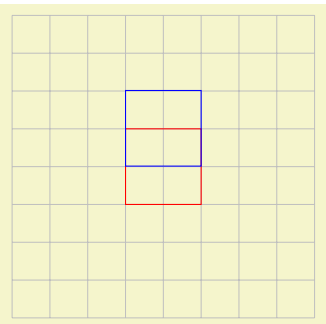
La syntaxe minimale pour créer une `node` est `\node {};`, mais cela n'est pas très utile. Il faut spécifier où se trouve la `node` en écrivant `\node at` ⟨*position*⟩ `{};` (où ⟨*position*⟩ est une coordonnée, comme `(0,1cm)`). Cela n'est toujours pas très utile car rien n'est dessiné. C'est le rôle des `shape` que de déterminer ce qui est dessiné.

Dessinons ainsi un rectangle rouge en `(0,0)` (ici, c'est le centre du canevas), et de côté `1cm` [8]



```
\begin{tikzpicture}
  \draw[style=help lines,gray!50] (-2cm,-2cm) grid[step=0.5cm] (2cm,2cm);
  \node[rectangle,minimum height=1cm,minimum
width=1cm,draw=red] at (0,0) {};
\end{tikzpicture}
```

J'ai ajouté une grille de pas `0.5cm` pour mieux voir les coordonnées. Dans cet exemple, c'est le centre du rectangle qui est placé au point `(0,0)`. Pour être plus précis, il s'agit de son ancre `center`. C'est le comportement par défaut, mais tout l'intérêt est qu'on peut placer les autres ancres définies par la `shape` au point spécifié. Ajoutons donc à ce dessin un rectangle bleu dont le bas est situé en `(0,0)`. Comme nous allons avoir plusieurs `node` qui partagent les caractéristiques `rectangle,minimum height=1cm,minimum width=1cm`, je les ai placées dans un *style* que j'ai appelé `joli rectangle` (il ne faudrait pas l'appeler `rectangle` car cela existe déjà). Comme le bas est représenté par l'ancre `south`, il faut ajouter l'option `anchor=south` à la `node` :

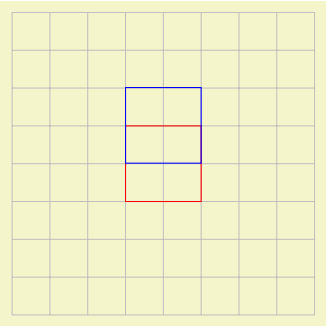


```
\begin{tikzpicture}
  \draw[style=help lines,gray!50] (-2cm,-2cm) grid[step=0.5cm] (2cm,2cm);
  \tikzstyle{joli rectangle}=[rectangle,minimum height=1cm,minimum
width=1cm]
  \node[joli rectangle,draw=red] at (0,0) {};
  \node[joli rectangle,draw=blue,anchor=south] at (0,0) {};
\end{tikzpicture}
```

7. L'option `use optics` charge les éléments pertinents de `/tikz/optics/` dans `/tikz/`, ce qui permet de les utiliser directement. Séparer a priori les commandes dans des espaces de noms différents permet de réduire le risque de collision de noms.

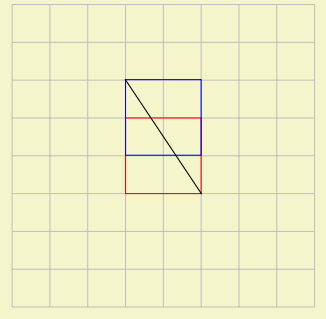
8. Dans cette documentation, les longueurs ne sont généralement pas écrites avec les conventions du système international, mais avec les conventions habituelles pour du code tikz.

J'aimerais maintenant tracer un trait noir entre coin en haut à gauche du carré bleu et le coin en bas à droite du carré rouge. Pour cela, il faut pouvoir accéder aux ancres `north west` (en haut à gauche) et `south east` (en bas à droite) des deux `nodes`. Nous allons donc donner un nom à chacune de ces `nodes`, par exemple `Esther` et `Athalie` :



```
\begin{tikzpicture}
  \draw[style=help lines,gray!50] (-2cm,-2cm) grid[step=0.5cm] (2cm,2cm);
  \tikzstyle{joli rectangle}=[rectangle,minimum height=1cm,minimum
width=1cm]
  \node[joli rectangle,draw=red]
    (Esther) at (0,0) {};
  \node[joli rectangle,draw=blue,anchor=south]
    (Athalie) at (0,0) {};
\end{tikzpicture}
```

Pour le moment, cela ne change absolument rien au dessin, mais on peut désormais accéder aux ancres nécessaires par `(Athalie.north west)` et `(Esther.south east)` et dessiner le trait voulu.



```
\begin{tikzpicture}
  \draw[style=help lines,gray!50] (-2cm,-2cm) grid[step=0.5cm] (2cm,2cm);
  \tikzstyle{joli rectangle}=[rectangle,minimum height=1cm,minimum
width=1cm]
  \node[joli rectangle,draw=red]
    (Athalie) at (0,0) {};
  \node[joli rectangle,draw=blue,anchor=south]
    (Esther) at (0,0) {};
  \draw (Esther.north west) --(Athalie.south east);
\end{tikzpicture}
```

Pour placer les points, il est possible d'utiliser une options du genre `left=of (autre node)`, ou encore `above=5mm of (Esther)`. C'est ce qu'on fait dans l'exemple 2.3 p. 10.

1.5 Programmer des images : faire des calculs avec tikz

Pour plus de détails, lire le manuel de tikz.

Un des principaux avantages [9] d'utiliser du code pour générer des schémas est que cela permet de concevoir les schémas comme un programme informatique. Ainsi, on gagne plusieurs choses :

- en utilisant des variables pour représenter les longueurs et angles du schéma, on peut facilement l'ajuster immédiatement ainsi que le modifier plus tard au besoin ;
- en utilisant des styles pour l'aspect visuel (couleurs, hachures, etc.), il est plus facile de maintenir une cohérence visuelle entre différents schémas. La modification du style se répercutant sur tous les schémas (tant qu'il n'est pas défini à nouveau), on peut facilement décider de changer le bleu représentant l'eau, ou bien, si on veut distribuer un document en noir et blanc, de la représenter par un motif (`pattern`) approprié ;
- on peut utiliser les diverses ressources de la programmation que sont les boucles, les conditions, et les calculs pour effectuer des tâches répétitives ou complexes ;
- il est possible de réutiliser [10] des bouts de schéma

Le but de cette section est de rappeler rapidement comment faire des calculs avec `pgfmath`. Une référence complète se trouve dans le manuel de `tikz`. Les commandes principales que nous allons utiliser sont la boucle `\foreach`, et les commandes de calcul `\pgfmathcalc` et `\pgfmathresult`, `\pgfmathdef`, et `\pgfmathsetlength`.

1.5.1 Calculs

La commande `\pgfmathcalc{<expression>}` parse calcule l'`<expression>` qui lui est donnée (selon des règles qui sont spécifiées dans le manuel de tikz) et enregistre le résultat dans la macro `\pgfmathresult`. Si on

9. Un des principaux inconvénients est que cela demande un peu plus de travail, surtout au début.
10. J'aimerais ajouter « facilement » ici, mais ça n'est hélas pas le cas. Le plus simple est de définir une macro contenant le bout de schéma, et cette méthode marche très bien, mais elle a certaines limites.

a besoin de garder ce résultat, il faut le copier dans une autre macro (disons par exemple `\reflexionAngle`) en utilisant `\let\reflexionAngle\pgfmathresult`. La syntaxe des expressions mathématiques est assez naturelle. Pour les détails, je vous renvoie au manuel tikz ; voici néanmoins quelques exemples [11] :

33.0
```
\pgfmathparse{5*6+3}
```

10.0
```
\pgfmathparse{(4^2 + 4)/2}
```

24.0
```
\pgfmathparse{factorial(4)}
```

0.7071
```
% les angles sont en degres !
\pgfmathparse{cos(45)}
```

0.70709
```
% r convertit les degres en radians
\pgfmathparse{cos(pi/4 r)}
```

0.7071
```
\pgfmathparse{abs(-sqrt(2)/2)}
```

Des fonctions comme `acos`, `sinh`, `ln`, etc. sont disponibles [12]. Il est possible de définir des fonctions via la commande `\pgfmathdeclarefunction` ou la clé `/pgf/declare function`, en fournissant éventuellement une implémentation bas niveau :

1.74144
```
\pgfkeys{/pgf/declare function={f(\x)=1.7280+0.01345/(\x^2);}}
\pgfmathparse{f(1)}
\pgfmathresult
```

À partir de tikz 3.0, la bibliothèque tikz `math` permet de simplifier l'écriture des calculs.

Il existe aussi des fonctions logiques, et les opérateurs booléens habituels (`not(x)=!x`, `or(x,y)=x||y`, `and(x,y)=x&&y`). Par convention, le booléen `true` est représenté par `1` et `false` par `0`. On a alors divers outils avec plusieurs syntaxes :

1
```
\pgfmathparse{2 <= 4}
```

0
```
\pgfmathparse{greater(1,2)}
```

1
```
\pgfmathparse{not(1.5 == 0 && 1.5 <= 2)}
```

La commande `\pgfmathsetmacro{<macro>}{<expression>}` parse l'`<expression>` mathématique de la même manière que `\pgfmathcalc`, et enregistre le résultat du calcul dans `<macro>`. Par exemple, `\pgfmathsetmacro{\transitionProbability}{0.5+0.1}` enregistre `0.6` dans `\transitionProbability`. Par contre, si le résultat du calcul est une chaîne de caractères, il faut utiliser quelque chose comme `\pgfmathparse{<expression>}\let<macro>\pgfmathresult`.

La commande `\pgfmathtruncatemacro` s'utilise comme `\pgfmathsetmacro`, mais elle renvoie un entier (le résultat tronqué), qui peut être utilisé dans une boucle, ou dans une condition. Par exemple, `\pgfmathtruncatemacro\nmax{5/4}` stocke `1` dans `\nmax`.

Une remarque sur les définitions et les groupes. `\pgfmathsetmacro` et les commandes assimilées utilisent la primitive TeX `\def` pour définir les macros, ce qui fait qu'on peut redéfinir une macro sans se préoccuper de savoir si elle existait déjà ou non. De manière générale, je préfère utiliser `\def` plutôt que `\newcommand` pour définir et modifier des variables, car elles peuvent devoir changer lors du tracé du dessin. Cela implique de veiller à ne pas redéfinir des commandes existantes par erreur, mais cela évite aussi beaucoup de tracas. Dans la mesure où le contenu d'une `tikzpicture` est contenu à l'intérieur d'un groupe, les éventuelles redéfinitions intempestives n'ont de conséquences qu'à l'intérieur du dessin. Cela veut aussi dire que pour accèder à une variable définie dans une `tikzpicture`, il faut ajouter un `\global` quelque part.

11. Pour essayer une expression mathématique, le plus simple est d'écrire `\pgfmathparse{2+2}\pgfmathresult` dans le corps d'un document tex : cela affichera `4.0` dans le pdf. Ici, j'ai omis les `\pgfmathresult` pour plus de clareté, mais la commande `\pgfmathparse` n'affiche rien toute seule !

12. Des outils rudimentaires pour travailler avec des vecteurs et un générateur de nombres pseudo-aléatoires existent aussi.

1.5.2 Conditions

Quelques mots sur les fonctions logiques de tikz et leur utilisation.

Une expression comme `greater(\x,pi)` est évaluée à `1` si `\x` est plus grand que π, et à `0` sinon. Toutes les fonctions logiques se comportent de cette manière. Cela permet de définir des fonctions par morceaux, etc. Pour exécuter du code latex sous condition, une méthode (parmi d'autres) consiste en ce qui suit :

1 est plus petit que π ; 2 est plus petit que π ; 3 est plus petit que π ; 4 est plus grand que π ;

```
\foreach \x in {1,2,3,4}
{
  \pgfmathparse{greater(\x,pi)}
  \ifnum\pgfmathresult=1
    \x{} est plus grand que $\pi$ ;
  \else
    \x{} est plus petit que $\pi$ ;
  \fi
}
```

Il peut être nécessaire de stocker le résultat de `\pgfmathresult` dans une macro ; la commande `\pgfmathsetmacro` stockerait `1.0` ou `0.0` (des flottants au lieu d'entiers), qui ne peuvent pas être traités par `\ifnum` : il faut donc faire appel à `\pgfmathtruncateresult`. Ainsi, le code suivant reproduit le même résultat :

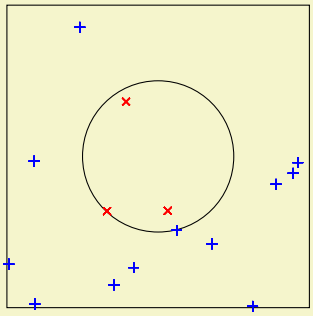
1 est plus petit que π ; 2 est plus petit que π ; 3 est plus petit que π ; 4 est plus grand que π ;

```
\foreach \x in {1,2,3,4}
{
  \pgfmathtruncatemacro\plusGrandQuePi{greater(\x,pi)}
  \ifnum\plusGrandQuePi=1
    \x{} est plus grand que $\pi$ ;
  \else
    \x{} est plus petit que $\pi$ ;
  \fi
}
```

On peut aussi vouloir spécifier un aspect visuel selon une condition. Une méthode consiste à définir des styles, qu'on assigne selon la condition. Ici, nous tirons un certain nombre de points au hasard dans un carré. Nous voulons distinguer les points qui se trouvent dans un certain cercle des autres. Pour cela, on définit deux styles `inside` et `outside`. Pour le moment, nous allons colorer les points dedans en rouge et deux dehors en bleu :

```
\tikzset{inside/.style={red},outside/.style={blue}}
```

Ensuite, il suffit d'utiliser la fonction `ifthenelse` pour définir une macro `\pointStyle` qui vaut `inside` ou `outside`, selon la position du point situé en `(\posx,\posy)`. Dans l'exemple qui suit, je me suis rendu compte que les couleurs ne se voient pas quand on imprime en noir et blanc, et j'ai donc dessiné des croix rouges ainsi que des plus bleus en modifiant les styles `inside` et `outside` :



```
\begin{tikzpicture}
  \tikzset{inside/.style={red,mark=x},outside/.style={blue,mark=+}}
  \newdimen\radius
  \pgfmathsetlength\radius{1cm}
  \pgfmathsetmacro\k{2}
  \pgfmathtruncatemacro\NumberOfPoints{15}

  \draw (0,0) circle(\radius);
  \draw (-\k*\radius,-\k*\radius) rectangle (\k*\radius,\k*\radius);

  \newdimen\posx
  \newdimen\posy
  \foreach \nil in {1,2,...,\NumberOfPoints}
  {
    \pgfmathsetlength\posx{\k*\radius*rand}
    \pgfmathsetlength\posy{\k*\radius*rand}
    \pgfmathparse{ifthenelse(veclen(\posx,\posy)<\radius,"inside", "outside")}
    \let\pointStyle\pgfmathresult
    \draw[\pointStyle] plot (\posx,\posy);
  }
\end{tikzpicture}
```

1.5.3 Boucles `foreach`

La boucle `\foreach` s'utilise grossièrement de la manière suivante :

1,2,3,4,5,6,7,8,9,10,

```
\foreach \x in {1,2,...,10} {\x,}
```

0.42,0.5,12.56,

```
\foreach \x in {0.42,0.5,12.56} {\x,}
```

Il y a un certain nombre de subtilités dans la syntaxe qui permettent de faire des choses élaborées : je vous renvoie au manuel de tikz qui y consacre un chapitre entier ainsi qu'aux exemples. Remarquez que les boucles peuvent être imbriquées.

