Ipe

Release 7.2.28

Otfried Cheong

Aug 13, 2023

CONTENTS:

Preparing figures for a scientific article is a time-consuming process. If you are using the LaTeX document preparation system in an environment where you can include PDF figures, then the extensible drawing editor Ipe may be able to help you in the task. Ipe allows you to prepare and edit drawings containing a variety of basic geometry primitives like lines, splines, polygons, circles etc.

Ipe also allows you to add text to your drawings, and unlike most other drawing programs, Ipe treats these text object as LaTeX text. This has the advantage that all usual LaTeX commands can be used within the drawing, which makes the inclusion of mathematical formulae (or even simple labels like q_i) much simpler. Ipe processes your LaTeX source and includes the PDF rendering produced by LaTeX in the figure.

In addition, Ipe offers you some editing functions that can usually only be found in professional drawing programs or CAD systems. For instance, it incorporates a context sensitive snapping mechanism, which allows you to draw objects meeting in a point, having parallel edges, objects aligned on intersection points of other objects, rectilinear and c-oriented objects and the like. Whenever one of the snapping modes is enabled, Ipe shows you *Fifi*, a secondary cursor, which keeps track of the current aligning.

One of the nicest features of Ipe is the fact that it is *extensible*. You can write your own functions, so-called *ipelets*. Once registered with Ipe by adding them to your ipelet path, you can use those functions like Ipe's own editing functions. (In fact, some of the functions in the standard Ipe distribution are actually implemented as ipelets.) Ipelets can be written in Lua, an easy-to-learn interpreted language that is embedded into Ipe, or also in C++. Among others, there is an ipelet to compute Voronoi diagrams.

Making a presentation is another task that requires drawing figures. You can use Ipe to prepare presentations in PDF format. Ipe offers many features to make attractive presentations.

Ipe tries to be self-explanatory. There is online help available, and most commands tell you about options, shortcuts, or errors. Nevertheless, it would probably be wise to read at least a few sections of this manual. The chapter on general concepts and the chapter explaining the snapping functions would be a useful read. If you want to use Ipe to prepare presentations, you should also read the *Presentations* section.

CHAPTER

ONE

ABOUT IPE FILES

Ipe 7 creates PDF files. These files can be used in any way that PDF files are used, such as viewed with a PDF viewer, edited with a PDF editor, or included in Latex/Pdflatex documents. However, Ipe cannot read arbitrary PDF files, only files it has created itself. This is because files created by Ipe contain a special hidden stream that describes the Ipe objects. (So if you edit your Ipe-generated PDF file in a different program such as Adobe Acrobat or MacOS Preview, Ipe will not be able to read the file again afterwards.)

You decide in what format to store a figure when saving it for the first time. Ipe gives you the option of saving with extensions `pdf` (PDF), and `ipe` (XML). Files saved with extension `ipe` are XML files and contain no PDF information. The precise XML format used by Ipe is documented *later in this manual*. XML files can be read by any XML-aware parser, and it is easy for other programs to generate XML output to be read by Ipe. You probably don't want to keep your figures in XML format, but it is excellent for communicating with other programs, and for converting figures between programs.

There are a few interesting uses for Ipe documents:

1.1 Figures for Latex documents

Ipe was originally written to make it easy to make figures for Latex documents. If you are not familiar with including figures in Latex, you can find details in the *Using Ipe figures in Latex* section.

1.2 Presentations

Ipe is not a presentation tool like Powerpoint or Keynote. An Ipe presentation is simply a PDF file that has been created by Ipe, and which you present during your talk using any PDF reader (for instance Acrobat Reader).

However, Ipe now comes with a presentation tool IpePresenter to make presentations more comfortable. The *Presentations* section explains Ipe features meant for making presentations.

1.3 SVG files

Figures in SVG format can be used to include scalable figures in web pages. Ipe does not save in SVG format directly, but the tool **`iperender`** allows you to convert an Ipe document to SVG format. This conversion is one-way, although the auxiliary tool **`svgtoipe`** also allows you to convert SVG figures to Ipe format.

1.4 Bitmaps

Sometimes Ipe can be useful for creating bitmap images. Again, the **iperender** tool can render an Ipe document as a bitmap in PNG format.

CHAPTER

TWO

GENERAL CONCEPTS

After you start up Ipe, you will see a window with a large gray area containing a white rectangle. This area, the *canvas*, is the drawing area where you will create your figures. The white rectangle is your *sheet of paper*, the first page of your document. (While Ipe doesn't stop you from drawing outside the paper, such documents generally do not print very well.)

At the top of the window, above the canvas, you find two toolbars: one for snapping modes, grid size and angular resolution; and another one to select the current mode.

On the left hand side of the canvas you find an area where you can select object properties such as stroke and fill color, pen width, path properties, text size, and mark size. Below it is a list of the *layers* of the current page.

All user interface elements have tool tips - if you move the mouse over them and wait a few seconds, a short explanation will appear.

The mode toolbar allows you to set the current **Ipe mode**. Roughly speaking, the mode determines what the left mouse button will do when you click it in the figure. The leftmost five buttons select modes for selecting and transforming objects, the remaining buttons select modes for creating new objects.

Pressing the right mouse button pops up the object context menu in any mode.

In this chapter we will discuss the general concepts of Ipe. Understanding these properly will be essential if you want to get the most out of Ipe.

2.1 Order of objects

An Ipe drawing is a sequence of geometric objects. The order of the objects is important—wherever two objects overlap, the object which comes first in Ipe's sequence will hide the other ones. When new objects are created, they are added in *front* of all other objects. However, you can change the position of an object by putting it in front or in the back, using *Edit* → *Front* and *Edit* → *Back*.

2.2 The current selection

Whenever you call an Ipe function, you have to specify which objects the function should operate on. This is done by *selecting* objects. The selected objects (the *selection*) consists of two parts: the *primary* selection consists of exactly one object (of course, this object could be a group). All additional selected objects form the *secondary* selection. Some functions (like the context menu) operate only on the primary selection, while others treat primary and secondary selections differently (the align functions, for instance, align the secondary selections with respect to the primary selection.)

The selection is shown by outlining the selected object in color. Note that the primary selection is shown with a slightly different look.

The primary and secondary selections can be set in selection mode. Clicking the left mouse button close to an object makes that object the primary selection and deselects all other objects. If you keep the `Shift` key pressed while clicking with the left mouse key, the object closest to the mouse will be added to or deleted from the current selection. You can also drag a rectangle with the mouse—when you release the mouse button, all objects inside the rectangle will be selected. With the `Shift` key, the selection status of all objects inside the rectangle will be switched.

To make it easier to select objects that are below or close to other objects, it is convenient to understand exactly how selecting objects works. In fact, when you press the mouse button, a list of all objects is computed that are sufficiently close to the mouse position (the exact distance is set as the `select_distance` in `prefs.lua`). This list is then sorted by increasing distance from the mouse and by increasing depth in the drawing. If `Shift` was not pressed, the current selection is now cleared. Then the first object in the list is presented. Now, while still keeping the mouse button pressed, you can use the `Space` key to step through the list of objects near the mouse in order of increasing depth and distance. When you release the right mouse button, the object is selected (or deselected).

When measuring the distance from the mouse position to objects, Ipe considers the boundary of objects only. So to select a filled object, don't just click somewhere in its interior, but close to its boundary.

Another way to select objects is using the *Select all* function from the *Edit* menu. It selects all objects on the page. Similarly, the *Select all in layer* function in the *Layer* menu selects all objects in the active layer.

2.3 Moving and scaling objects

There are four modes for transforming objects: *translate*, *stretch*, *rotate*, and *shear*. If you hold the shift key while pressing the left mouse button, the stretch function keeps the aspect ratio of the objects (an operation we call *scaling*), and the translate function is restricted to horizontal and vertical translations.

Normally, the transformation functions work on the current selection. However, to make it more convenient to move around many different objects, there is an exception: When the mouse button is pressed while there is no current selection, then the object closest to the cursor is moved, rotated, scaled, or sheared.

By default, the *rotate* function rotates around the center of the bounding box for the selected objects. This behavior can be overridden by specifying an *axis system*. If an axis system is set, then the origin is used as the center.

The *scale*, *stretch*, and *shear* functions use a corner of the bounding box for the selected objects as the fix-point of the transformation. Again, if an axis system is set, the origin of the system is used instead. In this case, the *shear* function will use your x-axis as the fixed axis of the shear operation (rather than a horizontal line).

It is often convenient to rotate or scale around a vertex of an object. This is easy to achieve by setting the origin of the axis system to that vertex, using the *Snap to vertex function* for setting the axis system.

2.4 Stroke and fill colors

Path objects can have two different colors, one for the boundary and one for the interior of the object. The Postscript/PDF terms *stroke* and *fill* are used to denote these two colors. Stroke and fill color can be selected independently in the *Properties* window. Imagine preparing a drawing by hand, using a pen and black ink. What Ipe draws in its *stroke* color is what you would stroke in black ink with your pen. Probably you would not use your pen to fill objects, but you would use a brush, and maybe even a different kind of paint like water color. Well, the *fill* color is Ipe's "brush."

When you create a path object, you'll have to tell Ipe whether you want it stroked, filled, or both. This is set in the *Path properties* field. Clicking near the right end of the field will cycle through the three modes *stroked*, *stroked & filled*, and *filled*. You can also use the context menu of the path properties field.

Text objects and arrows only use the stroke color, even for the filled arrows. You would also use a pen for these details, not the brush.

The mark shapes *disk* and *square* also use only the stroke color. You can make bicolored marks using the mark shapes *fdisk* and *fsquare*.

2.5 Pen, dash style, arrows, and tiling patterns

The *path properties* field is used to set all properties of path objects except for the pen width, which is set using the selector just above the path properties field. The dash-dot pattern (solid line, dashed, dotted etc.) effect for the boundaries of *path objects*, such as polygons and polygonal lines, splines, circles and ellipses, rectangles and circular arcs. It does not effect text or marks.

Line width is given in Postscript/PDF points (1/72 inch). A good value is something around 0.4 or 0.6.

By clicking near the ends of the segment shown in the path properties field, you can toggle the front and rear arrows. Only polygonal lines, splines, and circular arcs can have arrows.

If you draw a single line segment with arrows and set it to *filled only*, then the arrows will be drawn using the fill color (instead of the stroke color), and the segment is not drawn at all. This is sometimes useful to place arrows that do not appear at the end of a curve.

Various shapes and sizes of arrows are available through the context menu in the path properties field. You can add other shapes and sizes using a stylesheet.

The arrow shapes *arc* and *farc* are special. When the final segment of a path object is a circular arc, then these arcs take on a curved shape that depends on the radius of the arc. They are designed to look right even for arcs with rather small radius.

The arrow shapes whose name starts with *mid* are also special: Those arrows are not drawn at the endpoint of a curve, but at its midpoint. (This is currently only implemented for polylines, that is curves that do not contain circular arcs or splines.)

A tiling pattern allows you to hatch a path object instead of filling it with a solid color. Only path objects can be filled with a tiling pattern. The pattern defines the slope, thickness, and density of the hatching lines, their color is taken from the object's fill color. You can select a tiling pattern using the context menu in the path properties field. You can define your own tiling patterns in the documents stylesheet.

2.6 Transparency

Ipe supports a simple model of transparency. You can set the opacity of path objects, text objects, and images: an opacity of 1.0 means a fully opaque object, while 0.5 would mean that the object is half-transparent. All opacity values you wish to use in a document must be defined in its stylesheet.

