



## Simulating hand-drawn lines

Asked 11 years, 4 months ago   Modified 3 years, 6 months ago   Viewed 26k times



182

I am working on a project that will have some vector graphics, perhaps using `TikZ` (or a similar package). All of the graphics consists of lines, in shades of gray. However, the lines from `TikZ` appear too clean.



Is there any way to make the lines appear as if they were produced by a graphite pencil?



tikz-pgf

diagrams

pstricks

context

metapost

[Edit tags](#)

Share Edit Follow Close

Flag

edited May 17, 2014 at 14:57



strpeter

5,075

7

33

61

asked Dec 25, 2011 at 12:32



Village

13.1k

23

112

215

3



Tried the same a couple of month ago, but never came up with a Latex solution. Found a simple code example using google, so maybe it might help: [stevehanov.ca/blog/index.php?id=33](http://stevehanov.ca/blog/index.php?id=33) – Christian Dec 25, 2011 at 20:50

14 Answers

Sorted by:

[Reset to default](#)

Date modified (newest first)



I'm posting this just because you mentioned specifically the lines.

182

I modified the `bent` decoration to make it look like more of a hand drawing. It truly has problems and you can't use it on curves for now... well you can but the result is unexpected at the least.



+500



```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{calc,decorations.pathmorphing,patterns}

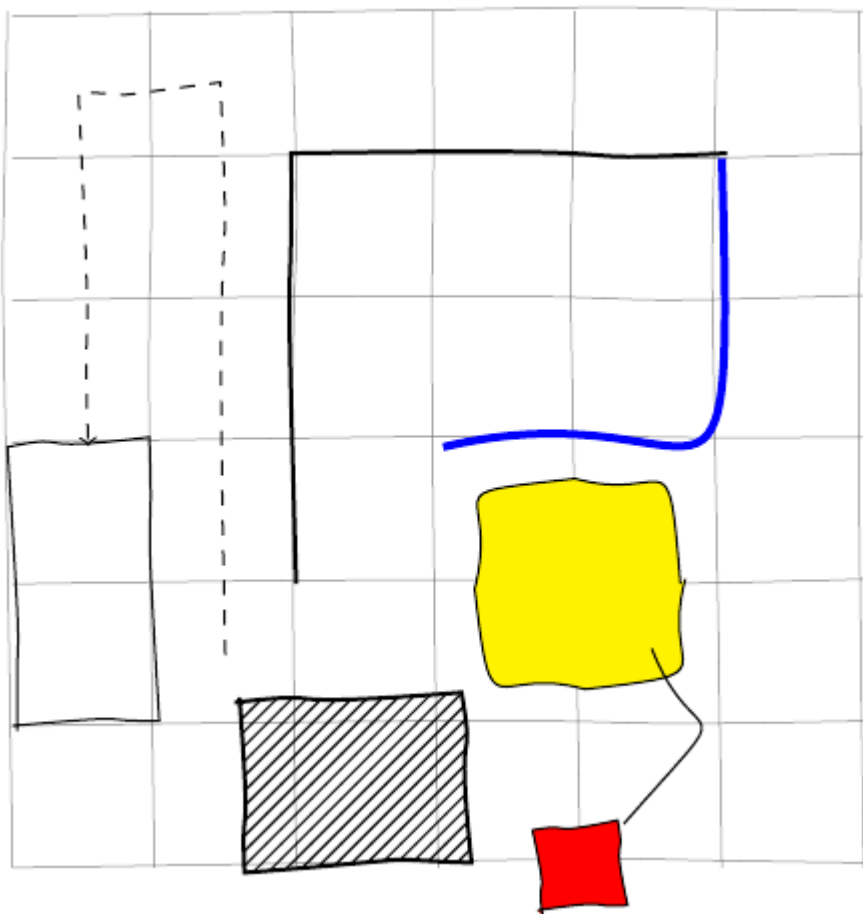
\makeatletter

\pgfdeclaredecoration{penciline}{initial}{
  \state{initial}[width=+\pgfdecoratedinputsegmentremainingdistance,auto
corner on length=1mm,]{
    \pgfpathcurveto%
      {% From
        \pgfqpoint{\pgfdecoratedinputsegmentremainingdistance}
                      {\pgfdecorationsegmentamplitude}
      }
      {% Control 1
        \pgfmathrand
        \pgfpointadd{\pgfqpoint{\pgfdecoratedinputsegmentremainingdistance}
{0pt}}
                      {\pgfqpoint{-\pgfdecorationsegmentaspect
\pgfdecoratedinputsegmentremainingdistance}%
                      {\pgfmathresult
\pgfdecorationsegmentamplitude}
                      }
      }
      {%TO
        \pgfpointadd{\pgfpointdecoratedinputsegmentlast}{\pgfpoint{1pt}{1pt}}
      }
    }
  \state{final}{}
}

\makeatother
\begin{document}
\begin{tikzpicture}[decoration=penciline]
\draw[decorate,style=help lines] (-2,-2) grid[step=1cm] (4,4);
\draw[decorate,thick] (0,0) -- (0,3) -- (3,3);
\draw[decorate,ultra thick,blue] (3,3) arc (0:-90:2cm); %% This is supposed to
be an arc!!
\draw[decorate,thick,pattern=north east lines] (-0.4cm,-0.8cm) rectangle (1.2,-
2);
\node[decorate,draw,inner sep=0.5cm,fill=yellow,circle] (a) at (2,0) {}; %%
That's not even an ellipse !!
\node[decorate,draw,inner sep=0.3cm,fill=red] (b) at (2,-2) {};
\draw[decorate] (b) to[in=-45,out=45] (a); %% This was supposed to be an edge!!
\node[decorate,draw,minimum height=2cm,minimum width=1cm] (c) at (-1.5,0) {};
\draw[decorate,->,dashed] (-0.5cm,-0.5cm) -- (-0.5cm,3.5cm) -| (c.north);
```

```
\end{tikzpicture}
\end{document}
```

Here is the output:



I'm currently studying the `markings` decorations to see how one can move along a path without giving explicit coordinates on the curve. That would hopefully make it possible to use it on arcs. Note that the background grid is also decorated :)

Share Edit Follow Flag

edited Mar 29, 2012 at 16:18

answered Mar 29, 2012 at 15:05



**percusse**

**156k**

15

324

560

- 1 Is there a way to "fix" the lines once they are drawn? If using this with overlays in beamer slides, the lines change on every slide that is generated. – [user38931](#) Dec 5, 2013 at 12:42
- 11 @user38931 You can fix the seed of the random number generator. See example here [tex.stackexchange.com/questions/144621/...](https://tex.stackexchange.com/questions/144621/...) to start with and replace the seed with a fixed number. I think you can take it from there. – [percusse](#) Dec 5, 2013 at 13:49
- 1 @SergioParreiras I'll start this weekend sorry for the delay. – [percusse](#) Sep 16, 2014 at 1:38
- 13 @percusse: Please never apologize to me. I will never be able to repay all the stuff I learned with your posts here. – [Sergio Parreiras](#) Sep 16, 2014 at 15:04
- 4 @percusse, Sorry to be a pest, but was the arc portion ever completed? Much appreciated! – [aiag](#) Jan 2, 2018 at 0:41



[EDIT on August 13, 2018: better and shorter code, but unfortunately a lot slower now...]

