pst-slpe package
version 1.0

Martin Giese[*]

98/09/15

1 Introduction

As of the 97 release, PSTricks contains the `pst-grad` package, which provides a gradient fill style for arbitrary shapes. Although it often produces nice results, it has a number of deficiencies:

1. It is not possible to go from a colour A to B to C, etc. The most evident application of such a multi-colour gradient are of course rainbow effects. But they can also be useful in informative contexts, eg to identify modes of operation in a scale of values (normal/danger/overload).

2. Colours are interpolated linearly in the RGB space. This is often OK, but when you want to go from red $(1, 0, 0)$ to green $(0, 1, 0)$, it looks much better to get there via yellow $(1, 1, 0)$ than via brown $(0.5, 0.5, 0)$. The point is, that to get from one saturated colour to another, the colours on the way should also be saturated to produce an optically pleasing result.

3. `pst-grad` is limited to *linear* gradients, ie there is a (possibly rotated) rectilinear coordinate system, such that the colour at every point depends only on the x coordinate of the point. In particular, there is no way to get circular patterns.

`pst-slpe` solves *all* of the mentioned problems in *one* package.

Problems 1. is addressed by permitting the user to specify an arbitrary number of colours, along with the points at which these are to be reached. A special form of each of the fill styles is provided, which just needs two colours as parameters, and goes from one to the other. This makes the fill styles easier to use in that simple case.

Problem 2. is solved by interpolating in the hue-saturation-value colour space. Conversion between RGB and HSV is done behind the scenes. The user specifies colours in RGB.

Finally, `pst-slpe` provides *concentric* and *radial* gradients. What these mean is best explained with a polar coordinate system: In a concentric pattern, the colour of a point depends on the radius coordinate, while in a radial pattern, it depends on the angle coordinate.

[*]email:`giese@ira.uka.de`

As a special bonus, the PostScript part of `pst-slpe` is somewhat optimized for speed. In `ghostscript`, rendering is about 30% faster than with `pst-grad`.

For most of these problems, solutions have been posted in the appropriate TEX newsgroup over the years. `pst-slpe` has however been developed independently from these proposals. It is based on the original PSTricks 0.93 `gradient` code, most of which has been changed or replaced. The author is indebted to Denis Girou, whose encouragement triggered the process of making this a shipable package instead of a private experiment.

The new fill styles and the graphics parameters provided to use them are described in section 2 of this document. Section 3, if present, documents the implementation consisting of a generic TEX file and a PostScript header for the `dvi`-to-PostScript converter. You can get section 3 by calling LATEX as follows on most relevant systems:

```
latex '\AtBeginDocument{\AlsoImplementation}\input{pst-slpe.dtx}'
```

2 Package Usage

To use `pst-slpe`, you have to say

```
    \usepackage{pst-slpe}
```

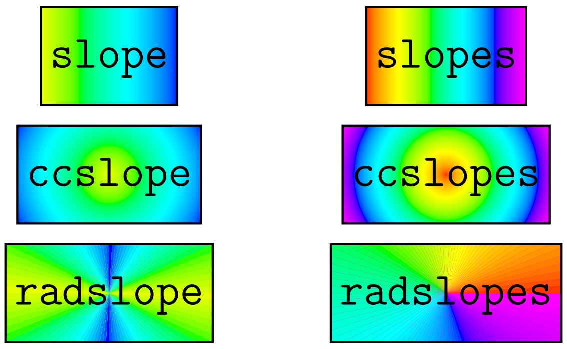
in the document prologue for LATEX, and

```
    \input pst-slpe.tex
```

in “plain” TEX.

`slope` `slopes` `ccslope` `ccslopes` `radslope` `radslopes`

`pst-slpe` provides six new fill styles called `slope`, `slopes`, `ccslope`, `ccslopes`, `radslope` and `radslopes`. These obviously come in pairs: The . . .`slope`-styles are simplified versions of the general . . .`slopes`-styles.[1] The `cc`. . . styles paint concentric patterns, and the `rad`. . . styles do radial ones. Here is a little overview of what they look like:



These examples were produced by saying simply

```
    \psframebox[fillstyle=slope]{...}
```

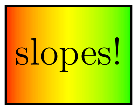
[1]By the way, I use slope as a synonym for gradient. It sounds less pretentious and avoids name clashes.

etc. without setting any further graphics parameters. The package provides a number of parameters that can be used to control the way these patterns are painted.