2.7 Symbolic and absolute attributes

Attributes such as color, line width, pen, mark size, or text size, can be either absolute (a number, or a set of numbers specifying a color) or symbolic (a name). Symbolic attributes must be translated to absolute values by a *stylesheet* for rendering.

One purpose of stylesheets is to be able to reuse figures from articles in presentations. Obviously, the figure has to use much larger fonts, markers, arrows, and fatter lines in the presentation. If the original figure used symbolic attributes, this can be achieved by simply swapping the stylesheet for another one.

The Ipe user interface is tuned for using symbolic attribute values. You can use absolute colors, pen width, text size, and mark size by clicking the button to the left of the selector for the symbolic names.

When creating an object, it takes its attributes from the current user interface settings, so if you have selected an absolute value, it will get an absolute attribute. Absolute attributes have the advantage that you are free to choose any value you wish, including picking arbitrary colors using a color chooser. In symbolic mode, you can only use the choices provided by the current *stylesheet*.

The choices for symbolic attributes provided in the Ipe user interface are taken from your stylesheet.

2.8 Zoom and pan

You can zoom in and out the current drawing using a mouse wheel or the zoom functions. The minimum and maximum resolution can be customized. Ipe displays the current resolution at the bottom right (behind the mouse coordinates).

Related are the functions *Normal size* (which sets the resolution to 72 pixels per inch), *Fit page* (which chooses the resolution so that the current page fills the canvas), *Fit objects* (which chooses the resolution such that the objects on the page fill the screen), and *Fit selection* (which does the same for the selected objects only). All of these are in the *Zoom* menu.

You can *pan* the drawing either with the mouse in *Pan* mode, or by pressing the `x` key (*here*) with the mouse anywhere on the canvas. The drawing is then panned such that the cursor position is moved to the center of the canvas. This shortcut has the advantage that it also works while you are in the middle of any drawing operation. Since the same holds for the *zoom in* and *zoom out* buttons and keys, you can home in on any feature of your drawing *while* you are adding or editing another object.

2.9 Groups

It is often convenient to treat a collection of objects as a single object. This can be achieved by *grouping* objects. The result is a geometric object, which can be moved, scaled, rotated etc. as a whole. To edit its parts or to move parts of it with respect to others, however, you have to *un-group* the object, which decomposes it into its component objects. To un-group a group object, select it, bring up the object menu, and select the *Ungroup* function.

Group objects can be elements of other groups, so you can create a hierarchy of objects.

A second function of groups is that they allow you to add additional information to an object or group of objects. In particular, group objects allow you to set a *clipping path* on part of your drawing, to *create links* to external documents or websites, and to *decorate* objects. See *group objects* for details.

2.10 Layers

A page of an Ipe document consists of one or more layers. Each object on the page belongs to a layer. There is no relationship between layers and the *back-to-front ordering* of objects, so the layer is really just an attribute of the object.

The layers of the current page are displayed in the layer list, at the bottom left of the Ipe window. The checkmark to the left of the layer name determines whether the layer is visible. The layer marked with a yellow background is the *active* layer. New objects are always created in the active layer. You can change the active layer by left-clicking on the layer name (on Windows, double-click on the layer name).

By right-clicking on a layer name, you open the layer context menu that allows you to change layer attributes, to rename layers, to delete empty layers, and to change the ordering of layers in the layer list (this ordering has no other significance).

A layer may be editable or locked. Objects can be selected and modified only if their layer is editable. Locked layer are displayed in the layer list with a pink background. You can lock and unlock layers from the layer context menu, but note that the active layer cannot be locked.

A layer may have snapping on or off—objects will behave magnetically only if their layer has snapping on. By default, snapping is on when the layer is visible, but you can choose to turn it off entirely, or to keep it on even when the layer is not visible.

Layers are also used to create pages that are displayed incrementally in a PDF viewer. Once you have distributed your objects over various layers, you can create *views*, which defines in what order which layers of the page are shown.

2.11 Mouse shortcuts

For the beginner, choosing a selection or transformation mode and working with the left mouse button is easiest. Frequent Ipe users don't mind to remember the following shortcuts, as they allow you to perform selections and transformations without leaving the current mode:

Modifiers	Left Mouse	Right mouse
Plain	*mode-dependent*	context menu
Shift	*mode-dependent*	pan
Ctrl	select	stretch
Ctrl+Shift	select non-destructively	scale
Alt	translate	rotate
Alt+Shift	translate horizontal/vertical	rotate

The middle mouse button always pans the canvas. Without a keyboard-modifier, the right mouse button brings up the object context menu.

If you have to use Ipe with a two-button mouse, where you would normally use the middle mouse button (for instance, to move a vertex when editing a path object), you can hold the Shift-key and use the right mouse button.

If you are not happy with these shortcuts, they *can be changed easily*.

CHAPTER

THREE

OBJECT TYPES

Ipe supports five different types of objects that can be placed on a page, namely path objects (which includes all objects with a stroked contour and filled interior, such as (poly)lines, polygons, splines, splinegons, circles and ellipses, circular and elliptic arcs, and rectangles), text objects, image objects, group objects, and reference objects (which means that a symbol is used at a certain spot on the page).

Path and text objects are created by clicking the left mouse button somewhere on the canvas using the correct Ipe mode. Group objects are created using the *Group* function in the *Edit* menu. Image objects are added to the document using *Insert image* in the *File* menu. Reference objects can be created either using mark mode, or using the *Symbols* ipelet.

3.1 Path objects

Path objects are defined by a set of *subpaths*, that is, curves in the plane. Each subpath is either open or closed, and consists of straight line segments, circular or elliptic arc segments, parabola segments (or, equivalently, quadratic Bézier splines), cubic Bézier splines, and cubic B-spline segments. The curves are drawn with the stroke color, dash style, and line width; the interior of the object specified is filled using the fill color.

The distinction between open and closed subpaths is meaningful for stroking only, for filling any open subpath is implicitely closed. Stroking a set of subpaths is identical to stroking them individually. This is not true for filling: using several subpaths, one can construct objects with holes, and more complicated pattern. The filling algorithm is normally the *even-odd rule* of Postscript/PDF: To determine whether a point lies inside the filled shape, draw a ray from that point in any direction, and count the number of path segments that cross the ray. If this number is odd, the point is inside; if even, the point is outside.

Ipe can draw arrows on the first and last segment of a path object, but only if that segment is part of an open subpath.

There are several Ipe modes that create path objects in different ways. All modes create an object consisting of a single subpath only. To make more complicated path objects, such as objects with holes, you create each boundary component separately, then select them all and use the *Compose paths* function in the *Edit* menu. The reverse operation is *Decompose path*, you find it in the context menu of a path object that has several subpaths.

You can also create complicated paths by joining curves sequentially. For this to work, the endpoint of one path must be (nearly) identical to the begin point of the next—easy to achieve using snapping. You select the path objects you wish to join, and call *Join paths* in the *Edit* menu. You can also join several open path objects into a single closed path this way.

Circles can be entered in three different ways. To create an ellipse, create a circle and shear or stretch and rotate it. Circular arcs can be entered by clicking three points on the arc or by clicking the center of the supporting circle as well as the begin and end vertex of the arc. They can be filled in Postscript/PDF fashion, and can have arrows. You can stretch a circular arc to create an elliptic arc.

A common application for arcs is to mark angles in drawings. The snap keys are useful to create such arcs: set arc creation to *center & 2 pts*, select *snap to vertex* and *snap to boundary*, click first on the center point of the angle (which is magnetic) and click then on the two bounding lines.

There are two modes for creating more complex general path objects. The difference between *line* mode and *polygon* mode is that the first creates an open path, the latter generates a closed one. As a consequence, the *line* mode uses the current arrow settings, while the *polygon* mode doesn't.

The path object created using line or polygon mode consists of segments of various types. The initial setting is to create straight segments. By holding the shift-key when pressing the left mouse button one can switch to splines. Ipe offers three different kinds of splines: uniform B-splines, cardinal splines, and clothoid splines. You can select the type of spline in the *Properties* menu. Quadratic and cubic Bézier spline segments can be created as special cases of uniform B-splines with three and four control points.

Circular arcs can be added as follows: Click twice in polyline mode, once on the starting point of the arc, then on a point in the correct tangent direction. Press the a key, and click on the endpoint of the arc.

To make curves where segments of different type are joined with identical tangents, you can press the y key whenever you are starting a new segment: this will set the coordinate system centered at the starting point of the segment, and aligned with the tangent to the previous segment.

For the mathematically inclined, a more precise description of the segments that can appear on a subpath follows.

A subpath consists of a sequence of segments. Each segment is either a straight line segment, an elliptic arc, or a spline of one of the three supported types.

Note that a uniform B-spline with only three control points will be drawn as a *quadratic* Bézier spline. A B-spline with four control points is by definition a cubic Bézier spline.

The quadratic Bézier spline defined by control points p_0, p_1, and p_2, is the curve

$$P(t) = (1-t)^2 p_0 + 2t(1-t)p_1 + t^2 p_2,$$

where t ranges from 0 to 1. This implies that it starts in p_0 tangent to the line p_0p_1, ends in p_2 tangent to the line p_1p_2, and is contained in the convex hull of the three points. Any segment of any parabola can be expressed as a quadratic Bézier spline.

For instance, the piece of the unit parabola $y = x^2$ between $x = a$ and $x = b$ can be created with the control points

$$p_0 = (a, a^2)$$
$$p_1 = (\tfrac{a+b}{2}, ab)$$
$$p_2 = (b, b^2)$$

Any piece of any parabola can be created by applying some affine transformation to these points.

The cubic Bézier spline with control points p_0, p_1, p_2, and p_3, is the curve

$$R(t) = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3.$$

It starts in p_0 being tangent to the line p_0p_1, ends in p_3 being tangent to the line p_2p_3, and lies in the convex hull of the four control points.

Uniform cubic B-splines approximate a series of n control points p_0, p_1, …, p_{n-1}, $n \geq 3$, with a curve consisting of $n-3$ cubic Bézier splines, see, for instance, Sederberg[1]. To clamp the spline to the first and last control point, the first and last knot are repeated three times. If the curve is closed (a *splinegon*), there is no clamping and n control points define n Bézier splines.

Since each point on a Bézier curve is a convex combination of the four control points, the curve segment lies in the convex hull of the control points. Furthermore, any affine transformation can be applied to the curve by applying it to the control points. Note that a control point has influence on only a few Bézier segments, so when you edit a spline object and move a control point, only a short piece of the spline in the neighborhood of the control point will move.

A *cardinal spline* is an *interpolating spline*—unlike a uniform B-spline, it will go *through* the control points. Ipe converts a sequence of n control points into $n-1$ Bézier segments, each starting and ending at a control point. The tangent at each control point is parallel to the segment connecting the previous and the next control point.

[1] Thomas Sederberg, Computer-Aided Geometric Design, Chapter 6.

Finally, a *clothoid spline* (spline type `spiro`) is created by solving an optimization problem that tries to minimize the change in curvature (so they look much "rounder" and are generally more pleasing). Details can be found in Raph Levien's thesis.

Existing polygonal objects can be edited, using Ipe's *edit mode*. You select the object you want to modify, and then press `M-e` (or select *Edit* → *Edit object*). The object will be placed in edit mode.

3.2 Text objects

Text objects come in two flavors: simple *labels*, and *minipages*. There are two variants of these: *titles* (a label that serves as the title of the page), and *textbox* (a minipage that spans the entire width of the page).