39

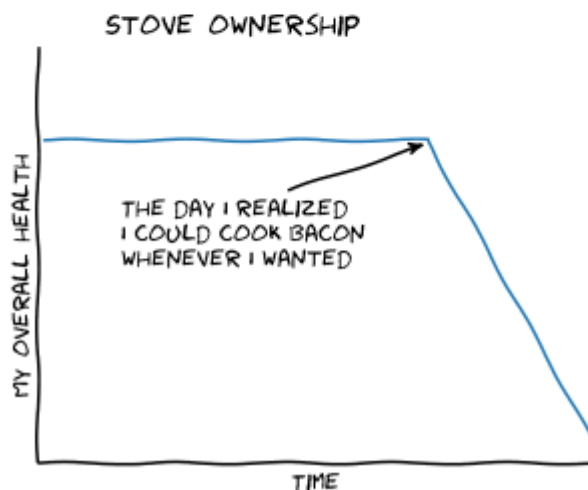
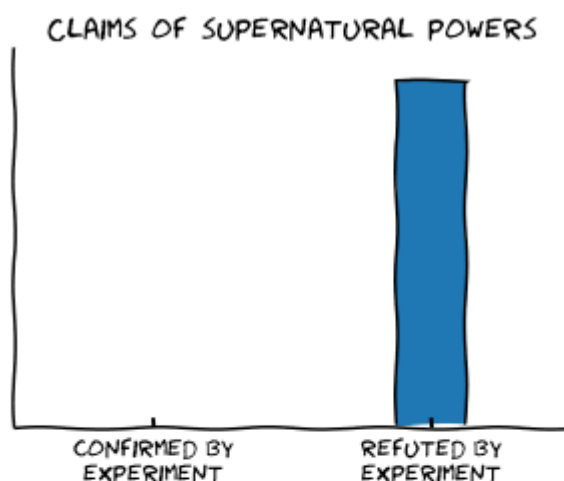
I realize I'm a bit late to the party, but here's my attempt. I tried to implement the ideas they used in `matplotlib` to create an "XKCD" style. Example code and images below.

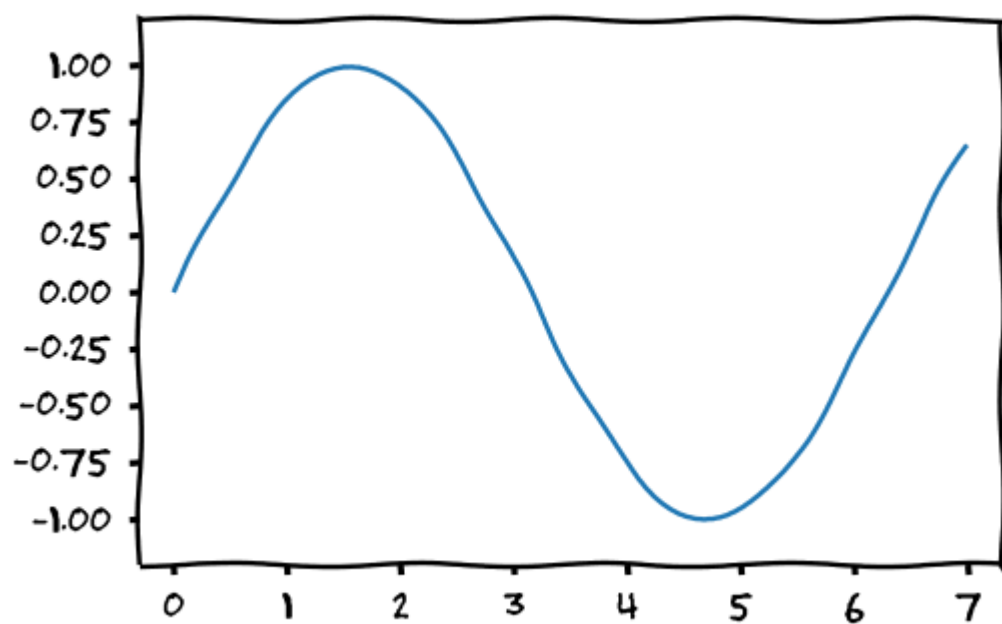
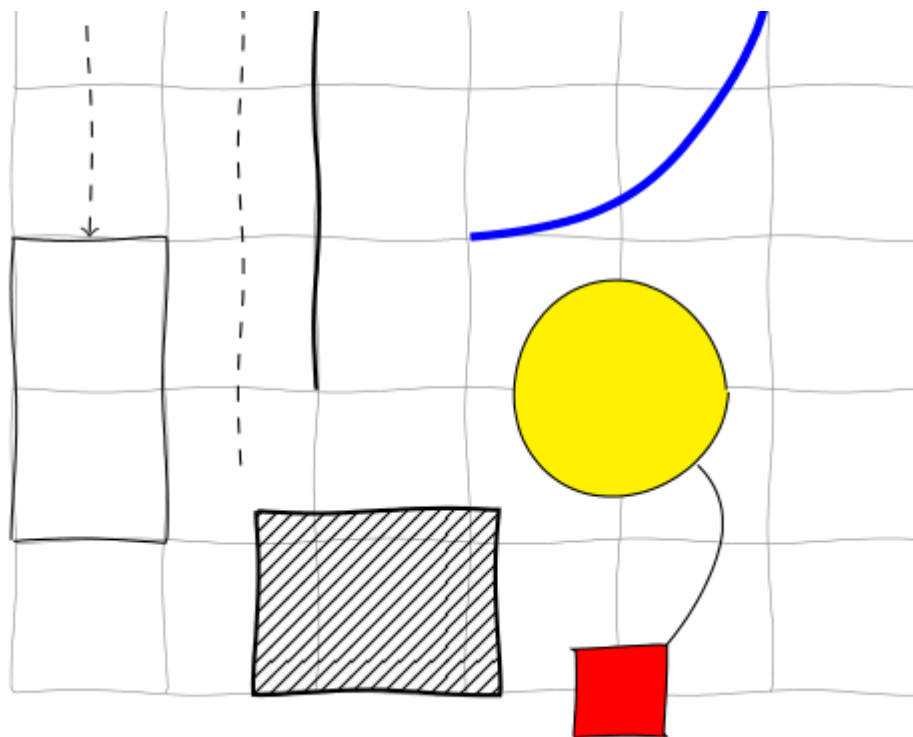


It's the first time I wrote code at the PGF level, so I'm happy to receive corrections and/or suggestions.



The code needs LuaLaTeX for the fonts, but if you remove the font stuff it should work with normal LaTeX as well.





```

\documentclass{article}

\usepackage{fontspec}
\setmainfont{Humor Sans}
\usepackage{tikz}
\usetikzlibrary{calc,decorations,patterns,arrows,decorations.pathmorphing}
\definecolor{pltblue}{HTML}{1F77B4}

\makeatletter
\pgfset{
  /tikz/line cap=round, /tikz/line join=round, /tikz/line width=1pt
}

