UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR
THEORETICAL COMPUTER SCIENCE

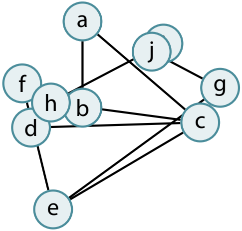
Algorithmic Graph Drawing in Ti*k*Z with Lua

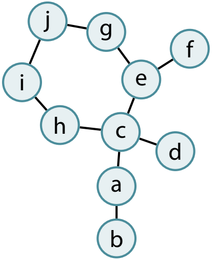
Till Tantau

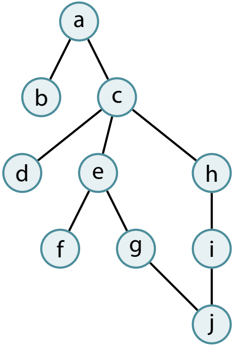
FOSDEM 2015

IM FOCUS DAS LEBEN

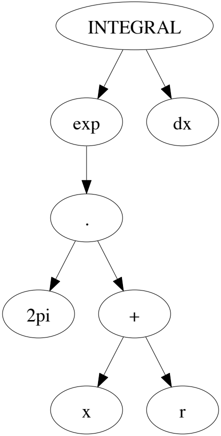
Which drawing of the graph would you choose?

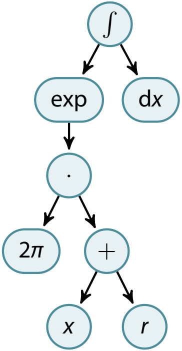




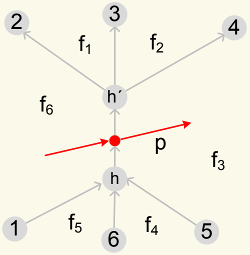


Typography and graph drawing do not mix easily.





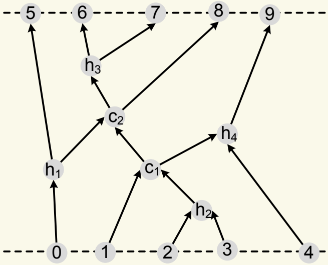
Typography and graph drawing do not mix easily.



(b) A realization of p.

al arc can reduce the crossings

aints, are restricted to the



(a)

3. Steps towards a final layout: (a) PR $\mathcal{R}$, (b) fine-layering of the subgra

Outline

Graph Drawing

- Aims
- Solutions: Force-Based Methods
- Solutions: The Sugiyama Method

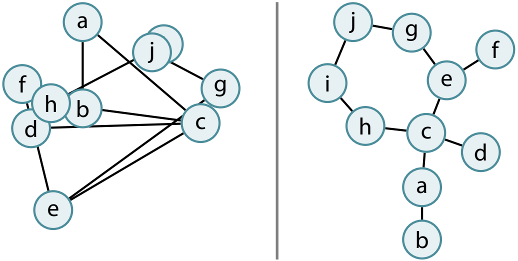
Graph Drawing in Ti*k*Z

- What is Ti*k*Z?
- How to Draw a Graph with Ti*k*Z

Graph Drawing in Ti*k*Z with Lua

- Programming in TeX
- Programming in Lua
- How to Implement a Graph Drawing Algorithm

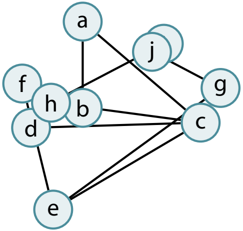
Why is the right drawing better than the left one?

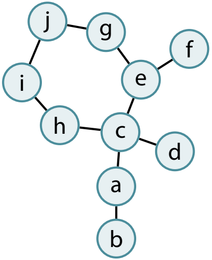


Some observations:

1. On the right, there are less *crossings*.
2. On the right, there are less *overlaps*.
3. On the right, there are more *symmetries*.
4. On the right, the *edge lengths* are similar.
5. On the right, the *angles between edges* are similar.

Why is the right drawing better than the left one?





This leads to an *optimzation problem:* "Draw the graph such that

1. *edge crossings* are minimized,
2. *node overlaps* are minimized,
3. *symmetries* are maximized,
4. *deviations in edge lengths* are minimized
5. *angular variance* is minimized."

