



TikZ in $\text{\LaTeX}$

Prof. Pranay Kumar Saha

April 9, 2025





What is TikZ?

"TikZ ist kein Zeichenprogramm"

- TikZ (along with its backend PGF) is a powerful package for creating vector graphics directly within $\text{\LaTeX}$.
- It allows you to describe graphics programmatically using $\text{\LaTeX}$ commands.
- Often used for diagrams, graphs (though `pgfplots` is often easier for data plots), illustrations, and annotating images.



Why TikZ?

"TikZ ist kein Zeichenprogramm"

Why Use TikZ?

- **Consistency:** Graphics use the same fonts and styles as your document.
- **Quality:** Produces high-quality, scalable vector graphics (PDF).
- **Integration:** Code lives with your text; easy integration of math and labels.
- **Power & Control:** Extremely flexible and customizable for complex diagrams.
- **Code-Based:** Graphics are described logically, can be version controlled.

The name is a recursive acronym: "TikZ ist kein Zeichenprogramm" (TikZ is not a drawing program).



Loading the Package

Basic Setup

Using TikZ requires loading the package in your preamble.

Preamble Code

```
\usepackage{tikz}

% Optionally load libraries for extra features:
\usetikzlibrary{shapes, arrows.meta, positioning, calc}
% Example libraries
```

- `\usepackage{tikz}` loads the core TikZ/PGF system.
- Many advanced features (specific shapes, arrowheads, relative positioning tools) are in **libraries** that need to be loaded explicitly using `\usetikzlibrary{...}`. The `pgfplots` package loads TikZ automatically.



The tikzpicture Environment

The Drawing Canvas

All TikZ graphics are created inside a `tikzpicture` environment.

Basic Structure

```
\begin{tikzpicture}[optional settings]
  % TikZ drawing commands go here...
\end{tikzpicture}
```

- This environment creates a "picture" region within your document.
- You can provide default options for the whole picture in the optional argument [...].
- Everything inside is drawn relative to a coordinate system.



The tikzpicture Environment

The Drawing Canvas

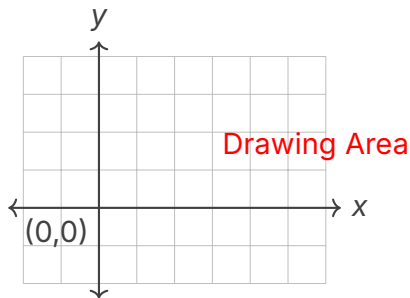


Figure: Conceptual TikZ coordinate system



Coordinates and Paths

Specifying Points and Drawing Lines

Coordinates

Specified as (x, y) , e.g., $(0, 0)$, $(2\text{cm}, 1\text{cm})$, $(3, -1)$. Units default to cm unless specified (pt, mm, in).

The `\draw` Command

The fundamental command for drawing paths (lines, curves, shapes).

```
\draw [options] (point1) <path spec> (point2) ... ;
```



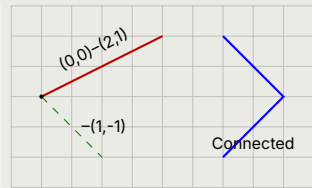
Coordinates and Paths

Specifying Points and Drawing Lines

Code

```
1 \begin{tikzpicture}
2   % Draw a single line segment
3   \draw (0,0) -- (2,1);
4
5   % Draw multiple connected segments
6   \draw[blue] (3,1) -- (4,0) -- (3,-1);
7
8   % Draw starting from origin implicitly
9   % \draw -- (1,-1); % equivalent to (0,0)--(1,-1)
10 \end{tikzpicture}
```

Output



Path Specifiers:

- `--` : Straight line segment.