```

```

/pgf/decoration/randomness/.initial=z,
/pgf/decoration/wavelength/.initial=100
}
\pgfdeclaredecoration{sketch}{init}{
  \state{init}[width=0pt,next state=draw,persistent precomputation={
    \pgfmathsetmacro\pgf@lib@dec@sketch@t0
  }]{
  \state{draw}[width=\pgfdecorationsegmentlength,
    auto corner on length=\pgfdecorationsegmentlength,
    persistent precomputation={
      \pgfmathsetmacro
\pgf@lib@dec@sketch@t{mod(\pgf@lib@dec@sketch@t+pow(\pgfkeysvalueof{/pgf
/decoration/randomness},rand),\pgfkeysvalueof{/pgf/decoration/wavelength})}
    }]{
      \pgfmathparse{sin(2*\pgf@lib@dec@sketch@t*pi/\pgfkeysvalueof{/pgf
/decoration/wavelength} r)}
      \pgfpathlineto{\pgfqpoint{\pgfdecorationsegmentlength}{\pgfmathresult
\pgfdecorationsegmentamplitude}}
    }
  \state{final}{}
}
\tikzset{xkcd/.style={decorate,decoration={sketch,segment
length=0.5pt,amplitude=0.5pt}}}
\makeatother

\pgfmathsetseed{1}

\pagestyle{empty}

\begin{document}
\renewcommand{\baselinestretch}{0.85}
\begin{center}
  \begin{tikzpicture}[xscale=4, yscale=0.05]
    \node at (0.5,115) {\large Claims of supernatural powers};
    \begin{scope}[very thick, every path/.style={xkcd}]
      \fill[fill=pltblue] (0.875,1) -- ++(0,99) -- ++(0.25,0) -- +(0,-99) --
cycle;
      \draw (0.875,1) -- ++(0,99) -- ++(0.25,0) -- +(0,-99);
      \draw[ultra thick] (0,3) -- (0,0) node[below, text width=3cm,
align=center] {confirmed by\ \ experiment};
      \draw[ultra thick] (1,3) -- (1,0) node[below, text width=3cm,
align=center] {refuted by\ \ experiment};
      \draw (-0.5,0) -- (1.5,0);
      \draw (-0.5,0) -- (-0.5,110);
    \end{scope}
  \end{tikzpicture}
  \vspace{2cm}
  \begin{tikzpicture}[xscale=0.08, yscale=0.15]
    \node at (35,12) {\large Stove ownership};
    \node[rotate=90] at (-3,-12) {My overall health};
    \node[text width=4cm] (N) at (40, -8) {The day I realized\ \ I could cook
bacon\ \ whenever I wanted};
    \begin{scope}[very thick, every path/.style={xkcd}]

```

```

\draw (0,-30) -- node[below] {Time} (100,-30);
\draw (0,-30) -- (0,10);
\draw[pltblue] (1,1) -- (70,1) -- (100,-28);
\draw[->, >=stealth'] ($(N.north)+(5cm,0)$) -- (69.5,0.5);
\end{scope}
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
\draw[xkcd,help lines] (-2,-2) grid[step=1cm] (4,4);
\draw[xkcd,thick] (0,0) -- (0,3) -- (3,3);
\draw[xkcd,ultra thick, blue] (3,3) arc (0:-90:2cm);
\draw[xkcd,thick, pattern=north east lines] (-0.4cm,-0.8cm) rectangle
(1.2,-2);
\node[xkcd, draw, inner sep=0.5cm, fill=yellow, circle] (a) at (2,0) {};
\node[xkcd,draw, inner sep=0.3cm, fill=red] (b) at (2,-2) {};
\draw[xkcd] (b) to[in=-45,out=45] (a);
\node[xkcd,draw, minimum height=2cm, minimum width=1cm] (c) at (-1.5,0) {};
\draw[xkcd,->, decoration={post=lineto, post length=5mm}, dashed] (-0.5cm,-
0.5cm) -- (-0.5cm,3.5cm) -| (c.north);
\end{tikzpicture}
\vspace{2cm}
\begin{tikzpicture}[very thick, yscale=2, every path/.style={xkcd}]
\draw (-0.3,-1.2) rectangle (7.3,1.2);
\foreach \y in {-1.00,-0.75,...,1.00} {
\draw[ultra thick] (-0.3,\y) -- (-0.4,\y) node[left]
{\pgfmathprintnumber[fixed,fixed zerofill,precision=2,assume math mode=true]
{\y}};
}
\foreach \x in {0,...,7} {
\draw[ultra thick] (\x,-1.2) -- (\x,-1.25) node[below] {\x};
}
\draw[domain=0:7, pltblue, samples=100] plot (\x, {\sin(\x r)});
\end{tikzpicture}
\end{center}
\end{document}

```

Share Edit Following Flag

edited Aug 13, 2018 at 17:48

answered Aug 11, 2018 at 21:09



Frunobulax

1,900 13 14



68



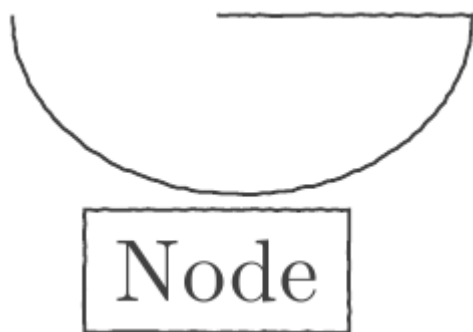
The pgf manual has a tutorial called "Putting it in Chains". It starts with an example, which does what you're looking for.

**EDIT:**

Since the PGF/TikZ manual has a small hyperlink problem due to a typo here is an example of the relevant decoration style `random steps`. You can customize it further by playing around with the `segment length` which adjusts the chunks that go in the random direction and `amplitude` which controls the maximum distance between random step and the main path.

Sample/Simple Example Style:

```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{decorations.pathmorphing}
\begin{document}
\begin{tikzpicture}[pencildraw/.style={
  black!75,
  decorate,
  decoration={random steps,segment length=0.8pt,amplitude=0.1pt}
}]
\node[pencildraw,draw] {Node};
\draw[pencildraw] (0,1) -- (1,1) arc (0:-180:9mm and 7mm);
\end{tikzpicture}
\end{document}
```



Share Edit Follow Flag

edited Jan 10, 2017 at 6:37

answered Dec 25, 2011 at 12:56  
user10274





---

Could you expand on this answer? Putting in some minimal demonstration code would be best, or at the very least pointing to a section or page in the manual (along with the version number). – [qubyte](#) Dec 25, 2011 at 15:52

19



Added a small example. Since the people who end up on this page vary in terms of TikZ skills, it does no harm to have a demoexample at each post. I hope you don't mind. – [percusse](#) Dec 25, 2011 at 17:53



---

Good answer ! and the solution works with circles ! – [Alain Matthes](#) Mar 24, 2012 at 15:49

3



---

I voted to undelete you answer and since two other undelete votes were already casted, now it's undeleted. I think it's a nice answer that deserves to be visible to all the community. – [Gonzalo Medina](#) Sep 17, 2015 at 19:09

---

21

Has anyone noticed how it looks more natural when we just superpose lines? I guess that when we hand drawn we are not only introducing a random component, we some times pass the pencil again, maybe with a different pressure. Here an example:

```
\draw[<-,gray,thick] (0,1) to[bend left=10] ++(1,1);
\draw[<-,gray] (0,1) to[bend left=5] ++(1,1);
```