Dans l'exemple (inutile) suivant, nous voulons placer une barre tous les `\shift` sur un trait de longueur `\length`. De plus, on veut qu'un trait sur deux soit rouge, et l'autre bleu. On calcule donc le nombre `\imax` de traits à dessiner, puis on utilise une boucle. Dans la boucle, on utilise la parité de la variable de boucle `\i` pour savoir si la barre doit être rouge ou bleue.

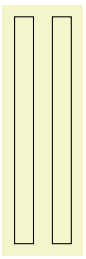
```
\begin{tikzpicture}
  \pgfmathsetmacro\length{4}
  \pgfmathsetmacro\shift{0.5}
  \pgfmathtruncatemacro\imax{\length/\shift}

  \draw (0,0) -- (\length,0);
  \foreach \i in {0,1,2,...,\imax}
  {
    \pgfmathparse{ifthenelse(equal(mod(\i,2),0),"red","blue")}
    \let\chunkStyle\pgfmathresult
    \draw[\chunkStyle] (\i*\shift cm,-0.3cm) -- (\i*\shift cm,0.3cm);
  }
\end{tikzpicture}
```

1.5.4 Les longueurs

En tikz, on travaille toujours, de manière implicite ou explicite, avec des longueurs (et des angles). Par exemple, la coordonnée `(1,2)` signifie généralement `(1cm,2cm)` (en fait, `(1,2)` est vu comme une coordonnée dans un système dont les vecteurs unité ont, par défaut, une longueur d'un centimètre, mais qui peuvent être modifiés). Le plus souvent, il est raisonnable de préciser explicitement les unités et donc de travailler avec des longueurs.

En (la)tex, les longueurs doivent être déclarées avant d'être utilisées, et contrairement aux macros, il n'est pas d'usage de les déclarer et de les définir en même temps. La commande `\newdimen{<longueur>}`

permet de déclarer une longueur[13]. Pour lui donner une valeur, le plus simple est d'utiliser la commande `\pgfmathsetlength{<longueur>}{<expression math>}`, qui parse l'`<expression math>` et l'enregistre dans `<longueur>`. Ainsi,



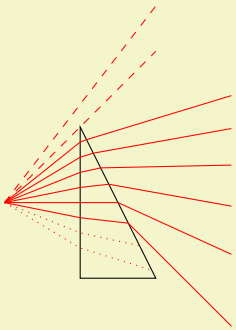
```
\begin{tikzpicture}
  \newdimen\hauteurFente
  \pgfmathsetlength\hauteurFente{3cm}
  \newdimen\largeurFente
  \pgfmathsetlength\largeurFente{0.25cm}
  \newdimen\ecartFentes
  \pgfmathsetlength\ecartFentes{0.5cm}

  \coordinate (L) at (-\ecartFentes/2,0);
  \coordinate (R) at (\ecartFentes/2,0);
  \foreach \x in {L,R}
  \draw[shift={(\x)}] (-\largeurFente/2,-\hauteurFente/2)
    rectangle (\largeurFente/2,\hauteurFente/2);
\end{tikzpicture}
```

La commande `\pgfmathsetlength` suppose qu'une grandeur sans unité est donnée en points (`pt`). Ainsi, `\pgfmathsetlength\hauteurFente{3}` équivaut à `\pgfmathsetlength\hauteurFente{3pt}`. Néanmoins, il n'est pas une bonne idée de ne pas spécifier l'unité.

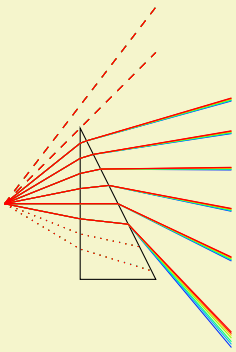
Les fonctions de comparaison de `pgfmath` (comme `greater`, `less`, `equal`) peuvent être utilisées avec des longueurs : par exemple, `\pgfmathparse{2cm <= 4cm}` donne `1`, alors que `\pgfmathparse{2cm <= 4pt}` donne `0`.

1.5.5 Un prisme

Armés de ces outils, nous pouvons dessiner le parcours de rayons lumineux à travers un prisme en verre. Quelques calculs d'optique géométrique permettent de déterminer les coordonnées nécessaires au tracé, et on obtient ainsi la figure suivante :



Comme nous avons utilisé des paramètres ajustables, il est très facile d'adapter le code de manière à tracer les rayons pour plusieurs longueurs d'ondes (en utilisant la loi de Cauchy) :



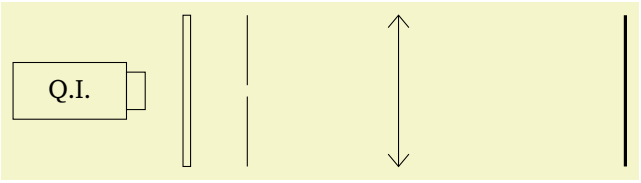
Le code servant à générer cette figure étant un peu long, je ne l'ai pas inclus dans le corps du document, mais en pièce jointe : prisme_dispersion.pgf.

13. Pour la même raison que je préfère `\def` à `\newcommand` pour définir des variables dans les dessins, je préfère `\newdimen` à `\newlength`.

2 Exemples

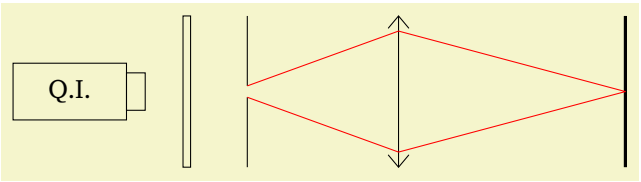
2.1 Image d'une fente sur un écran

La première étape est de placer tous les éléments. On commence de manière très logique par la lampe, qu'on place en `(0,0)`. Ensuite, tous les éléments sont placés à droite les uns des autres. Par exemple, l'anticalorique est placé à `0.5cm` de la sortie de la lampe (ancre `aperture east` de la `node quartz iode`) par l'option `right=0.5cm of (quartz iode.aperture east)`. L'écran et la lentille sont situés relativement à la fente parce que j'ai jugé ça plus pratique pour les modifier indépendamment, mais ce genre de choix dépend de l'humeur et de la situation. J'ai nommé les différents éléments de manière peu cohérente pour souligner que les noms des `nodes` sont très libres : on peut y mettre des espaces, des accents [14], etc. Il faut juste éviter d'y mettre de point, virgule, double-point, etc. qui ont un sens particulier (le point sépare le nom d'une `node` de celui de l'ancre visée, par exemple).




```
\begin{tikzpicture}[use optics]
  \node[halogen lamp] (quartz iode) at (0,0) {Q.I.};
  \node[heat filter,right=0.5cm of quartz iode.aperture east] (AC) {};
  \node[slit,right=0.75cm of AC] (fente) {};
  \node[lens,right=2cm of fente] (L) {};
  \node[screen,right=5cm of fente] (screen) {};
\end{tikzpicture}
```

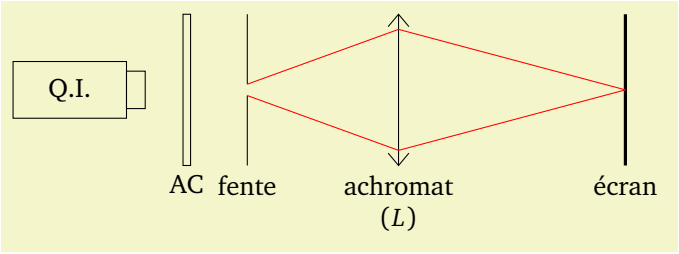
Une deuxième étape consiste à tracer les rayons lumineux. Ici, on a de la chance car toutes les positions nécessaires sont des ancres des différentes `node`. Par exemple, `(L.lens south)` est située juste avant le bas de `L` de manière à ce que le dessin soit joli (mais on peut quand même accéder à ce bas via `(L.south)` - l'idée est qu'il s'agit alors du bas du support annulaire de la lentille). On relie donc les ancres par des `--` dans une commande `\draw`.




```
\begin{tikzpicture}[use optics]
  \node[halogen lamp] (quartz iode) at (0,0) {Q.I.};
  \node[heat filter,right=0.5cm of quartz iode.aperture east] (AC) {} ;
  \node[slit,right=0.75cm of AC] (fente) {};
  \node[lens,right=2cm of fente] (L) {};
  \node[screen,right=5cm of fente] (screen) {};

  \draw[red] (fente.slit north) -- (L.lens north) -- (screen.center)
  (fente.slit south) -- (L.lens south) -- (screen.center);
\end{tikzpicture}
```

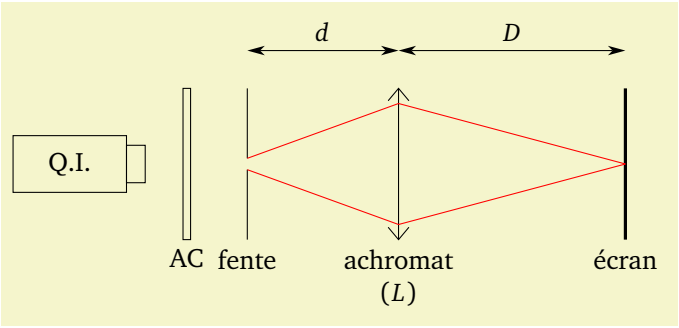
Il faut ensuite ajouter les étiquettes des différents éléments. Cela est fait via l'option `label`, de la forme `label=`⟨*texte*⟩, ou plus généralement `label=[`⟨*opts*⟩`]`⟨*pos*⟩`:`⟨*texte*⟩. En particulier, pour pouvoir sauter une ligne, il faut par exemple utiliser l'option `align=center`.

14. Vous remarquerez que je n'utilise pas d'accents dans les exemples de code. C'est parce que le code qui traite ces exemples n'est pas capable de les traiter. Par contre, il n'y a aucun problème à les utiliser dans tikz (une `node` peut s'appeler `(écran)` sans aucun problème, par exemple).




```
\begin{tikzpicture}[use optics]
  \node[halogen lamp] (quartz iode) at (0,0) {Q.I.};
  \node[heat filter,right=0.5cm of quartz iode.aperture east,label={below:AC}] (AC) {} ;
  \node[slit,right=0.75cm of AC,label={below:fente}] (fente) {};
  \node[lens,right=2cm of fente,label={[align=center]below:achromat \\ $(L)$}] (L) {};
  \node[screen,right=5cm of fente,label={below:\'ecran}] (screen) {};

  \draw[red] (fente.slit north) -- (L.lens north) -- (screen.center)
  (fente.slit south) -- (L.lens south) -- (screen.center);
\end{tikzpicture}
```

Enfin, on ajoute les flèches. Pour ce faire, je définis une coordonnée (c'est une `node` qui n'a qu'une seule ancre et qui ne dessine rien) et j'utilise la position verticale de cette coordonnée (point 1) ainsi que les positions horizontales des (centres des) divers objets (point 2), grace à la syntaxe `(point 1 -| point 2)`.

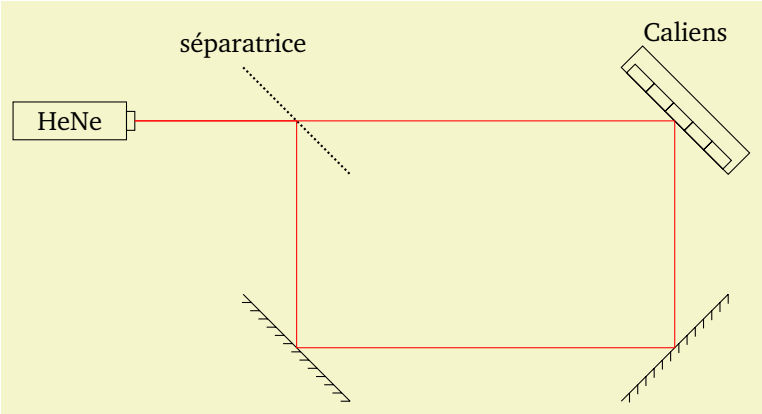



```
\begin{tikzpicture}[use optics]
  \node[halogen lamp] (quartz iode) at (0,0) {Q.I.};
  \node[heat filter,right=0.5cm of quartz iode.aperture east,label={below:AC}] (AC) {} ;
  \node[slit,right=0.75cm of AC,label={below:fente}] (fente) {};
  \node[lens,right=2cm of fente,label={[align=center]below:achromat \\ $(L)$}] (L) {};
  \node[screen,right=5cm of fente,label={below:\'ecran}] (screen) {};

  \draw[red] (fente.slit north) -- (L.lens north) -- (screen.center)
  (fente.slit south) -- (L.lens south) -- (screen.center);

  \coordinate (arrow origin) at (0,1.5cm);

  \draw[>=technical,<->] (arrow origin -| fente) -- (arrow origin -| L) node[midway,above] {$d$};
  \draw[>=technical,<->] (arrow origin -| L) -- (arrow origin -| screen) node[midway,above] {$D$};
\end{tikzpicture}
```

Quand le style sera au point, il sera possible d'utiliser `dim arrow` pour indiquer les dimensions au lieu de cette méthode légèrement plus laborieuse.

2.2 Des interférences

Ici, je n'ai pas utilisé `right=of` et ses amis pour placer les `nodes`, mais des calculs de coordonnées.



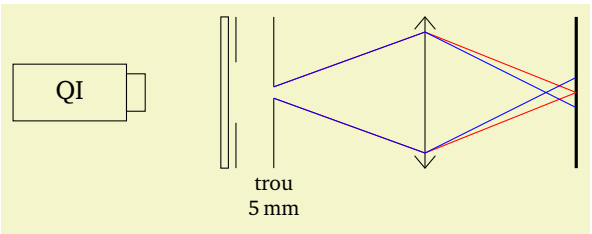

```
\begin{tikzpicture}[use optics]
  \node[laser] (L) at (0,0) {\ce{HeNe}};
  \node[semi-transparent mirror,rotate=45] (ST) at ($(L)+(3cm,0)$) {};
  \node[above] at (ST.north) {s\'eparatrice};
  \node[mirror,rotate=-135] (M1) at ($(ST)+(0,-3cm)$) {};
  \node[mirror,rotate=-45] (M2) at ($(M1)+(5cm,0)$) {};
  \node[sensor line,rotate=45,anchor=pixel 3 west,label={[label distance=0.5cm]above right:Caliens}]
    (Caliens) at ($(ST)+(5cm,0)$) {};
  \draw[red] (L.aperture east) -- (ST.center) -- (M1.center) -- (M2.center) -- (Caliens.pixel 3 west);
  \draw[red] (L.aperture east) -- (ST.center) -- (Caliens.pixel 3 west);
\end{tikzpicture}
```

On a utilisé `anchor=pixel 3 west` pour que le centre du capteur soit placé au point `($(ST)+(5cm,0)$)`.