There are many other possible objectives: Important stuff near the center, clustered clusters, . . .

Outline

Graph Drawing

- Aims
- Solutions: Force-Based Methods
- Solutions: The Sugiyama Method

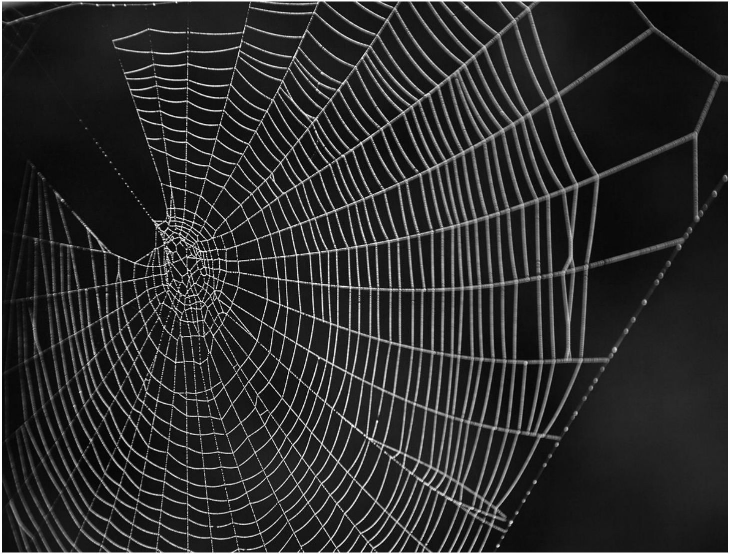
Graph Drawing in Ti*k*Z

- What is Ti*k*Z?
- How to Draw a Graph with Ti*k*Z

Graph Drawing in Ti*k*Z with Lua

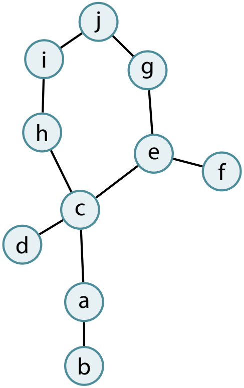
- Programming in TEX
- Programming in Lua
- How to Implement a Graph Drawing Algorithm

Mother nature draws graphs beautifully.



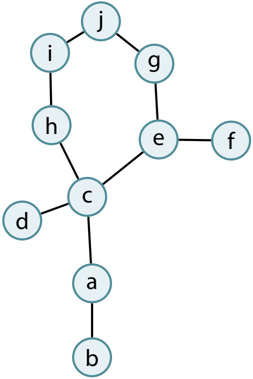
The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.



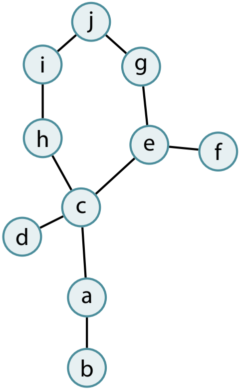
The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.



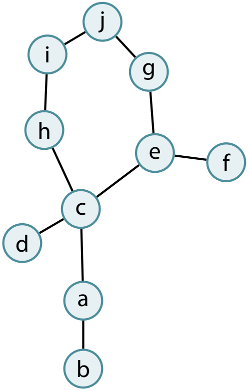
The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.



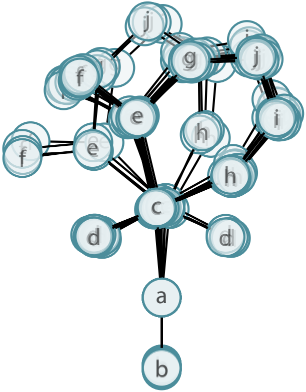
The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.



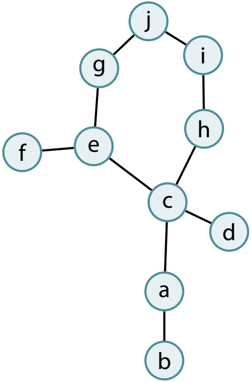
The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.



The force-based approach: Apply forces to nodes iteratively.

- Nodes are *movable*.
- Edges cause *forces* between nodes.
- We *simulate* the resulting node movements until an *equilibrium* is reached.