For instance, I took the code in the answer of @renard and I have just repeated some lines and shapes:

```
\draw[penciline={jag ratio=2},style=help lines,thick] (-2,-2) grid[step=1cm]
(4,4);
\draw[penciline={jag ratio=2},style=help lines] (-2,-2) grid[step=1cm] (4,4);

\draw[penciline={jag ratio=0},thick] (0,0) -- (0,3) -- (3,3);
\draw[penciline={jag ratio=2}] (0,0) -- (0,3) -- (3,3);

\draw[decorate,thick,pattern=north east lines] (-0.4cm,-0.8cm) rectangle
(1.2,-2);
\draw[penciline={jag ratio=0},ultra thick,blue] (3,3) arc (0:-90:2cm);

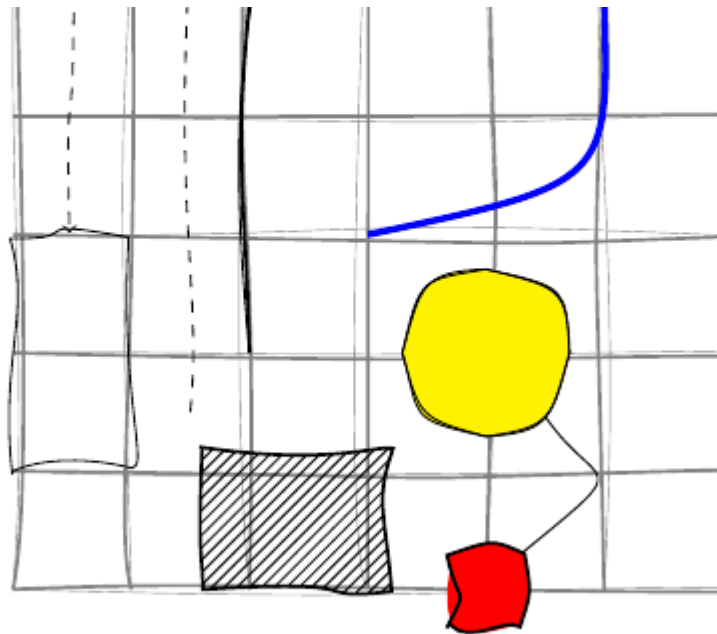
\node[penciline={jag ratio=0},draw,inner sep=0.5cm,fill=yellow,circle] (a) at
(2,0) {};
\node[penciline={jag ratio=0},penciline={jag ratio=0},draw,inner
sep=0.5cm,circle] (a) at (2,0) {};

\node[decorate,inner sep=0.3cm,fill=red] (b) at (2,-2) {};
\node[decorate,draw,thick,inner sep=0.3cm,fill=red] (b) at (2,-2) {};

%% This was supposed to be an edge!!
\draw[decorate] (b) to[in=-45,out=45] (a);
\node[decorate,draw,minimum height=2cm,minimum width=1cm] (c) at (-1.5,0) {};
\draw[decorate,->,dashed] (-0.5cm,-0.5cm) -- (-0.5cm,3.5cm) -| (c.north);
```

Might a mix of the repetition and the randomness of the other solutions increase the natural sensation?. Note that I also have changed a little the different jag ratio or pressure in the repetitions. Here the result:





Share Edit Follow Flag

answered Mar 7, 2016 at 17:17



alexis

481

3

12

I worked on @percusse example and I managed to fix the circle issues. I might be perfectible but I guess the idea is there.

19

```
\documentclass[border=1cm]{standalone}
```

```
\RequirePackage{tikz}
```

```
\usepackage{ifthen}
```

```
\usetikzlibrary{calc,patterns,decorations,plotmarks}
```

```
\usepackage{pgfplots}
```

```
\pgfplotsset{compat=1.8}
```

```
\begin{document}
```

```
\pgfdeclaredecoration{penciline}{initial}{
```

```
  \state{initial}[width=+\pgfdecoratedinputsegmentremainingdistance,
```

```
    auto corner on length=1pt,
```

```
]{
```

```
  \ifthenelse
```

```
    {\pgfkeysvalueof{/tikz/penciline/jag ratio}=0} {
```

```
      \pgfpathcurveto%
```

```
        {% 1st control point
```

```
          \pgfpoint
```

```
            {\pgfdecoratedinputsegmentremainingdistance/2}
```

```
            {2*rnd*\pgfdecorationsegmentamplitude}
```

```
        }
```

```
        {% 2nd control point
```

```
          \pgfpoint
```

```
          %% Make sure random number is always between origin and target
```

```
points
```

```
          {\pgfdecoratedinputsegmentremainingdistance/2}
```

```
          {2*rnd*\pgfdecorationsegmentamplitude}
```

```
        }
```

```
        {% 2nd point (1st one is implicit)
```

```
          \pgfpointadd
```

```
            {\pgfpointdecoratedinputsegmentlast}
```

```
            {\pgfpoint{0*rnd*1pt}{0*rnd*1pt}}
```

```
        }
```

```
    } {
```

```
      \pgfpathcurveto%
```

```
        {% 1st control point
```

```
          \pgfpoint
```

```
            {\pgfdecoratedinputsegmentremainingdistance*rnd*1pt}
```

```
            {\pgfkeysvalueof{/tikz/penciline/jag ratio}*}
```

```
            rand*\pgfdecorationsegmentamplitude}
```

```
        }
```

```
        {% 2nd control point
```

```
          \pgfpoint
```

```
          %% Make sure random number is always between origin and target
```

```
points
```

```

        {(0.5+0.25*rand)*\pgfdecoratedinputsegmentremainingdistance}
        {\pgfkeysvalueof{/tikz/penciline/jag ratio}*
         rand*\pgfdecorationsegmentamplitude}
    }
    {% 2nd point (1st one is implicit)
    \pgfpointadd
    {\pgfpointdecoratedinputsegmentlast}
    {\pgfpoint{rand*1pt}{rand*1pt}}
    }
  }
}
\state{final}{}
}

```

```

\tikzset{
  penciline/.code={\pgfqkeys{/tikz/penciline}{#1}},
  penciline={
    jag ratio/.initial=5,
    decoration/.initial = penciline,
  },
  penciline/.style = {
    decorate,
    %%decoration={\pgfkeysvalueof{/tikz/penciline/decoration}},
    penciline/.cd,
    #1,
    /tikz/.cd,
  },
  decorate,
  decoration={\pgfkeysvalueof{/tikz/penciline/decoration}},
}

```

```

\def\Grid{
  \draw[penciline={jag ratio=6},style=help lines] (-2,-2) grid[step=1cm] (4,4);
}

```

```

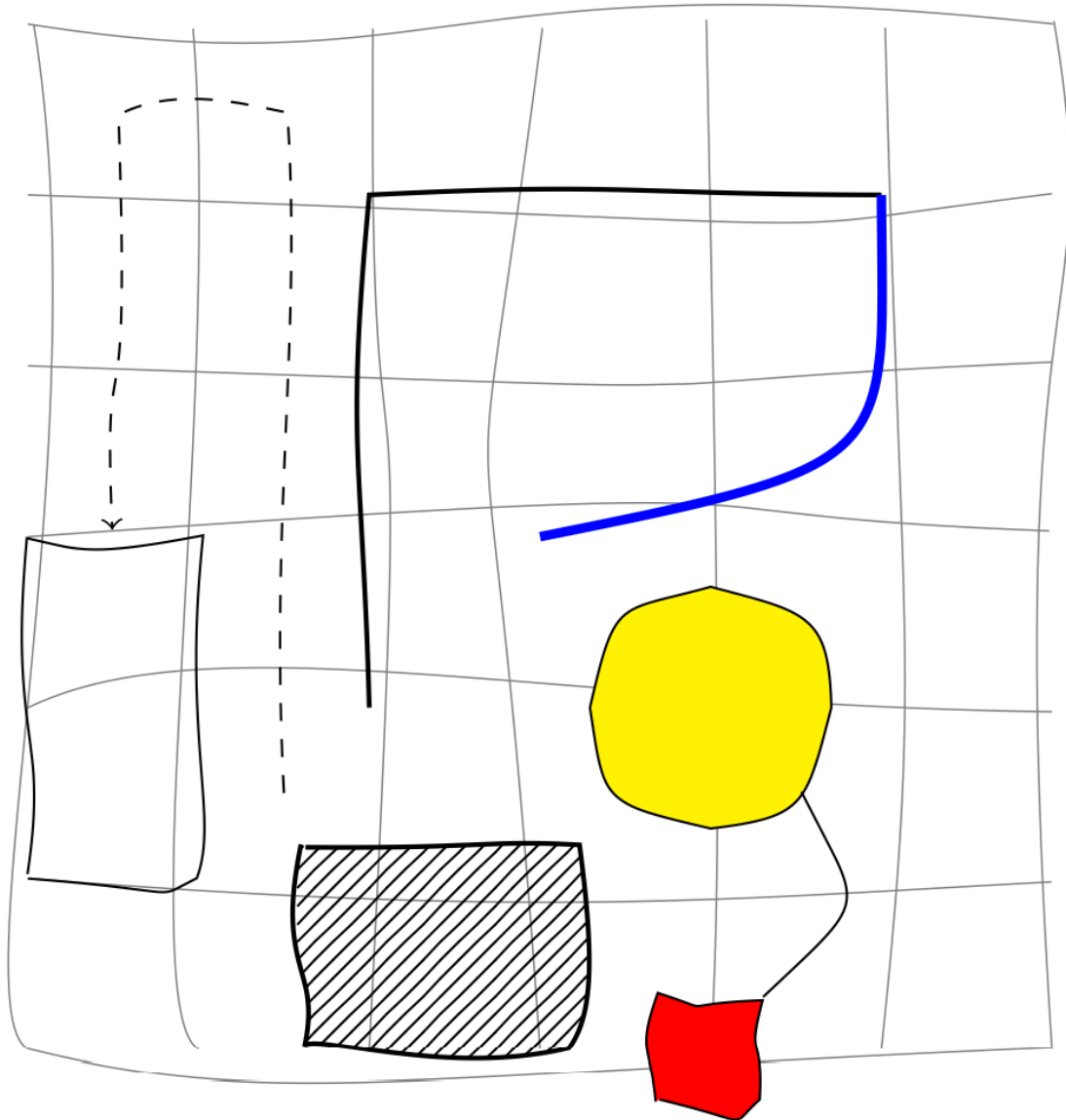
\begin{tikzpicture}[penciline={jag ratio=2}]
  \Grid{}
  \draw[penciline={jag ratio=0},thick] (0,0) -- (0,3) -- (3,3);

  \draw[decorate,thick,pattern=north east lines] (-0.4cm,-0.8cm) rectangle
  (1.2,-2);
  \draw[penciline={jag ratio=0},ultra thick,blue] (3,3) arc (0:-90:2cm);
  \node[penciline={jag ratio=0},draw,inner sep=0.5cm,fill=yellow,circle] (a) at
  (2,0) {};
  \node[decorate,draw,inner sep=0.3cm,fill=red] (b) at (2,-2) {};
  %% This was supposed to be an edge!!
  \draw[decorate] (b) to[in=-45,out=45] (a);
  \node[decorate,draw,minimum height=2cm,minimum width=1cm] (c) at (-1.5,0) {};
  \draw[decorate.->.dashed] (-0.5cm.-0.5cm) -- (-0.5cm.3.5cm) -| (c.north):