Plusieurs possibilités pour ajouter un label sont illustrées : créer une `node` au bon endroit avec le texte voulu (ici *séparatrice*), et utiliser la clé `label` (cf. manuel pgf/tikz).

2.3 De la dispersion

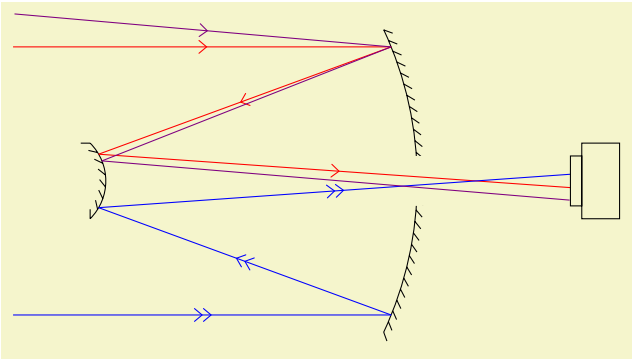
La syntaxe `($(A)!0.6!(B)$)` (*partway modifiers*, lisez le manuel tikz pour les détails) permet de déterminer un point situé à la distance `0.6` entre les points `(A)` et `(B)` (`0` correspond à `(A)` et `1` à `(B)`).




```
\begin{tikzpicture}[use optics]
  % [align=center] permet les labels multiligne
  % [font=\footnotesize] fait un texte plus petit
  \tikzset{every label/.style={align=center,font=\footnotesize}}

  \node[halogen lamp] (S) at (0,0) {QI};
  \node[heat filter, right=of S] (AC) {};
  \node[diaphragm, right=0.1cm of AC] (diaphragme) {};
  \node[slit, right=0.5cm of diaphragme,label={south:{trou \\ \SI{5}{\milli\meter}}}] (T) {};
  \node[lens,right=2cm of T] (L) {};
  \node[screen,right=2cm of L] (ecran) {};

  \draw[red]
  (T.slit north) -- (L.lens north) -- (ecran.center)
  (T.slit south) -- (L.lens south) -- (ecran.center);

  \draw[blue]
  (T.slit north) -- (L.lens north) -- ($(ecran.north)!0.6!(ecran.south)$)
  (T.slit south) -- (L.lens south) -- ($(ecran.north)!0.4!(ecran.south)$);
\end{tikzpicture}
```

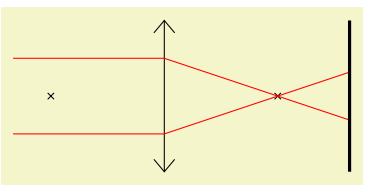
2.4 Un téléscope de Cassegrain

Pour dessiner un miroir troué, on utilise la commande `\clip` de tikz (qu'on place dans un `scope` dont le seul contenu est affecté par le `\clip`). On trace alors les rayons à la main, sans se soucier de l'exactitude du schéma.



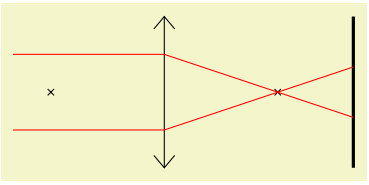
```
\begin{tikzpicture}[use optics]
% mirror with hole
\begin{scope}
  \clip (-0.75cm,-2.2cm) rectangle (1cm,0-0.33cm) (-0.75cm,2.2cm) rectangle (1cm,0+0.33cm);
  \node[spherical mirror, object height=4cm, spherical mirror angle=50] (M1) at (0cm,0) {};
\end{scope}

% small mirror
\node[convex mirror, spherical mirror orientation=rtl,
    object height=1cm, spherical mirror angle=90] (M2) at (-4cm,0) {};

% convergence point
\coordinate (F) at (1cm,0);

% red ray
\begin{scope}[red]
  \draw[-<-] (M1.22) coordinate (P1) -- +(-5cm,0);
  \draw[->-] (P1) -- (M2.30) coordinate (Q1);
  \draw[->-] (Q1) -- ($(Q1)!1.25!(F)$) coordinate (R1);
\end{scope}

% blue ray
\begin{scope}[blue]
  \draw[-<<-] (M1.-22) coordinate (P2) -- +(-5cm,0);
  \draw[->>-] (P2) -- (M2.-30) coordinate (Q2);
  \draw[->>-] (Q2) -- ($(Q2)!1.25!(F)$) coordinate (R2);
\end{scope}

% violet ray
\begin{scope}[violet]
  \draw[-<-] (M1.22) coordinate (P3) -- +(175:5cm);
  \draw (P3) -- (M2.22) coordinate (Q3);
  \draw (Q3) -- ($(Q3)!1.25!($(F)+(0,-0.15cm)$)$) coordinate (R3);
\end{scope}

% sensor
\node[generic sensor, anchor=aperture west] at ($(R1)!0.5!(R2)$) {};
\end{tikzpicture}
```

2.5 De l'optique géométrique et des calculs



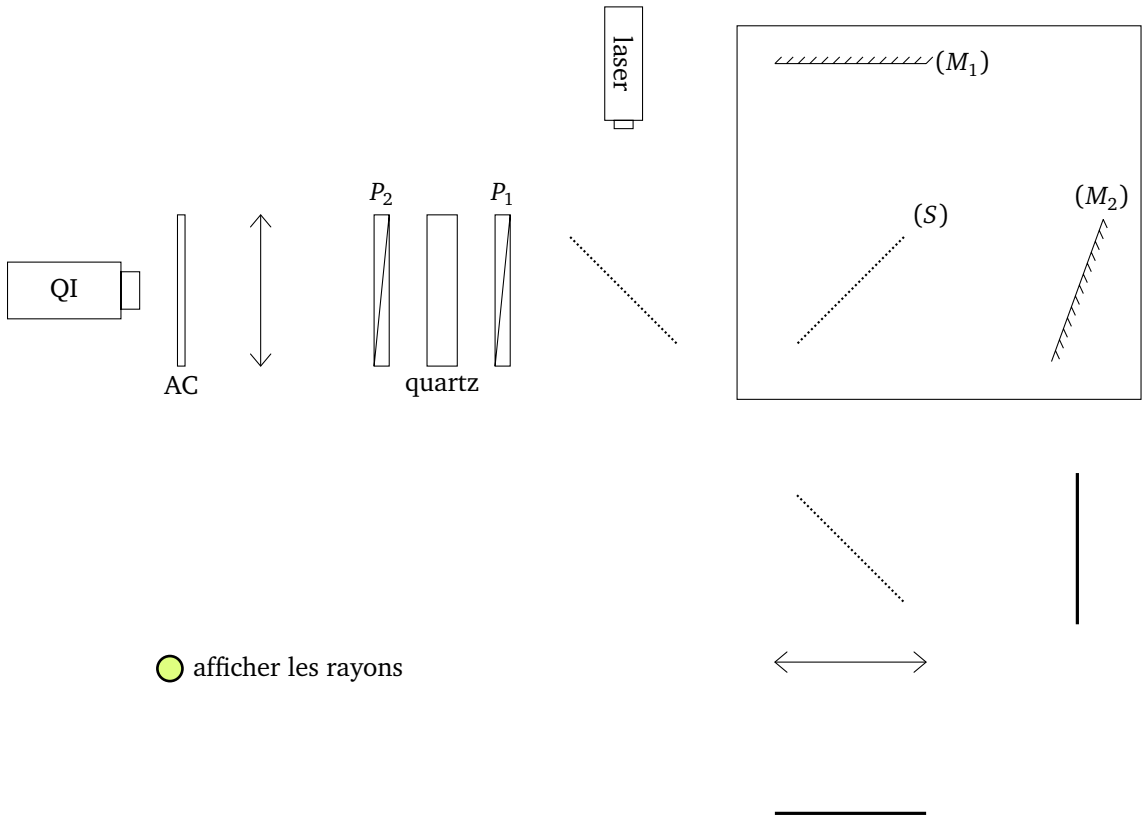
```
\begin{tikzpicture}[use optics]
  \node[lens,draw focal points,focal length=1.5cm,object height=2cm] (L) at (0,0) {};
  \coordinate (P) at (-2cm,0.5cm);
  \coordinate (Q) at (-2cm,-0.5cm);
  \draw[red,shorten >=-1cm] (P) -- ($(L.north)!(P)!(L.south)$) -- (L.east focus);
  \draw[red,shorten >=-1cm] (Q) -- ($(L.north)!(Q)!(L.south)$) -- (L.east focus);
  \node[screen] at (2.45cm,0) {};
\end{tikzpicture}
```

Cet exemple n'est pas très propre, car j'ai dû allonger manuellement les rayons pour qu'ils aillent jusqu'à l'écran. La méthode suivante, qui consiste à calculer le point d'intersection grâce à une fonction `\toVerticalProjection`, est beaucoup plus propre :



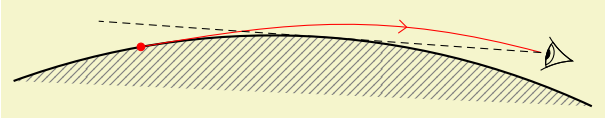
```
\begin{tikzpicture}[use optics]
  \node[lens,draw focal points,focal length=1.5cm,object height=2cm] (L) at (0,0) {} ;
  \coordinate (P) at (-2cm,0.5cm) ;
  \coordinate (Q) at (-2cm,-0.5cm) ;
  \node[screen] (S) at (2.5cm,0) {};

  \def\toVerticalProjection#1#2#3{let \p{1} = #1, \p{2} = #2, \p{3} = #3 in
   -- (\x{3},{\y{1}+(\y{2}-\y{1})/(\x{2}-\x{1})*(\x{3}-\x{1})})}

  \draw[red] (P) -- ($(L.north)!(P)!(L.south)$) coordinate (Plens)
  \toVerticalProjection{(Plens)}{(L.east focus)}{(S)};

  \draw[red] (Q) -- ($(L.north)!(Q)!(L.south)$) coordinate (Qlens)
  \toVerticalProjection{(Qlens)}{(L.east focus)}{(S)};

\end{tikzpicture}
```

2.6 De la biréfringence avec un interféromètre de Michelson



Cet exemple utilise des intersections pour tracer des rayons lumineux : la syntaxe n'est pas la plus agréable et concise qu'on puisse concevoir, mais elle marche. Cet exemple est aussi l'occasion de montrer l'utilisation du package `ocgx` avec `tikz`. Le code de la figure est dans le fichier birefringence_michelson.pgf.

2.7 Un mirage



```
\begin{tikzpicture}[use optics]
  \pgfmathsetmacro\thetaA{110}
  \pgfmathsetmacro\thetaB{65}
  \pgfmathsetmacro\thetaP{100}
  \pgfmathsetmacro\thetaQ{70}
  \pgfmathsetmacro\deltaTheta{5}
  \newdimen\radius
  \pgfmathsetlength\radius{10cm}

  \newdimen\height
  \pgfmathsetlength\height{0.4cm}

  \draw[thick, pattern=north east lines, pattern color=gray] (0,0)
    arc [start angle=\thetaA, end angle=\thetaB, radius=\radius];
  \path (0,0) arc [start angle=\thetaA, end angle=\thetaP, radius=\radius] coordinate (P);
  \path (0,0) arc [start angle=\thetaA, end angle=\thetaQ, radius=\radius] coordinate (Q);
  \coordinate (QE) at ($(Q)+(\thetaQ:\height)$);

  \node[circle, draw,fill,red, inner sep=0, minimum size=0.1cm] at (P) {};
  \draw[red,->>={at=0.65}] (P) to[out={\thetaP-90}, in={\thetaQ+\deltaTheta+90}] (QE);
  \pic[scale=0.75,rotate={\thetaQ+\deltaTheta-90}] (eye) at (QE) {optics eye};

  \pgfmathsetmacro\thetaTangential{86}
  \path (0,0) arc [start angle=\thetaA, end angle=\thetaTangential, radius=\radius] coordinate (H);
  \draw[densely dashed,shorten >=-3cm] (eye-in) -- (H);
\end{tikzpicture}
```

3 Référence

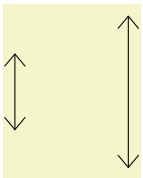
3.1 Généralités

3.1.1 Options communes

Certaines options sont communes à beaucoup des `shapes` (à l'exception notable des lampes et capteurs).

`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

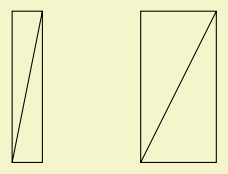
L'option `object height` contrôle la hauteur de la plupart des objets (c'est le cas si rien n'est précisé).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens, object height=1cm] (L1) at (0,0) {};
  \node[lens, object height=2cm] (L2) at (3cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/object aspect ratio`=⟨*number or length*⟩ (no default, initially `0.2`)

L'option `object aspect ratio` contrôle le rapport d'aspect de la plupart des objets ayant une largeur. Si ⟨*number*⟩ vaut `1`, la largeur de l'objet est égale à sa hauteur. Si ⟨*number*⟩=`1/2`, la largeur de l'objet vaut la moitié de sa hauteur. Quand ⟨*number or length*⟩ est un nombre sans dimension (e.g. `0.5`), il est interprété comme un rapport d'aspect, relatif à la hauteur. Quand il s'agit d'une longueur dimensionnée (e.g. `1cm`), il est directement interprété comme la largeur (*width*) de l'objet.



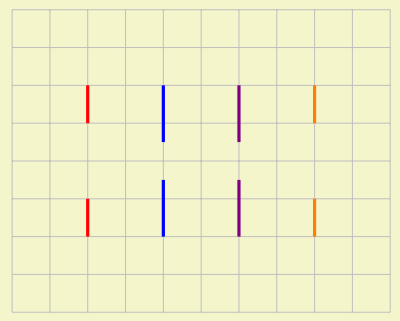
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[polarizer, object aspect ratio=0.2] (L1) at (0,0) {};
  \node[polarizer, object aspect ratio=0.5] (L2) at (4cm,0) {};
\end{tikzpicture}
```

/tikz/optics/object width (style, no value)

L'option `object width` est un alias pour `object aspect ratio`.

3.1.2 Longueurs absolues et relatives

Les divers éléments ont plusieurs longueurs ajustables. La plupart du temps, il n'est nécessaire de spécifier qu'une seule longueur (généralement la hauteur de l'objet) de manière absolue, c'est-à-dire avec une unité de longueur (`cm`, `pt`, `em`, etc.). Les autres longueurs peuvent être spécifiées en unité de cette longueur absolue de référence (il faut alors spécifier un nombre), ou, si cela s'avère plus pratique, comme des longueurs absolues (il faut alors spécifier une unité). Voici un exemple avec `slit height`, qui peut être spécifié relativement à `object height`.