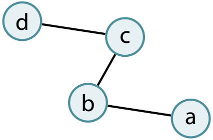


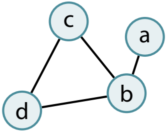
Forces 1 (Tutte): Spring forces

Public Domain

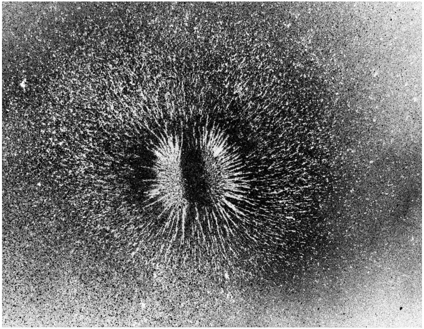
- Edges are *springs*.
- Springs have a *natural length*.
- If an edge is too short, "it pushes the nodes apart."
- If an edge is too long, "it pulls the nodes together."

Forces 1 (Tutte): Spring forces



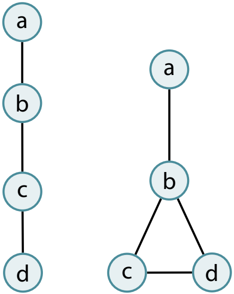


Forces 2 (Eades): Electrical forces

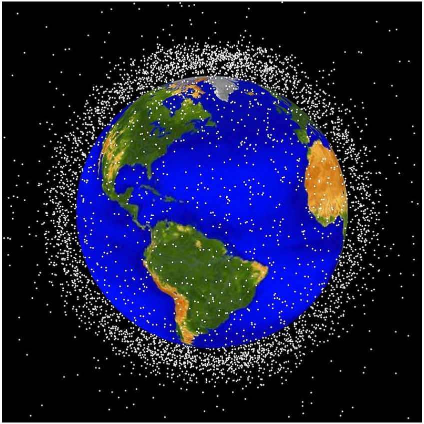


- There are additional *repulsive forces* between nodes.
- Nodes hence tend to form *circles* and *lines*.
- Angles tend to be equal.

Forces 2 (Eades): Electrical forces



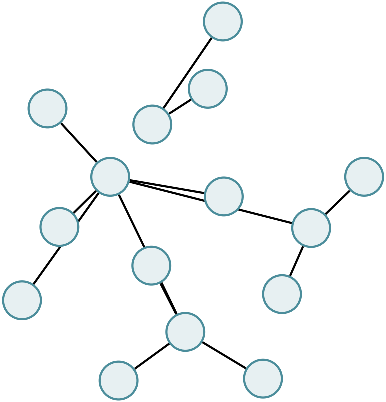
Forces 3: Gravitational forces



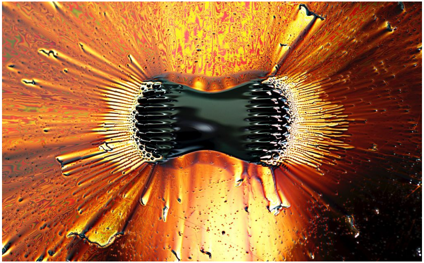
Public Domain

- Nodes are additionally pulled to the center.
- The "heavy, important" nodes tend to be in the center.

Forces 3: Gravitational forces



Forces 4: Magnetic fields



GNU Free Documentation License, Author Gregory F. Maxwell

- Edges try to align with the direction of a force field.
- For instance, we can cause edges to become *horizontal* or *vertical*.

Summary of force-based algorithms.

Advantages

+ "There is a force for every aesthetic objective."
+ Easy iterative implementation.
+ Edge routing is easily incorporated.

Disadvantages

- Iterative algorithms are slow.
- Difficult to implement *well*.
- Difficult to reproduce and predict drawings.