```

```
\end{tikzpicture}
```

```
\end{document}
```



**Edit:**

Using test from [Conditional using non-integer numbers](#) allow to use floats for `jag ratio`:

```
...
\ifthenelse
  {\lengthtest{\pgfkeysvalueof{/tikz/penciline/jag ratio}pt = 0pt}} {
    \pgfpathcurveto%
  }
...
```

Share Edit Follow Flag

edited Apr 13, 2017 at 12:35

answered Dec 17, 2014 at 23:10



Community Bot

1



renard

742

5

14

66

This is another attempt to simulate hand drawn curves in Metapost. This is heavily inspired by the MetaFun macros. The main advantage of this approach is that you don't have change your drawing macros at all. Simply `input` the file and call `sketchypaths`, and all paths become a bit sketchy.



Save the following as `mp-sketch.mp`



```
%D The variable \type{sketch_amount} determines the amount of randomness in the
%D drawing
numeric sketch_amount; sketch_amount := 3bp;
```

```
%D The macro \type{sketchdraw} randomized the path before drawing it. The
%D \type{expr} ... \type{text} trick is copied from the definition of
%D \type{drawarrow}
```

```
def sketchdraw expr p =
  do_sketchdraw(p if (path p): randomized sketch_amount fi)
enddef;
```

```
def do_sketchdraw(expr p) text t =
  normaldraw p t ;
enddef;
```

```
%D The macro \type{sketchfill} randomizes the path before filling it.
path _sketch_path_;
```

```
def sketchfill expr p =
  _sketch_path_ := p randomized sketch_amount;
  do_sketchfill
enddef ;
```

```
def do_sketchfill text t =
  normalfill _sketch_path_ t ;
enddef ;
```

```
%D The macro \type{sketchypaths} is modeled after \type{visualizepaths} from
%D \filename{mp-tool}.
```

```
def sketchypaths =
  let draw = sketchdraw ;
  let fill = sketchfill ;
enddef ;
```

The main macro is `sketchypaths` which changes the definition of `draw` and `fill` to be sketchy. To recover the normal behavior use the `naturalizepahts` macro from `metafun`.

To use these, you also need to load `mp-tool.mkiv`. I'll show a test file in ConTeXt, which already loads the MetaFun macros.



```

\startMPinclusions
  input mp-sketch;
\stopMPinclusions

\starttext

\startMPpage[offset=3mm]
  sketchypaths;

  draw (0,0) -- (1cm,0) -- (1cm,1cm);

  fill fullsquare scaled 1cm shifted (3cm,3cm) withcolor red;

  drawarrow (1cm,1cm) -- (2cm,2cm);

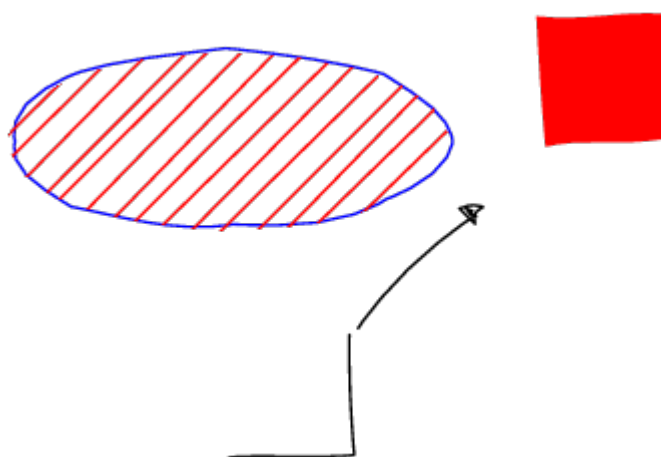
  stripe_path_a
    (withcolor red)
    (draw)
    fullcircle xscaled 100 yscaled 40 shifted (0, 2.5cm) withcolor blue;

\stopMPpage

\stoptext

```

which gives



Since the `sketchypaths` macro changes the `draw` and `fill` commands, it is easy to combine this with existing MetaPost packages. For example, to use `boxes.mp`:

```

\startMPdefinitions
  input mp-sketch;
  input boxes;
\stopMPdefinitions

\starttext
\startMPpage[offset=3mm]
  . . .

```

```

sketchypaths;

boxit.one (btex One etex);
boxit.two (btex Two etex);
boxit.three (btex Three etex);

three.w - two.e = two.w - one.e = (1cm,0);
one.c = origin;

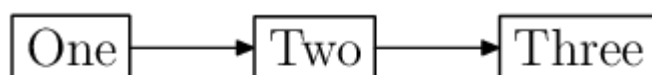
drawboxed (one, two, three);

drawarrow one.e -- lft two.w;
drawarrow two.e -- lft three.w;

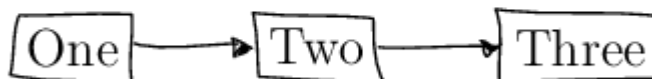
\stopMPpage
\stoptext

```

The figure on the top shows the output without `sketchypaths` and the figure on the bottom shows the output with `sketchyparts`.