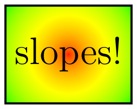
`slopebegin` `slopeend` The graphics parameters `slopebegin` and `slopeend` set the colours between which the three ...`slope` styles should interpolate. Eg,

```
\psframebox[fillstyle=slope,slopebegin=red,slopeend=green]{...}
```

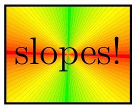
produces:



The same settings of `slopebegin` and `slopeend` for the `ccslope` and `radslope` fillstyles produce

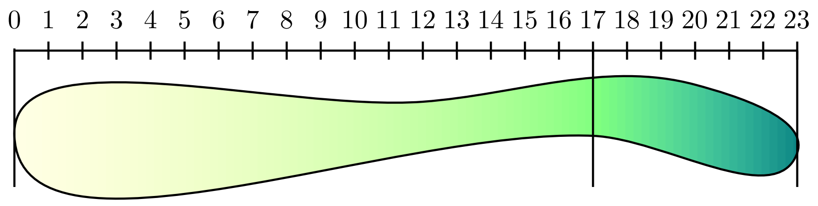


resp.



The default settings go from a greenish yellow to pure blue.

`slopecolors` If you want to interpolate between more than two colours, you have to use the ...`slopes` styles, which are controlled by the `slopecolors` parameter instead of `slopebegin` and `slopeend`. The idea is to specify the colour to use at certain points 'on the way'. To fill a shape with `slopes`, imagine a linear scale from its left edge to its right edge. The left edge must lie at coordinate 0. Pick an arbitrary value for the right edge, say 23. Now you want to get light yellow at the left edge, a pastel green at 17/23 of the way and dark cyan at the right edge, like this:



The RGB values for the three colours are $(1, 1, 0.9)$, $(0.5, 1, 0.5)$ and $(0, 0.5, 0.5)$. The value for the `slopecolors` parameter is a list of 'colour infos' followed by the number of 'colour infos'. Each 'colour info' consists of the coordinate value where a colour is to be specified, followed by the RGB values of that colour. All these values are separated by white space. The correct setting for the example is thus:

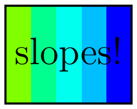
```
slopecolors=0 1 1 .9   17 .5 1 .5   23 0 .5 .5   3
```

For `ccslopes`, specify the colours from the center outward. For `radslopes` (with no rotation specified), 0 represents the ray going 'eastward'. Specify the colours anti-clockwise. If you want a smooth gradient at the beginning and starting ray of `radslopes`, you should pick the first and last colours identical.

Please note, that the `slopecolors` parameter is not subject to any parsing on the TEX side. If you forget a number or specify the wrong number of segments, the PostScript interpreter will probably crash.

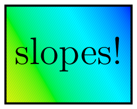
The default value for `slopecolors` specifies a rainbow.

`slopesteps` The parameter `slopesteps` controls the number of distinct colour steps rendered. Higher values for this parameter result in better quality but proportionally slower rendering. Eg, setting `slopesteps` to 5 with the `slope` fill style results in

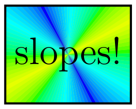


The default value is 100, which suffices for most purposes. Remember that the number of distinct colours reproducible by a given device is limited. Pushing `slopesteps` to high will result only in loss of performance at no gain in quality.

`slopeangle` The `slope(s)` and `radslope(s)` patterns may be rotated. As usual, the angles are given anti-clockwise. Eg, an angle of 30 degrees gives

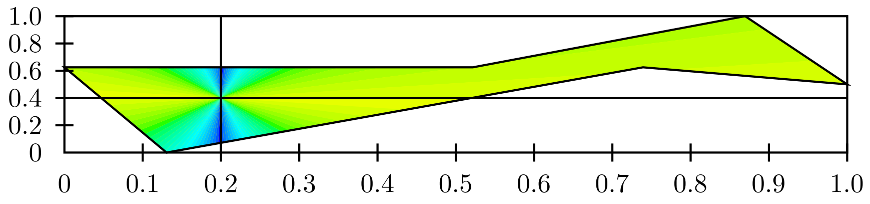


and



with the `slope` and `radslope` fillstyles.

`slopecenter` For the `cc`... and `rad`... styles, it is possible to set the center of the pattern. The `slopecenter` parameter is set to the coordinates of that center relative to the bounding box of the current path. The following effect:

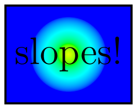


was achieved with

```
fillstyle=radslope,slopecenter=0.2 0.4
```

The default value for `slopecenter` is `0.5 0.5`, which is the center for symmetrical shapes. Note that this parameter is not parsed by TEX, so setting it to anything else than two numbers between 0 and 1 might crash the PostScript interpreter.

`sloperadius` Normally, the `cc`... and `rad`... styles distribute the given colours so that the center is painted in the first colour given, and the points of the shape furthest from the center are painted in the last colour. In other words the maximum radius to which the `slopecolors` parameter refers is the maximum distance from the center (defined by `slopecenter`) to any point on the periphery of the shape. This radius can be explicitly set with `sloperadius`. Eg, setting `sloperadius=0.5cm` gives



Any point further from the center than the given `sloperadius` is painted with the last colour in `slopeclours`, resp. `slopeend`.

The default value for `sloperadius` is 0, which invokes the default behaviour of automatically calculating the radius.