Outline

Graph Drawing

- Aims
- Solutions: Force-Based Methods
- Solutions: The Sugiyama Method

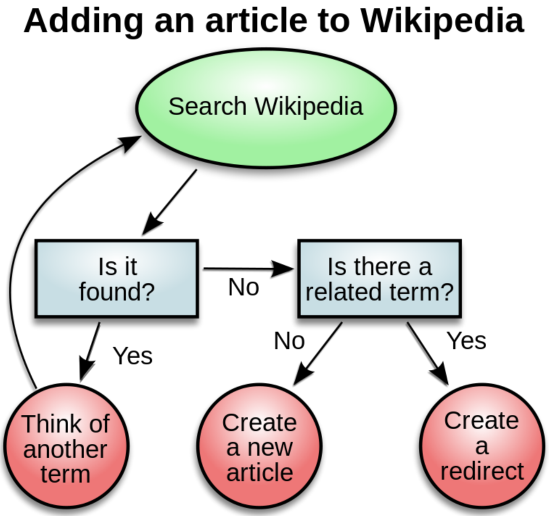
Graph Drawing in Ti*k*Z

- What is Ti*k*Z?
- How to Draw a Graph with Ti*k*Z

Graph Drawing in Ti*k*Z with Lua

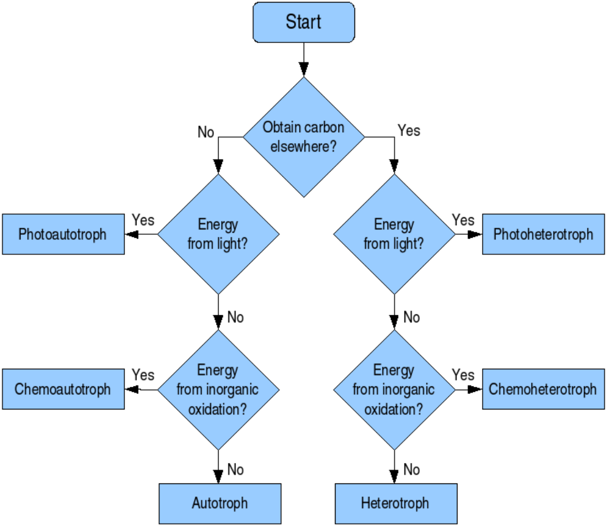
- Programming in TEX
- Programming in Lua
- How to Implement a Graph Drawing Algorithm

Many graphs are layered.

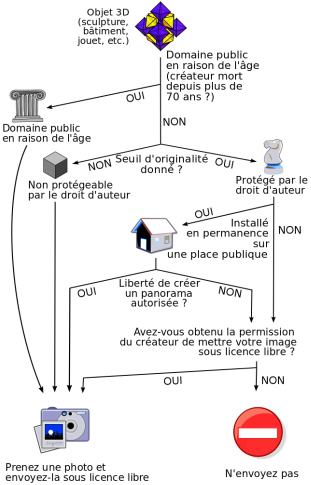


Public Domain

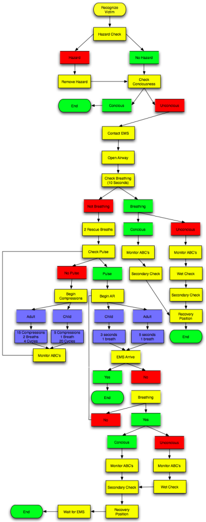
Many graphs are layered.



Many graphs are layered.



Many graphs are layered.



Creative Commons License

The Sugiyama method.

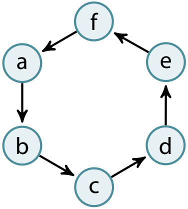
On input of a *directed graph*, do:

1. Make the graph *acyclic*, if necessary.
2. Assign a *layer* to each node, so that edges are only between nodes of adjacent layers.
3. Minimize the number of *edge crossings.*
4. Position the nodes on each layer *nicely.*

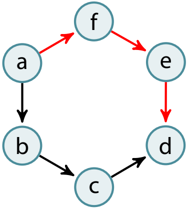
(Unfortunately, all but the last step are NP-complete. . .)

Sugiyama step 1: Make the graph acyclic

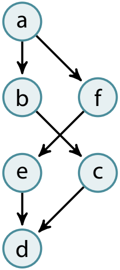
Input



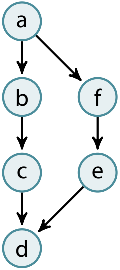
Redirecting edges makes it acyclic



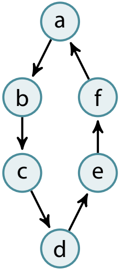
Sugiyama step 2: Assign layers



Sugiyama step 3: Minimize crossings



Sugiyama step 4: Position nodes nicely



Summary of Sugiyama's method

Advantages

+ Produces nice drawings of layered graphs.
+ Can handle edges crossing several layers.
+ Extremely fast when good heuristics are used.