The position you have to click to start creating a *label* object is normally the leftmost baseline point (but this can be changed by changing the object's horizontal and vertical alignment). A popup window appears where you can enter Latex source code.

A *minipage* object is different from a simple text object in that its width is part of its definition. When you create a minipage object, you first have to drag out a horizontal segment for the minipage. This is used as the top edge of the minipage—it will extend downwards as far as necessary to accomodate all the text. Minipages are formatted using, not surprisingly, Latex's `minipage` environment. Latex tries to fill the given bounding box as nicely as possible. It is possible to include center environments, lemmas, and much more in minipages.

To create a *textbox* object, simply press `F10`. Ipe automatically places the object so that it spans the entire width of the page (the *layout* settings in the stylesheet determine how much space is left on the sides), and places it vertically underneath the textboxes already on the page. This is particularly convenient for creating presentations with a lot of text, or with items that appear *one by one*.

Title objects are managed by Ipe automatically. They are special labels that are created using *Edit title & sections* in the *Page* menu. Their color, size, alignment, and position on the page is determined by the stylesheet.

You can use any LaTeX-command that is legal inside a `\makebox` (for labels) or inside a `minipage` (for minipages). You cannot use certain commands that involve a non-linear translation into PDF. In particular, you cannot use commands to generate hyperlinks (the `\href` command from the `hyperref` package will not work). If you need to add links to your Ipe document, you will need to use a *group object*.

You can use color in your text objects, using the `\textcolor` command, like this:

```
This is in black. \textcolor{red}{This is in red.} This is in black.
```

All the symbolic colors of your current stylesheet are also available as arguments to `\textcolor`. You can also use absolute colors, for instance:

```
This is in black. \textcolor[rgb]{1,1,0}{This is in yellow.} This is in black.
```

If you need LaTeX-commands that are defined in additional LaTeX packages, you can include (`\usepackage`) those in the LaTeX preamble, which can be set in *Document properties* in the *Edit* menu.

Note that the `xcolor`-package is loaded automatically by Ipe, without any options. If you need to use package options of the `xcolor`-package, place the command

```
\ipedefinecolors{options}
```

in your preamble. It has to go before the first use of `xcolor` commands in your document.

After you have created or edited a text object, the Ipe screen display will show the beginning of the Latex source. You can select *Run Latex* from the *File* menu to create the PDF representation of the object. This converts all the text objects in your document at once, and Ipe will display a correct rendition of the text afterwards.

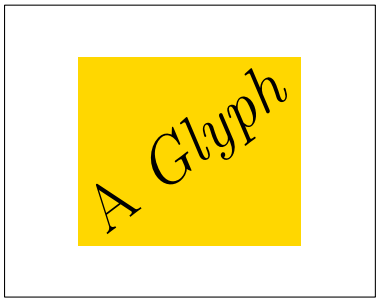
If the Latex conversion process results in errors, Ipe will automatically show you the log file created by the Latex run. If you cannot figure out the problem, look in *the section on troubleshooting*.

You can use Unicode text, such as accented characters, Greek, Cyrillic, Chinese, Japanese, or Korean, in your text objects, once *you have set up the necessary style files and fonts*.

When Ipe computes the bounding box for a piece of text, it relies entirely on the dimensions that Latex provides. Sometimes glyphs are larger than their `official` dimensions, and as a result this bounding box is too tight. In the following figure, `A` and `G` stick out of the golden rectangle (the bounding box computed by Ipe based on the Latex dimensions) at the top, `y` sticks out at the bottom:

When you experience that text in your figures is clipped, you may have to enlarge the figure's bounding box using a `BBOX` layer.

The opposite problem can occur when you use transformed text. Ipe computes the bounding box for the transformed text by transforming the bounding box for the original text, and the result can be too large:



If this is a problem, you can put the text object inside a group and set a clipping path for the group.

3.3 Image objects

Images are inserted using *Insert image* (in the *File* menu). Once in a drawing, you can scale, stretch, shear, and rotate an image. You can read in some scanned drawing and draw on top of it within Ipe. This is useful if you have a drawing on paper and want to make an Ipe version of it.

Insert image can handle images in JPEG and PNG format (and possibly others such as GIF, BMP, TIFF, depending on the operating system). JPEG images are stored as is (PDF supports the JPEG standard), all other images are stored as a (compressed) bitmap, with full-color resolution (24 bits per pixel).

It is also possible to simply paste an image into Ipe (this should not be used for JPEG images, as you would then store the bitmap without JPEG compression).

Images are stored efficiently in PDF format. It is reasonable to create PDF presentations with lots of JPEG photographs in Ipe.

3.4 Group objects

Group objects are created by selecting any number of objects and using the *Group* function from the *Edit* menu. The grouped objects then behave like a single object. To modify a group object, it has to be decomposed into its parts using *Ungroup*.

3.4.1 Clipping

You can set a *clipping path* for a group. The group will then be clipped to this path—nothing will be drawn outside the clipping path. This is useful, for instance, to clip out an interesting part of an existing drawing or bitmap.

To add a clipping path, select a group as the primary selection, and a path object as the secondary selection. Then select *Add clipping path* from the group's context menu.

3.4.2 Decorations

Group objects can be *decorated*. A decoration consists of one or more path objects that are drawn around the group. The decoration is automatically resized to fit the bounding box of the group.

To use decorations, you first need to add a *stylesheet* to your document that defines decoration symbols—you may want to start with the provided style sheet *decorations.isy*. Then use the group's object menu to choose a decoration for the group.

3.4.3 Editing text in group

Groups often contain some text. For instance, a graph vertex is nicely represented as a group consisting of a text label and either a mark symbol such as a disk, or a path object (a circle, rectangle, or a more complicated shape). When drawing a graph, one can place the vertices by copying and pasting these vertex objects, but then one needs to set the text label in each vertex.

To make this easy, the *Edit object* operation (which is otherwise used to edit the text in text objects and the shape of path objects) can also be used for group objects that contain at lease one text object. It allows you to update the text inside the *top-most* text object of the group.

3.4.4 Recursive group edit

Often you want to modify the contents of a group without disturbing the rest of your drawing. To make this easy, Ipe provides the *Edit group* operation, available either from the *Edit* menu or from the group's context menu.

Group editing is implemented by un-grouping the group into a newly created layer whose name will start with *EDIT-GROUP*. Ipe locks all other layers, so that you can concentrate on editing the objects in the group. When you are done, you select *End group edit* from the *Edit* menu. Ipe will take all the objects in the group edit layer, group them together, and place the group back in its original layer.

You can edit groups recursively: If your group contains another group that you want to modify, you can perform another group edit operation. Each *End group edit* closes one group edit layer, until you return to your normal drawing workflow.

Group edit is not a special mode—all the state needed by Ipe to manage editing the group is stored inside the drawing. This means that you can save your drawing during a group edit (also, auto-saving works during a group edit). If you have a document with multiple pages, you can also start group edits on several pages in parallel, for instance to copy and paste objects between groups.

It is legal to unlock the other layers of the page so that you can move objects into and out of the group. You should, however, be careful with changing anything about the layers of the page—do not re-order or rename the layers. If you need new layers, create them at the end of the layer list.

When you perform the *End group edit* operation, the group edit layer must be the active layer. If you changed the active layer, you will have to change it back to be able to return from the group edit.

Note that group edit does not currently preserve a clipping path set on the group. It does preserve the group's decoration and link destination.

3.4.5 Links to websites, videos, and other pages

Group objects allow you to create *active links*: when the PDF document is viewed in a PDF viewer, one can click inside the bounding box of the group to cause some action.

You add a link to a group by bringing up the object menu, and using *Set link URL*. Note that only top-level group objects on a page (that is, a group object that is not inside another group) are turned into active links.

Ipe supports several types of link actions:

Websites

If you provide a URL such as `http://ipe.otfried.org` as a link action, then clicking on the link will open the webpage in a web browers.

Launching media

If the link action starts with `launch:`, then the action starts a program to view the given file. For instance, the link action `launch:apollo17.avi` would start a video player playing the file `apollo17.avi` in the directory that also contains the PDF file.

The PDF presentations apps Presentation and pdfpc understand these links, and play the video inside the presentation app. Pdfpc will even embed the video into the current slide, using the bounding box of the group object to place the video.

Pdfpc allows you to provide a few more parameters: for instance, the link action

```
launch:apollo17.avi?autostart&loop&start=5&stop=12
```

will start the movie as soon as the slide is viewed, plays the movie in a loop, starting at 5 seconds and ending at 12 seconds into the movie.

You can also launch other types of documents. For instance, clicking on a group object with link action `launch:summary.txt` will open the text file `summary.txt` in a text editor.

Navigating inside the document

If the link action starts with *goto:*, then clicking on the group object will navigate to another page of the same PDF document. You identify pages using their *section name*. So if you have a page with section name *chapter3* in your document, then the link action *goto:chapter3* will jump to that page.

Standard actions

PDF currently defines four standard actions: *NextPage*, *PrevPage*, *FirstPage*, and *LastPage*. You use these by prefixing them with *named:*. For example, the link action *named:PrevPage* creates an action to go to the previous page. This can be used, for instance, to place buttons in a background layer that appears on every page of a document.

3.5 Reference objects and symbols

A symbol is a single Ipe object (which can of course be a group) that is defined in a document's stylesheet. A reference object is a reference to a symbol, placed at a given position on the page.

Symbols can be parameterized with stroke and fill color, pen width, and symbol size. Whether or not a symbol accepts which parameter is visible from the symbol's name: if it takes any parameter, the name must end in a pair of parentheses containing some of the letters `s`, `f`, `p`, `x` (in this order), for the parameters stroke, fill, pen, and size. References to parameterized symbols allow all the attributes that the symbol accepts.

All references can be translated around the page. Whether or not the symbol can be rotated or stretched depends on the definition of the symbol in the stylesheet.

If a symbol named `Background` exists in your stylesheet, it is automatically displayed on each page, at the very back. To suppress the automatic display, create a layer named `BACKGROUND` on the page. (If such a layer is present, it suppresses the `Background` symbol. It does not matter if the layer itself is visible or not.)

Marks are symbols with special support from the Ipe user interface. They are used to mark points in the drawing, and come in several different looks (little circles, discs, squares, boxes, or crosses). You can define your own mark shapes by defining appropriate symbols in your stylesheet.

Note that marks behave quite different from path objects. In particular, you should not confuse a disc mark with a little disc created as a circle object:

- a solid mark (type *disk* and *square*) only obeys the stroke color (but *fdisk* and *fsquare* marks are filled with the fill color);
- when you scale a mark, it will not change its size (you can change the mark size from the properties panel, though);
- when you rotate a mark, it does not change its orientation.

You can change a mark's shape and size later.

CHAPTER

FOUR

SNAPPING

One of the nice features of Ipe is the possibility of having the mouse *snap* to other objects during entry or moving. Certain features on the canvas become "magnetic", and it is very easy to align objects to each other, to place new objects properly with respect to the present objects and so on.

Snapping comes in three flavors: *grid snapping*, *context snapping*, and *angular snapping*.

In general, you turn a snapping mode on by pressing one of the buttons in the *Snap* toolbar, or selecting the equivalent functions in the Snap menu. The buttons are independent, you can turn them on and off independently. (The snapping modes, however, are not independent. See below for the precise interaction.) The keyboard shortcuts are rather convenient since you will want to toggle snapping modes on and off while in the middle of creating or editing some object.

