What is PGF/Ti*k*z?

PGF/Ti*k*z is a pair of languages for generating *vector graphics*. That is, graphics constructed from geometric primitives (points/vertices, line segments, polyhedra, parametric surfaces, etc.), with each primitive assigned a set of properties (colour, line weight, dash patterns, etc.). This is in contrast to *raster graphics*, in which each pixel of the image is stored (e.g. with colour, luminance, transparency values, etc.).

Because vector graphics consist of coordinates/lines/curves, the size of a representation on a screen is independent of the size of the object – you can zoom in to a vector image arbitrarily close and it will remain crisp.

Furthermore, the parameters of vector objects can be later modified, so that moving, scaling, rotating, filling, etc. does not degrade the quality of a vector image. In contrast, repeated modifications to a raster image will continually degrade the image quality as elements are snapped to a pixel grid between each edit.

Your First Ti*k*z Diagram

Every

Specifying Points

There are many ways to specify *points* or *coordinates* in Ti*k*z. You can declare your coordinate system explicitly (this will be more relevant later) with the syntax

```
([coordinate system] cs: (system-specific coordinates))
```

but for various common coordinate systems, special implicit syntax is available.

For instance, cartesian coordinate may be specified explicitly using the `canvas` coordinate system:

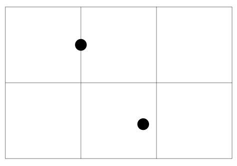
```
(canvas cs:x=2ex,y=5pt)
```

which takes two keys, `x` and `y`, that accept TEX dimensions. Or, they can be specified implicitly by listing the two dimensions separated by commas in round brackets, as in

```
(2ex,5pt)
```

This coordinate means "2ex upwards and 5pt to the right of the origin".

Note that you can also write things like `1em+2cm` in a dimension, since the maths engine is used to evaluate the coordinates.



```
\begin{tikzpicture}
    \draw[help lines] (0,0) grid (3,2);

    \filldraw (1cm,1.5cm)   circle (2pt);
    \filldraw (2cm-5pt,3ex) circle (2pt);
\end{tikzpicture}
```

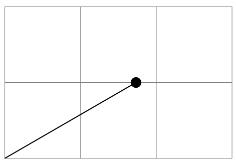
Polar coordinates may be specified using the `canvas polar` system

```
(canvas polar cs:angle=30,radius=2cm)
```

or implicitly by providing the angle and a radius separated by a colon:

```
(30:2cm)
```

This coordinate means "2cm from the origin, 30 degrees counterclockwise from the positive x-axis".



```
\begin{tikzpicture}
    \draw[help lines] (0,0) grid (3,2);
    \draw (0,0) -- (30:2cm);

    \fill (30:2cm) circle (2pt);
\end{tikzpicture}
```

There are other coordinate systems, but we'll only discuss those on request. From now on, we'll also omit the explicit system declaration unless it is otherwise relevant.

If units are not provided, e.g. `(1,2)`, then the coordinates are specified in PGF's internal `xy`-coordinate system. By default, the unit x-vector points 1cm to the right, and the unit y-vector points 1cm upwards.*

By giving three numbers, as in `(1,2,3)`, the coordinates are specified in PGF's internal xyz-coordinate system. By default, the unit x-vector points 1cm to the right, and the unit y-vector points 1cm upwards, and the unit z-vector points to `(-3.85mm,-3.85mm)`.

It is also possible to use an anchor (explained later) of an existing shape as in `(my\_node.centre)` as a coordinate.

You can add two plus signs before a coordinate to specify a coordinate relative to the previously defined coordinate. For example, `(1,0) ++(1,0) ++(0,1)` specifies the three coordinates `(1,0)`, then `(2,0)` [=(1,0)+(1,0)], and `(2,1)` [= (2,0) + (0,1)].

Instead of two plus signs, you can also add a single one. This also specifies a point in a relative manner, but doesn't update the reference coordinate. For example, `(1,0) +(1,0) +(0,1)` specifies the three coordinates `(1,0)`, then `(2,0)` [= (1,0) + (1,0)], and `(1,1)` [= (1,0) + (0,1)].

Paths

A *path* is a series of curves between points, which need not be connected.

To specify a straight line path between two coordinates, put two dashes between the coordinates. For instance,

```
(0,0) -- (0,1) -- (1,1) -- cycle
```

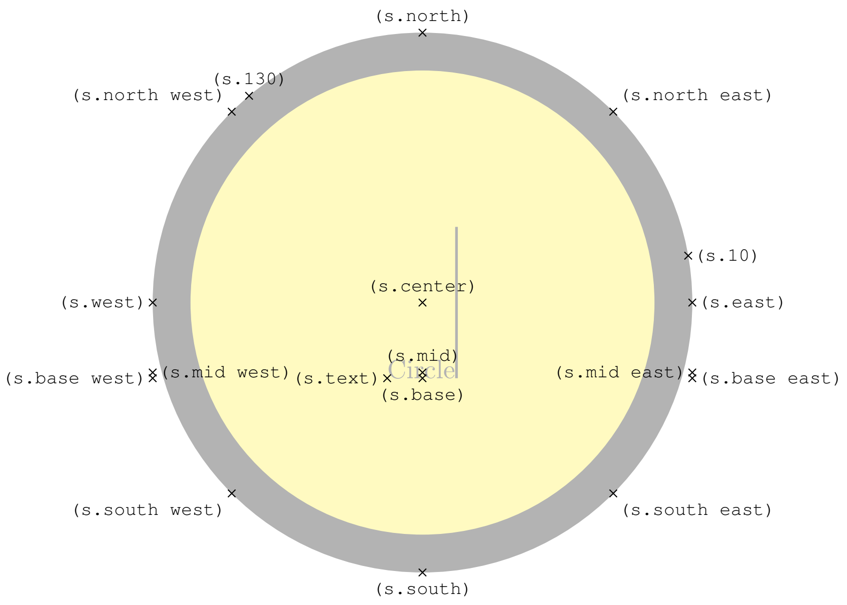
specifies a trianglar path, with the special `cycle` coordinate being defined to be the first point in the path (e.g. `(0,0)` in this case).

