

 已深度思考 (用时 11.4 秒)



好的，我们来逐行/逐段分析这段使用 TikZ 绘图的 LaTeX 代码。这段代码定义了两种绘制规则多边形层级结构的方法，并使用一个矩阵来展示不同参数下的绘图结果。

代码分析

1. 文档类和 TikZ 设置

<LATEX>

```
1 % https://tex.stackexchange.com/a/731431/322482
2 \documentclass[tikz]{standalone}
```

- `% ...` : 这是一个注释行，提供了代码的来源链接。
- `\documentclass[tikz]{standalone}` :
 - `standalone` : 这是一个特殊的文档类，它会生成一个紧密包裹内容的 PDF 文件，大小与内容完全一致，没有多余的页边距。非常适合用来创建独立的图形。
 - `[tikz]` : 这是给 `standalone` 文档类的一个选项，它会自动加载 `tikz` 宏包，并设置好环境，让我们能直接开始绘图。

<LATEX>

```
1 \tikzset{
2   @every rp/.code=\edef\rpSides{\pgfinteval{\pgfkeysvalueof{/tikz/rp sides}-2}}\pgfmathsetmacro\rpAngle{360/(\pgfkeysvalueof{/tikz/rp sides})},
3   @every rp/.append style={local bounding box=rpbox, x=\pgfkeysvalueof{/tikz/rp side length}, y=\pgfkeysvalueof{/tikz/rp side length}}},
```

- `\tikzset{ ... }` : 这个命令用于定义一系列 TikZ 的样式 (styles) 和键 (keys)，方便后续重复使用。
- `@every rp/.code= ...` :
 - 定义了一个名为 `@every rp` 的内部样式。`.code` 表示当这个样式被调用时，会执行一段代码。
 - `\edef\rpSides{ ... }` : 定义一个宏 `\rpSides`。它的值是 `多边形边数 - 2`。`\pgfkeysvalueof{/tikz/rp sides}` 获取 `rp sides` 键的值 (稍后会定义)，`\pgfinteval` 用来做整数运算。这个 `\rpSides` 变量在 `\foreach` 循环中用来控制循环次数。
 - `\pgfmathsetmacro\rpAngle{ ... }` : 定义一个宏 `\rpAngle`。它的值是 `360 / 多边形边数`，也就是每个顶点之间的夹角。
- `@every rp/.append style={ ... }` :
 - 给 `@every rp` 样式追加更多的设置。
 - `local bounding box=rpbox` : 为当前路径创建一个局部的边界框，并命名为 `rpbox`。这使得我们可以通过 `rpbox.south` 等方式引用图形的边界，用于放置图例。
 - `x= ... , y= ...` : 设置 TikZ 绘图的局部坐标系。这里将 `x` 和 `y` 向量的长度都设置为 `rp side length` 键的值。这使得在极坐标系中，半径为 1 的单位长度等于 `rp side length` 的值。

<LATEX>

```
1   rp sides/.initial=5, rp side length/.initial=+7mm,
```

- `rp sides/.initial=5` : 定义一个名为 `rp sides` 的键，并设置其初始值为 5。这个键用来控制多边形的边数。
- `rp side length/.initial=+7mm` : 定义一个名为 `rp side length` 的键，并设置其初始值为 7mm。这个键用来控制图形的尺寸 (具体来说是每一层的半径增量)。

<LATEX>

```
1   rp node/.style={shape=circle, draw, inner sep=+0pt, minimum size=+1.3ex, rp node \pgfintmod{#1}{2}},
2   rp node 0/.code=, rp node 1/.style=fill,
```

- `rp node/.style={ ... }` : 定义了节点 (圆点) 的基础样式 `rp node`。它接受一个参数 `#1`。
 - `shape=circle, draw` : 节点是圆形的，并且有边框。
 - `inner sep=+0pt, minimum size=+1.3ex` : 内部间距为 0，最小尺寸为 `1.3ex` (`ex` 是一个与当前字体相关的单位，约等于字母 'x' 的高度)。
 - `rp node \pgfintmod{#1}{2}` : 这是一个非常巧妙的技巧。它会根据传入的参数 `#1` 的奇偶性来调用不同的样式。`\pgfintmod{#1}{2}` 计算 `#1` 除以 2 的余数 (0 或 1)。所以，如果 `#1` 是偶数，这里会调用 `rp node 0` 样式；如果是奇数，则调用 `rp node 1` 样式。
- `rp node 0/.code=` : 定义 `rp node 0` 样式。`.code=` 表示这个样式是空的，不做任何事。所以偶数层的节点是空心的。
- `rp node 1/.style=fill` : 定义 `rp node 1` 样式。`.style=fill` 表示这个样式会填充节点。所以奇数层的节点是实心的。