Whenever one of the snapping modes is enabled, you will see a little cross near the cursor position. This is the secondary cursor *Fifi*[1]. Fifi marks the position the mouse is snapped to.

4.1 Grid snapping

Grid snapping is easy to explain. It simply means that the mouse position is rounded to the nearest grid point. Grid points are points whose coordinates are integer multiples of the *grid size*, which can be set in the box in the *Snap* field. You have a choice from a set of possible grid sizes. The units are Postscript/PDF points (in LaTeX called bp), which are equal to 1/72 of an inch.

You can ask Ipe to show the grid points by selecting the function *View* → *Grid visible*. The same function turns it off again.

4.2 Context snapping

When *context snapping* is enabled, certain features of the objects of your current drawing become magnetic. There are three buttons to enable three different features of your objects: vertices, the boundary, and intersection points.

When the mouse is too far away from the nearest interesting feature, the mouse position will not be "snapped". The snapping distance can be changed by setting *Snapping distance* value in the preference dialog. If you use a high setting, you will need to toggle snapping on and off during drawing. Some people prefer to set snapping on once and for all, and to set the snap distance to a very small value like 3 or 4.

The features that you can make "magnetic" are the following:

vertices are vertices of polygonal objects, control points of multiplicity three of splines, centers of circles and ellipses, centers and end points of circular arcs, and mark positions.

[1] Fifi is called after the dog in the `rogue` computer game installed on most Unix systems in the 1980's, because it also keeps running around your feet.

boundaries are the object boundaries of polygonal objects, splines and splinegons, circles and ellipses, and circular arcs.

intersections are the intersection points between the boundaries of path objects.

4.3 Custom grid snapping

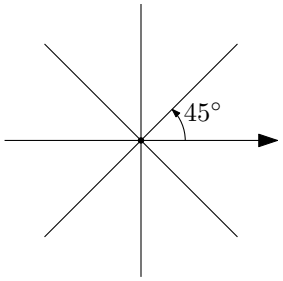
Sometimes you need a special grid, for instance a triangular grid, or a grid for making perspective drawings. Since Ipe cannot offer every possible grid under the sun, it instead offers you snapping to custom grids.

To use custom grid snapping, create a layer with name *GRID*, and draw your grid in this layer. You can then snap to the intersection points between the objects in the *GRID* layer. It does not matter if the layer is visible or not.

The *grid maker* ipelet offers a few ready-made grids to be used as custom grids.

4.4 Angular snapping

When *angular snapping* is enabled, the mouse position is restricted to lie on a set of lines through the *origin* of your current *axis system*. The lines are the lines whose angle with the *base direction* is an integer multiple of the snap angle. The snap angle can be set in the second box in the Snap toolbar. The values are indicated in degrees. So, for a snapping angle of $45°$, we get the snap lines indicated in the figure below. (In the figure the base direction—indicated with the arrow—is assumed horizontal.)



For a snap angle of 180 degrees, snapping is to a single line through the current origin.

In order to use angular snapping, it is important to set the axis system correctly. To set the origin, move the mouse to the correct position, and press the `F1`-key. Note that angular snapping is *disabled* while setting the origin. This way you can set a new origin for angular snapping without leaving the mode first. Once the origin has been set, the base direction is set by moving to a point on the desired base line, and pressing the `F2`-key. Again, angular snapping is disabled. Together, origin and base direction determine the current *axis system*. Remember that the origin is also used as the fix-point of scale, stretch, and rotate operations, if it is set.

You can hide the current axis system by pressing `Ctrl+F1`. This also turns off angular snapping, but preserves origin and orientation of the axes. To reset the orientation (such that the x-axis is horizontal, use `Ctrl+F2`).

You can set origin and base direction at the same time by pressing `F3` when the mouse is very near (or snapped to) an edge of a polygonal object. The origin is set to an endpoint of the edge, and the base direction is aligned with it. This is useful to make objects parallel to a given edge.

For drawing rectilinear or c-oriented polygons, the origin should be set to the previous vertex at every step. This can be done by pressing `F1` every time you click the left mouse button, but that would not be very convenient. Therefore, Ipe offers a second angular snap mode, called *automatic angular snapping*. This mode uses an independent origin, which is automatically set every time you add a vertex when creating a polygonal object. Note that while the origin is independent of the origin set by `F1`, the base direction and the snap angle used by automatic angular snapping is the

same as for angular snapping. Hence, you can align the axis system with some edge of your drawing using F3, and then use automatic angular snapping to draw a new object that is parallel or orthogonal to this edge.

This snapping mode has another advantage: It remains silent and ineffective until you start creating a polygonal object. So, even with automatic angular snapping already turned on, you can still freely place the first point of a polygon, and then the remaining vertices will be properly aligned to make a c-oriented polygon.

The automatic angular snapping mode is never active for any non-polygonal object. In particular, to *move* an object in a prescribed direction, you have to use normal angular snapping.

A final note: Many things that can be done with angular snapping can also be done by drawing auxiliary lines and using context snapping. It is mostly a matter of taste and exercise to figure out which mode suits you best.

4.5 Interaction of the snapping modes

Not all the snapping modes can be active at the same time, even if all buttons are pressed. Here we have a close look at the possible interactions, and the priorities of snapping.

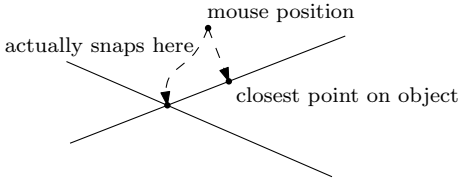
The two angular snapping modes restrict the possible mouse positions to a *one-dimensional* subspace of the canvas. Therefore, they are incompatible with the modes that try to snap to a zero-dimensional subspace, namely vertex snapping, intersection snapping, and grid snapping. Consequently, when one of the angular snapping modes is *on*, vertex snapping, intersection snapping, and grid snapping are ineffective.

On the other hand, it is reasonable to snap to boundaries while in an angular snapping mode, and this function is actually implemented correctly. When both angular and boundary snapping are *on*, Ipe will compute intersections between the snap lines with the boundaries of your objects, and whenever the mouse position *on* the snap line comes close enough to an intersection, the mouse is snapped to that intersection.

The two angular snapping modes themselves can also coexist in the same fashion. If both angular and automatic angular snapping are enabled, Ipe computes the intersection point between the snap lines defined by the two origins and snaps there. It the snap lines are parallel or coincide, automatic angular snapping is used.

When no angular snapping mode is active, Ipe has three priorities. First, Ipe checks whether the closest vertex or intersection point is close enough. If that is not the case, the closest boundary edge is determined. If even that is too far away, Ipe uses grid snapping (assuming all these modes are enabled).

Note that this can actually mean that snapping is *not* to the *closest* point on an object. Especially for intersections of two straight edges, the closest point can never be the intersection point, as in the figure below!

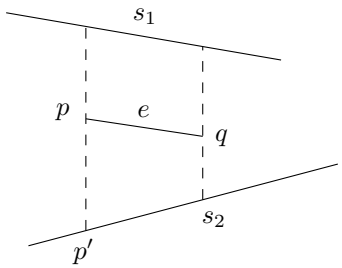


4.6 Examples

It takes some time and practice to feel fully at ease with the different snapping modes, especially angular snapping. Here are some examples showing what can be done with angular snapping.

4.6.1 Example 1:

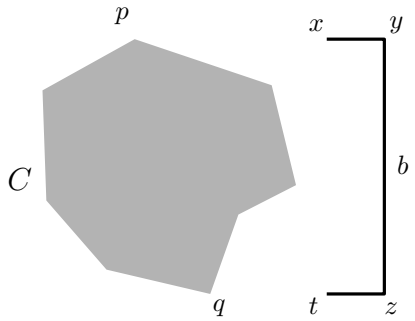
We are given segments s_1, s_2, and e, and we want to add the dashed vertical extensions through p and q.



1. set `F4` and `F5` snapping on, go into *line* mode, and reset axis orientation with `Ctrl+F2`,
2. go near p, press `F1` and `F8` to set origin and to turn on angular snap.
3. go near p', click left, and extend segment to s_1.
4. go near q, press `F1` to reset origin, and draw second extension in the same way.

4.6.2 Example 2:

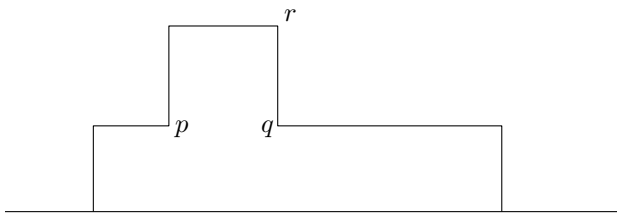
We are given the polygon C, and we want to draw the bracket b, indicating its vertical extension.



1. set `F4` and `F9` snapping on, go into *line* mode, reset axis orientation with `Ctrl+F2`, set snap angle to $90°$.
2. go near p, press `F1` and `F8` to set origin and angular snapping
3. go to x, click left, extend segment to y, click left
4. now we want to have z on a horizontal line through q: go near q, and press `F1` and `F8` to reset origin and to turn on angular snapping. Now both angular snapping modes are on, the snap lines intersect in z. item click left at z, goto x and press `F1`, goto t and finish bracket.

4.6.3 Example 3:

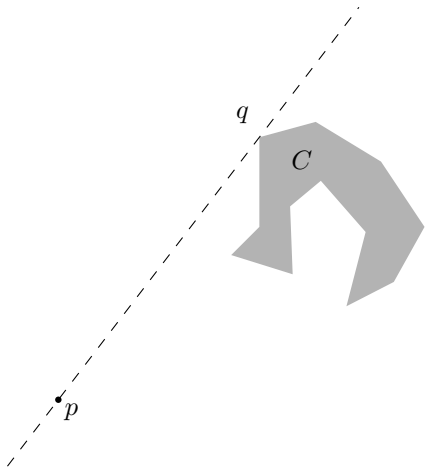
We want to draw the following "skyline". The only problem is to get q horizontally aligned with p.



1. draw the baseline using automatic angular snapping to get it horizontal.
2. place p with boundary snapping, draw the rectilinear curve up to r with automatic angular snapping in $90°$ mode.
3. now go to p and press `F1` and `F8`. The snap lines intersect in q. Click there, turn off angular snapping with `Shift-F2`, and finish curve. The last point is placed with boundary snapping.

4.6.4 Example 4:

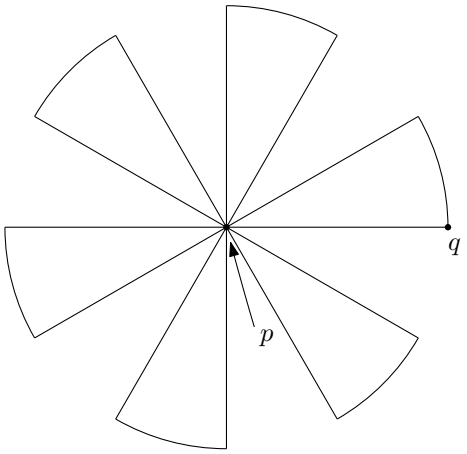
We want to draw a line through p, tangent to C in q.



1. with vertex snapping on, put origin at p with `F1`
2. go to q and press `F2`. This puts the base direction from p to q.
3. set angular snapping with `F8` and draw line.

4.6.5 Example 5:

We want to draw the following "windmill". The angle of the sector and between sectors should be $30°$.