More Path Operations

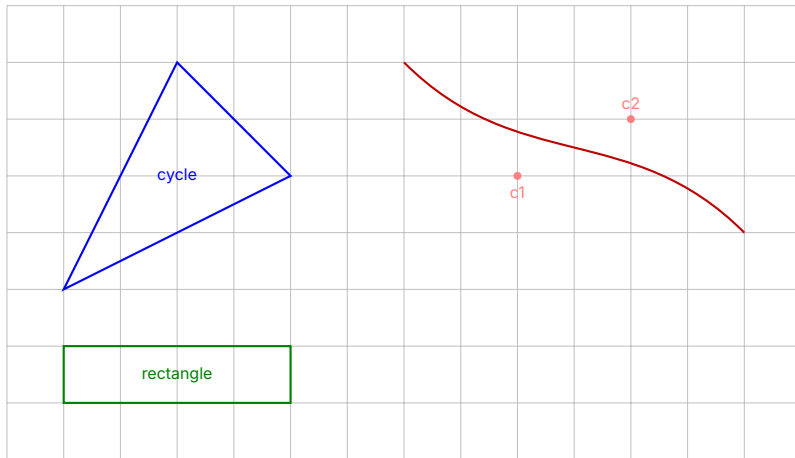
Curves, Closing Paths

Code

```
1 \begin{tikzpicture}
2   % 1. Closed path (triangle)
3   \draw[blue] (0,0) -- (2,1) -- (1,2) -- cycle;
4
5   % 2. Simple Bezier curve
6   \draw[red, thick] (3,2) .. controls (4,1) and (5,1.5) .. (6,0.5);
7   % Format: (start) .. controls (c1) and (c2) .. (end)
8
9   % 3. Rectangle path operation
10  \draw[green!50!black] (0,-1) rectangle (2,-0.5);
11 \end{tikzpicture}
```



Output





More Path Specifiers:

- `-- cycle` : Draws a line back to the start of the current path section.
- `.. controls (c1) and (c2) ..` : Bezier curve.
- `rectangle` : Draws rectangle between two corner points.
- Many others: `arc`, `circle`, `ellipse`, `grid`, `parabola`...



Nodes: Adding Text

Placing Text Labels

Code

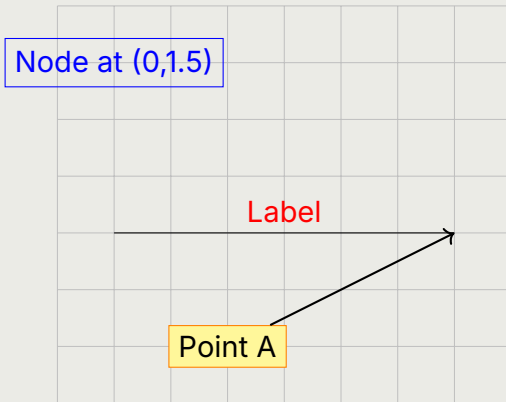
```
1 \begin{tikzpicture}
2   % 1. Node at a coordinate
3   \draw (0,1.5) node[draw=blue] {Node at (0,1.5)};
4
5   % 2. Node along a path
6   \draw (0,0) -- (3,0) node[midway, above, red] {Label};
7
8   % 3. Named node for later reference
9   \node (A) at (1,-1) [fill=yellow!50] {Point A};
10  % Can now use (A) as a coordinate
11  \draw[->, thick] (A) -- (3,0);
12 \end{tikzpicture}
```



Nodes: Adding Text

Placing Text Labels

Output





Nodes: Adding Text

Placing Text Labels

Node Syntax:

- `\node (name) at (coord) [options] {text};` (General form)
- `\draw (coord) node[options] {text};` (Draws only the node)
- Place `node[options] {text}` after a coordinate or path specifier (like `--`) to put it on the path.
- Options control position (`above`, `below`, `left`, `right`, `midway`), appearance (`draw`, `fill`, `shape=circle`, `rounded corners`), etc.
- Naming nodes (`name`) is very useful for connecting elements.