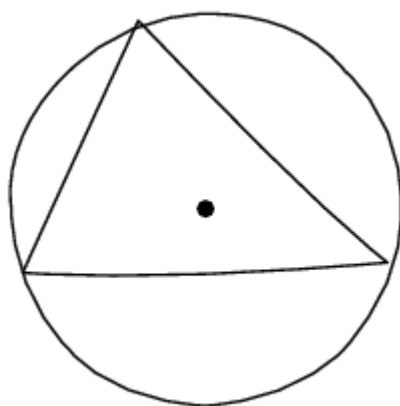


Without `sketchyparts`



With `sketchyparts`

And here is a slightly tweaked version of example 83 of [metapost examples](#)



In my biased opinion, this looks more *natural* than many of the other solutions.

Share Edit Following Flag

edited Feb 4, 2013 at 1:56

answered Feb 4, 2013 at 1:12











Aditya

61 0k

3

127

262

- 
- 16  It looks like a fact to me rather than a bias. – [percusse](#) Feb 4, 2013 at 9:13  

- 
- 4  @percuße now that you have seen its power, you must come over to the dark side :)  
– [Aditya](#) Feb 4, 2013 at 15:07  

- 
- 3  Like the blue-light fly zapper, I'm drawn to it. – [percusse](#) Feb 5, 2013 at 2:50  

- 
- 1  I like this style! But I don't understand how to make it work. How do I use Metapost and  
load `mp-tool.mkiv` in LaTeX? – [StrawberryFieldsForever](#) Sep 29, 2019 at 1:59  

-

24

There are lots of nice answers here addressing how to get that "hand drawn" look with the shape of the lines. I'm going to sketch an approach to getting the texture right. I spent a bit of time this afternoon staring at some lines drawn with a 9B pencil on paper and decided that what it looked most like was a *brass rubbing*. That is, the stroke of the pencil reveals the texture of the paper underneath. This is most like a *fading* in TikZ/PGF parlance.

The cheap and nasty paper that I buy doesn't have much of a grain to it (I have kids, they do art, I buy about 5 reams every six months). The pattern looks more like *noise*. So having learnt about [Perlin noise](#) recently, I thought that would be an appropriate thing to simulate the paper texture with. I soon decided that a TeX implementation would be daft, so went hunting for a Lua version and found one of the closely related [Simplex noise](#) (also due to Perlin; the link goes to a Lua implementation; the author uses strong language both in comments and function names). So I hacked that into a Lua file for LuaTeX and wrote a little demonstration LaTeX file.

I'm calling this an *approach* because it needs considerable work to be useful. For one thing, even though I'm using Lua to do the heavy processing then it takes time to render the fading. So given that noise isn't usually required to be too random, one should do a lot of caching: both of the original generated numbers and the graphic used for the fading itself. There's also a fair bit about fadings that I don't understand, particularly related to how the fading and the picture match up.

What should happen is that one uses *one* fading for the whole picture so that strokes laid over each other end up using the same noise and thus work as though they are on top of the same piece of paper. This would also need a little work to do with positioning.

The code isn't too complicated. Here's the LaTeX file:

```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{fadings}

\directlua{
  dofile('tikzgraphite.lua')
}

\makeatletter
\tikzset{show bounding box/.code={
  \message{^^JBounding box is (\the\pgf@picminx,\the\pgf@picminy) to
(\the\pgf@picmaxx,\the\pgf@picmaxy)^^J}
}}
\makeatother

\begin{document}
\begin{tikzfadingfrompicture}[name=graphite]
\foreach \i in {0,...,160} \foreach \j in {0,...,40}
  \draw[graphite] (\i pt, \j pt) -- (\i+1 pt, \j pt) -- (\i+1 pt, \j+1 pt) -- (\i pt, \j+1 pt) -- cycle;
```

```

\fill[transparent! \directlua{tex.print(math.floor(50*(Noise2D(\i,\j)+1)))}]
(\i pt, \j pt) rectangle (\i + 1 pt, \j + 1 pt);
\end{tikzfadingfrompicture}
\begin{tikzpicture}
\draw[path fading=graphite,fit fading=false,fading transform={shift=
{(70pt,15pt)}}],line width=.5cm,line cap=round,black!75] (0,0) to[bend left]
(5,0);
\tikzset{show bounding box}
\end{tikzpicture}
\end{document}

```

The `show bounding box` key is so that I can get an idea of how big to make the fading. I don't want to scale it as I want a good texture, and I don't want to generate it too big as it takes a loooooonnnnnnggggg time to render.

Here's the lua file:

```

local Gradients3D = {{1,1,0},{-1,1,0},{1,-1,0},{-1,-1,0},
{1,0,1},{-1,0,1},{1,0,-1},{-1,0,-1},
{0,1,1},{0,-1,1},{0,1,-1},{0,-1,-1}}

local simplex = {
{0,1,2,3},{0,1,3,2},{0,0,0,0},{0,2,3,1},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{1,2,3,0},
{0,2,1,3},{0,0,0,0},{0,3,1,2},{0,3,2,1},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{1,3,2,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},
{1,2,0,3},{0,0,0,0},{1,3,0,2},{0,0,0,0},{0,0,0,0},{0,0,0,0},{2,3,0,1},
{2,3,1,0},
{1,0,2,3},{1,0,3,2},{0,0,0,0},{0,0,0,0},{0,0,0,0},{2,0,3,1},{0,0,0,0},
{2,1,3,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},
{2,0,1,3},{0,0,0,0},{0,0,0,0},{0,0,0,0},{3,0,1,2},{3,0,2,1},{0,0,0,0},
{3,1,2,0},
{2,1,0,3},{0,0,0,0},{0,0,0,0},{0,0,0,0},{3,1,0,2},{0,0,0,0},{3,2,0,1},
{3,2,1,0}}

local p = {151,160,137,91,90,15,
131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
130 226 205 02 222 114 67 20 24 72 242 141 120 105 70 66 215 61 156 100}

```

```
130,230,205,95,222,114,07,29,24,72,245,141,120,195,70,00,215,01,150,100,

for i=1,#p do
    p[i-1] = p[i]
    p[i] = nil
end

for i=1,#Gradients3D do
    Gradients3D[i-1] = Gradients3D[i]
    Gradients3D[i] = nil
end

local perm = {}

for i=0,255 do
    perm[i] = p[i]
    perm[i+256] = p[i]
end

function Dot2D(tbl, x, y)
    return tbl[1]*x + tbl[2]*y;
end

local Prev2D = {}

-- 2D simplex noise

function Noise2D(xin, yin)
    if Prev2D[xin] and Prev2D[xin][yin] then return Prev2D[xin][yin] end

    local n0, n1, n2; -- Noise contributions from the three corners
    -- Skew the input space to determine which simplex cell we're in
    local F2 = 0.5*(math.sqrt(3.0)-1.0);
    local s = (xin+yin)*F2; -- Hairy factor for 2D
    local i = math.floor(xin+s);
    local j = math.floor(yin+s);
    local G2 = (3.0-math.sqrt(3.0))/6.0;

    local t = (i+j)*G2;
    local X0 = i-t; -- Unskew the cell origin back to (x,y) space
    local Y0 = j-t;
    local x0 = xin-X0; -- The x,y distances from the cell origin
    local y0 = yin-Y0;

    -- For the 2D case, the simplex shape is an equilateral triangle.
    -- Determine which simplex we are in.
    local i1, j1; -- Offsets for second (middle) corner of simplex in (i,j)
coords
    if(x0>y0) then
        i1=1
        j1=0 -- lower triangle, XY order: (0,0)->(1,0)->(1,1)
    else
        i1=0
```

```

    j1=1 -- upper triangle, YX order: (0,0)->(0,1)->(1,1)
end

-- A step of (1,0) in (i,j) means a step of (1-c,-c) in (x,y), and
-- a step of (0,1) in (i,j) means a step of (-c,1-c) in (x,y), where
-- c = (3-sqrt(3))/6

local x1 = x0 - i1 + G2; -- Offsets for middle corner in (x,y) unskewed
coords
local y1 = y0 - j1 + G2;
local x2 = x0 - 1.0 + 2.0 * G2; -- Offsets for last corner in (x,y)
unskewed coords
local y2 = y0 - 1.0 + 2.0 * G2;

-- Work out the hashed gradient indices of the three simplex corners
local ii = i%256
local jj = j%256
local gi0 = perm[ii+perm[jj]] % 12;
local gi1 = perm[ii+i1+perm[jj+j1]] % 12;
local gi2 = perm[ii+1+perm[jj+1]] % 12;

-- Calculate the contribution from the three corners
local t0 = 0.5 - x0*x0-y0*y0;
if t0<0 then
    n0 = 0.0;
else
    t0 = t0 * t0
    n0 = t0 * t0 * Dot2D(Gradients3D[gi0], x0, y0); -- (x,y) of Gradients3D
used for 2D gradient
end

local t1 = 0.5 - x1*x1-y1*y1;
if (t1<0) then
    n1 = 0.0;
else
    t1 = t1*t1
    n1 = t1 * t1 * Dot2D(Gradients3D[gi1], x1, y1);
end

local t2 = 0.5 - x2*x2-y2*y2;
if (t2<0) then
    n2 = 0.0;
else
    t2 = t2*t2
    n2 = t2 * t2 * Dot2D(Gradients3D[gi2], x2, y2);
end

-- Add contributions from each corner to get the final noise value.
-- The result is scaled to return values in the localerval [-1,1].

local retval = 70.0 * (n0 + n1 + n2)