Disadvantages

– Difficult implementation.
– Works only for inherently layered graphs.

Outline

TikZ ist *kein* Zeichenprogramm

```
Let $\int_0^1 \sqrt{x}\, dx$
be the integral\dots
```

Let $\int_0^1 \sqrt{x}\, dx$ be the integral. . .

```
Let \tikz \fill[red]
  (0,0) circle[radius=1mm];
be the circle\dots
```

Let ● be the circle. . .

- TikZ is a library of *TeX macros* for specifying graphics.
- I developed it about 10 years ago in order to produce the 10 figures of my PhD thesis.
- Today, the manual has over a 1000 pages.

How does it work?

```
The triangle \tikz \draw (0,0)
  -- (30:10pt) -- (60:10pt) -- cycle;
```

The triangle 

Ti*k*Z first transforms the code into a *series of graphics commands*:

```
\pgfpathmoveto{\pgfpointxy{0}{0}}
\pgfpathlineto{\pgfpointpolar{30}{10pt}}
\pgfpathlineto{\pgfpointpolar{60}{10pt}}
\pgfpathclose
\pgfusepath{draw}
```

How does it work?

```
The triangle \tikz \draw (0,0)
  -- (30:10pt) -- (60:10pt) -- cycle;
```

The triangle 

These, in turn, get transformed into *abstract graphics primitives*:

```
\pgfsys@moveto{0pt}{0pt}
\pgfsys@lineto{8.660254pt}{5pt}
\pgfsys@lineto{5pt}{8.660254pt}
\pgfsys@closepath
\pgfsys@stroke
```

How does it work?

```
The triangle \tikz \draw (0,0)
  -- (30:10pt) -- (60:10pt) -- cycle;
```

The triangle 

Finally, these are translated into *concrete graphics primitives:*

```
\special{ps:: 0 0 moveto}
\special{ps:: 8.627899 4.98132 lineto}
\special{ps:: 4.98132 8.627899 lineto}
\special{ps:: closepath}
\special{ps:: stroke}
```

(for PostScript output)

How does it work?

```
The triangle \tikz \draw (0,0)
  -- (30:10pt) -- (60:10pt) -- cycle;
```

The triangle 

Finally, these are translated into *concrete graphics primitives:*

```
\special{pdf: 0 0 m}
\special{pdf: 8.627899 4.98132 l}
\special{pdf: 4.98132 8.627899 l}
\special{pdf: h}
\special{pdf: S}
```

(for PDF output)

How does it work?

```
The triangle \tikz \draw (0,0)
  -- (30:10pt) -- (60:10pt) -- cycle;
```

The triangle 

Finally, these are translated into *concrete graphics primitives:*

```
\special{dvisvgm:raw
  <path d = " M 0 0
              L 8.660254 5
              L 5 8.660254
              Z"
        style = "stroke">}
```

(for SVG output)

Outline

Graph Drawing

- Aims
- Solutions: Force-Based Methods
- Solutions: The Sugiyama Method

Graph Drawing in Ti*k*Z

- What is Ti*k*Z?
- How to Draw a Graph with Ti*k*Z

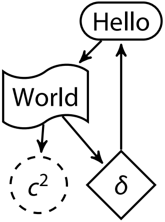
Graph Drawing in Ti*k*Z with Lua

- Programming in TEX
- Programming in Lua
- How to Implement a Graph Drawing Algorithm

The TikZ syntax for graphs

- A succinct, well-designed syntax for graphs is important when *humans* specify graphs *“by hand”.*
- The TikZ syntax *mixes the philosophies* of DOT and TikZ.