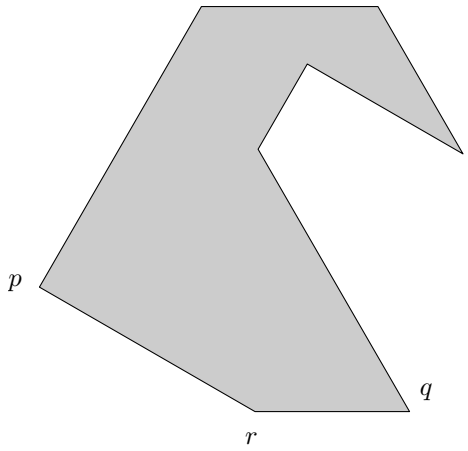


1. set vertex snapping, snap angle to $30°$, reset axis orientation with `Ctrl+F2`,
2. with automatic angular snapping, draw a horizontal segment pq.
3. go to p, place origin and turn on angular snapping with `F1` and `F8`,

4. duplicate segment with `d`, go to q and pick up q for rotation (with `Alt` and the right mouse button, or by switching to rotate mode). Rotate until segment falls on the next snap line.
5. turn off angular snapping with `F8`. Choose arc mode, variant *center & two points*,
6. go to p, click for center. Go to q, click for first endpoint of arc, and at r for the second endpoint. Select all, and group.
7. turn angular snapping on again. Duplicate sector, and rotate by $60°$, using angular snapping.
8. duplicate and rotate four more times.

4.6.6 Example 6:

We want to draw a c-oriented polygon, where the angles between successive segments are multiples of $30°$. The automatic angular snapping mode makes this pretty easy, but there is a little catch: How do we place the ultimate vertex such that it is at the same time properly aligned to the penultimate and to the very first vertex?



1. set snap angle to $30°$, and turn on automatic angular snapping.
2. click first vertex p and draw the polygon up to the penultimate vertex q.
3. it remains to place r such that it is in a legal position both with respect to q and p. The automatic angular snapping mode ensures the position with respect to q. We will use angular snapping from p to get it right: Go near p and turn on vertex snapping. Press `F1` to place the origin at p and `F8` to turn on angular snapping. Now it is trivial to place r.

CHAPTER

FIVE

STYLESHEETS

The symbolic attributes appearing in an Ipe document are translated to absolute values for rendering by a *stylesheet* that is attached to the document. Documents can have multiple *cascaded* stylesheets, the sheets form a stack, and symbols are looked up from top to bottom. At the bottom of any stylesheet cascade is always the minimal *standard* style sheet, which is built into Ipe.

When you create a new empty document, it automatically gets a copy of this standard style sheet (which does little more than define the *normal* attribute for each kind of attribute). In addition, Ipe inserts a predefined list of stylesheets. The list of these stylesheets can be *customized* using an ipelet, using Ipe's command line options, or an environment variable. By default, a new document gets the stylesheet *basic* that comes with Ipe.

The stylesheet dialog (in *Edit* → *Stylesheets*) allows you to inspect the cascade of stylesheets associated with your document, to add and remove stylesheets, and to change their order. You can also save individual stylesheets.

The stylesheets of your document also determine the symbolic choices you have in the Ipe user interface. If you feel that Ipe does not offer you the right choice of colors, pen widths, etc., you are ready to make your own style sheet!

5.1 Make your own stylesheet!

So you are ready to roll your own stylesheet, to have the colors and pen widths you've always wanted? Here you go:

First, decide on a name for your stylesheet. In this section, let's pick the name *personal*.

Open an Ipe document (or make a new one and save it in a file), then start up your favorite text editor and create the file `personal.isy`. The file must be in the **same folder** as your Ipe document!

Enter the following contents into the file, and save it:

```
<ipestyle name="personal">
<color name="yellowgreen" value="0.604 0.804 0.196"/>
</ipestyle>
```

(The `name` attribute in the first line must match the filename, without the extension `.isy`.)

In Ipe, use *Edit* → *Style sheets* to bring up the stylesheet dialog. Press the *Add* button, and select your file `personal.isy`. You'll see *personal* appear at the top of the list of stylesheets. Click *Ok* to confirm adding the stylesheet.

You will notice that a new color named `yellowgreen` is now available in the dropdown for stroke and fill color. Congratulations—you made your first stylesheet!

You can now add colors, pen widths, and sizes for symbols (markers), arrows, and the grids, by imitating the following examples:

```
<pen name="light" value="0.7"/>
<symbolsize name="giant" value="20"/>
<arrowsize name="small" value="6"/>
<gridsize name="10 pts" value="10"/>
<anglesize name="20 deg" value="20"/>
```

Each line contains the symbolic `name`, and an absolute numeric value in `value`. The name must start with a letter (grid size and angle size are exceptions).

To add a definition, update the file `personal.isy` in your text editor, then use *Edit* → *Update style sheets*. This will cause Ipe to read the file again, and to replace the copy of the stylesheet inside your document with this newest version.

This way, you can quickly test out new definitions by editing the stylesheet in your text editor, and pressing `Ctrl`+`Shift`+`U` in Ipe to try out the new definitions.

You may wonder how to get your favorite colors right, so here is a little trick: draw a small box in Ipe, then press the absolute stroke color button (the top-left button in the *Properties* panel). It will allow you to select a color using a graphical user interface. Once you have found the right color, apply it to the box (by selecting `<absolute>` in the stroke color selector), then right-click on the box and select *Edit as XML*. A dialog will appear showing the current definition of the box in Ipe's internal XML format, like this:

```
<path stroke="0.561 0.349 0.008" pen="ultrafat">
64 816 m
64 800 l
80 800 l
80 816 l
h
</path>
```

You can now copy the color definition (in this case `"0.561 0.349 0.008"`, a nice shade of brown) to your stylesheet.

You can find inspiration for more colors in the `colors.isy` stylesheet in Ipe's `styles` folder. It defines all the colors of the X11 color database—you could make a selection of these for your own use.

There is much more you can do with stylesheets. Have a look in the stylesheets that come with Ipe for some inspiration, or keep reading this chapter and the *next*. The ultimate reference is, of course, the *description of the stylesheet file format*.

5.2 Stylesheet theory

When a stylesheet is "added" to an Ipe document, the contents of the stylesheet file is copied into the Ipe document. Subsequent modification of the stylesheet file has no effect on the Ipe document. The right way to modify your stylesheet is to either "add" it again, and then to delete the *old* copy from your stylesheet cascade (the one further down in the list), or to use the *Update stylesheets* function in the *Edit* menu. This function assumes that the stylesheet file is in the same directory as the document and that the filename coincides with the name of the stylesheet (plus the extension `.isy`).

Removing or replacing a stylesheet can cause some of the symbolic attributes in your document to become undefined. This is not a disaster—Ipe will simply use some default value for any undefined symbolic attribute. To allow you to diagnose the problem, Ipe will show a warning listing all undefined symbolic attributes.

We discuss a few stylesheet topics in this chapter. Other stylesheet definitions that are (mostly) meant for PDF presentations are discussed in the next chapter.

5.3 Symbols

Style sheets can also contain *symbols*, such as marks and arrows, background patterns, or logos. These are named Ipe objects that can be referenced by the document. If your document's stylesheets define a symbol named *Background*, it will be displayed automatically on all pages. (If a layer named `BACKGROUND` is present on a page, it suppresses the `Background` symbol for that page. It does not matter if the layer itself is visible or not.) You can create and use symbols using the *Symbols* ipelet. Here is a (silly) example of a style sheet that defines such a background:

```
<ipestyle name="background">
<symbol name="Background" xform="yes">
<text pos="10 10" stroke="black" size="LARGE">
Background text
</text>
</symbol>
</ipestyle>
```

Note the use of the `xform` attribute—it ensures that the background is embedded only once into PDF document. This can make a huge difference if your background is a complicated object.

Symbols can be parameterized with a stroke color, fill color, pen size, and symbol size. This means that the actual value of these attributes is only set when the symbol is used in the document (not in the symbol definition). The name of a parameterized symbol must end with a pair of parentheses containing some of the letters `s` (stroke), `f` (fill), `p` (pen), `x` (symbol size), in this order. The symbol definition can then use the special attribute values `sym-stroke`, `sym-fill`, and `sym-pen`. A resizable symbol is automatically magnified by the symbol size set in the symbol reference.

A symbol can define several snap positions for the symbol object. These positions are then active in vertex snap mode. Symbols with snap positions are also presented differently in the current selection (the entire symbol is outlined, like a group, rather than just showing a cross at the symbol location), and you can select such symbols by clicking near any of the snap positions.

You can also use a stylesheet to define additional mark shapes, arrow shapes, or tiling patterns.

5.4 Decorations

A *decoration* is a symbol that can be used to decorate a group object. Its name must start with the string *decoration/*, and it should contain either a path object or a group of path objects.

Ipe resizes these path objects so that they fit nicely around the bounding box of the group object being decorated. For this to work correctly, the decoration object must be drawn such that it decorates the rectangle with corners at $(100, 100)$ and $(300, 200)$.

To make a decoration symbol, follow these steps:

1. Draw a rectangle. Select *Edit as XML* from its context menu, and change the coordinates to look as follows:

   ```
   <path stroke="black" fill="lightblue">
   100 100 m
   300 100 l
   300 200 l
   100 200 l
   h
   </path>
   ```

2. Draw your decoration so that it fits this rectangle. You should draw only path objects.

3. Delete the rectangle from the first step.

4. If the decoration consists of more than one object, group them all together.
5. Save this object as a symbol whose name starts with *decoration/* (including the slash). You can do this either using *create new symbol* in the *symbols* ipelet, or by selecting *Edit as XML* from the object's context menu and copying the code into a stylesheet open in your text editor.

For inspiration, have a look at the decoration symbols in the stylesheet `decorations.isy` that comes with Ipe.

5.5 More about stylesheets

5.5.1 Paper size

The style sheet is also responsible for determining the paper and frame size. Ipe's default paper size is the ISO standard A4. If you wish to use letter size paper instead, include this style sheet:

```
<ipestyle name="letterpaper">
  <layout paper="612 792" origin="0 0" frame="612 792"/>
</ipestyle>
```

5.5.2 Latex preamble.

Stylesheets can also define a piece of LaTeX-preamble for your document. When your text objects are processed by LaTeX, the preamble used consists of the pieces on the style sheet cascade, from bottom to top, followed by the preamble set for the document itself.

Note that when putting LaTeX code in your style sheet, you have to escape the characters that are special in XML. For instance, for < you would have to write `<`.

CHAPTER

SIX

DOCUMENTS WITH TEXT THAT IS NOT IN ENGLISH

If you make figures containing text objects in languages other than English, you will need to enter accented characters, or characters from other scripts such as Greek, Cyrillic, Farsi, Arabic, Hangul, Kana, or Chinese characters.

Of course you can still use the LaTeX syntax `K\"onig` to enter the German word "König", but for larger runs of text it's more convenient to enter text in a script supported by your system. When Ipe writes the LaTeX source file, it writes the text in UTF-8 encoded Unicode. You have to make sure that your LaTeX setup can handle this file.

6.1 Pdflatex for latin and cyrillic scripts

An easy solution, sufficient for German, French, and other languages for which support is already in a standard LaTeX-setup, is to add the line

```
\usepackage[utf8]{inputenc}
```

in your *Latex preamble* (set in the *Document properties* dialog, available on the *Edit* menu).