```

```
if not Prev2D[xin] then Prev2D[xin] = {} end
Prev2D[xin][yin] = retval

return retval;
end
```

It is almost entirely just clipped out from the lua implementation linked above. There's no obvious licence information there, but consider it covered under whatever-licence-it-would-be-there.

Here's the result of that code:



Share Edit Follow Flag

edited Dec 7, 2012 at 20:56

answered Mar 29, 2012 at 21:25



**cJORSSen**

9,709 4 34 121



**Andrew Stacey**

151k 42 377 733

---

4 Did you see [this article](#)? – cJORSSen Dec 7, 2012 at 20:59



@cJORSSen I hadn't seen it, thanks for bringing to my attention. Something to work on, I deem. – Andrew Stacey Dec 8, 2012 at 20:06

---





28



+500



Here are two pens with MetaPost (using ConTeXt format), I feel the code is pretty self-explanatory:

```
\starttext
\startMPenvironment
  color lightgray ; lightgray := (.8,.8,.8) ;
  color darkgray ; darkgray := (.2,.2,.2) ;
\stopMPenvironment

Pen based on a circle:
\startMPcode
  z0 = (0.5cm,1.5cm) ; z1 = (2.5cm,2.5cm) ;
  z2 = (6.5cm,0.5cm) ; z3 = (3.0cm,1.5cm) ;

  pickup pencircle xscaled 2mm yscaled 4mm rotated 30 ;
  draw z0..z1..z2..z3..z0..cycle withcolor lightgray ;
\stopMPcode

\bigskip
Pen based on a diamond:
\startMPcode
  z0 = (0.5cm,1.5cm) ; z1 = (2.5cm,2.5cm) ;
  z2 = (6.5cm,0.5cm) ; z3 = (3.0cm,1.5cm) ;

  pickup pensquare xyscaled 2mm rotated 45 ;
  draw z0..z1..z2..z3...z0..cycle withcolor darkgray ;
\stopMPcode
\stoptext
```

Pen based on a circle:



Pen based on a diamond:



This is the first time I have used MetaPost with ConTeXt, and was happy how to see how easy it is to use.

Share Edit Follow Flag

answered Mar 30, 2012 at 10:20



**morbusg**

**25.2k**

4

78

161

---



55



Only for the fun, I took Marc's suggestion and I create a new decoration free hand with fixed parameters, a new style free hand and a new macro freedraw

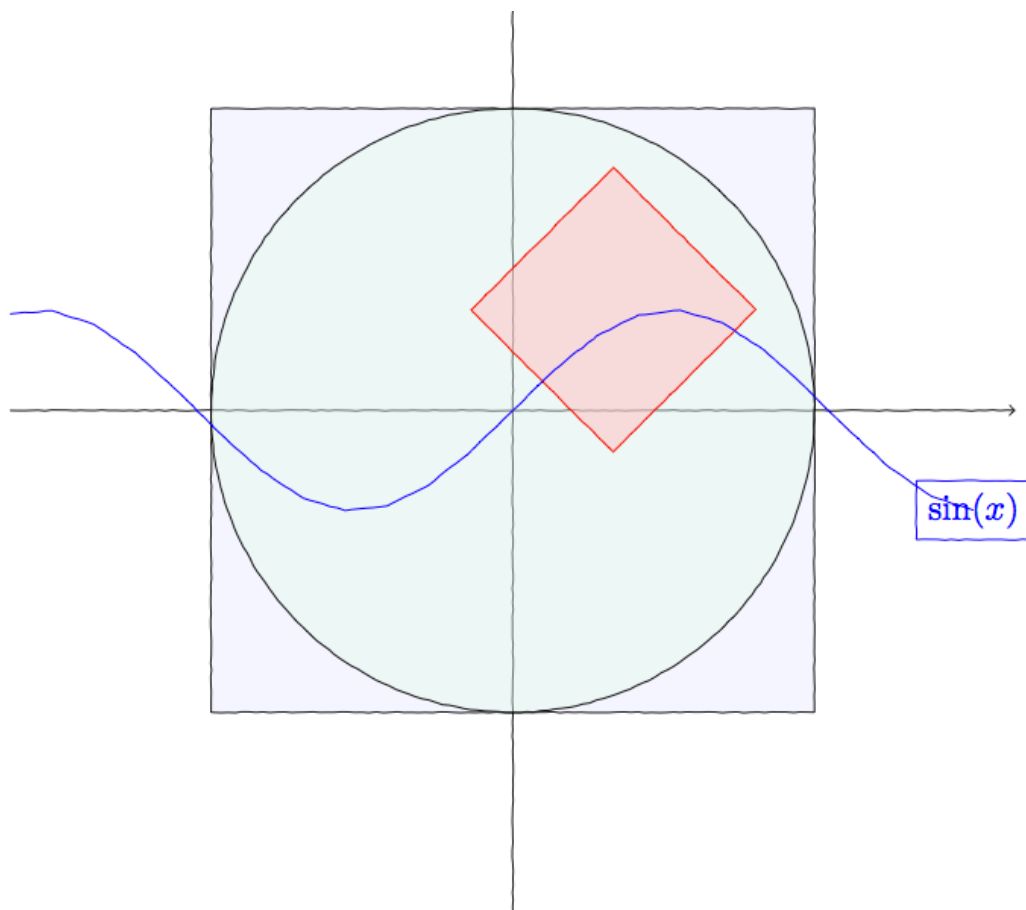
```
\documentclass[border=1cm]{standalone}
\usepackage{tikz}
\usetikzlibrary{decorations.pathmorphing}
\begin{document}

\pgfdeclaredecoration{free hand}{start}
{
  \state{start}[width = +0pt,
    next state=step,
    persistent precomputation = \pgfdecoratepathhascornerstrue]{}
  \state{step}[auto end on length = 3pt,
    auto corner on length = 3pt,
    width=+2pt]
  {
    \pgfpathlineto{
      \pgfpointadd
      {\pgfpoint{2pt}{0pt}}
      {\pgfpoint{rand*0.15pt}{rand*0.15pt}}
    }
  }
  \state{final}
  {}
}
\tikzset{free hand/.style={
  decorate,
  decoration={free hand}
}}
\def\freedraw#1;{\draw[free hand] #1;}

\begin{tikzpicture}
\freedraw [->](-5,0) -- (5,0);
\freedraw [->](0,-5) -- (0,5);

\freedraw[fill=blue!15,fill opacity=.25] (-3,-3) rectangle (3,3);
\freedraw[fill=green!15,fill opacity=.25] circle [radius=3cm];
\freedraw[free hand,red,shift={(1,1)},rotate=45,fill=red!25,fill opacity=.5] %
(1,1) -- (1,-1) -- (-1,-1) -- (-1,1) -- (1,1);
\freedraw[color=blue] plot (\x,{sin(\x r)}) node [free hand,draw] {$\sin(x)$}
;
\end{tikzpicture}
\end{document}
```





Share Edit Follow Flag

answered Mar 24, 2012 at 17:08



**Alain Matthes**

**92.7k**

9

203

339

6

You might want to create a brush (possibly with transparency) and apply it repeatedly on the paths created by the other replies, possibly with some transformations of the brush itself to make it appear even more random. Intensity effects can be achieved with a highly transparent brush and changing "velocity" of the path movement (or changing frequency of applying the brush).