Drawing Shapes

Predefined Geometric Shapes

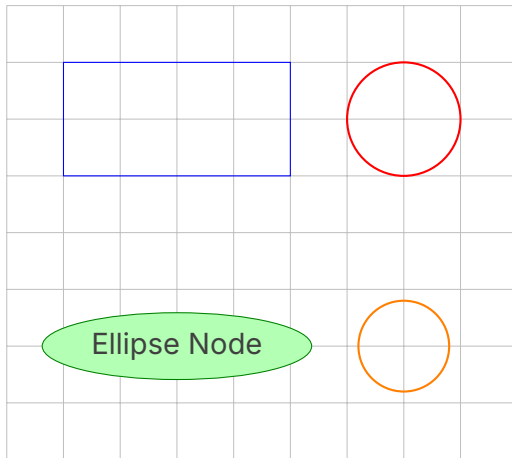
TikZ provides commands and path operations for common shapes.

```
1 \begin{tikzpicture}
2   % 1. Rectangle path operation
3   \draw[blue] (0,1) rectangle (2,2);
4
5   % 2. Circle path operation
6   \draw[red, thick] (3,1.5) circle (0.5cm); % Needs radius
7
8   % 3. Using node shapes (needs shapes library)
9   % \usetikzlibrary{shapes.geometric} % Needed! (Added in preamble)
10  \node[draw=green!50!black, shape=ellipse, fill=green!30] at (1,-0.5) {Ellipse Node};
11
12  % 4. Alternative circle syntax
13  \draw[orange] (3,-0.5) circle [radius=0.4cm];
14 \end{tikzpicture}
```



Drawing Shapes

Predefined Geometric Shapes





Drawing Shapes

Predefined Geometric Shapes

- Path operations: `rectangle`, `circle`, `ellipse`, `arc`. Syntax varies (corners vs center/radius).
- Node shapes: Set the `shape=...` option for a `\node` (e.g., `circle`, `ellipse`, `rectangle`, `star` - needs `shapes.geometric` library). The node text goes inside the shape.



Styling Paths

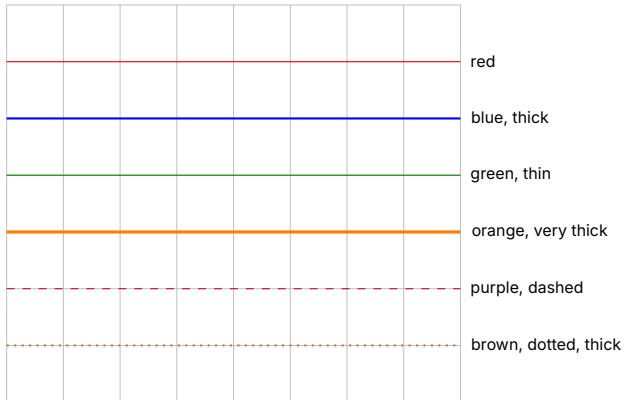
Colors, Thickness, Line Styles

```
1 \begin{tikzpicture}
2   \draw[help lines] (0,-1) grid (4,2);
3
4   \draw [red] (0,1.5) -- (4,1.5);
5   \draw [blue, thick] (0,1) -- (4,1);
6   \draw [green!50!black, thin] (0,0.5) -- (4,0.5);
7   \draw [orange, very thick] (0,0) -- (4,0);
8   \draw [purple, dashed] (0,-0.5) -- (4,-0.5);
9   \draw [brown, dotted, thick] (0,-1) -- (4,-1);
10 \end{tikzpicture}
```



Styling Paths

Colors, Thickness, Line Styles





Styling Paths

Colors, Thickness, Line Styles

- Colors: red, blue, green, black, gray, white, custom colors (color=mycolor, requires xcolor package loaded), mixing (red!50!blue).
- Line Width: thin, thick, very thick, ultra thick, line width=<dimen>.
- Line Style: solid (default), dashed, dotted, dashdotted.
- Arrows: ->, <-, <->, -stealth, latex- etc. (many require arrows.meta library).