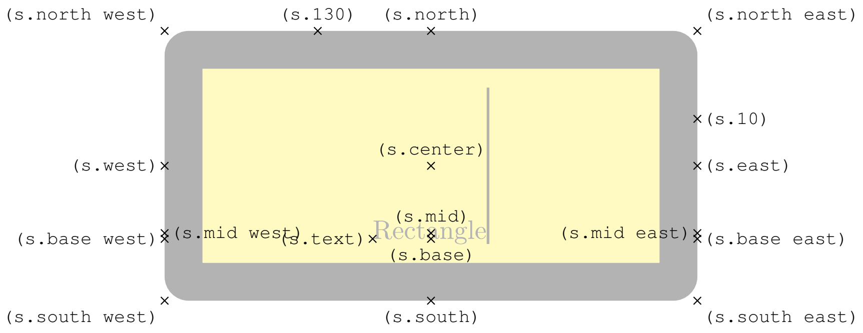
*It is possible to use coordinates like `(1,2cm)`, which is neither an `xy`-coordinate nor a `canvas` coordinate. Roughly speaking, if you have mixed coordinates, Ti*k*z will typecast the coordinate `(X,Ycm)` to the sum `(X,0)+(0pt,Y)`, and similarly for any mixed dimensions. In particular, `(2+3cm,0)` is *not* the same as `(2cm+3cm,0)`, as it will be typecast to sum `(2pt,0) + (3cm,0) = (2pt+3cm,0)`.

Nodes

The Calc Library

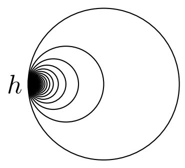
Anchors

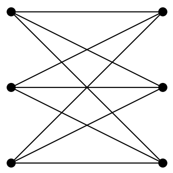




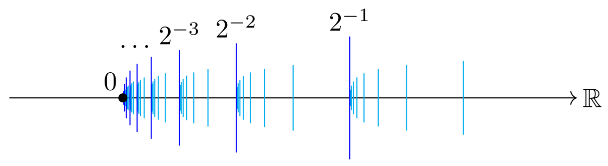
Samples

Iteration

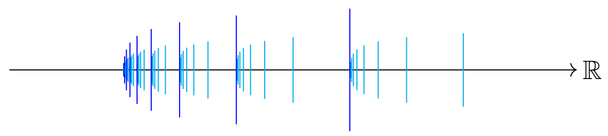




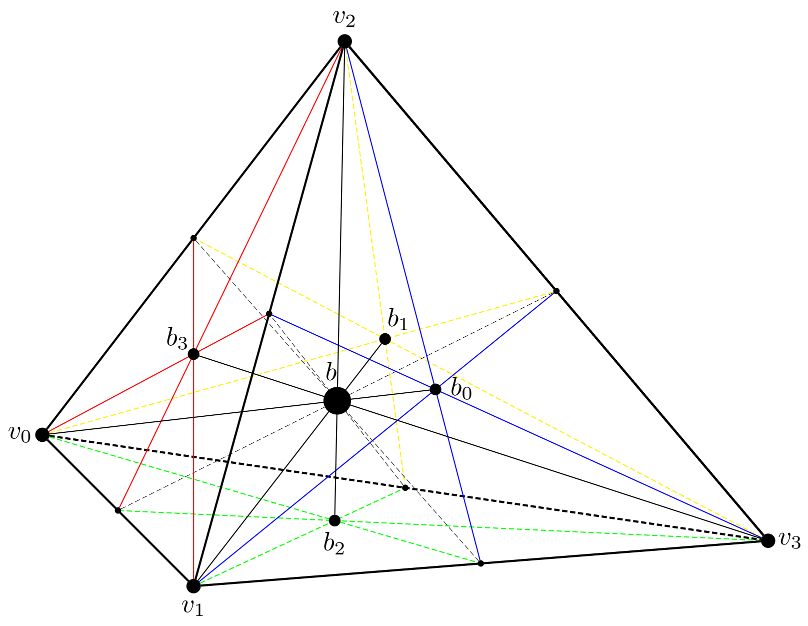
$$Z = \{0\} \cup \{2^{-n} : n \in \mathbb{Z}^+\} \cup \{2^{-n} + 2^{-n-m} : n,m \in \mathbb{Z}^+\}$$



Simpler version without if/else clauses for labels:



Calc Library


v_2
b_1
b_3
b
b_0
v_0
b_2
v_3
v_1