In addition, you may need to setup your document language. So, to use UTF-8 encoded Russian in LaTeX, your preamble would look like this:

```
\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}
```

When setting this up, you have to keep in mind that Ipe can only handle scalable fonts, such as Postscript Type1 fonts. You'll have to choose a setup that uses such scalable fonts. Quite often it is sufficient to install the package `cm-super`, which contains Type1 versions of standard fonts.

If your LaTeX setup does not contain scalable fonts for your document, it will often fall back on Metafont fonts in bitmapped Type3 format. Ipe will show an error message informing you about Type3 fonts in your LaTeX output. In that case, try installing `cm-super`, or use a package that explicitly uses Postscript fonts. For instance, for Russian you could install the *PsCyr* package, and the following preamble:

```
\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}
\usepackage{pscyr}
```

6.2 Using Xetex

If you set the LaTeX engine to `xetex` (you set this in the *Document properties* dialog, available on the *Edit* menu), then Ipe will use Xelatex to convert your text objects to PDF. Xetex supports Unicode natively, and will give you access to many scalable fonts on your system.

For instance, to use Thai in your Ipe document, it suffices to set the Latex engine to `xetex`, and to select a Thai font in the preamble:

```
\usepackage{fontspec}
\setmainfont{Garuda}
```

(On a Unix-system, you can determine which fonts on your system support Thai by saying `fc-list :lang=th` on the command line.)

Unfortunately, running Xetex is significantly slower than Pdflatex, so you may want to turn off the automatic running of LaTeX in the *File* menu.

If your script uses right-to-left writing, you will need *some more effort*.

6.3 Chinese, Japanese, and Korean

Using Korean Hangul and Hanja in Ipe is quite easy, by placing this in your preamble:

```
\usepackage[utf]{kotex}
```

Japanese LaTeX-users had long used a specialized version of TeX. Fortunately, the new `luatex` implementation makes this unnecessary, so we can now typeset Japanese by setting the *Latex engine* to `luatex` and including the `luatexja` package in the preamble:

```
\usepackage{luatexja}
```

To typeset Chinese, the *xeCJK* package of CTeX also works with Ipe. Set the *Latex engine* to `xetex`, and include the package in your preamble:

```
\usepackage{xeCJK}
```

6.4 Right-to-left writing: Farsi, Hebrew, and Arabic

Scripts with right-to-left writing require some extra care. The main document needs to be processed left-to-right for Ipe to work correctly. Only individual text objects can be translated using right-to-left mode.

Here are solutions that work for Farsi (Persian), Hebrew, and Arabic.

6.4.1 Persian

Include the stylesheet *right-to-left.isy* from the Ipe stylesheet folder. It defines a text style `rtl` for right-to-left text objects.

In the document properties (that is, in the *Document properties* dialog, available on the *Edit* menu), set `Latex engine` to `xetex`, and the `Latex preamble` to

```
\usepackage[documentdirection=lefttoright]{xepersian}
\settextfont{FreeFarsi}
```

I needed to install the packages `texlive-lang-arabic` and `fonts-freefarsi` on my Linux system to use this. On a Unix-system, you can determine which fonts on your system support Farsi by saying `fc-list :lang=fa` on the command line.

It is important to set the option `documentdirection=lefttoright` for the *xepersian* package, to make sure the main document is processed in left-to-right mode.

You can now have text objects with Latin script using the `normal` text style, and text objects with Persian script using the `rtl` text style.

If you want, you can make `rtl` the default text style, with the following *customization*:

```
prefs.initial_attributes.textstyle = "rtl"
```

If you do not use the *right-to-left.isy* stylesheet, then you have to put one more line in the preamble:

```
\ipedefinecolors{}
\usepackage[documentdirection=lefttoright]{xepersian}
\settextfont{FreeFarsi}
```

This is necessary, because Ipe normally loads the `xcolors` package after processing the document preamble. Some packages (like `bidi` and `xepersian`) require to be loaded *after* `xcolors`, so you need to use `\ipedefinecolors{}` to load `xcolors` early. We didn't need this above, because the stylesheet `right-to-left.isy` already contains the command.

6.4.2 Hebrew

Include the stylesheet *right-to-left.isy* from the Ipe stylesheet folder. It defines a text style `rtl` for right-to-left text objects.

In the document properties (that is, in the *Document properties* dialog, available on the *Edit* menu), set `Latex engine` to `xetex`, and the `Latex preamble` to

```
\ipedefinecolors{}
\usepackage{fontspec}
\setmainfont{Liberation Serif}
\setmonofont{Liberation Mono}
\setsansfont{Liberation Sans}
\usepackage{bidi}
```

I needed to install the package `texlive-lang-arabic` to use the `bidi` package. On a Unix-system, you can determine which fonts on your system support Hebrew by saying `fc-list :lang=he` on the command line.

You can now have text objects with Latin script using the `normal` text style, and text objects in Hebrew using the `rtl` text style.

If you want, you can make `rtl` the default text style, with the following *customization*:

```
prefs.initial_attributes.textstyle = "rtl"
```

6.4.3 Arabic

In the document properties (that is, in the *Document properties* dialog, available on the *Edit* menu), set `Latex engine` to `luatex`, and the `Latex preamble` to

```
\usepackage{arabluatex}
```

If you don't want to use the standard Amiri font, select another font in the preamble:

```
\newfontfamily\arabicfont[Script=Arabic]{KacstLetter}
```

On a Unix-system, you can list the fonts on your system supporting Arabic by saying `fc-list :lang=ar` on the command line.

You can now create text objects in Arabic using the macro verb+arb+ and the environment `arab`.

The following stylesheet *arabic.isy* makes this more comfortable:

```
<ipestyle name="arabic">
<textstyle name="arabic" type="minipage" begin="\begin{arab}" end="\end{arab}"/>
<textstyle name="arabic" type="label" begin="\arb{" end="}"/>
</ipestyle>
```

If you add this stylesheet to your document, you can select the `arabic` style for text objects, and directly write in Arabic inside these objects.

If you want, you can make `arabic` the default text style, with the following *customization*:

```
prefs.initial_attributes.textstyle = "arabic"
```

CHAPTER

SEVEN

PRESENTATIONS

An Ipe presentation is an Ipe PDF document that is presented using a PDF viewer and a video projector. Ipe has a number of features that make it easier to make such presentations.

Ipe comes with the presentation tool IpePresenter. It shows the current slide in one window (which you can make full screen on the external display), while showing the current slide, the next slide, notes for the current page, as well as a timer on your own display. IpePresenter not only works for Ipe presentations, but also for presentations made with beamer. It easily fits on and runs from a USB-stick, and does not require Latex on the computer where you give the presentation.

The following sections explain Ipe features that are useful for making presentations.

Ipe comes with a somewhat basic style sheet `presentation.isy` for making presentations. For a more sophisticated presentation style sheet, have a look at Jens' webpage.

7.1 Presentation stylesheets

A presentation *must* use a dedicated stylesheet. Presentations must use much larger fonts than what is normal for a figure. Don't try to make use of the `LARGE` and `huge` textsizes, but use a stylesheet that properly defines `normal` to be a large textsize.

Ipe comes with a style sheet `presentation.isy` that can be used for presentations. To create a new presentation, you can simply say:

```
ipe -sheet presentation
```

Note that `presentation.isy` is meant to be used *instead* of `basic.isy` (not in addition to it).

This presentation stylesheet enlarges all standard sizes by a factor 2.8. Note the use of the `<textstretch>` element to magnify text:

```
<textstretch name="normal" value="2.8"/>
<textstretch name="large" value="2.8"/>
```

The text size you choose from the Ipe user interface (*large*, for instance) is in fact used for *two* symbolic attributes, namely `textsize` (where *large* maps to `\large`) and `textstretch` (where it maps to no stretch in the standard style sheet). By setting the text stretch, you can magnify fonts.

In addition, the `<layout>` element in this stylesheet redefines the paper size to be of the correct proportions for a projector, and defines a smaller area of the paper as the *frame*. The frame is the area that should be used for the contents. The *Insert text box* function, for instance, creates text objects that fill exactly the width of the frame.

The `<titlestyle>` element defines the style of the page *title* outside the frame. You can set the title for each page using *Page* → *Edit title & sections*.

The LaTeX-preamble defined in the <preamble> element redefines the standard font shape to cmss (Computer Modern Sans Serif). Many people find sans-serif fonts easier to read on a screen. In addition, it redefines the list environments to use less spacing, and the text styles to not justify paragraphs (the <textstyle> elements).

If you wish to use the page transition effects of Acrobat Reader, you can define the effects in the stylesheet (using <effect> elements), and set them using *View* → *Edit effect*.

If you want to number the pages of the presentation, you'll need to use the <pagenumberstyle> element. It defines color, size, position, and alignment of the page number, and provides a template for the page number text. The template can use the following Latex counters:

- ipePage for the current page number,
- ipeView for the current view number,
- ipePages for the total number of pages,
- ipeViews for the number of views of the current page,

and the special macro \ipeNumber{x}{y} that evaluates to x when the page has only one view, and to y if there is more than one view.

If the template is empty, this has the same effect as the following definition (this is also the default definition):

```
\ipeNumber{\arabic{ipePage}}{\arabic{ipePage} - \arabic{ipeView}}
```

The following example definition indicates the views of a single page with letters A, B, C, . . . :

```
<pagenumberstyle pos="300 100" size="Huge" color="navy"
halign="center">\ipeNumber{\arabic{ipePage}}{\arabic{ipePage}
\Alph{ipeView}}</pagenumberstyle>
```

The following definition uses roman numerals for the pages and does not indicate different views at all:

```
<pagenumberstyle pos="580 20" size="Huge" color="purple"
halign="right">\roman{ipePage}</pagenumberstyle>
```

And this shows both the current page and the total number of pages:

```
<pagenumberstyle pos="20 820" size="Huge" color="0.5 0 0"
valign="top">\arabic{ipePage}/\arabic{ipePages}</pagenumberstyle>
```

7.2 Views

When making a PDF presentation for a PDF Viewer or for IpePresenter, one would often like to present a page incrementally. For instance, I would first like to show a polygon, then add its triangulation, and finally color the vertices. *Views* make it possible to do this nicely.

An Ipe document consists of several pages, each of which can consist of an arbitrary number of views. When saving as PDF, each view generates a separate PDF page (if you only look at the result in a PDF viewer, you cannot tell whether two pages are actually two views of the same Ipe page or two different Ipe pages).

An Ipe page consists of a number of objects, a number of layers, and a number of views. Each object belongs to exactly one layer. A layer can be shown by any number of views—a view is really just a list of layers to be presented. In addition, a view keeps a record of the current active layer—this makes it easy to move around your views and edit them.

Views can also give a different meaning to symbolic attributes, so that objects that appear in blue on one view will be red on the next one. One can also change the dash pattern and pen width, and can even replace one symbol by a different one.

As a somewhat experimental feature, views can transform individual layers of the page. This allows you to move around or rotate objects from view to view, without having to make copies of the objects.

Finally, views can specify a graphic effect to be used by the PDF viewer when proceeding to the following PDF page.

7.2.1 Example

To return to our polygon triangulation example, let's create an empty page. We draw a polygon into the default layer *alpha*. Now use *Views* → *New layer, new view*, and draw the triangulation into the new layer *beta*. Note that the function not only created a new layer, but also a second view showing both *alpha* and *beta*. Try moving back and forth between the two views (using the `PageUp` and `PageDown` keys). You'll see changes in the layer list on the left: in view 1, layer *alpha* is selected and active, in view 2, both layers are selected and *beta* is active. Create a third layer and view, and mark the vertices.