<LATEX>

```
1   rp edge/.style=draw, rp edge in/.style={rp edge, dashed}, rp edge out/.style={rp edge},
```

- `rp edge/.style=draw` : 定义了边 (线段) 的基础样式 `rp edge`，就是简单地画出一条线。
- `rp edge in/.style={rp edge, dashed}` : 定义了“内部”边的样式，它继承 `rp edge` 并添加了 `dashed` (虚线) 属性。
- `rp edge out/.style={rp edge}` : 定义了“外部”边的样式，它与基础样式 `rp edge` 完全相同 (实线)。

<LATEX>

```
1   rp legend/.style={below, at=(rpbox.south), node font=\bfseries}}
```

- `rp legend/.style={ ... }` : 定义了图例 (显示节点总数的文本) 的样式。
 - `below, at=(rpbox.south)` : 放置在之前定义的局部边界框 `rpbox` 的正下方。
 - `node font=\bfseries` : 文本使用粗体。

2. 第一个绘图命令 \tikzRepPolyA

这个命令的策略是: 逐层放置节点，并在放置新节点的同时连接到前一层的节点。

<LATEX>

```
1 \newcommand*\tikzRepPolyA[2][]{% #2 is an integer number
2 \path[#1,@every rp
3   node[rp node=0, alias/.list={rp-0, rp- ..., rp-\rpSides}](rp){}
```

- `\newcommand*\tikzRepPolyA[2][]` : 定义一个新命令 `\tikzRepPolyA`，它有两个参数。`[#1]` 是可选参数 (用于传入额外的 TikZ 样式)，`{#2}` 是必需参数 (一个整数，表示图形的层数)。
- `\path[#1,@every rp]` : 开始一个 TikZ 路径，并应用可选样式 `#1` 和我们之前定义的 `@every rp` 核心样式。
- `node[rp node=0, alias/.list={ ... }](rp){}` : 在原点 `(0,0)` 放置第一个节点 (第 0 层)。
 - `rp node=0` : 使用 `rp node` 样式，并传入参数 `0`，所以它会是空心的。
 - `alias/.list={ ... }` : 为这个节点创建一系列别名。`rp-0`，`rp-1`，..., `rp-n` 都将指向这个中心节点。这是一种聪明的技巧，用于简化后续循环中边的连接。
 - `(rp){}` : 将这个节点命名为 `rp`。

<LATEX>

```
1   \ifnum#2>0 foreach \rpR in {1, ..., #2}{
2     (rp) foreach \rpS in {0, ..., \rpSides}{
```

- `\ifnum#2>0 ... \fi` : 只有当层数 `#2` 大于 0 时，才执行循环来绘制更多的层。
- `foreach \rpR in {1, ..., #2}{ ... }` : 外层循环，遍历每一层 (从第 1 层到第 `#2` 层)。`\rpR` 代表当前层的半径。
- `(rp) foreach \rpS in {0, ..., \rpSides}{ ... }` : 内层循环，遍历当前层上的每一个顶点。`\rpS` 代表顶点的索引。`(rp)` 将循环的起点重置为中心节点，但 `++` 坐标会使其从上一个节点的位置开始计算。

<LATEX>

```
1     ++(\rpS*\rpAngle:\rpR) node[rp node=\rpR](rp-\rpS){}
```

- `++(\rpS*\rpAngle:\rpR)` : 使用极坐标系在前一个节点的基础上进行相对移动。`\rpS*\rpAngle` 计算出当前顶点的角度，`\rpR` 是半径。这会在一个以原点为中心、半径为 `\rpR` 的圆上定位节点。
- `node[rp node=\rpR](rp-\rpS){}` : 在计算出的位置放置一个新节点。
 - `rp node=\rpR` : 节点的样式 (空心/实心) 由层数 `\rpR` 的奇偶性决定。
 - `(rp-\rpS)` : 给这个新节点一个临时名字，例如 `rp-0'`，`rp-1'` 等。

