Graph Drawing in TikZ

Till Tantau

Graph Drawing Conference 2012
3. Steps towards a final layout: (a) PR \( \mathcal{R} \), (b) fine-layering of the subgraph.
The Problem: Integrating Graph Drawings Into Documents

**GraphViz** yields

```
INTEGRAL

exp  dx

2pi  +

x    r
```

**\TeX** yields

```
\int \exp \cdot 2\pi + x \cdot r \, dx
```
The Problem: Integrating Graph Drawings Into Documents

GraphViz yields

\[
\int \exp \cdot 2\pi + x \cdot r \, dx
\]

\TeX\ yields

\[
\int \exp \cdot 2\pi + x \cdot r \, dx
\]

What we actually want:

\[
\int \exp \cdot 2\pi + x \cdot r \, dx
\]
A Solution: Graph Drawing in TikZ

- Take an *existing document description language* (TEX) with an *embedded graphics description language* (TikZ).
- Add options and syntactic extensions for specifying graphs easily.
- Run graph drawing algorithms *as part of the document processing*.

**Advantages**

+ Styling of nodes and edges matches main document.
+ Graph drawing algorithms know size of nodes and labels precisely.
+ No external programs needed.
+ Algorithm designers can concentrate on algorithmic aspects.
Talk Outline

How Do I Use It?

How Does It Work?

How Do I Implementing An Algorithm?
TikZ in a Nutshell: The Idea

\usepackage{tikz}

...  
A circle like \tikz{}
  \fill[red] (0,0) circle[radius=.5ex]; 
} is round.

- TikZ is a package of \TeX-macros for specifying graphics.
- The macros transform highlevel descriptions of graphics into lowlevel \textit{PDF-}, \textit{PostScript-}, or \textit{SVG}-primitives during a \TeX run.

A circle like \textbullet{} is round.
TikZ in a Nutshell: Nodes and Edges

\begin{tikzpicture}
  \node (a) at (0,2) [rounded rectangle] {Hello};
  \node (b) at (2,2) [tape] {World};
  \node (c) at (4,2) [circle, dashed] {$ c^2 $};
  \node (d) at (4,0) [diamond] {$ \delta $};
  \draw (a) edge[->] (b) (b) edge[->] (c) (b) edge[->] (d) (d) edge[->] (a);
\end{tikzpicture}
Using the Graph Drawing System = Adding an Option

\usetikzlibrary{graphdrawing.layered}
...
\tikz [layered layout] {
  \node (a) at (0,1) [rounded rectangle] {Hello};
  \node (b) at (2,1) [tape] {World};
  \node (c) at (4,1) [circle, dashed] {$c^2$};
  \node (d) at (4,0) [diamond] {$\delta$};
  \draw (a) edge[->] (b) (b) edge[->] (c) (b) edge[->] (d) (d) edge[->] (a);
}
Using the Graph Drawing System = Adding an Option

\usetikzlibrary{graphdrawing.force}

\tikz [spring layout, node distance=1.25cm] {
    \node (a) at (0,1) [rounded rectangle] {Hello};
    \node (b) at (2,1) [tape] {World};
    \node (c) at (4,1) [circle, dashed] {$ c^2 $};
    \node (d) at (4,0) [diamond] {$ \delta $};
    \draw (a) edge[->] (b) (b) edge[->] (c)
             (b) edge[->] (d) (d) edge[->] (a);
}\n
A concise syntax for graphs is important when *humans specify graphs* “by hand.”

The chosen syntax *mixes the philosophies* of DOT and TikZ.

```latex
\begin{tikzpicture}[layered layout]
  \graph {
    Hello -> World -> "$c^2$";
    World -> "$\delta" -> Hello;
  }
\end{tikzpicture}
```
Key Features of TikZ’s Syntax for Graphs

- **Node options follow nodes in square brackets.**
- **Edge options follow edges in square brackets.**
- **Additional edge kinds.**
- **Natural specification of trees.**

\begin{tikzpicture}
  \graph [layered layout] {
    Hello [rounded rectangle] -> World [tape] -> "$c^2$" [circle, dashed];
    World -> "$\delta$" [diamond] -> Hello;
  }
\end{tikzpicture}
Key Features of TikZ’s Syntax for Graphs

- Node options follow nodes in square brackets.
- *Edge options follow edges in square brackets.*
- Additional edge kinds.
- Natural specification of trees.

\begin{tikzpicture}
\graph [layered layout] {
  Hello [rounded rectangle] -> World [tape] -> "$c^2$" [circle, dashed];

  World -> [dashed, blue] \(\delta\) [diamond] -> [bend right, "foo"] Hello;
};
\end{tikzpicture}
Key Features of TikZ’s Syntax for Graphs

- Node options follow nodes in square brackets.
- Edge options follow edges in square brackets.
- *Additional edge kinds.*
- Natural specification of trees.

\begin{tikzpicture}
\graph [tree layout] {
  a -> b -- c <- d <-> e;
};
\end{tikzpicture}
Key Features of TikZ’s Syntax for Graphs

- Node options follow nodes in square brackets.
- Edge options follow edges in square brackets.
- Additional edge kinds.
- Natural specification of trees.

\begin{code}
\begin{verbatim}
\tikz \graph [binary tree layout] {
  root -> {
    left -> {
      1, 
      2 
    },
    right -> {
      3 -> { , 4 }
    }
  }
};
\end{verbatim}
\end{code}
Talk Outline

How Do I Use It?