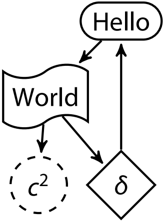
```
\tikz \graph {
      Hello    [rounded rectangle]
  ->  World    [tape]
  ->  "$c^2$"  [circle, dashed];

  World
  ->  "$\delta$" [diamond]
  ->  Hello;
};
```



The TikZ syntax for graphs

- *Node options follow nodes.*
- Edge options follow edges.
- Special notation for edges.
- Natural syntax for trees.

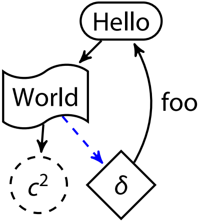
```
\tikz \graph {
     Hello    [rounded rectangle]
 ->  World    [tape]
 ->  "$c^2$"  [circle, dashed];

 World
 ->  "$\delta$" [diamond]
 ->  Hello;
};
```



The TikZ syntax for graphs

- Node options follow nodes.
- *Edge options follow edges.*
- Special notation for edges.
- Natural syntax for trees.

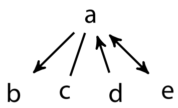
```
\tikz \graph {
     Hello    [rounded rectangle]
  -> World    [tape]
  -> "$c^2$"  [circle, dashed];

  World
  ->[dashed, blue] "$\delta$"[diamond]
  ->[bend right, "foo"'] Hello;
};
```



The TikZ syntax for graphs

- Node options follow nodes.
- Edge options follow edges.
- *Special notation for edges.*
- Natural syntax for trees.

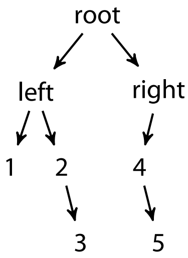
```
\tikz \graph {
  a -> b -- c <- d <-> e;
};
```



The TikZ syntax for graphs

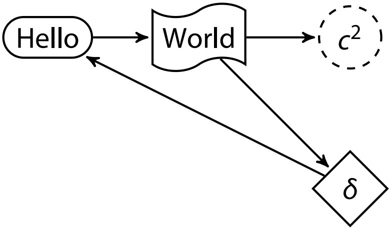
- Node options follow nodes.
- Edge options follow edges.
- Special notation for edges.
- *Natural syntax for trees.*

```
\tikz \graph [binary tree layout] {
  root -> {
    left -> {
      1,
      2 -> 3 [second]
    },
    right -> {
      4 -> { , 5 }
    }
  }
};
```



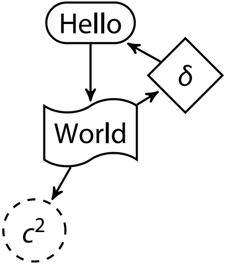
Explicit coordinates . . .

```
\tikz \graph {
  Hello        [x=0, y=2, rounded rectangle];
  World        [x=2, y=2, tape];
  "$c^2$"      [x=4, y=2, circle, dashed];
  "$\delta$"   [x=4, y=0, diamond];

  Hello -> World -> "$c^2$";
  World -> "$\delta$" -> Hello;
};
```



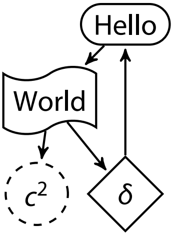
. . . versus algorithmic graph drawing.

```
\usegdlibrary{force}
\tikz \graph [spring layout, node distance=1.2cm] {
  Hello      [x=0, y=2, rounded rectangle];
  World      [x=2, y=2, tape];
  "$c^2$"    [x=4, y=2, circle, dashed];
  "$\delta$" [x=4, y=0, diamond];

  Hello -> World -> "$c^2$";
  World -> "$\delta$" -> Hello;
};
```

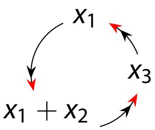


. . . versus algorithmic graph drawing.

```
\usegdlibrary{layered}
\tikz \graph [layered layout] {
 Hello      [           rounded rectangle];
 World      [           tape];
 "$c^2$"    [           circle, dashed];
 "$\delta$" [           diamond];

 Hello -> World -> "$c^2$";
 World -> "$\delta$" -> Hello;
};
```

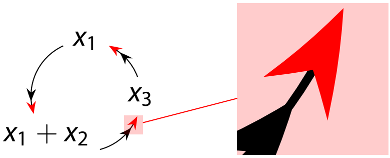


Graph drawing is even useful in seemingly trivial cases.




```
\tikz [>={Stealth[bend] Stealth[bend, red]}]
  \graph [simple necklace layout, necklace routing] {
    "$x_1$" -> "$x_1+x_2$" -> "$x_3$" -> "$x_1$"
  };
```

Graph drawing is even useful in seemingly trivial cases.




```
\tikz [>={Stealth[bend] Stealth[bend, red]}]
  \graph [simple necklace layout, necklace routing] {
   "$x_1$" -> "$x_1+x_2$" -> "$x_3$" -> "$x_1$"
  };
```

Outline

TEX und LuaTEX

TEX is great, but . . .