Styling Nodes and Shapes

Fill, Border, Text Colors

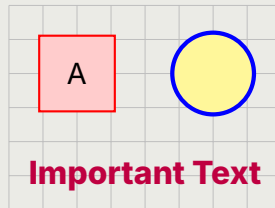
Code

```
\begin{tikzpicture}
% Node with fill and border color
\node [draw=red, thick, fill=red!20,
       minimum size=1cm] at (0,0) {A};

% Shape with different draw/fill
\draw [draw=blue, ultra thick, fill=yellow!50]
      (2,0) circle (0.6cm);

% Node with text color
\node [text=purple, font=\large\bfseries]
      at (1,-1.5) {Important Text};
\end{tikzpicture}
```

Output





Styling Nodes and Shapes

Fill, Border, Text Colors

Common Node/Shape Options:

- `draw=<color>`: Color of the border line (omit for no border). Can add thickness etc.
- `fill=<color>`: Background fill color (omit for transparent).
- `text=<color>`: Color of the text inside the node.
- `font=<cmds>`: $\text{\LaTeX}$ font commands (`\itshape`, `\bfseries`, `\small`).
- Sizing: `minimum width=`, `minimum height=`, `inner sep=` (padding).
- Shape: `circle`, `rectangle`, `rounded corners`, `ellipse`, etc. (Use `shape=ellipse`, etc.)



Scopes: Localizing Options

Applying Styles to Groups

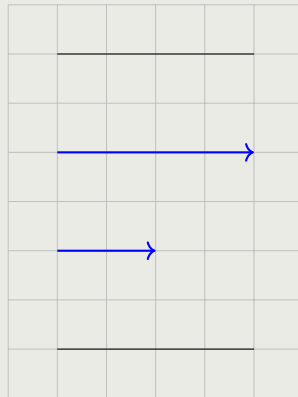
Code

```
\begin{tikzpicture}
  % Default drawing style
  \draw (0,1) -- (2,1);

  \begin{scope}[draw=blue, thick, ->]
    % These inherit blue, thick, arrow tip
    \draw (0,0) -- (2,0);
    \draw (0,-1) -- (1,-1);
  \end{scope} % Options end here

  % Back to default style
  \draw (0,-2) -- (2,-2);
\end{tikzpicture}
```

Output





Scopes: Localizing Options

Applying Styles to Groups

Purpose of `scope`:

- Group related elements.
- Apply common options without repeating them.
- Avoid options "leaking" to later parts of the picture.
- Useful for transformations (shifting, scaling, rotating parts) - advanced.



Defining Reusable Styles

Using `\tikzset` in the Preamble

Repeating options like `[draw=blue, thick, fill=blue!10]` is tedious and error-prone. Define named styles in the preamble for consistency.

The `\tikzset` Command

Preferred method for defining styles globally (in preamble).

```
\tikzset{
  style name 1/.style = { option1=value, option2, ... },
  style name 2/.style = { style name 1, option3, ... },
  another name/.style = { ... }
}
```



Defining Reusable Styles

Using `\tikzset` in the Preamble

The `\tikzset` Command

- `style name`: The name you will use later (e.g., `my node`).
- `/.style = {...}`: Defines a style.
- `options`: Any valid TikZ options (e.g., `draw=red`, `shape=circle`, `thick`).
- Styles can inherit from other styles (e.g., `style name 1` used in `style name 2`).



Defining Reusable Styles

```
\tikzset{
  basic node/.style={draw=gray, minimum height=1.5em, font=\small
  },
  input node/.style={ basic node, shape=rectangle, fill=blue!10,
    draw=blue!80!black, thick
  },
  process node/.style={ basic node, shape=rectangle, rounded corners,
    fill=green!10, draw=green!50!black, thick
  },
  decision node/.style={ basic node, shape=diamond, fill=orange!20,
    draw=orange!80!black, thick, aspect=2 % Make diamond wider
  },
  my arrow/.style={-{Stealth[length=2mm]}} % Needs arrows.meta library
```



Defining Reusable Styles

Example Usage (using styles defined in preamble)

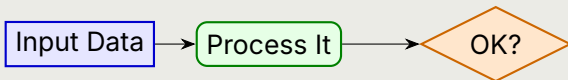
```
\begin{tikzpicture}
  \node [input node] (in) at (0,1) {Input Data};
  \node [process node] (proc) at (2,1) {Process It};
  \node [decision node] (dec) at (4,1) {OK?};

  \draw [my arrow] (in) -- (proc);
  \draw [my arrow] (proc) -- (dec);
\end{tikzpicture}
```



Defining Reusable Styles

Output (using preamble styles)



Note: `\tikzstyle` is an older, less flexible alternative. Use `\tikzset`.