<LATEX>

```
1     edge[rp edge \if\ifnum\rpS>0 1\fi\ifnum\rpS<\rpSides\space 1\fi in\else out\fi](rp-\rpS)
```

- `edge[...] (rp-\rpS)` : 从新创建的节点 `(rp-\rpS')` 连接一条边到上一层对应位置的节点 `(rp-\rpS)`。
- `rp edge \if ... \fi in\else out\fi` : 这是一个复杂的条件判断，用于决定边的样式。它判断 `\rpS` 是否既大于 0 又小于 `\rpSides`。如果是，则使用 `rp edge in` (虚线) 样式；否则使用 `rp edge out` (实线) 样式。这导致了图中径向连线样式的变化。

<LATEX>

```
1     \ifnum\rpS>0 edge[rp edge out](rp-\pgfinteval{\rpS-1'})\fi
```

- `\ifnum\rpS>0 ... \fi` : 如果当前顶点不是第一个顶点 (`\rpS > 0`)。
- `edge[rp edge out](rp-\pgfinteval{\rpS-1'})` : 从当前节点 `(rp-\rpS')` 连接一条边到同一层的前一个节点 `(rp-\pgfinteval{\rpS-1'})`。这形成了多边形的边。

<LATEX>

```
1     node also[alias=rp-\rpS](rp-\rpS')
2   } \fi
```

- `node also[alias=rp-\rpS](rp-\rpS')` : 这是一个关键步骤。它给刚刚创建的临时节点 `(rp-\rpS')` 赋予一个新的、永久的别名 `(rp-\rpS)`。这样，在下次外层循环 (绘制下一层) 时，`(rp-\rpS)` 就会指向当前层的节点，从而实现层与层之间的连接。

<LATEX>

```
1     node[rp legend]{\inteval{1+#2*(\pgfkeysvalueof{/tikz/rp sides}-1)}};
```

- `node[rp legend]{ ... }` : 在图形的底部添加图例。
- `\inteval{ ... }` : 计算节点总数并显示。公式是 `1 + 层数 * (边数 - 1)`。

3. 第二个绘图命令 \tikzRepPolyB

这个命令的策略是: 先画出所有的线，再在顶点上覆盖节点。

<LATEX>

```
1 \newcommand*\tikzRepPolyB[2][]{% #2 is an integer number
2 \path[rp node 0/.style={fill=white},#1,@every rp, rp edge out]
```

- `\newcommand*\tikzRepPolyB[2][]` : 定义第二个绘图命令，参数与 `A` 相同。
- `\path[rp node 0/.style={fill=white}, ...]` : 开始一个路径。
 - `rp node 0/.style={fill=white}` : 临时覆盖 `rp node 0` 样式，使其填充为白色。这是为了让中心节点能盖住穿过中心的线条。
 - `#1,@every rp, rp edge out` : 应用可选样式、核心样式和默认的 `rp edge out` (实线) 样式。

<LATEX>

```
1   \ifnum#2>0
2     (0,0) foreach \rpS in {0, ..., \rpSides}{-- ++(\rpS*\rpAngle:#2)
3     \if\ifnum\rpS>0 1\fi\ifnum\rpS<\rpSides\space 1\fi edge[rp edge in, draw](0,0) \fi -- cycle
```

- `\ifnum#2>0 ... \fi` : 同样，只在层数大于 0 时绘图。
- 这一长串代码做了两件事:
 - `(0,0) foreach ... -- ++(...) -- cycle` : 从原点开始，通过循环绘制出最外层的多边形。
 - `\if ... \fi edge[rp edge in, draw](0,0)` : 在循环内部，根据与 `A` 命令中相同的条件，从每个顶点向中心 `(0,0)` 绘制一条径向线 (虚线或实线)。

<LATEX>

```
1     foreach \rpR in {1, ..., \pgfinteval{#2-1}}{
2       (0:\rpR) foreach \rpS in {1, ..., \rpSides}{-- ++(\rpS*\rpAngle:\rpR)}}
3
```

- `foreach \rpR in {1, ..., #2-1}` : 循环绘制除了最外层之外的所有内层多边形。
- `(0:\rpR) foreach ...` : 从半径为 `\rpR` 的 `x` 轴正半轴上的点开始，通过循环绘制一个完整的多边形。