- implementing large algorithms is *practically impossible* since
- we miss *floating point numbers*, *strings*, *control structures*, *arrays*, *records*, *modules*, . . .

```
The sum of the first 100 numbers is
\newcount\i \newcount\sum
\loop
  \advance\sum by \i\relax
  \ifnum\i<100
  \advance\i by 1\relax
\repeat
\the\sum
```

The sum of the first 100 numbers is 5050

TEX und LuaTEX

TEX is great, but . . .

- implementing large algorithms is *practically impossible* since
- we miss *floating point numbers*, *strings*, *control structures*, *arrays*, *records*, *modules*, . . .

Lua is a minimalistic, elegant language, . . .

- . . . that has been integrated into recent TEX versions:

```
The sum of the first 100 numbers is
\directlua{
  local sum = 0
  for i=1,100 do
    sum = sum + i
  end
  tex.print(sum)
}
```

The sum of the first 100 numbers is 5050

Lua by examples: Hello World.

```
print "Hello World!"
```

- Lua is an imperative scripting language. . .
- . . . that gets you going quickly . . .
- . . . and is really tiny (compiler and libraries around 200kB).

Lua by examples: Variables and types.

```
local x = 1
local y = 2
local z = "Hello there"

if 2*x == y then
  print "Ok"
end

if z == "Hello there" then
  print "Ok"
end
```

- The syntax is a bit “Pascal-like”.
- There are only few types (numbers, strings, functions, tables).
- You cannot and need not specify types.

Lua by examples: Functions.

```
function factorial (n)
  if n <= 1 then
    return 1
  else
    return n*factorial(n-1)
  end
end
```

- Functions are first-order citizens.
- They can be passed around and closures are fully supported.

Lua by examples: Everything is a (hash) table.

```
local array1 = { 2, 3, "hallo" }
local array2 = { 4, 3, 2, 1 }
local record = {
  start = 1,
  stop = 2
}
```

Lua's "everything is a table" paradigm:

- An *array* hashes positive integers to entries.
- A *struct* hashes strings to entries
  (so `record.start` is syntactic sugar for `record["start"]`).

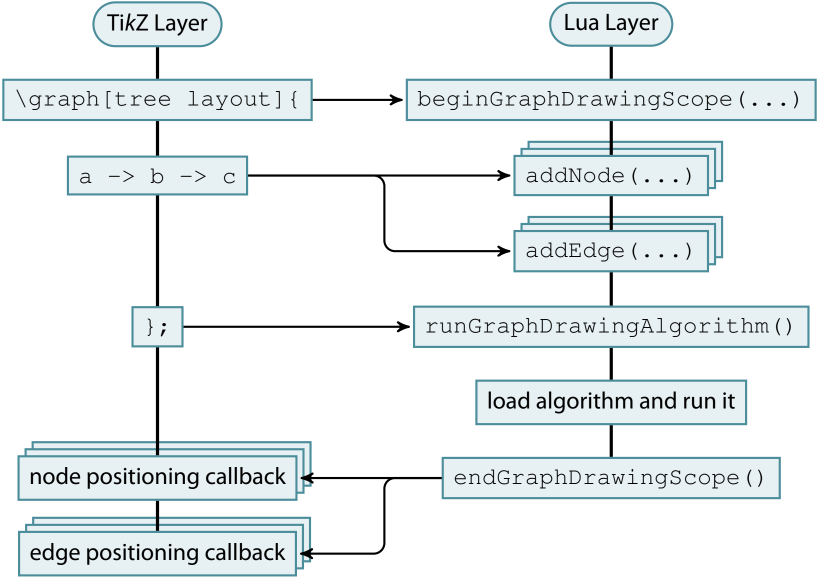
Lua's hash tables are

- *incredibly fast* (strings are prehashed, integers are not hashed but internally form an array),
- *incredibly easy* (they grow and shrink automatically, the syntax is very well designed).