Why Avoid Absolute Coordinates?

The Problem with Hardcoding (x, y)

While specifying every point like $(2.5, 3.1)$ works, it becomes difficult for complex diagrams:

- **Tedious:** Calculating exact positions is time-consuming.
- **Hard to Modify:** If you move one element, you might need to recalculate and change coordinates for many others.
- **Poor Readability:** Code filled with numbers doesn't clearly express the *relationships* between elements (e.g., "A is above B").

The Solution: Relative Positioning

TikZ offers several ways to place elements relative to each other, making diagrams:

- Easier to create and understand.
- Much easier to modify and maintain.
- More robust if content changes slightly.



1. Named Coordinates and Nodes

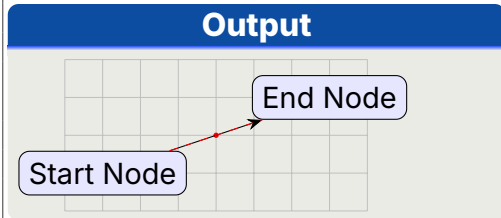
The Foundation for Relative Drawing

Give points or nodes names, then refer to those names later.

```
\begin{tikzpicture}
  % Define Node A at (0,0)
  \node[my node] (A) at (0,0) {Start Node};

  % Define Node B at (3,1)
  \node[my node] (B) at (3,1) {End Node};

  % Define a coordinate (just a point)
  \coordinate (MidHelper) at (1.5, 0.5);
  % Draw using names
  \draw[my arrow] (A) -- (B);
  \draw[red, dashed] (A) -- (MidHelper) -- (B);
\end{tikzpicture}
```





1. Named Coordinates and Nodes

The Foundation for Relative Drawing

Key Points:

- `\node (<name>) ... ;` Defines a node with content/shape.
- `\coordinate (<name>) at (pos);` Defines a named point (no visible content).
- Use `(<name>)` as a coordinate in subsequent path commands.



2. Relative Coordinates ('++')

Steps from the Current Position

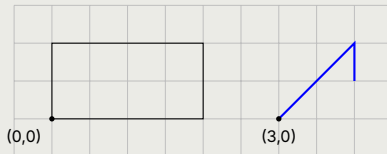
Specify a coordinate as an offset from the *previous* coordinate on the path using ++(dx,dy).

Code

```
\begin{tikzpicture}
% Draw a 2x1 rectangle starting at (0,0)
\draw (0,0) -- ++(2,0)    % Move 2 right
      -- ++(0,1)         % Move 1 up
      -- ++(-2,0)        % Move 2 left
      -- cycle;          % Close path to start

% Draw a separate path starting from (3,0)
\draw[blue, thick] (3,0) -- ++(1,1) -- ++(0,-0.5);
\end{tikzpicture}
```

Output





2. Relative Coordinates ('++')

Steps from the Current Position

- $++(dx, dy)$ adds dx, dy to the last point reached.
- Useful for drawing sequences or grids where relative steps are known.
- Can still be tedious if the overall structure changes.
- Single $+(dx, dy)$ moves relative but doesn't update the "last point reached" for subsequent $++$.

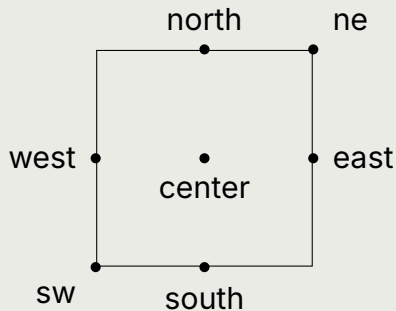


3. Node Anchors

Connecting to Specific Points on Nodes

Nodes have named "anchors" on their borders and center. Use (`<node name>.<anchor>`) to connect precisely.