```
\begin{tikzpicture}[use optics]
  \draw[style=help lines,gray!50]
    (-3cm,-2cm) grid[step=0.5cm] (2cm,2cm);
  \node[slit,object height=2cm,slit height=0.5,red,very thick]
    at (-2cm,0) {};
  \node[slit,object height=2cm,slit height=0.25,blue,very thick]
    at (-1cm,0) {};
  \node[slit,object height=2cm,slit height=0.5cm,violet,very
thick]
    at (0cm,0) {};
  \node[slit,object height=2cm,slit height=1cm,orange,very thick]
    at (1cm,0) {};
\end{tikzpicture}
```

L'intérêt d'utiliser des longueurs *relatives* est que la *forme générale* d'un objet n'est pas modifiée par un changement d'échelle (il n'y a qu'à changer la longueur absolue de référence).

Les éléments optiques ont plusieurs « hauteurs ». La hauteur totale est toujours appelée `objet height`, mais par exemple la taille d'une fente est appelée `slit height`, la taille (sans conséquence sur le dessin, mais qui affecte les ancres) d'une lentille sur son support est appelée `lens height`, etc.

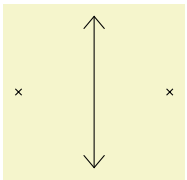
3.2 Éléments optiques

3.2.1 Lentille

Shape `lens`

Dessine une lentille.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens] (L) at (0,0) {};
\end{tikzpicture}
```

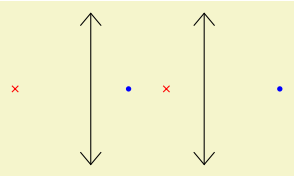


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens,draw focal points] (L) at (0,0) {};
\end{tikzpicture}
```

/tikz/optics/draw focal points=⟨*style*⟩ (style, default `empty`)

On peut tracer les foyers avec `draw focal points`. En donnant ⟨*style*⟩ en argument à la clé `draw docal points`, on peut déterminer comment sont tracés les foyers. Par exemple `draw focal`

`points=red` les trace en rouge, et `draw focal points=circle,draw=none,fill=blue` trace un cercle rempli en bleu.



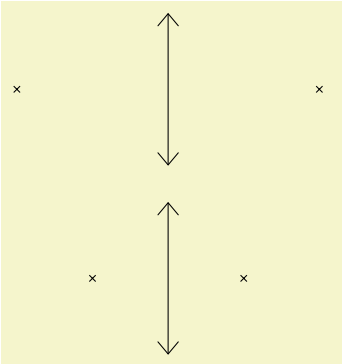
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens,draw focal points={red}]
  (L1) at (0,0) {};
  \node[lens,draw focal points={circle,draw=none,fill=blue}]
  (L2) at (3cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

L'option `object height` est applicable.

`/tikz/optics/focal length`=⟨*length*⟩ (no default, initially `1cm`)

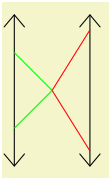
L'option `focal length` contrôle la distance focale de la lentille.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens, focal length=1cm] (L1) at (0,0) {};
  \node[lens, focal length=2cm] (L2) at (0,5cm) {};
\end{tikzpicture}
```

`/tikz/optics/lens height`=⟨*number or length*⟩ (no default, initially `0.8`)

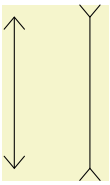
L'option `lens height` contrôle la hauteur de la lentille. Il s'agit soit une longueur abolue (avec unité), soit d'une longueur relative mesurée par rapport à la hauteur totale de la lentille avec support.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens] (L1) at (1cm,0) {};
  \node[lens, lens height=0.5] (L2) at (-1cm,0) {};
  \draw[red] (0,0) -- (L1.lens north) (0,0) -- (L1.lens south);
  \draw[green] (0,0) -- (L2.lens north) (0,0) -- (L2.lens south);
\end{tikzpicture}
```

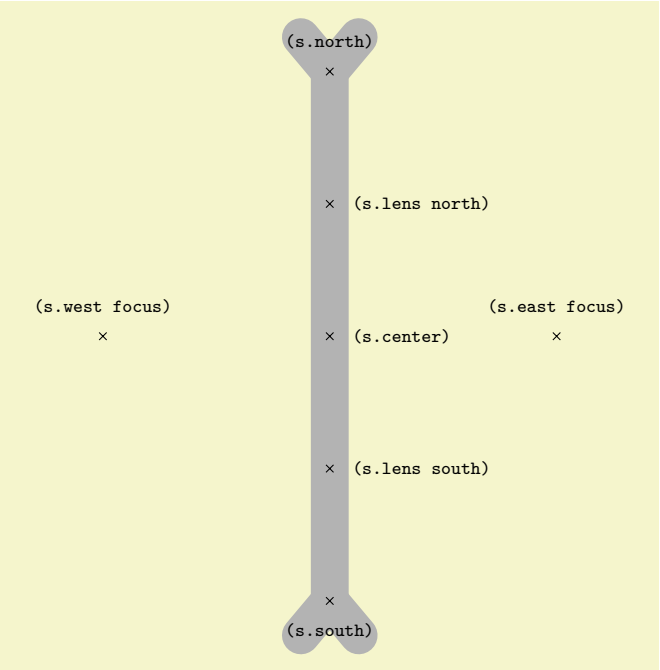
`/tikz/optics/lens type` (no value)

L'option `lens type` contrôle le type de lentille. Utiliser `lens type=converging` permet de dessiner une lentille convergente (c'est l'option par défaut), alors que `lens type=diverging` permet de dessiner une lentille divergente.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens,lens type=converging] (L1) at (-1cm,0) {};
  \node[lens,lens type=diverging] (L2) at (1cm,0) {};
\end{tikzpicture}
```

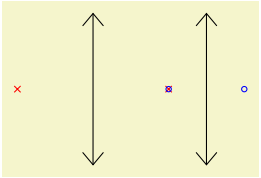
La figure suivante récapitule les ancres définies par `lens`.




```
\Huge
\begin{tikzpicture}[use optics]
\node[name=s,lens,object height=7cm,focal length=3cm,
lens height=0.5,line shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,lens north/right,lens south/right,center/right,
east focus/above,west focus/above}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

Les clés `east focus` et `west focus` ont pour synonyme `east focal point` et `west focal point`, respectivement.

On peut utiliser la clé `tikz anchor=` pour placer les lentilles les unes par rapport aux autres.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[lens,draw focal points={red}]
  (L1) at (0,0) {};
  \node[lens,draw focal points={circle,draw=blue},
  focal length=0.5cm,anchor=west focus]
  (L2) at (L1.east focus) {};
\end{tikzpicture}
```

3.2.2 Fente

Shape `slit`

Dessine une fente.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[slit] (S) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

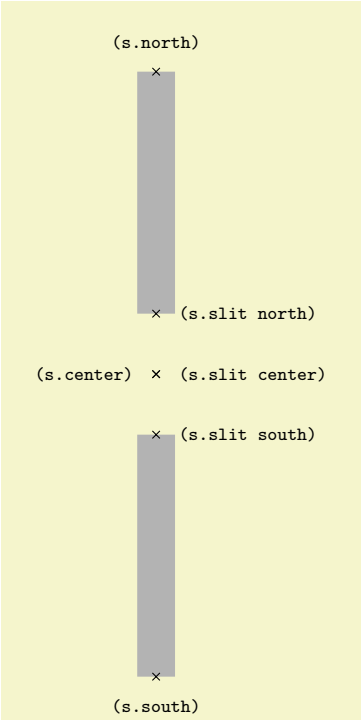
L'option `object height` est applicable.

`/tikz/optics/slit height`=⟨*number or length*⟩ (no default, initially `0.075`)

La clé `slit height` définit la hauteur de la fente (s'il s'agit d'une longueur relative, elle est exprimée en unités de la hauteur de l'objet). Par exemple, ⟨*number*⟩=`0.5` donne une fente faisant la moitié de la hauteur du support. De même, ⟨*length*⟩=`1cm` donne une fente de `1cm`. Les valeurs de ⟨*number*⟩ doivent être inférieures à l'unité, et celles de ⟨*length*⟩ inférieures à celles de `object height`, sans quoi les résultats ne sont pas garantis.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[slit, slit height=0.5] (S) at (0,0) {};
  \node[slit, slit height=0.3] (S) at (1cm,0) {};
  \node[slit, slit height=0.1] (S) at (2cm,0) {};
\end{tikzpicture}
```

La figure suivante récapitule les ancres définies par `slit`.




```
\Huge
\begin{tikzpicture}[use optics]
\node[name=s,slit,object height=8cm,
slit height=0.2,line shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,slit north/right,slit south/right,center/left,
slit center/right}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.3 Double fente

Shape `double slit`

Dessine une fente double.

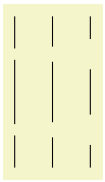

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[double slit] (S) at (0,0) {};
\end{tikzpicture}
```

/tikz/optics/object height=⟨*length*⟩ (no default, initially 2cm)

L'option object height est applicable.

/tikz/optics/slit height=⟨*number or length*⟩ (no default, initially 0.075)

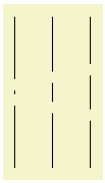
La clé slit height définit la hauteur d'une fente (s'il s'agit d'une longueur relative, elle est en unités de la hauteur de l'objet). Chacune des fentes a une hauteur ⟨*number or length*⟩.



```
\begin{tikzpicture}[use optics,scale=.5,optics/slit separation=0.5]
  \node[double slit, slit height=0.075] (S) at (0,0) {};
  \node[double slit, slit height=0.1] (S) at (1cm,0) {};
  \node[double slit, slit height=0.2] (S) at (2cm,0) {};
\end{tikzpicture}
```

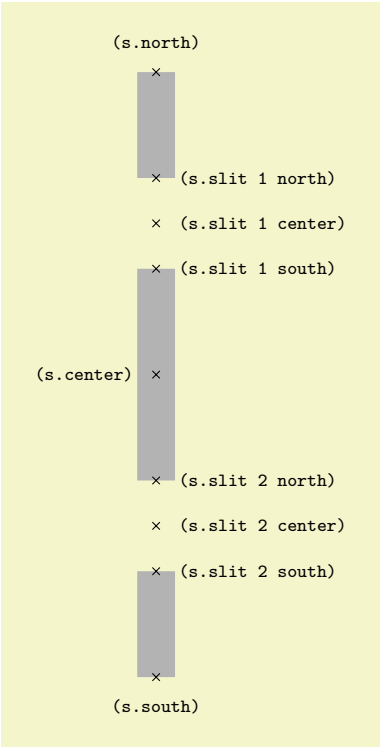
/tikz/optics/slit separation=⟨*number or length*⟩ (no default, initially 0.2)

La clé slit separation définit la distance entre les deux fentes (s'il s'agit d'une longueur relative, elle est en unités de la hauteur de l'objet).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[double slit, slit separation=0.1] (S) at (0,0) {};
  \node[double slit, slit separation=0.2] (S) at (1cm,0) {};
  \node[double slit, slit separation=0.3] (S) at (2cm,0) {};
\end{tikzpicture}
```

La figure suivante récapitule les ancres définies par double slit.




```
\Huge
\begin{tikzpicture}[use optics]
\node[double slit,name=s,object height=8cm, slit height=0.15,
slit separation=0.5, line shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,center/left,
slit 1 north/right,slit 1 south/right,slit 1 center/right,
slit 2 north/right,slit 2 south/right,slit 2 center/right}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.4 Miroir

Shape `mirror`

Dessine un miroir plan.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[mirror] (S) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

L'option `object height` est applicable.

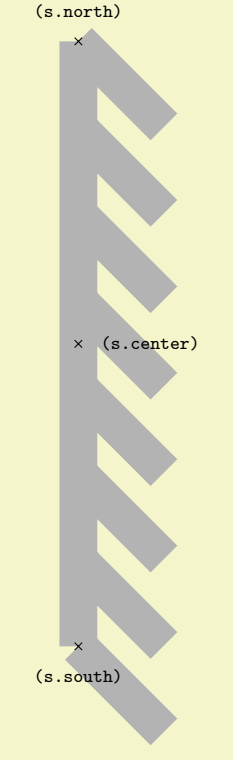
`/tikz/optics/mirror decoration separation`=⟨*number or length*⟩ (no default, initially `0.15cm`)

Correspond à la clé `/pgf/decoration/segment length` pour la décoration `border` utilisée pour dessiner le miroir (cf. manuel de pgf/tikz). La valeur correspondant à `/pgf/decoration/segment length` est obtenue en multipliant ⟨*number*⟩ par la hauteur du miroir.

`/tikz/optics/mirror decoration amplitude`=⟨*number or length*⟩ (no default, initially `0.125cm`)

Correspond à la clé `/pgf/decoration/amplitude` pour la décoration `border` utilisée pour dessiner le miroir (cf. manuel de pgf/tikz). La valeur correspondant à `/pgf/decoration/amplitude` est obtenue en multipliant ⟨*number*⟩ par la hauteur du miroir.

La figure suivante récapitule les ancres définies par `mirror`.



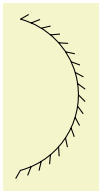

```
\Huge
\begin{tikzpicture}[use optics]
\node[mirror,name=s,object height=8cm,line shape example,
mirror decoration separation=0.141, mirror decoration amplitude=0.2] {};
\foreach \anchor/\placement in
{north/above,south/below,center/right}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.5 Miroir sphérique

Shape `spherical mirror`

Dessine un miroir sphérique (concave ou convexe).

⚠ Cette partie n'est pas encore au point et peut changer sans préavis.



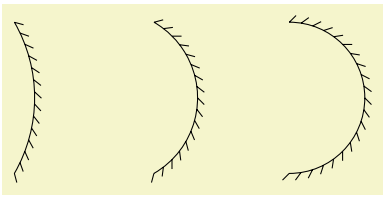
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[spherical mirror] (M) at (0,0) {};
\end{tikzpicture}
```

/tikz/optics/object height=⟨*length*⟩ (no default, initially 2cm)

La clé object height est applicable.

/tikz/optics/spherical mirror angle=⟨*angle*⟩ (no default, initially 150)

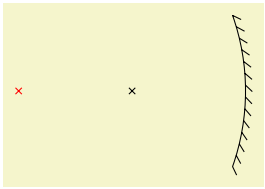
La clé spherical mirror angle contrôle l'angle d'ouverture du miroir sphérique (un arc de cercle de hauteur spécifiée par object height et d'ouverture angulaire ⟨*angle*⟩ est dessiné). Attention à ne pas choisir ⟨*angle*⟩ = 0 : utiliser un miroir plat mirror à la place.