How Does It Work?

How Do I Implementing An Algorithm?
TeX is great, . . .

- but implementing advanced algorithms is next to impossible.

Lua is a small, simple, elegant language, . . .

- . . . that has been integrated into modern versions of TeX:

\[ \sum_{n=1}^{100} n = \]
\[
\text{\directlua{
local sum = 0
for i=1,100 do
  sum = sum + i
end
tex.print(sum)
}}\]

\[\sum_{n=1}^{100} n = 5050\]
How a Graph is Drawn

TikZ Layer

```latex
\graph [tree layout]{
  a -> b -> c
};
```

Lua Layer

```lua
beginGraphDrawingScope(...)
addNode(...)
addEdge(...)
runGraphDrawingAlgorithm()
endGraphDrawingScope()
```

node positioning callback

edge positioning callback

load algorithm and run it
Talk Outline

How Do I Use It?

How Does It Work?

How Do I Implementing An Algorithm?
“Graph Drawing” can be seen as . . .

- starting with a *graph*, . . .
- . . . applying a *series of transformations* to it. . .
- . . . and ending with a *drawn graph*.

Graph drawing in TikZ follows this philosophy:
- Algorithms *declare* what kind of graphs they expect
- and also the properties of the graphs they *produce*. 
Implementing a New Graph Drawing Algorithm

-- File VerySimpleDemo.lua
local VerySimpleDemo = pgf.gd.new_algorithm_class {
    works_only_on_connected_graphs = true,
}

function VerySimpleDemo:run()
    local graph = self.ugraph -- The graph model
    local radius = graph.options['/graph drawing/radius']
    local alpha = (2*math.pi) / #graph.vertices

    -- Iterate over all vertices:
    for i,vertex in ipairs(graph.vertices) do
        vertex.pos.x = math.cos(i*alpha) * radius
        vertex.pos.y = math.sin(i*alpha) * radius
    end
end

return VerySimpleDemo -- This return is a quirk of Lua
Using the Graph Drawing Algorithm

\tikz \graph [ layout=VerySimpleDemo, radius=1cm] {  
  a -- b -- c -- a;  
  d -- e;  
  f -- g -- h -- d -- f;  
  e -- g;  
};

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{graph1}
\caption{Graph layout example using \texttt{VerySimpleDemo} layout.}
\end{figure}
Using the Graph Drawing Algorithm

\tikz \graph [ layout=VerySimpleDemo, radius=1cm] {
  a -- [orient=right] b -- c -- a;
  d -- e;
  f -- g -- h -- d -- f;
  e -- g;
};

---

![Graph 1](image1)

![Graph 2](image2)
Using the Graph Drawing Algorithm

\begin{tikzpicture}
  \graph [layout=VerySimpleDemo, radius=1cm]
  a --[orient=right] b -- c -- a;
  d -- e;
  f -- g -- h -- d --[stub,red] f;
  e --[stub,red] g;
\end{tikzpicture}
Using the Graph Drawing Algorithm

\begin{tikzpicture}
    \GraphInit[vstyle=VerySimpleDemo]
    \SetGraphUnit{1cm}
    \GraphInit[vstyle=VerySimpleDemo]
    \SetGraphUnit{1cm}
    \SetVertexFillColor{lightgray}
    \SetVertexShape{circle}
    \SetUpEdge[style=circle connection bar]
    \begin{scope}
        \Vertex{x}
        \Vertex{y}
        \Vertex{z}
        \Vertex{w}
        \Vertex{v}
        \Vertex{u}
        \Vertex{t}
    \end{scope}
    \begin{scope}
        \Edges(x,y,y,z,z,x,x,w,w,v,v,w,w,t,t)
    \end{scope}
\end{tikzpicture}
Conclusion

Graph drawing in TikZ is aimed at

- *users* who want to draw graphs with up to $\approx 100$ nodes inside \TeX documents and
- *researchers* who want to implement new algorithms.

Already implemented algorithms:

- Reingold–Tilford tree drawing.
- Layered Sugiyama method.
- Multi-level force-based algorithms.

*Available as* part of (the development version of) TikZ.