Save in PDF format, and voila, you have a lovely little presentation. The result is available here.

7.2.2 Text boxes

In presentations, one often has slides with mostly text. The *textbox* object is convenient for this, as one doesn't need to use the mouse to create it. To create a slide where several text items appear one by one, one only needs to press F10 to create a textbox, then `Shift+Ctrl+I` to make a new view, F10 again for the next textbox, and so on. Finally, one moves the textboxes vertically for the most pleasing effect (*Shift+Alt+Left Mouse* does a constrained vertical translation, or *Shift+Left Mouse* in *Translate* mode).

7.2.3 Changing symbolic values inside a view

Imagine you have an object that is shown in dark orange on your current page, but on one specific view you want to highlight the object in purple.

To achieve this, go to the view, and open *Edit view* from the *Views* menu. In the attribute map field, write this line:

```
color:darkorange=purple;
```

You will notice that all objects on the page that had the `darkorange` color now show in purple.

You can similarly map other attribute values, for instance:

```
color:navy=red;
color:darkorange=purple;
symbolsize:large=tiny;
pen:ultrafat=normal;
dashstyle:dotted=dashed;
symbol:mark/disk(sx)=mark/box(sx);
```

The attribute kinds you are allowed to modify are "pen", "symbolsize", "arrowsize", "opacity", "color", "dashstyle", and "symbol". Both the original and the new attribute must be symbolic attributes defined in your style sheet.

It's probably not a good idea to remap common attributes in your document—this may quickly become confusing. A better approach would be to create dedicated attribute values like `emphasis`, that map to some standard value in the stylesheet, and which you can then redefine in the views where you want to emphasize the object.

7.2.4 Transforming objects inside a view

The second large text field inside the *Edit view* dialog allows you to specify transformations for each layer of the page. For instance, you could write these definitions:

```
alpha=[1 0 0 1 20 100];
beta=[1 0 0 0.8 0 0];
```

This will translate all objects in layer alpha by the vector (20, 100) (that is, 20 units to the right and 100 units upwards), while all objects in layer beta are scaled in the *y*-direction by factor 0.8.

This is somewhat experimental—there is no good UI to figure out the right transformation matrices (you may want to draw a helper object, apply the transformation, and then use *Edit as XML* to look at the object's matrix field).

Also, Ipe still considers the objects to be at their original place, it just displays them elsewhere. For instance, to modify an object you have to select it by clicking at its original location. (So, to modify an object, it is best is to go back to the original view where you made the object.)

Currently, the layer transformation is not taken into account when computing the bounding box of a page, so make sure you have other objects creating a suitable bounding box.

7.2.5 Controlling the PDF bounding box of a view

Note that all views of a page receive the same bounding box, containing all objects visible on some view, plus all objects in a layer named `BBOX` (even if that layer is not visible). This can be used to force a larger bounding box without adding a white rectangle or the like.

If you need independent bounding boxes for each view, create a layer named `VIEWBBOX`. Any view in which this layer is visible will receive a bounding box computed for the objects visible in this view only.

7.3 Bookmarks

You can set a section title and a subsection title for each page of an Ipe document. Theses titles will be shown in the *bookmarks list* (right-click on a toolbar to make it visible). Double-clicking a title brings you directly to its page, making navigation of long documents much easier. The titles are also exported to PDF, and are visible in the bookmarks view of PDF viewers.

You can also use the section title of a page to refer to this page in **`ipetoipe`** and **`iperender`**.

7.4 Excluding a page from the presentation

If you add a layer with name `NOPDF` to a page (it does not need to be visible in any view of the page), then the page is not included in the PDF representation of the document. So while Ipe will show the page normally, viewing the document with a PDF reader (including IpePresenter) will not show that page.

7.5 Gradient patterns

Gradient patterns allow to shade objects with continuously changing colors. This is often used for backgrounds, or to achieve the illusion of three-dimensional spheres or cylinders.

The intended use of gradients is to allow the creation of attractive symbols inside the style sheet, for backgrounds, as bullets in item lists (see next section), or simply to define attractive glassy-ball symbols and the like that can be used through the *Use symbol* ipelet.

The Ipe user interface does not offer any way of creating or editing gradients. If your stylesheet defines a gradient, then it is possible to fill a path object with this gradient, but getting the gradient coordinate system right is not trivial. (The trick is to draw the path object at gradient coordinates, and translate/rotate it to the final location afterwards.)

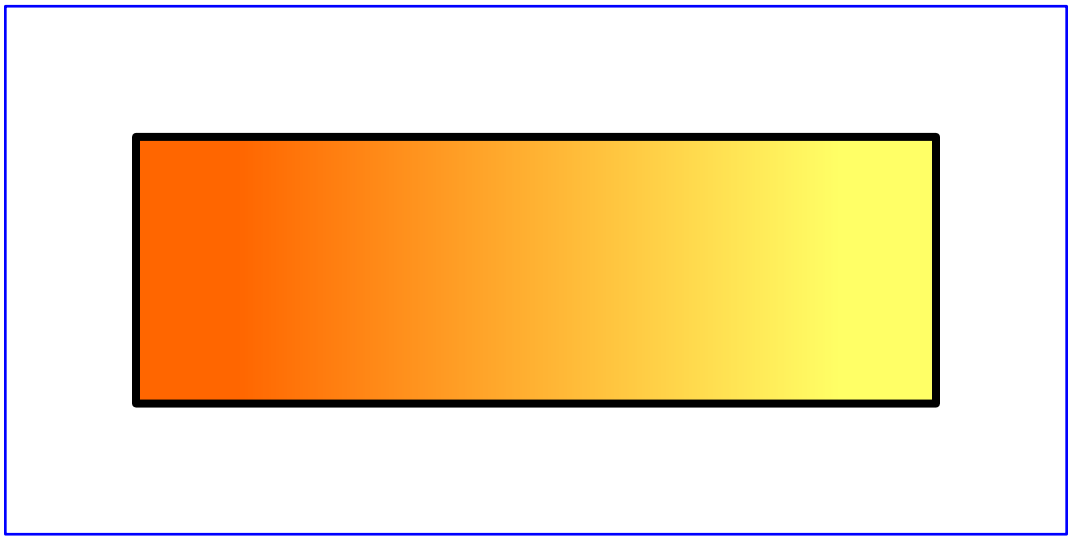
The definition of a linear (axial) gradient looks like this:

```
<gradient name="linear" type="axial" extend="yes" coords="75 0 325 0">
  <stop offset="0.05" color="1 0.4 0"/>
  <stop offset="0.95" color="1 1 0.4"/>
</gradient>
```

If used like this:

```
<path stroke="0" fill="1" gradient="linear" pen="3">
  50 50 m 350 50 l 350 150 l 50 150 l h
</path>
```

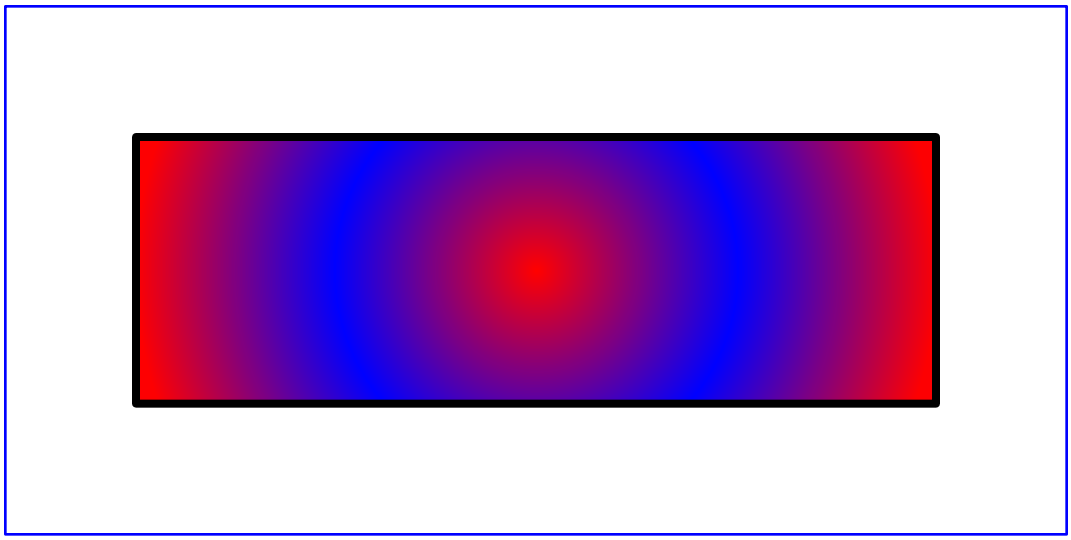
it will look like this:



A radial gradient looks like this:

```
<gradient name="radial" type="radial" extend="yes"
          coords="200 100 0 200 100 150">
  <stop offset="0" color="1 0 0"/>
  <stop offset="0.5" color="0 0 1"/>
  <stop offset="1" color="1 0 0"/>
</gradient>
```

It will look like this:

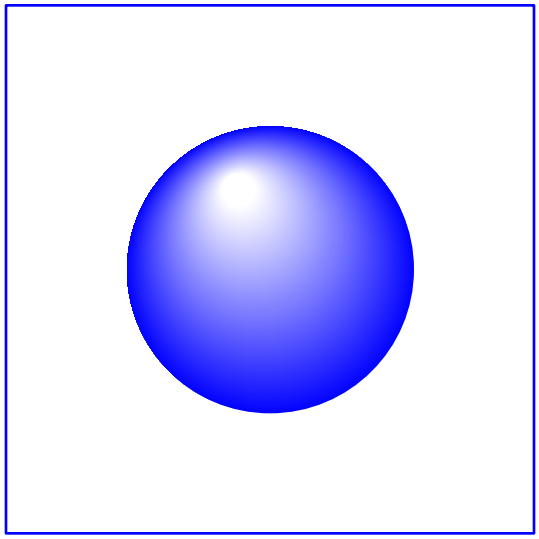


A common use of radial gradients is to define glassy balls like this:

```
<gradient name="ball" type="radial" coords="-4 10 2 0 0 18">
  <stop offset="0" color="1 1 1"/>
  <stop offset="1" color="0 0 1"/>
</gradient>
```

Note that the gradient is centered at *0 0*, so it needs to be moved to the location where it is used:

```
<path matrix="3 0 0 3 100 100" fill="1" gradient="ball">
  18 0 0 18 0 0 e
</path>
```



Normally, you would define a symbol looking like a glassy ball in your style sheet:

```
<ipestyle>
  <gradient name="ball" type="radial" coords="-4 10 2 0 0 18">
    <stop offset="0" color="1 1 1"/>
    <stop offset="1" color="0 0 1"/>
```

(continues on next page)

(continued from previous page)

```
  </gradient>
  <symbol name="ball(x)" transformations="translations">
    <path fill="1" gradient="ball"> 18 0 0 18 0 0 e </path>
  </symbol>
</ipestyle>
```

The glassy ball can then be used in the document using the *Use symbol* ipelet. Note that `transformations="translations"` ensures that stretching your drawing does not change the glassy ball. Adding the `(x)` suffix to the symbol name allows you to resize the glassy ball by changing the symbol size from the properties (the same selector used to change the mark size).

For the precise syntax of the gradient definition see *here*. The easiest method of creating gradients, though, is to use an SVG editor such as **Inkscape** and to convert the SVG gradient to Ipe format using **svgtoipe**.