```
\begin{tikzpicture}[use optics]
  \node[spherical mirror, spherical mirror angle=60] at (0,0) {};
  \node[spherical mirror, spherical mirror angle=120] at (2cm,0) {};
  \node[spherical mirror, spherical mirror angle=180] at (4cm,0) {};
\end{tikzpicture}
```

⚠ La fonction from_radius est expérimentale.

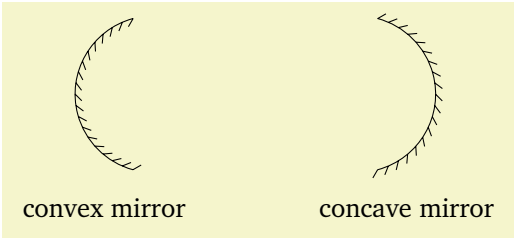
Il peut être utile de spécifier non pas l'angle d'ouverture, mais le rayon de courbure du miroir ; dans ce but, une fonction from_radius(R) calcule l'angle d'ouverture correspondant au rayon R, à hauteur /tikz/optics/object height imposée. Bien entendu, il n'est pas possible que la hauteur soit supérieure au double du rayon du miroir.



```
\begin{tikzpicture}[use optics]
  \node[spherical mirror, object height=2cm,
  spherical mirror angle=from_radius(3cm),
  draw mirror focus, draw mirror center={red}] (M) {};
\end{tikzpicture}
```

/tikz/optics/spherical mirror type (no value)

La clé spherical mirror type contrôle le type de miroir : utiliser spherical mirror type=concave donne un miroir concave (c'est l'option par défaut), alors que spherical mirror type=convex donne un miroir convexe. Il est plus commode d'utiliser les styles convex mirror et concave mirror, qui sont des raccourcis pour cette clé.



```
\begin{tikzpicture}[use optics]
  \node[convex mirror, label={[label distance=0.25cm]south:convex mirror}] at (0cm,0) {};
  \node[concave mirror, label={[label distance=0.25cm]south:concave mirror}] at (4cm,0) {};
\end{tikzpicture}
```

/tikz/optics/concave mirror (style, no value)

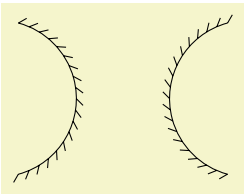
Le style concave mirror correspond à spherical mirror, spherical mirror type=concave, et dessine un miroir concave. Voir spherical mirror type pour un exemple.

/tikz/optics/convex mirror (style, no value)

Le style convex mirror correspond à spherical mirror, spherical mirror type=convex, et dessine un miroir convexe. Voir spherical mirror type pour un exemple.

/tikz/optics/spherical mirror orientation=⟨*type*⟩ (no default, initially ltr)

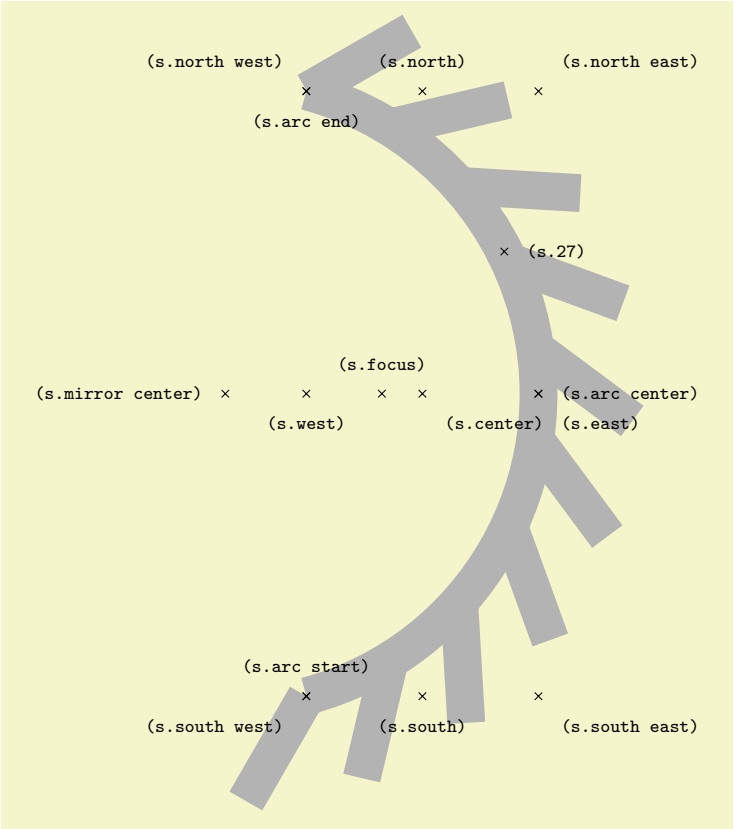
La clé spherical mirror orientation contrôle l'orientation du miroir (i.e. dans quel sens on suppose que la lumière se propage). Les valeurs possibles sont ltr (« left to right ») et rtl (« right to left »).



```
\begin{tikzpicture}[use optics]
  \node[spherical mirror, spherical mirror orientation=ltr] at (0cm,0) {};
  \node[spherical mirror, spherical mirror orientation=rtl] at (2cm,0) {};
\end{tikzpicture}
```

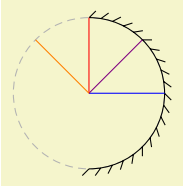
La décoration du miroir (hachures) est contrôlée par les mêmes clés que celle de mirror : mirror decoration separation et mirror decoration amplitude.

La figure suivante récapitule certaines des ancres définies par spherical mirror.



```
\Huge
\begin{tikzpicture}[use optics]
\node[spherical mirror,name=s,object height=8cm,line shape example,
mirror decoration separation=0.141, mirror decoration amplitude=0.2] {};
\foreach \anchor/\placement in
{north/above,south/below,center/below right,
east/below right,
west/below,
north east/above right,
north west/above left,
south east/below right,
south west/below left,
mirror center/left,
arc start/above,
arc end/below,
arc center/right,
27/right,
focus/above}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

Avec les miroirs sphériques, les « border anchor » de tikz sont particulièrement utiles. Des ancres numériques (de la forme `node.27`) sont définies, qui permettent d'accèder au bord de la `shape` dans la direction angulaire correspondante (ici à 27°). Rappelons que dans tikz, les angles sont comptés dans le sens trigonométrique à partir de l'axe Ox, et exprimés en degrés. Suit un exemple, où on a dessiné le cercle sous-jacent.

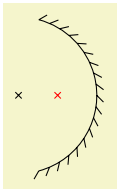


```
\begin{tikzpicture}[use optics,scale=.5]
\coordinate (O) at (0,0);
\node[circle,draw, inner sep=0, outer sep=0,minimum height=2cm, densely
dashed, gray!60] (C) at (O) {};
\node[spherical mirror,draw,object height=2cm,anchor=mirror center,
    spherical mirror angle=180] (M) at (O) {};
\draw[blue] (O) -- (M.0);
\draw[violet] (O) -- (M.45);
\draw[red] (O) -- (M.90);
\draw[orange] (O) -- (M.135);
\end{tikzpicture}
```

⚠ Pour le moment, les angles supérieurs à ±`spherical mirror angle`/2 sont traités comme si le miroir était un cercle entier (et pas un arc). Il n'est pas garanti que ce comportement soit maintenu.

`/tikz/optics/draw mirror center`=⟨*style*⟩ (style, no default)

Le style `draw mirror center` permet de dessiner le centre du miroir (avec le style ⟨*style*⟩ s'il est spécifié).



```
\begin{tikzpicture}[use optics]
\node[concave mirror,draw mirror center, draw mirror focus={red}] {};
\end{tikzpicture}
```

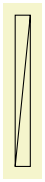
`/tikz/optics/draw mirror focus`=⟨*style*⟩ (style, no default)

Le style `draw mirror focus` permet de dessiner le foyer du miroir (avec le style ⟨*style*⟩ s'il est spécifié). Voir `draw mirror center` pour un exemple.

3.2.6 Polariseur

Shape `polarizer`

Dessine un polariseur.



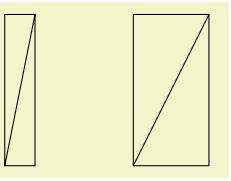
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[polarizer] (S) at (0,0) {};
\end{tikzpicture}
```

/tikz/optics/object height=⟨*length*⟩ (no default, initially 2cm)

L'option object height est applicable.

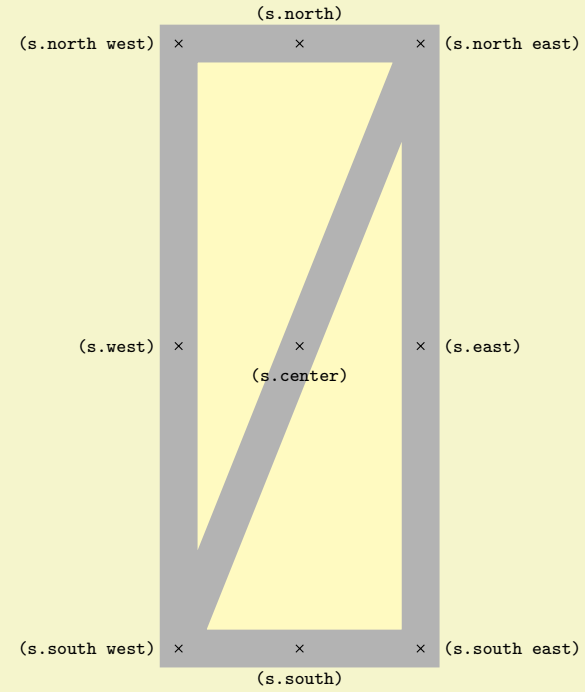
/tikz/optics/object aspect ratio=⟨*number*⟩ (no default, initially 0.2)

L'option object aspect ratio contrôle le rapport d'aspect du polariseur. Si ⟨*number*⟩ vaut 1, la largeur de l'objet est égale à sa hauteur. Si ⟨*number*⟩=1/2, la largeur du polariseur vaut la moitié de sa hauteur



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[polarizer, object aspect ratio=0.2] (L1) at (0,0) {};
  \node[polarizer, object aspect ratio=0.5] (L2) at (4cm,0) {};
\end{tikzpicture}
```

La figure suivante récapitule les ancres définies par polarizer.

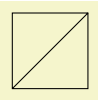



```
\Huge
\begin{tikzpicture}[use optics]
\node[polarizer,name=s,object height=8cm,object aspect ratio=0.4,shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,east/right,west/left,center/below,
north east/right,north west/left,south east/right,south west/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.7 Séparateur de faisceau

/tikz/optics/beam splitter (style, no value)

Dessine un séparateur de faisceau. Les options et ancres sont les mêmes que pour `polarizer`.

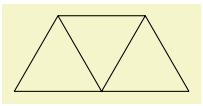


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[beam splitter] at (0,0) {};
\end{tikzpicture}
```

3.2.8 Prisme à vision directe

Shape `double amici prism`

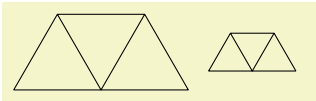
Dessine un prisme à vision directe (double prisme d'Amici).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[double amici prism] (PVD) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/prism height`=⟨*length*⟩ (no default, initially `1.5cm`)

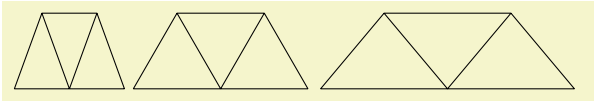
La clé `prism height` contrôle la hauteur des trois prismes identiques constituant le PVD (la longueur d'un côté est donc $2/\sqrt{3}$ fois cette hauteur).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[double amici prism, prism height=1cm] (PVD1) at (0,0) {};
  \node[double amici prism, prism height=0.5cm] (PVD2) at (4cm,0) {};
\end{tikzpicture}
```

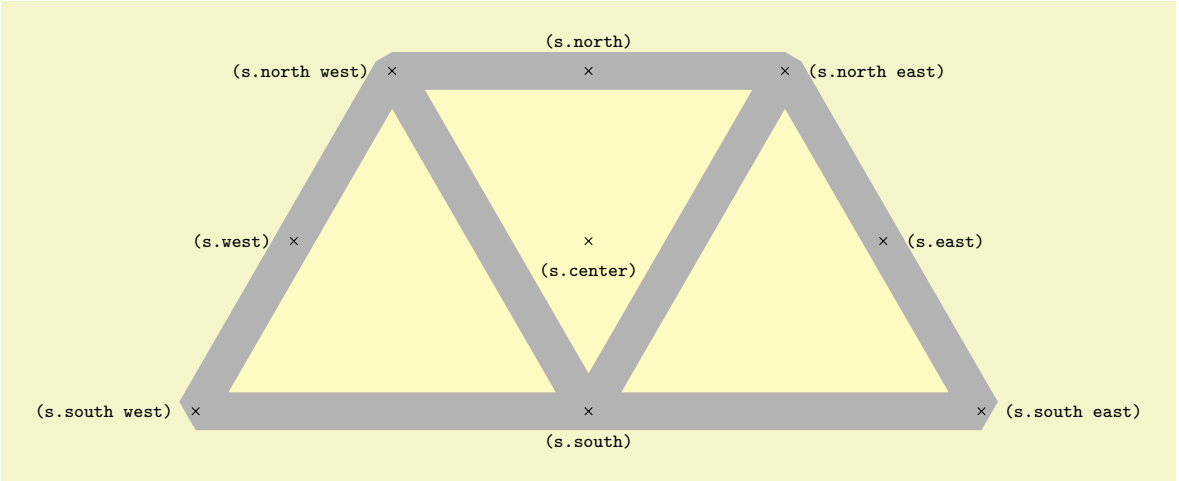
`/tikz/optics/prism apex angle`=⟨*angle*⟩ (no default, initially `60`)

La clé `prism apex angle` contrôle l'angle au sommet des trois prismes identiques. ⟨*angle*⟩ est exprimé en degrés. Si ⟨*angle*⟩ vaut `60`, les prismes sont des triangles equilatéraux.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[double amici prism, prism apex angle=40] (PVD1) at (0,0) {};
  \node[double amici prism, prism apex angle=60] (PVD2) at (4cm,0) {};
  \node[double amici prism, prism apex angle=80] (PVD3) at (10cm,0) {};
\end{tikzpicture}
```

La figure suivante récapitule les ancres définies par `double amici prism`.



```
\Huge
\begin{tikzpicture}[use optics]
\node[double amici prism,name=s,prism height=4.5cm,shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,east/right,west/left,center/below,
north east/right,north west/left,south east/right,south west/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.9 Élément générique fin

Shape `thin optics element`

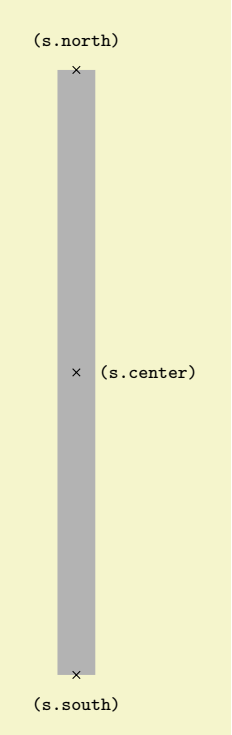
Dessine un élement générique (utilisé pour des éléments plus spécifiques). Cette `shape` peut être utile pour dessiner un objet non prévu, en lui donnant un style pertinent.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[thin optics element] (S) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

L'option `object height` est applicable.

La figure suivante récapitule les ancres définies par `thin optics element`.