Lua: What else?

- Lua supports coroutines.
- Lua supports meta-programming and thereby classes and objects.
- Lua does not crash.
- Lua does garbage collection.
- Lua integrates seamlessly with C in both directions.

The interplay of Lua and TikZ.



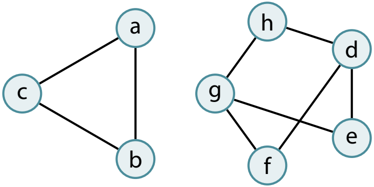
Outline

Complete code for a very simple graph drawing algorithm.

```lua
-- File SimpleDemo.lua
local MyAlgorithmClass = {}
function MyAlgorithmClass:run()
 local g = self.digraph
 local alpha = (2 * math.pi) / #g.vertices
 local radius = g.options.radius
 for i,vertex in ipairs(g.vertices) do
  vertex.pos.x = radius * math.cos(i * alpha)
  vertex.pos.y = radius * math.sin(i * alpha)
 end
end

-- "Publish" the algorithm
local graph_drawing_framework =
 require "pgf.gd.interface.InterfaceToAlgorithms"
graph_drawing_framework.declare {
 key           = "simple demo layout",
 algorithm     = MyAlgorithmClass,
 preconditions = { connected = true }
}
```

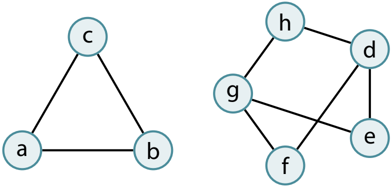
We can immediately use the algorithm – no compilation or installation is needed.

```
\usegdlibrary{SimpleDemo}
...
\tikz \graph [ simple demo layout, radius=1cm ] {
 a -- b -- c -- a;
 d -- e;
 f -- g -- h -- d -- f;
 e -- g;
};
```



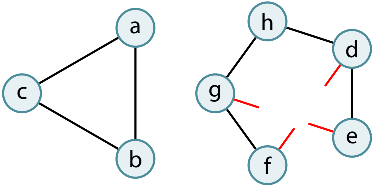
We can immediately use the algorithm – no compilation or installation is needed.

```
\usegdlibrary{SimpleDemo}
...
\tikz \graph [ simple demo layout, radius=1cm] {
  a --[orient=right] b -- c -- a;
  d -- e;
  f -- g -- h -- d -- f;
  e -- g;
};
```



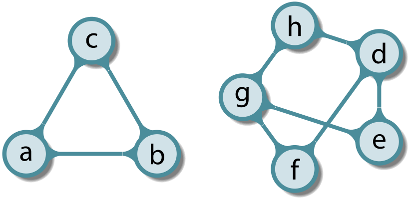
We can immediately use the algorithm – no compilation or installation is needed.

```
\usegdlibrary{SimpleDemo}
...
\tikz \graph [ simple demo layout, radius=1cm]
 a --[orient=right] b -- c -- a;
 d -- e;
 f -- g -- h -- d --[stub,red] f;
 e --[stub, red] g;
};
```



We can immediately use the algorithm – no compilation or installation is needed.

```
\usegdlibrary{SimpleDemo}
...
\tikz \graph [ simple demo layout, radius=1cm,
               nodes={circle, fill=..., ...},
               edges={circle connection bar, ...}] {
  a --[orient=right] b -- c -- a;
  d -- e;
  f -- g -- h -- d -- f;
  e -- g;
};
```



Summary

Graph drawing is about drawing graphs

- quickly,
- such that some aesthetic criteria are met and
- such that structure in the graphs becomes visible.

Graph drawing in TikZ is directed at

- *users* who wish to draw graphs as part of TEX documents
- and *researchers* who implement new algorithms.

Graph drawing in TikZwith Lua means that

- algorithms can and must be implement in the Lua language
- inside a *framework* that takes care of common tasks.