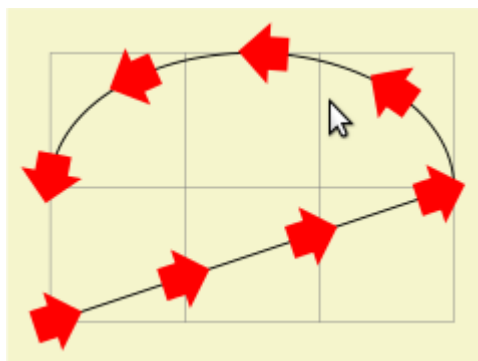
See how a graphite effect is created in this [drawing pad](#), or the airbrush tool in GIMP. I am not an expert in graphics so I cannot give much advice here; my feeling is that you will need to experiment quite a bit to achieve the desired result. (On that note: A sample drawing would help understand your question better.)

You can use marking decorations (Section 30.4 in the PGF manual). There is also an example that draws arrows along a path, so I'm pretty sure you can do the same thing with a vector or bitmap brush. This has been borrowed from the manual:

```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and 1 step 1cm
  with { \node [single arrow,fill=red,
    single arrow head extend=3pt,transform shape] {};}]}

  \draw [help lines] grid (3,2);
  \draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);

\end{tikzpicture}
```



The size of the PDF will not be affected much because TikZ can be instructed to load the brush only once. However, rendering may take quite a while.

Share Edit Follow Flag

edited Mar 23, 2012 at 22:56

answered Mar 23, 2012 at 22:50



krlmlr

12.3k

9

65

116



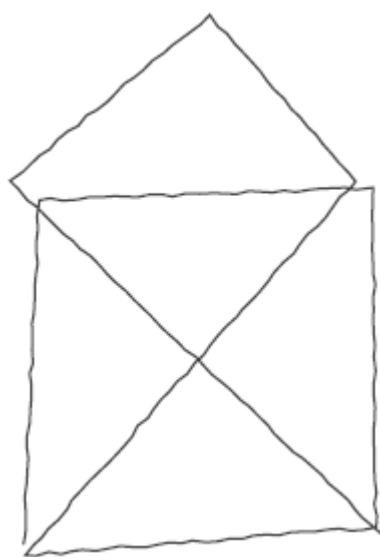
24



Based on Marcs solution I tried to mimic the ps-tricks results (randomized/imprecise coordinates) shown by Werner using tikz:

```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{decorations.pathmorphing}
\begin{document}
\begin{tikzpicture}[pencildraw/.style={
  black!75,
  decorate,
  decoration={random steps,segment length=2pt,amplitude=0.3pt},
  to path={ -- ([xshift=rand*1mm, yshift=rand*1mm] \tikztotarget)
\tikztonodes}
}
]
\draw[pencildraw] (0,0) to (0,2) to (2,2) to (2,0) to (0,0) to (2,2) to
(1,3) to (0,2) to (2,0);

\end{tikzpicture}
\end{document}
```



Maybe someone can improve this, i.e. find a more general way to randomize coordinates, as this solution for example will not randomize all corners of a rectangle automatically.

Share Edit Follow Flag

edited Dec 27, 2011 at 20:33

answered Dec 26, 2011 at 18:40



Christian

1,781 10 7

In terms of regular (straight) lines, `pstricks-add` offers

35

`\pslineByHand`[<options>](<x1>,<y1>)(<x2>,<y2>)(<x3>,<y3>)...

that draws a straight line with some "jagging." The "jagging" parameters that influence the line drawing are `VarStepEpsilon` (default is 2) and `varsteptol` (default is 0.8).



```
\documentclass{article}
\usepackage{xcolor}% http://ctan.org/pkg/xcolor
\usepackage{pstricks-add}% http://ctan.org/pkg/pstricks-add
\begin{document}
\begin{pspicture}(11,3)
  \psset{unit=1cm,linewidth=0.5pt,linecolor=black!70,linejoin=1}
  \psline(0,0)(0,2)(2,2)(2,0)(0,0)(2,2)(1,3)(0,2)(2,0)% Regular HOUSE with
straight lines
  \rput(3,0){\pslineByHand(0,0)(0,2)(2,2)(2,0)(0,0)(2,2)(1,3)(0,2)(2,0)}
  \rput(6,0){\pslineByHand[VarStepEpsilon=4](0,0)(0,2)(2,2)(2,0)(0,0)(2,2)
(1,3)(0,2)(2,0)}
  \rput(9,0){\pslineByHand[varsteptol=0.2](0,0)(0,2)(2,2)(2,0)(0,0)(2,2)
(1,3)(0,2)(2,0)}
\end{pspicture}
\end{document}
```

This requires a `latex -> dvips -> ps2pdf` or `xelatex` compile sequence.

Share Edit Follow Flag

answered Dec 26, 2011 at 2:12



Werner

587k 131 1382  
2260



0



 **This post is hidden.** It was [deleted](#) 8 years ago by [Stefan Kottwitz](#) ♦.

Since I can't add a comment, I'll post an answer instead. Can someone translate the answer provided by Aditya into LaTeX as I don't use ConTeXT and I'm not good enough with LaTeX to translate it myself.

Share Edit Follow Flag

answered Feb 14, 2015 at 11:29  
user72335



I think you should ask this as a separate question, in its own topic... – [Franck Pastor](#) Feb 14, 2015 at 11:35




There's a reason in the restriction regarding commenting. For sure it's not for getting fake answers, so it will be removed. – [Stefan Kottwitz](#) ♦ Feb 14, 2015 at 11:40

Comments disabled on deleted / locked posts / reviews



0



 **This post is hidden.** It was [deleted](#) 8 years ago by [Heiko Oberdiek](#), [Werner](#) ♦, [Claudio Fiandrino](#).

I found example here very useful.

I am planning to make a package with both `pencile` and `free hand`.

@percusse, alain-matthes: What are the license of these codes? is it possible to publish them under LaTeX Project Public License?

Thanks in advance. Cheers.

Share Edit Follow Flag

answered Dec 15, 2014 at 16:43



[renard](#)

742

5

14

1



Look at my profile page for the license ;) Oh and please don't put comments in the answers area. You can always come to the chat. Finally, if I can find time I'll make the decoration for the arc shapes too. – [percusse](#) Dec 15, 2014 at 16:52

1



This does not provide an answer to the question. To critique or request clarification from an author, leave a comment below their post – you can always comment on your own posts, and once you have sufficient [reputation](#) you will be able to [comment on any post](#).  
– [Heiko Oberdiek](#) Dec 15, 2014 at 17:06

Comments disabled on deleted / locked posts / reviews