```
\Huge
\begin{tikzpicture}[use optics]
\node[thin optics element,name=s,object height=8cm,line shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,center/right}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.10 Élément générique épais

Shape `thick optics element`

Dessine un élément générique épais.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[thick optics element] at (0,0) {};
\end{tikzpicture}
```

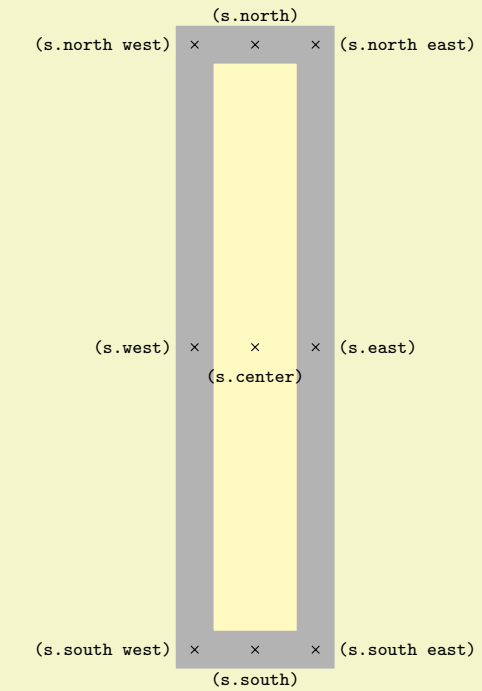
`/tikz/optics/object height`=⟨*length*⟩ (no default, initially `2cm`)

L'option `object height` est applicable.

`/tikz/optics/object aspect ratio`=⟨*number or length*⟩ (no default, initially `0.05`)

L'option `object aspect ratio` contrôle le rapport d'aspect de l'objet.

La figure suivante récapitule les ancres définies par `thick optics element`.




```
\Huge
\begin{tikzpicture}[use optics]
\node[thick optics element,name=s,object height=8cm,shape example,object aspect ratio=0.2] {};
\foreach \anchor/\placement in
{north/above,south/below,east/right,west/left,center/below,
north east/right,north west/left,south east/right,south west/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.2.11 Anticalorique

`/tikz/optics/heat filter` (style, no value)

Le style `heat filter` permet de dessiner un filtre anticalorique. Les options de `thick optics element` sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[heat filter] (S) at (0,0) {};
\end{tikzpicture}
```

3.2.12 Écran

/tikz/optics/screen (style, no value)

Le style screen permet de dessiner un écran. Les options de thin optics element sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[screen] (S) at (0,0) {};
\end{tikzpicture}
```

3.2.13 Réseau de diffraction

/tikz/optics/diffraction grating (style, no value)

Le style diffraction grating permet de dessiner un réseau de diffraction. Les options de thin optics element sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[diffraction grating] (S) at (0,0) {};
\end{tikzpicture}
```

3.2.14 Grille

/tikz/optics/grid (style, no value)

Le style grid permet de dessiner une grille. Les options de thin optics element sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[grid] (S) at (0,0) {};
\end{tikzpicture}
```

3.2.15 Lame semi-réfléchissante

/tikz/optics/semi-transparent mirror (style, no value)

Le style semi-transparent mirror permet de dessiner un miroir semi-réfléchissant. Les options de thin optics element sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[semi-transparent mirror] (S) at (0,0) {};
\end{tikzpicture}
```

3.2.16 Diaphragme

/tikz/optics/diaphragm (style, no value)

Le style diaphragm permet de dessiner un diaphragme (c'est un slit avec une fente large). Les options de slit sont applicables.

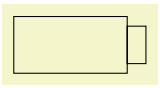
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[diaphragm] (S) at (0,0) {};
\end{tikzpicture}
```

3.3 Lampes et capteurs

3.3.1 Entrée/sortie optique générique

Shape `generic optics io`

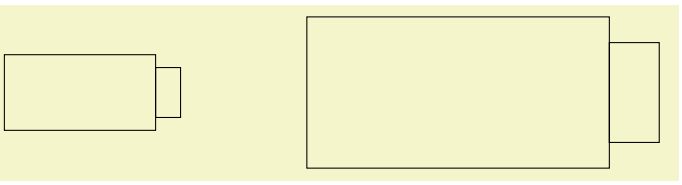
Dessine un système entrée-sortie générique (utilisé pour des objets plus spécifiques).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io] (S) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/io body height`=⟨*length*⟩ (no default, initially `0.75cm`)

La clé `io body height` contrôle la taille de la lampe. En particulier, ⟨*length*⟩ spécifie la hauteur du corps de la lampe, et les autres longueurs sont spécifiées relativement à celle-ci. En modifiant ⟨*length*⟩, on modifie donc la taille de la lampe sans la déformer.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io, io body height=1cm] (L1) at (0,0) {};
  \node[generic optics io, io body height=2cm] (L2) at (10cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/io body aspect ratio`=⟨*number or length*⟩ (no default, initially `2`)

La clé `io body aspect ratio` contrôle le rapport d'aspect de la lampe. Si ⟨*number*⟩ vaut `1`, la largeur de la lampe est égale à sa hauteur. Si ⟨*number*⟩=`1/2`, la largeur de la lampe vaut la moitié de sa hauteur

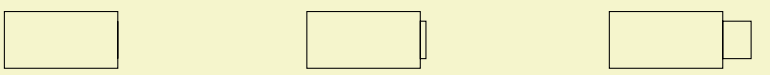
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io, io body aspect ratio=1] (L1) at (0,0) {};
  \node[generic optics io, io body aspect ratio=2] (L2) at (8cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/io body width`=⟨*number or length*⟩ (style, no default)

La clé `io body width` est un alias pour `io body aspect ratio`.

`/tikz/optics/io aperture width`=⟨*number or length*⟩ (no default, initially `0.33`)

La clé `io aperture width` contrôle la largeur du système de sortie de la lampe (qui représente un condenseur, une lentille de collimation, etc.), en unités de la hauteur du corps de la lampe. Si ⟨*number*⟩=`0`, le système de sortie n'est pas affiché.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io, io aperture width=0] at (0,0) {};
  \node[generic optics io, io aperture width=0.1] at (8cm,0) {};
  \node[generic optics io, io aperture width=0.5] at (16cm,0) {};
\end{tikzpicture}
```

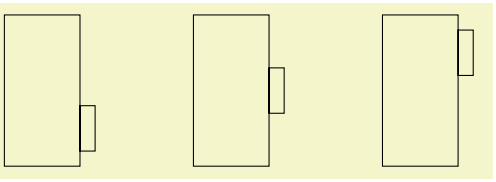
/tikz/optics/io aperture height=⟨*number or length*⟩ (no default, initially 0.66)

La clé io aperture width contrôle la hauteur du système de sortie de la lampe (qui représente un condenseur, une lentille de collimation, etc.), en unités de la hauteur du corps de la lampe.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io, io aperture height=0.5] at (0,0) {};
  \node[generic optics io, io aperture height=0.8] at (8cm,0) {};
\end{tikzpicture}
```

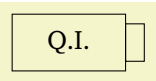
/tikz/optics/io aperture shift=⟨*number or length*⟩ (no default, initially 0)

L'option io aperture shift contrôle le décalage du système de sortie de la lampe par rapport au centre, en unités de la hauteur du corps de la lampe.



```
\begin{tikzpicture}[use optics,scale=.5,
optics,io body height=2cm,io body aspect ratio=0.5,io aperture height=0.3, io aperture width=0.1]
\node[generic optics io,io aperture shift=-0.25] at (-5cm,0) {};
  \node[generic optics io,io aperture shift=0] at (0,0) {};
  \node[generic optics io,io aperture shift=0.25] at (5cm,0) {};
\end{tikzpicture}
```

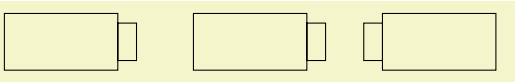
On peut afficher du texte dans le corps d'une lampe en utilisant le texte de la node.




```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io] at (-5cm,0) {Q.I.};
\end{tikzpicture}
```

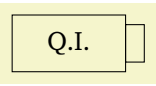
/tikz/optics/io orientation=⟨*type*⟩ (no default, initially ltr)

La clé io orientation détermine le sens dans lequel est dessiné l'objet (avec l'ouverture à droite (au niveau de l'ancre east) quand elle vaut ltr et à gauche (au niveau de l'ancre west) quand elle vaut rtl). Les noms correspondent à « left to right » et « right to left ». Seules les arguments ltr et rtl sont autorisés. La différence avec rotate est que io orientation modifie les ancres.



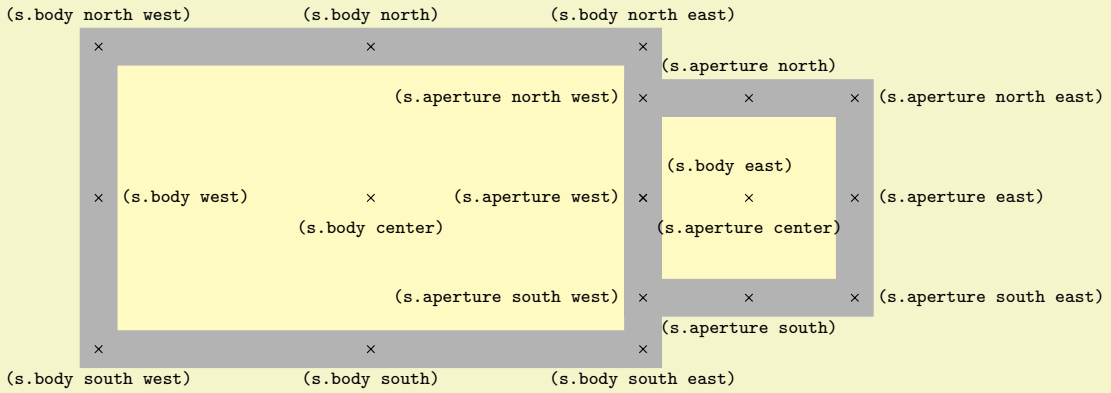
```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic optics io] at (0,0) {};
  \node[generic optics io,io orientation=ltr] at (5cm,0) {};
  \node[generic optics io,io orientation=rtl] at (10cm,0) {};
\end{tikzpicture}
```

On peut afficher du texte dans le corps d'une lampe en utilisant le texte de la `node`.




```
\begin{tikzpicture}[use optics,scale=.5]
\node[generic optics io] at (-5cm,0) {Q.I.};
\end{tikzpicture}
```

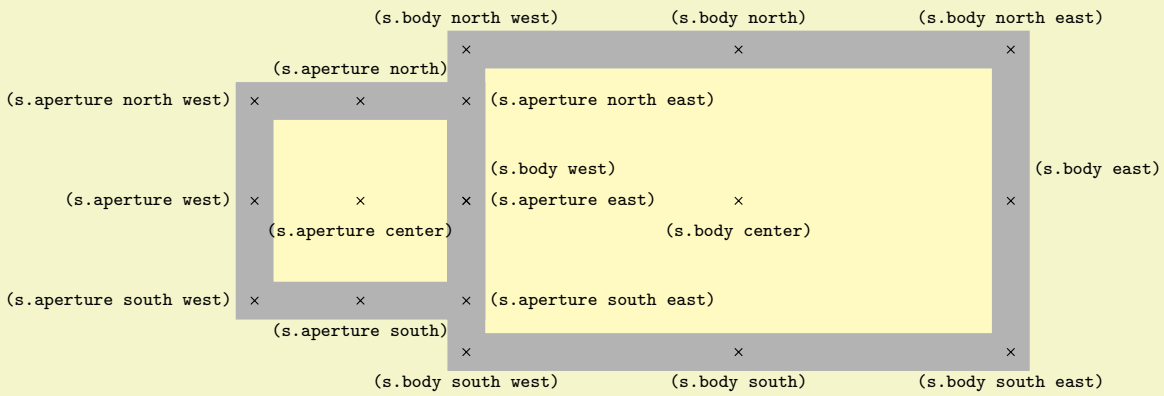
La figure suivante récapitule les ancres définies par `generic optics io`.




```
\Huge
\begin{tikzpicture}[use optics]
\node[generic optics io,name=s,io body height=4cm,io aperture width=0.7,io body aspect ratio=1.8,shape
example] {};
\foreach \anchor/\placement in
{body north/above,body south/below,body east/above right,body west/right,body center/below,
body north east/above,body north west/above,body south east/below,body south west/below,
aperture north/above,aperture south/below,aperture east/right,aperture west/left,aperture center/below,
aperture north east/right,aperture north west/left,aperture south east/right,aperture south west/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

Notons que les ancres `body east` et `aperture west` correspondent au même point. Les deux sont définies dans un souci de cohérence. Une ancre `east` est définie comme `body east` ou `aperture east` en fonction de la valeur de `io orientation` de manière à ce que `east` soit la clé la plus à l'est (à droite). La même chose a lieu pour l'ancre `west`.

La figure suivante récapitule les ancres définies par `generic optics io, io orientation=rtl`.

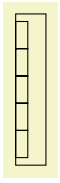


```
\Huge
\begin{tikzpicture}[use optics]
\node[generic optics io,name=s,io body height=4cm,io aperture width=0.7,io body aspect ratio=1.8,
io orientation=rtl,shape example] {};
\foreach \anchor/\placement in
{body north/above,body south/below,body east/above right,body west/above right,body center/below,
body north east/above,body north west/above,body south east/below,body south west/below,
aperture north/above,aperture south/below,aperture east/right,aperture west/left,aperture center/below,
aperture north east/right,aperture north west/left,aperture south east/right,aperture south west/left}
\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};
\end{tikzpicture}
```

3.3.2 Ligne de capteurs

Shape `sensor line`

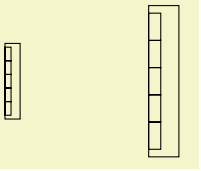
Dessine une ligne de capteurs (type Caliens).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line] (S) at (0,0) {};
\end{tikzpicture}
```

`/tikz/optics/sensor line height`=⟨*length*⟩ (no default, initially `2cm`)

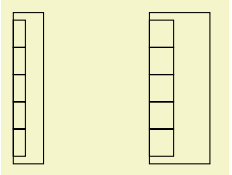
La clé `sensor line height` contrôle la taille de la ligne de capteurs. En particulier, ⟨*length*⟩ spécifie la hauteur du corps de la ligne, et les autres longueurs peuvent être spécifiées relativement à celle-ci (en ne donnant pas d'unité). Dans ce cas, en modifiant ⟨*length*⟩, on modifie donc la taille de la ligne de capteurs sans la déformer.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line, sensor line height=1cm] (L1) at (0,0) {};
  \node[sensor line, sensor line height=2cm] (L2) at (4cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/sensor line aspect ratio`=⟨*number*⟩ (no default, initially `0.2`)

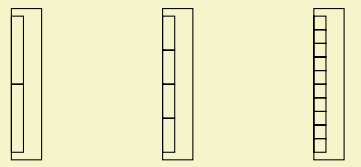
La clé `sensor line aspect ratio` contrôle le rapport d'aspect de la ligne de capteurs.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line, sensor line aspect ratio=0.2] (L1) at (0,0) {};
  \node[sensor line, sensor line aspect ratio=0.4] (L2) at (4cm,0) {};
\end{tikzpicture}
```

`/tikz/optics/sensor line pixel number`=⟨*number*⟩ (no default, initially `5`)

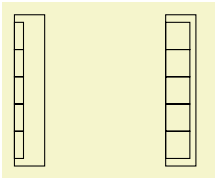
La clé `sensor line pixel number` contrôle le nombre de pixels ⟨*number*⟩ de la ligne de capteurs.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line, sensor line pixel number=2] at (0,0) {};
  \node[sensor line, sensor line pixel number=4] at (4cm,0) {};
  \node[sensor line, sensor line pixel number=10] at (8cm,0) {};
\end{tikzpicture}
```

/tikz/optics/sensor line pixel width=⟨*number or length*⟩ (no default, initially 0.4)

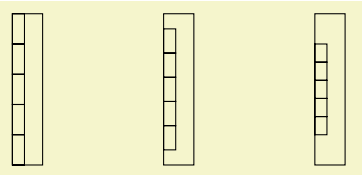
La clé sensor line pixel width contrôle la largeur des pixels : ⟨*number or length*⟩ est la largeur d'un pixel (par rapport à la largeur du capteur si elle est relative).