Common Anchors





3. Node Anchors

Connecting to Specific Points on Nodes

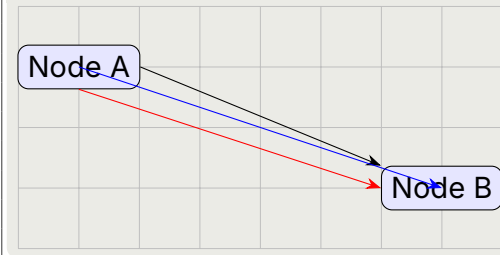
Nodes have named "anchors" on their borders and center. Use (`<node name>.<anchor>`) to connect precisely.

Code

```
\begin{tikzpicture}
  \node[my node] (A) at (0,1) {Node A};
  \node[my node] (B) at (3,0) {Node B};

  % Connect specific anchors
  \draw[my arrow] (A.east) -- (B.north west);
  \draw[my arrow, red] (A.south) -- (B.west);
  \draw[my arrow, blue] (A.center) -- (B.center);
\end{tikzpicture}
```

Output





3. Node Anchors

Connecting to Specific Points on Nodes

Key Points:

- Syntax: (`<node name>.<anchor>`).
- Ensures lines connect to the border/center cleanly, regardless of node size/content.
- Essential for flowcharts, diagrams where connections matter.



4. The 'positioning' Library

High-Level Relative Placement

This library provides a very intuitive syntax for placing nodes relative to *other named nodes*.

Loading the Library

Add to your preamble:

```
\usetikzlibrary{positioning}
```



4. The 'positioning' Library

High-Level Relative Placement

Syntax (used as node options)

`relation = [<distance>] of <other node name>`

- **Relations:** above, below, left, right, above left, below right, etc.
- **Distance:** Optional distance between node borders (default depends on node distance). E.g., 1cm, 5mm.
- You can combine: below = 1cm of NodeA, right = of NodeB.
- Compound: below left = <dy> and <dx> of <NodeC>.

This is often the preferred method for laying out block diagrams, flowcharts, etc.

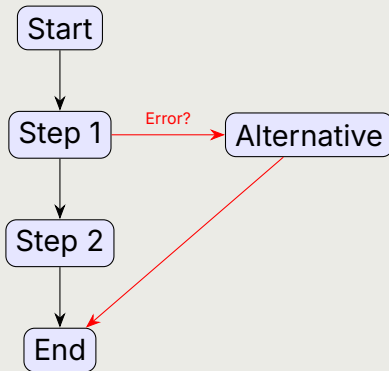


'positioning' Library Example

Creating a Simple Flowchart

```
\begin{tikzpicture}[
  node distance=0.8cm, % Default distance
  auto % Place nodes on paths automatically]
% Place nodes relative to each other
\node[my node] (start) {Start};
\node[my node] (step1) [below = of start] {Step 1};
\node[my node] (step2) [below = of step1] {Step 2};
\node[my node] (end) [below = of step2] {End};
\node[my node] (alt) [right = 1.5cm of step1] {Alternative};
% Draw connections (using node names)
\draw[my arrow] (start) -- (step1);
\draw[my arrow] (step1) -- (step2);
\draw[my arrow] (step2) -- (end);
% Label on path
\draw[my arrow, red] (step1) -- node {Error?} (alt);
\draw[my arrow, red] (alt) -- (end);
\end{tikzpicture}
```

Output





'positioning' Library Example

Creating a Simple Flowchart

Key Points:

- Define first node absolutely (or implicitly at $(0,0)$).
- Define subsequent nodes using `relation=of other`.
- Connect using node names (TikZ automatically uses appropriate anchors).



TikZ in L^AT_EX

Thank You for Listening!