<LATEX>

```
1     foreach \rpR in {1, ..., #2}{
2       (0,0) foreach \rpS in {0, ..., \rpSides}{++(\rpS*\rpAngle:\rpR) node[rp node=\rpR]{}
3     \fi
```

- `foreach \rpR ... foreach \rpS ...` : 这是一个双重循环，用来在所有应该有节点的位置放置节点。
- `++(\rpS*\rpAngle:\rpR) node[rp node=\rpR]{}` : 在每个顶点位置放置一个节点，节点的样式 (空心/实心) 由层数 `\rpR` 决定。这些节点会覆盖在线条的交点上。

<LATEX>

```
1     (0,0)node[rp node=0]{} node[rp legend]{\pgfinteval{1+#2*(\pgfkeysvalueof{/tikz/rp sides}-1)}};
```

- `(0,0)node[rp node=0]{}` : 最后，在原点 `(0,0)` 放置中心节点。由于它最后被绘制，它会位于所有径向线的顶层。
- `node[rp legend]{ ... }` : 添加与 `A` 命令完全相同的图例。

4. 文档主体和绘图

<LATEX>

```
1 \begin{document}
2 \foreach \tikzRepPoly in {\tikzRepPolyA, \tikzRepPolyB}
3 \tikz[column sep=1cm, row sep=5mm, column 4/.append style={rp side length=2/3*7mm}]
4 \matrix[rp sides=\inteval{\pgfmatrixcurrentrow+1}]{
```

- `\begin{document}` : LaTeX 文档主体开始。
- `\foreach \tikzRepPoly in {\tikzRepPolyA, \tikzRepPolyB}` : 一个循环，它会执行两次。第一次 `\tikzRepPoly` 代表 `\tikzRepPolyA`，第二次代表 `\tikzRepPolyB`。所以它会生成两组大的矩阵图。
- `\tikz[...]` : 开始一个 TikZ 图片。
 - `column sep=1cm, row sep=5mm` : 设置矩阵的列间距和行间距。
 - `column 4/.append style={ ... }` : 为矩阵的第 4 列追加一个特殊样式，将其 `rp side length` 缩小为原来的 2/3。
- `\matrix[rp sides=\inteval{\pgfmatrixcurrentrow+1}]{ ... }` : 创建一个 TikZ 矩阵。
 - `rp sides=\inteval{\pgfmatrixcurrentrow+1}` : 这是一个非常巧妙的设置。它将 `rp sides` 键的值设置为 `当前行号 + 1`。因为行号从 1 开始，所以第一行是 `1+1=2` 边 (实际是线段)，第二行是 `2+1=3` 边 (三角形)，以此类推。

<LATEX>

```
1 \tikzRepPoly{0} \pgfmatrixnextcell \tikzRepPoly{1} \pgfmatrixnextcell \tikzRepPoly{2} \pgfmatrixnextcell \tikzRepPoly{3} \\\
2 % ... (Repeated for 5 rows)
3 }
4 \end{document}
```

- `\tikzRepPoly{0} \pgfmatrixnextcell ... \\\` : 这是矩阵的内容。
- `\tikzRepPoly{0}` : 调用绘图命令 (`A` 或 `B`)，层数为 0。
- `\pgfmatrixnextcell` : 移动到下一个单元格。
- `\\` : 换行。
- 整个矩阵有 5 行 4 列。每一行代表一种多边形 (从 2 边到 6 边)。每一列代表不同的层数 (从 0 层到 3 层)。

总结

这段代码通过定义两种功能相同但实现方法不同的命令 (`\tikzRepPolyA` 和 `\tikzRepPolyB`) 来绘制分层的正多边形结构。

- `\tikzRepPolyA` 采用“节点优先”的策略，逐个添加节点并立即连接边。
- `\tikzRepPolyB` 采用“线优先”的策略，先画出所有背景线条，然后将节点覆盖在顶点上。

最后，代码利用 `\foreach` 循环和 `matrix` 环境，系统地展示了这两个命令在不同边数 (2 到 6) 和不同层数 (0 到 3) 下的绘图效果，形成了一个清晰的对比图表。代码中包含了大量高级的 TikZ 技巧，如自定义键、条件样式、节点别名和在矩阵选项中动态计算等。