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line, sensor line pixel width=0.3] at (0,0) {};
  \node[sensor line, sensor line pixel width=0.8] at (4cm,0) {};
\end{tikzpicture}
```

/tikz/optics/sensor line inner ysep=⟨*number or length*⟩ (no default, initially 0.05)

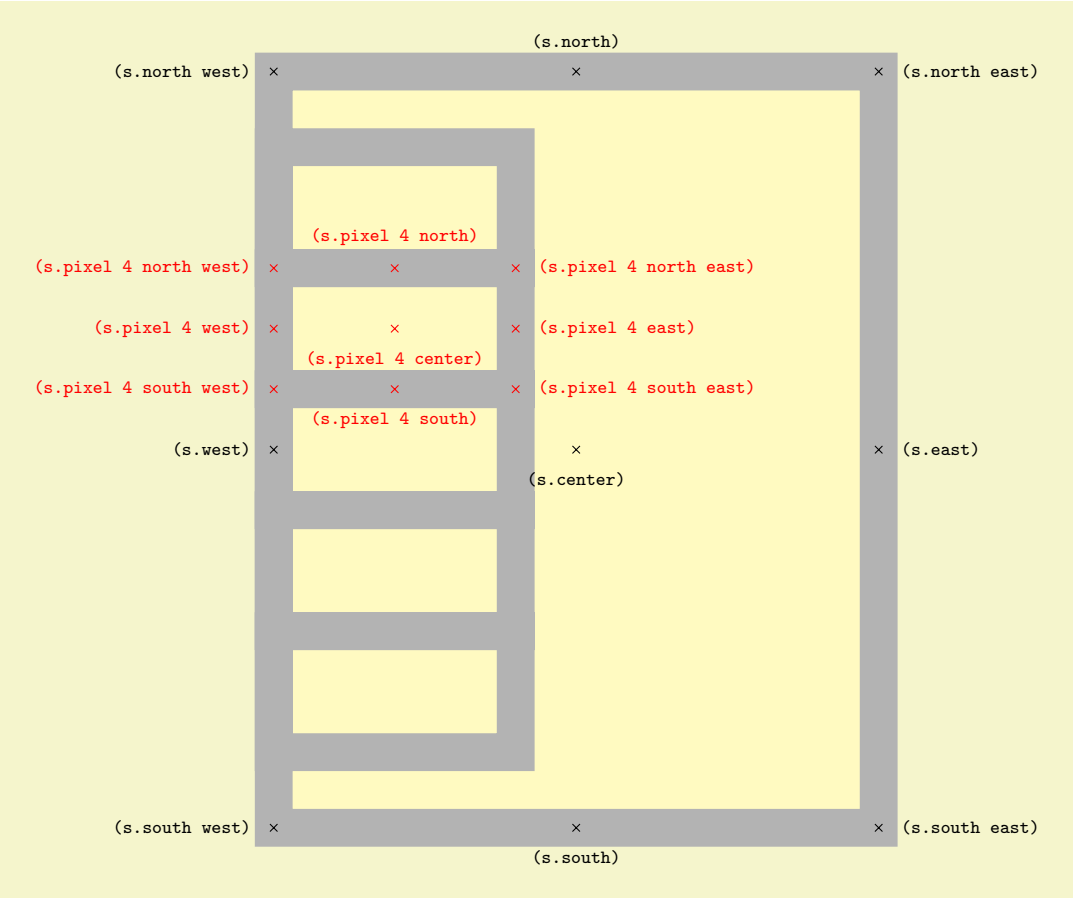
La clé sensor line inner ysep contrôle la séparation entre les bords haut et bas du capteur et les pixels. Si ⟨*number or length*⟩ est une hauteur relative, elle est comptée par rapport à la hauteur du capteur. La hauteur des pixels est calculée de manière à ce qu'il y en ait un nombre sensor line pixel number, en tenant compte de cette séparation.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[sensor line,sensor line inner ysep=0] at (0,0) {};
  \node[sensor line,sensor line inner ysep=0.1] at (4cm,0) {};
  \node[sensor line,sensor line inner ysep=0.2] at (8cm,0) {};
\end{tikzpicture}
```

Les ancres north, south, east, west, center, north east, north west, south east, south west, ainsi que pixel <i> <anchor> où <anchor> vaut north, south, etc. et où <i> est le numéro du pixel considéré (allant de 1 à sensor line pixel number) sont définies. Par exemple, on peut utiliser pixel 3 west.

La figure suivante récapitule les ancres définies par sensor line définies globalement ainsi que pour le pixel 4 (en rouge).




```
\Huge
\begin{tikzpicture}[use optics]
\node[sensor line,name=s,sensor line height=10cm,sensor line aspect ratio=0.8,
sensor line inner ysep=0.1,sensor line pixel number=5, shape example] {};
\foreach \anchor/\placement in
{north/above,south/below,east/right,west/left,center/below,
north east/right,north west/left,south east/right,south west/left}

\draw[shift=(s.\anchor)] plot[mark=x] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};

\foreach \anchor/\placement in
{pixel 4 north/above,pixel 4 south/below,pixel 4 east/right,pixel 4 west/left,pixel 4 center/below,
pixel 4 north east/right,pixel 4 north west/left,pixel 4 south east/right,pixel 4 south west/left}
% manque pixel <i> <subanchor>
\draw[red,shift=(s.\anchor)] plot[mark=x,red] coordinates{(0,0)}
node[\placement] {\scriptsize\texttt{(s.\anchor)}};

\end{tikzpicture}
```

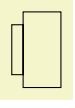
Remarques :

— Quand les ancres `east` et `west` ne sont pas définies explicitement, elles sont en fait des alias pour `center`. Cela permet aux clés `right=of...` et assimilées de fonctionner correctement.

3.3.3 Capteur générique

`/tikz/optics/generic sensor` (style, no value)

Le style `generic sensor` permet de dessiner un capteur. Les options de `generic optics io` sont applicables.



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic sensor] (S) at (0,0) {};
\end{tikzpicture}
```

3.3.4 Lampe générique

/tikz/optics/generic lamp (style, no value)

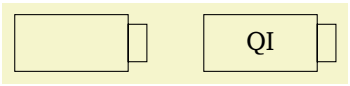
Le style generic lamp permet de dessiner une lampe. Les options de generic optics io sont applicables.

```
\begin{tikzpicture}[use optics,scale=.5]
  \node[generic lamp] (S) at (0,0) {};
\end{tikzpicture}
```

3.3.5 Lampe QI

/tikz/optics/halogen lamp (style, no value)

Le style halogen lamp permet de dessiner une lampe de type « lampe halogène Quartz Iode ». Les options de generic optics io sont applicables.

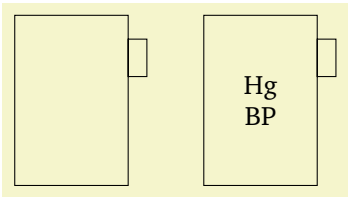



```
\begin{tikzpicture}[use optics,scale=.5]
  \node[halogen lamp] (S) at (0,0) {};
  \node[halogen lamp] (S) at (5cm,0) {QI};
\end{tikzpicture}
```

3.3.6 Lampe spectrale

/tikz/optics/spectral lamp (style, no value)

Le style spectral lamp permet de dessiner une lampe de type « lampe spectrale ». Les options de generic optics io sont applicables. Par ailleurs, un style est automatiquement appliqué pour autoriser le texte multiligne.




```
\begin{tikzpicture}[use optics,scale=.5]
  \node[spectral lamp] (S) at (0,0) {};
  \node[spectral lamp] (S) at (5cm,0) {\ce{Hg} \\ BP};
\end{tikzpicture}
```

3.3.7 Laser

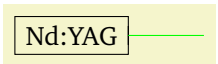
/tikz/optics/laser (style, no value)

Le style laser permet de dessiner un laser. Les options de generic optics io sont applicables.


```
\begin{tikzpicture}[use optics,scale=.5]
  \node[laser] (S) at (0,0) {};
  \node[laser] (S) at (5cm,0) {\ce{HeNe}};
\end{tikzpicture}
```

/tikz/optics/laser' (style, no value)

Le style laser' permet de dessiner un laser sans « orifice de sortie » (i.e. io aperture width=0pt). Les options de generic optics io sont applicables.




```
\begin{tikzpicture}[use optics,scale=.5]
  \node[laser'] (S) at (0,0) {\ce{Nd}:YAG};
  \draw[green] (S.aperture east) -- +(2cm,0);
\end{tikzpicture}
```

3.4 Divers

3.4.1 Marquer des rayons

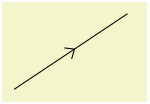
⚠ 2014-12-07 : cette partie a été substantiellement modifiée.

⚠ Cette partie n'est pas encore stable. Les raccourcis `->-`, etc. ne devraient normalement pas changer

Les flèches pour marquer les rayons sont conçues pour se trouver *vraiment* au milieu du rayon (contrairement à ce qui arrive en utilisant simplement `\arrow{>>}` avec `put arrow`).

`/tikz/optics/->-` (style, no value)

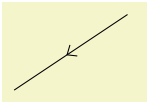
Ce style ajoute une flèche au milieu du chemin.



```
\begin{tikzpicture}[use optics]
  \draw[->-] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

`/tikz/optics/-<-` (style, no value)

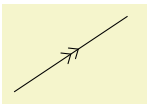
Ce style ajoute une flèche au milieu du chemin, dans le sens opposé à `->-`.



```
\begin{tikzpicture}[use optics]
  \draw[-<-] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

`/tikz/optics/->>-` (style, no value)

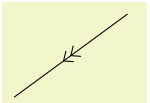
Ce style ajoute une double flèche au milieu du chemin.



```
\begin{tikzpicture}[use optics]
  \draw[->>-] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

`/tikz/optics/-<<-` (style, no value)

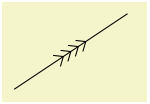
Ce style ajoute une double flèche au milieu du chemin, dans le sens opposé à `-<<-`.



```
\begin{tikzpicture}[use optics]
  \draw[-<<-] (0,0) -- (1.5cm,1.1cm);
\end{tikzpicture}
```

`/tikz/optics/->n-`={n=⟨*num*⟩, ⟨*specs*⟩} (style, no default)

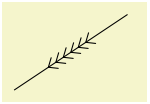
Ce style ajoute ⟨*num*⟩ flèches au milieu du chemin. Les spécifications ⟨*specs*⟩ sont appliquées aux flèches, suivant la syntaxe de `put arrow`.



```
\begin{tikzpicture}[use optics]
  \draw[->n-={n=4}] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

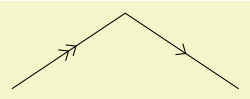
`/tikz/optics/-<n-`=n=⟨*num*⟩, ⟨*specs*⟩ (style, no default)

Ce style ajoute ⟨*num*⟩ flèches au milieu du chemin, dans le sens opposé à `->n-`.



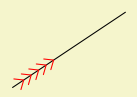
```
\begin{tikzpicture}[use optics]
  \draw[-<n-={n=6}] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

Des raccourcis allant de `->-` à `->>>>-` (idem pour `-<-`) sont disponibles. Les styles `->-`, `-<-`, etc. utilisent le style `put arrow` avec la flèche `multiple ray arrow`. Ils peuvent donc être configurés en suivant la syntaxe de `put arrow`, par exemple



```
\begin{tikzpicture}[use optics]
  \draw[->>-={at=0.25}, ->-={at=0.75}] (0,0) -- (1.5cm,1cm) -- (3cm, 0);
\end{tikzpicture}
```

Cela marche aussi avec `->n-` et `-<n-`.



```
\begin{tikzpicture}[use optics]
  \draw[->n-={n=5, at=0.2, style=red}] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

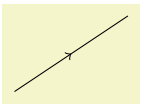
3.4.2 Placer des choses sur les chemins

⚠ 2014-12-07 : cette partie a été substantiellement modifiée.

⚠ Cette partie n'est pas encore stable. Tout risque de changer à tout moment.

`/tikz/put arrow` (no value)

La clé `put arrow` permet d'ajouter facilement une flèche sur un chemin.



```
\begin{tikzpicture}[use optics]
  \draw[put arrow] (0,0) -- (1.5cm,1cm);
\end{tikzpicture}
```

Par défaut, la flèche est `\arrow{>}` et elle est placée au milieu du chemin. Pour contrôler ces paramètres, il faut utiliser les sous-clés suivantes :

`/tikz/put arrow/pos`=⟨*pos*⟩ (no default, initially `0.5`)

Positionne la flèche à la position ⟨*pos*⟩ sur le chemin (par exemple, ⟨*pos*⟩=`0.5` place la flèche au milieu du chemin).

`/tikz/put arrow/at` (no value)

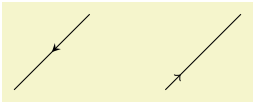
Alias pour `/tikz/put arrow/pos`.

`/tikz/put arrow/arrow`=⟨*arrow specification*⟩ (no default)

Utilise la flèche définie par ⟨*arrow specification*⟩ (par exemple `stealth` ou `latex`).

`/tikz/put arrow/arrow'`=⟨*arrow specification*⟩ (no default)

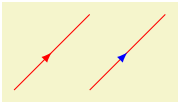
Utilise la flèche définie par ⟨*arrow specification*⟩ (par exemple `stealth` ou `latex`), mais avec une flèche inversée.



```
\begin{tikzpicture}[use optics]
  \draw[put arrow={arrow'=stealth}] (0,0) -- (1cm,1cm);
  \draw[put arrow={at=0.2}] (2cm,0) -- (3cm,1cm);
\end{tikzpicture}
```

`/tikz/put arrow/style`=⟨*style*⟩ (no default)

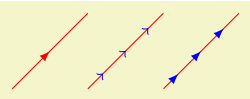
Le `\meta{style}` est passé à `\arrow[`⟨*style*⟩`]` pour dessiner les flèches.



```
\begin{tikzpicture}[use optics]
  \draw[red,put arrow={arrow=latex}]
  (0,0) -- (1cm,1cm);
  \draw[red,put arrow={arrow=latex,style={blue}}]
  (1cm,0) -- (2cm,1cm);
\end{tikzpicture}
```

`/tikz/put arrow/every arrow` (style, no value)

Il s'agit d'un style passé à toutes les flèches dessinées par `put arrow` (dans la portée du style). Il faut utiliser `every arrow/.style={`⟨*style*⟩`}` (ou `append style`, etc.).



```
\begin{tikzpicture}[use optics]
  \draw[red,put arrow={arrow=latex}]
  (0,0) -- (1cm,1cm);
  \draw[red, put arrow/every arrow/.style={blue},
    put arrow={at=0.2}, put arrow={at=0.5}, put arrow={at=0.8}]
  (1cm,0) -- (2cm,1cm);
  \draw[red, >=latex, put arrow/every arrow/.style={blue},
    put arrow={at=0.2}, put arrow={at=0.5}, put arrow={at=0.8}]
  (2cm,0) -- (3cm,1cm);
\end{tikzpicture}
```

Les différentes flèches utilisables sont détaillées dans le manuel de pgf/tikz.

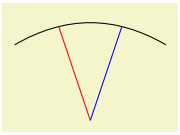
Pour un contrôle plus fin, il faut utiliser la bibliothèque tikz `markings`.

Par ailleurs, pour flécher des rayons lumineux, des styles spécifiques `->-`, `-<-`, `->>-` et `-<<-` ont été définis.

3.4.3 Placer des coordonnées sur les chemins

`/tikz/put coordinate`=⟨*coordinate*⟩ at ⟨*position*⟩ (no default)

Le style `put coordinate` crée une coordonnée nommée ⟨*coordinate*⟩ à l'abscisse curviligne ⟨*position*⟩ sur le chemin auquel est appliqué le style.

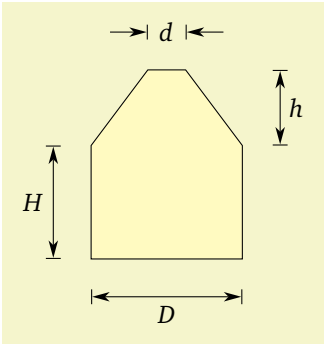


```
\begin{tikzpicture}[use optics]
  \draw[put coordinate=P at 0.3,put coordinate=Q at 0.7] (0,0)
    to[bend left] (2cm,0);
  \draw[red] (P) -- (1cm,-1cm);
  \draw[blue] (Q) -- (1cm,-1cm);
\end{tikzpicture}
```

3.4.4 Indiquer des dimensions sur les schémas

⚠ Cette partie est expérimentale et sujette à beaucoup de changements brusques sans préavis.

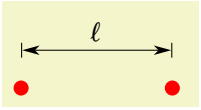
Commençons par un exemple.




```
\begin{tikzpicture}
\draw[fill=yellow!30]
   (-1cm,0) coordinate (A) -- (1cm,0) coordinate (B)
   -- (1cm,1.5cm) coordinate (B') -- (0.25cm,2.5cm) coordinate (b)
   -- (-0.25cm,2.5cm) coordinate (a) -- (-1cm,1.5cm) coordinate (A')
   -- cycle;

\draw (A) to[dim arrow'={label'=$D$}] (B);
\draw (A) to[dim arrow={label=$H$}] (A');
\draw (a) to[short dim arrow={label=$d$,label near middle}] (b);
\draw (B') to[dim arrow'={label'=$h$}] (b -| B');
\end{tikzpicture}
```

`/tikz/dim arrow`=⟨*sous-clés*⟩ (style, no default)

Le style `dim arrow` permet d'indiquer des dimensions sur les schémas. Il s'applique à un `to path`. Par exemple,




```
\begin{tikzpicture}[use optics]
  \node[circle,fill=red,inner sep=2pt] (a) at (0,0) {};
  \node[circle,fill=red,inner sep=2pt] (b) at (2cm,0) {};
  \draw (a.center) to[dim arrow={label=$\ell$}] (b.center);
\end{tikzpicture}
```

La flèche de dimension est décalée par rapport aux positions de départ et d'arrivée de manière à ne pas se superposer à l'objet dont on veut marquer la dimension.

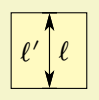
Plusieurs sous-clés permettent de spécifier des options en écrivant `dim arrow={`⟨*sous-clés*⟩`}`.

`/tikz/dim arrow/label=`⟨*text*⟩ (no default)

La clé `label` permet de spécifier le ⟨*texte*⟩ à afficher sur la flèche, qui est positionné à gauche de la flèche par rapport au sens du chemin (via `/tikz/auto=left`).

`/tikz/dim arrow/label'=`⟨*text*⟩ (no default)

La clé `label'` a le même rôle que `label`, mais le ⟨*texte*⟩ est affiché à droite dans le sens du chemin.



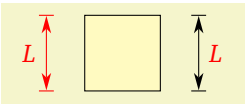

```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=1cm] (R) at (0,0) {};

  \draw (R.north east) to[dim arrow'={label'=$\ell'$}] (R.south east);
  \draw (R.north east) to[dim arrow'={label=$\ell$}] (R.south east);
\end{tikzpicture}
```

`/tikz/dim arrow/label text=`⟨*text*⟩ (no default)

La clé `label text` permet de spécifier le ⟨*texte*⟩ à afficher sur la flèche sans modifier son positionnement.

`/tikz/dim arrow/label style` (style, no value)

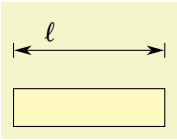
La clé `label style` permet de spécifier le style avec lequel le label doit être dessiné. À moins de le faire exprès, il n'est pas conseillé de remplacer ce style (qui est utilisé pour placer le label), mais plutôt d'y ajouter des spécifications grace à `/.append style`.




```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=1cm] (R) at (0,0) {};

  \draw (R.north east)
    to[dim arrow={label=$L$, label style/.append style=red}] (R.south east);
  \draw (R.north west)
    to[dim arrow'={label'=$L$}, red] (R.south west);
\end{tikzpicture}
```

`/tikz/dim arrow/label pos=`⟨*number*⟩ (no default, initially `0.5`)

La clé `label pos` permet de spécifier la position ⟨*number*⟩ à laquelle le label doit être dessiné. Cette clé ne s'applique pas à `short dim arrow`.




```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=2cm, minimum
height=0.5cm] (R) at (0,0) {};

  \draw (R.north west)
    to[dim arrow={label=$\ell$, label pos=0.25}] (R.north east);
\end{tikzpicture}
```

`/tikz/dim arrow/label near start` (no value)

Equivalent à `label pos=0` (pour `short dim arrow`, voir la documentation spécifique ci-dessous).

`/tikz/dim arrow/label near middle` (no value)

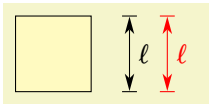
Equivalent à `label pos=0.5` (pour `short dim arrow`, voir la documentation spécifique ci-dessous).

`/tikz/dim arrow/label near end` (no value)

Equivalent à `label pos=1` (pour `short dim arrow`, voir la documentation spécifique ci-dessous).

/tikz/dim arrow/raise=⟨*length*⟩ (no default, initially 0.5cm)

La clé raise permet de spécifier la distance ⟨*length*⟩ à laquelle doit être dessinée la flèche de dimension par rapport au chemin initial.



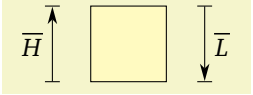

```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=1cm] (R) at (0,0) {};

  \draw (R.north east)
    to[dim arrow={label=$\ell$, raise=0.5cm}, black] (R.south east);
  \draw (R.north east)
    to[dim arrow={label=$\ell$, raise=1cm}, red] (R.south east);
\end{tikzpicture}
```

/tikz/dim arrow/no raise (style, no value)

Équivalent à raise=0. (Est-ce bien utile ?)

/tikz/dim arrow/-> (no value)

Quand la clé -> est utilisée, une flèche n'est dessinée que d'un côté (sélectionné de la même manière qu'avec la clé -> de tikz), par exemple pour marquer une grandeur algébrique.



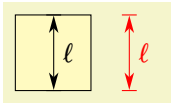

```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=1cm] (R) at (0,0) {};

  \draw (R.north east)
    to[dim arrow={->, label=$\overline{L}$, raise=0.5cm}] (R.south east);
  \draw (R.north west)
    to[dim arrow={<-, label'=$\overline{H}$, raise=-0.5cm}] (R.south west);
\end{tikzpicture}
```

/tikz/dim arrow/<- (no value)

Similaire à la clé /tikz/dim arrow/->, mais dans l'autre sens.

/tikz/dim arrow'=⟨*sous-clés*⟩ (style, no default)

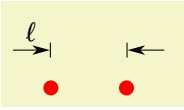
Le style dim arrow' a le même rôle que dim arrow, mais la valeur initiale de /tikz/dim arrow/raise est de -0.5cm au lieu de 0.5cm.




```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=1cm] (R) at (0,0) {};

  \draw (R.north east) to[dim arrow'={label=$\ell$}, black] (R.south east);
  \draw (R.north east) to[dim arrow={label=$\ell$}, red] (R.south east);
\end{tikzpicture}
```

/tikz/short dim arrow=⟨*sous-clés*⟩ (style, no default)

Le style short dim arrow a le même but que dim arrow, mais dans le cas où la dimension à marquer est petite, et où il faut que les flèches soient à l'extérieur.

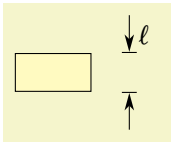



```
\begin{tikzpicture}[use optics]
  \node[circle,fill=red,inner sep=2pt] (a) at (0,0) {};
  \node[circle,fill=red,inner sep=2pt] (b) at (1cm,0) {};
  \draw (a.center)
  to[short dim arrow={label=$\ell$}]
  (b.center);
\end{tikzpicture}
```

Les options de /tikz/dim arrow s'appliquent, ainsi que quelques options supplémentaires.

/tikz/dim arrow/label near start (no value)

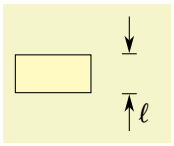
La clé label near start permet de placer l'étiquette donnant la dimension vers le début du chemin. C'est le comportement par défaut.



```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=0.5cm] (R) at (0,0) {};

  \draw (R.north east) to[short dim arrow={label=$\ell$}] (R.south east);
\end{tikzpicture}
```

`/tikz/dim arrow/label near end` (no value)

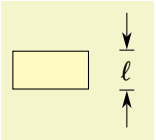
La clé `label near end` permet de placer l'étiquette donnant la dimension vers la fin du chemin.



```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=0.5cm] (R) at (0,0) {};

  \draw (R.north east)
      to[short dim arrow={label=$\ell$, label near end}] (R.south east);
\end{tikzpicture}
```

`/tikz/dim arrow/label near middle` (no value)

La clé `label near middle` permet de placer l'étiquette donnant la dimension vers le milieu du chemin. Dans ce cas, l'étiquette n'est pas décalée par rapport à la flèche (parce qu'il n'y a pas de superposition) ; cela peut être rétabli en spécifiant `/tikz/dim arrow/label style` explicitement.



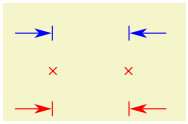
```
\begin{tikzpicture}
  \node[rectangle, draw=black, fill=yellow!30, minimum width=1cm, minimum
height=0.5cm] (R) at (0,0) {};

  \draw (R.north east)
      to[short dim arrow={label=$\ell$, label near middle}] (R.south east);
\end{tikzpicture}
```

`/tikz/dim arrow/arrow length`=⟨*length*⟩ (no default, initially `5mm`)

La clé `arrow length` permet spécifier la taille ⟨*length*⟩ des flèches extérieures.

`/tikz/short dim arrow'`=⟨*sous-clés*⟩ (style, no default)

Le style `short dim arrow'` a le même rôle que `short dim arrow`, mais la valeur initiale de `/tikz/dim arrow/raise` est de `-0.5cm` au lieu de `0.5cm`.

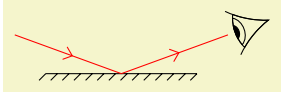


```
\begin{tikzpicture}[use optics]
  \coordinate (a) at (0,0);
  \coordinate (b) at (1cm,0);
  \draw[red,mark=x, draw=none] plot coordinates {(a) (b)};
  \draw (a.center) to[short dim arrow, blue] (b.center);
  \draw (a.center) to[short dim arrow', red] (b.center);
\end{tikzpicture}
```

3.4.5 Yeux

`/tikz/pics/optics eye` (style, no value)

La pic `optics eye` permet de dessiner un œil stylisé. Remarquez que `optics eye` n'est pas une `shape`, mais une `pic` (lire le manuel de pgf/tikz pour plus de détails).

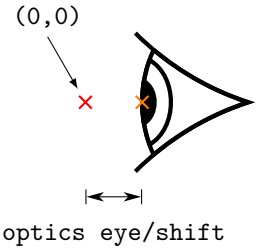


```
\begin{tikzpicture}[use optics]
  \node[mirror,rotate=-90] (M) at (0,0) {};

  \pgfmathsetmacro\rayangle{70}
  \draw[red,->-] ($(M.center)+({90+\rayangle}:1.5cm)$) -- (M.center);
  \draw[red,->-] (M.center) -- ($(M.center)+({90-\rayangle}:1.5cm)$)
    coordinate (out) {};
  \pic[rotate={90-\rayangle}] (eye) at (out) {optics eye};
\end{tikzpicture}
```

La manière dont l'oeil est dessiné peut-être modifiée en utilisant `optics eye={\meta{styles}}`. En particulier, les styles `pupil`, `cornea`, `iris` et `contour` peuvent être modifiés (par exemple avec `append style`).

```
\begin{tikzpicture}[use optics]
\pic (colorful eye) {optics eye={
cornea/.append style={red, fill=red!10},
iris/.append style={blue, fill=blue!10},
pupil/.append style={orange},
contour/.append style={green!60!black},
}};
\end{tikzpicture}
```

Lorsqu'une pic `optics eye` est ajoutée au point `(0,0)`, l'œil dessiné est déplacé de la longueur `optics eye/shift` de manière à ce que les rayons lumineux ne touchent pas l'œil. Cette distance peut être modifiée en utilisant `optics eye={shift=\meta{length}}`. Deux coordonnées `(\meta{name}-in)` et `(\meta{name}-center)` sont créées, où ⟨*name*⟩ est le nom de la pic. Par exemple `\pic (eye) at (0,0) {optics eye};` produira les coordonnées `(eye-in)` [à `(0,0)`] et `(eye-center)`. Ces points sont séparés de `optics eye/shift` et leurs positions sont expliquées dans la figure ci-dessous.



4 Remerciements

Je remercie les collègues et amis qui ont essuyé les plâtres en utilisant les premières versions de cette bibliothèque, et en particulier les auteurs du livre *Physique expérimentale* [15] pour lequel la bibliothèque a été développée.

15. http://www.physique-experimentale.com/