7.6 Prettier bullet points

Presentations often make use of `itemize` environments. You can make these prettier in a number of ways:

You can color your bullets:

```
<preamble>
\def\labelitemi{\LARGE\textcolor{red}{$\bullet$}}
</preamble>
```

Enumeration numbers could be put in a colored box:

```
<preamble>
\newcommand{\labelenumi}{\fbox{\Roman{enumi}}}
</preamble>
```

You could use the Dingbats font for nice enumerations, for instance by putting `\usepackage{pifont}` in your preamble, and then having text objects with `\begin{dinglist}{43}` or `\begin{dingautolist}{172}` (or use 182, 192, 202 for various circled numbers).

You can mark items as *good* and *bad* using these "bullets":

Bad item: `\textcolor{red}{\ding{55}}`
Good item: `\textcolor{green}{\ding{52}}`

7.7 Filming from the canvas

Some Ipe users make presentation videos directly from Ipe, as follows:

- Make Ipe full-screen (Zoom menu);
- Turn off the grid (Zoom menu);
- Enable pretty display (Zoom menu);
- Mac: Turn off the Snap toolbar and possibly the mode toolbar (Zoom menu);
- Linux: Turn off or rearrange toolbars and panels to taste;
- Windows: Turn off or rearrange toolbars by setting `prefs.win_toolbar_order`;

- Set up your video software to film the canvas area of Ipe (where your document is shown).

You can now go through your presentation, using the *Laser pointer* mode to point to items on your slide, and using ink mode to write annotations directly onto the slides.

For this to work well, it is essential that the width of the canvas remains fixed while you step through your document. Try it before you start filming! You can change the contents of the "View" and "Page" buttons by changing the preferences `prefs.view_button_prefix` and `prefs.page_button_prefix`. Furthermore, on MacOS, you can force the canvas width to remain fixed by making `prefs.osx_properties_width` large enough.

CHAPTER

EIGHT

ADVANCED TOPICS

8.1 Multiple figures in one Ipe document

When writing an article in Latex, you can store all the figures in a single Ipe document, by placing each figure on its own page. In Latex, each figure can then be included by saying, for instance,

```
\includegraphics[page=7]{figs}
```

to include the figure on page 7 of the Ipe document `figs.pdf`.

To avoid having to remember which figure is on which page (and having to renumber all figures when you insert or delete a figure), you can give *names* to the pages (figures) in your Ipe document, using *Pages* → *Edit title & sections*. Enter the name in the *Section* field, using only letters.

Then, run the *page-labels* script on your document, as follows:

```
ipescript page-labels figs.pdf
```

This reads the document `figs.pdf` and writes a new file `figsLabels.tex`.

In your Latex document, include this file in the preamble:

```
\input{figsLabels}
```

You can now include a figure using its symbolic name, like this:

```
\includegraphics[page=\ipeFigXXX]{figs}
```

where `XXX` is the symbolic name of the figure.

Whenever you add or delete a figure to the document, just run the *page-labels* script again, and your symbolic names will remain up to date.

If you want to refer to a specific view of a page, give the view a name (*Views* → *Edit view*). The *page-labels* script will then generate a label consisting of first the page name and the view name. For instance, for the view named *YYY* on the page named *XXX*, the label will be `\ipeFigXXXYYY`.

8.2 Sharing Latex definitions with your Latex document

When using Ipe figures in a Latex document, it is convenient to have access to some of the definitions from the document.

Ipe comes with a Lua script *update-master* that makes this easy.

In your Latex document, say `master.tex`, surround the interesting definitions using `%%BeginIpePreamble` and `%%EndIpePreamble`, for instance like this:

```
%%BeginIpePreamble
\usepackage{amsfonts}
\newcommand{\R}{\mathbb{R}}
%%EndIpePreamble
```

Running the script as

```
ipescript update-master master.tex
```

extracts these definitions and saves them as a stylesheet `master-preamble.isy`. (This filename is fixed, and does not depend on the document name.)

Running this script as

```
ipescript update-master master.tex figures/*.ipe
```

creates the stylesheet `master-preamble.isy` as above. In addition, it looks at all the Ipe figures mentioned on the command line. The script adds the new stylesheet to each figure, or updates the stylesheet to the newest version (if the figure already contains a stylesheet named `master-preamble`).

8.3 Writing ipelets

An ipelet is an extension to Ipe. Ipe 7 uses the scripting language Lua (in fact, most of the Ipe program itself is written in Lua), and loads ipelets written in Lua when it starts up. It is also possible to write ipelets in C++, using a small Lua wrapper that declares the methods available inside the ipelet.

Documentation about writing ipelets can be found in the Ipelib documentation.

8.4 Troubleshooting the LaTeX-conversion

Ipe converts text objects from their Latex source representation to a representation that can be rendered and included the PDF output by creating a Latex source file and running Pdflatex. This happens in a dedicated directory, which Ipe creates the first time it is used. The Latex source and output files are left in that directory and not deleted even when you close Ipe, to make it easy to solve problems with the Latex conversion process.

You can determine the directory used by Ipe using *Help* → *Show configuration*. If you'd prefer to use a different directory, set the environment variable `IPELATEXDIR` before starting Ipe.

If Ipe fails to translate your text objects, and you cannot find the problem by looking at the log file displayed by Ipe (or Ipe doesn't even display the log file), you can terminate Ipe, go to the conversion directory, and run Pdflatex manually:

```
pdflatex ipetemp.tex
```

8.5 Customizing Ipe

Since most of Ipe is writing in Lua, an interpreted language, much of Ipe's behavior can be changed without recompilation.

The main customization options are in the files `prefs.lua` (general settings), `shortcuts.lua` (keyboard shortcuts), and `mouse.lua` (mouse shortcuts). (Check the Lua code path in *Help* → *Show configuration* if you can't locate the files.)

If you have installed Ipe for your personal use only (for instance under Windows), you can simply modify the original Lua file. In all other cases, you need to provide a small Lua ipelet that will change the setting you wish to change.

A small example is the following ipelet that changes a keyboard shortcut and the maximum zoom:

```
----------------------------------------------------------------------
-- My customization ipelet: customize.lua
----------------------------------------------------------------------
prefs.max_zoom = 100
shortcuts.insert_text_box = "I"
shortcuts.mode_splines = "Alt+Ctrl+I"
```

The ipelet needs to be placed with the extension `.lua` somewhere on the ipelet path (check *Show configuration* again). On Unix, the directory `~/.ipe/ipelets` will do nicely. On Windows, you will have to set the environment variable IPELETPATH, see the next section.

8.6 Environment variables

Ipe, **ipetoipe**, **iperender**, and **ipescript** respect the following environment variables:

IPELATEXDIR the directory where Ipe runs Latex.

IPELATEXPATH the directory that contains the *pdflatex*, *xelatex*, and *lualatex* commands. If not set, Ipe assumes the commands are on your path.

IPEDEBUG set to 1 for debugging output.

IPETEXFORMAT if set, Ipe will not call pdflatex but pdftex requesting the pdflatex format (and similarly for xetex and luatex).

The Ipe program uses several additional environment variables:

EDITOR external editor to use for editing text objects.

IPESTYLES a list of directories, separated by semicolons on Windows and colons otherwise, where Ipe looks for stylesheets, for instance for the standard stylesheet `basic.isy`. You can write _ (a single underscore) for the system-wide stylesheet directory. If this variable is not set, the default consists of the system-wide stylesheet directory, plus `~/.ipe/styles` on Unix, plus `~/Library/Ipe/Styles` on OS X.

IPELETPATH a list of directories, separated by semicolons on Windows and colons otherwise, containing ipelets. You can write _ (a single underscore) for the system-wide ipelet directory. If this variable is not set, the default consists of the system-wide ipelet directory, plus `~/.ipe/ipelets` on Unix, plus `~/Library/Ipe/Ipelets` on OS X.

IPEICONDIR directory containing icons for the Ipe user interface.

IPEDOCDIR directory containing Ipe documentation.

IPELUAPATH path for searching for Ipe Lua code.

The **ipescript** program uses the following environment variable:

IPESCRIPTS a list of directories, separated by semicolons on Windows and colons otherwise, where *ipescript* looks for scripts. You can write _ (a single underscore) for the system-wide script directory. If this variable is not set, the default consists of the current directory and the system-wide script directory, plus `~/.ipe/scripts` on Unix, plus `~/Library/Ipe/Scripts` on OS X.

On Windows, you can use the special drive "letter" `ipe:` inside environment variables. Ipe translates it into the drive letter for the drive containing your Ipe executables.

8.6.1 ipe.conf

Ipe allows you to set environment variables by writing the definitions in a file *ipe.conf*. On Windows, the file has to be in the top level of the Ipe directory (the same place that contains the *readme.txt* and *gpl.txt* files), on Linux and OSX it is simply *.ipe/ipe.conf* in your home directory. Each line of the file contains a setting for one environment variable, for instance like this:

```
IPEDEBUG=1
IPELATEXDIR=C:\latexrun
```

8.7 Ipe on a USB-stick

Ipe for Windows is entirely self-contained—you can unzip the package anywhere, including on a USB-stick, and run Ipe from there. However, Ipe needs access to a LaTeX installation. One option is to use LaTeX online—you can enable this from the *Help* menu.

The other option is to install LaTeX as a portable installation on the same USB-stick:

- You can use the MiKTeX portable edition.
- The alternative is to use texlive. Download *install-tl-windows.exe* and run it, selecting the option for *Custom install*. When the full installer starts, change *Portable setup* to *Yes*, and change the *TEXDIR* directory to `X:\texlive`, where *X* is the drive letter of the USB-stick. You can also decrease the size of your installation by unselecting unnecessary packages and languages.

Installing LaTeX on the USB-stick will take quite a while. When it is done, unzip the Ipe package for Windows onto the root of the USB-stick. You should have the Ipe binary in `X:\ipe-7.x.y\bin\ipe.exe`.

Now create a directory `X:\latexrun` (it will be used by Ipe to run Latex).

Finally, you need to create a small *configuration file* as `X:\ipe-7.x.y\ipe.conf`. The contents of the file should be (for MiKTeX):

```
IPELATEXPATH=ipe:\miktex-portable\texmfs\install\miktex\bin\x64
IPELATEXDIR=ipe:\latexrun
```

or (for texlive):

```
IPELATEXPATH=ipe:\texlive\bin\win32
IPELATEXDIR=ipe:\latexrun
```

The `ipe:` part is translated by Ipe into the drive letter for the drive that Ipe is executed from, so it will correctly point to your USB-stick.

Double-check the setting for the `IPELATEXPATH`—it should be the directory that contains the `pdflatex.exe` program. Open Ipe and look *Help* → *Show configuration*, and check that all settings are correct.

It should now work to run Ipe from the USB-stick, whenever you plug it into a Windows computer.

If you want to make further *customization settings*, you can use the variable `config.ipedir` in your Lua code to refer to the USB-stick drive.

8.8 Running Ipe under Wine on Linux

Ipe itself works fine under Wine, but there is an issue: We don't want to create a new tex installation for Windows under Wine, we want to reuse the Linux tex installation!

So first we need to make the pdflatex program available to Wine, by putting a symbolic link in the simulated `C:` drive.

```
$ cd  /.wine/drive_c/windows/
$ ln -s /usr/bin/pdflatex pdflatex.exe
```

The second problem is that Ipe is not able to wait for the completion of the pdflatex call—it starts pdflatex, and then immediately tries to read the pdflatex output, which of course fails. The solution is to make Ipe wait for a specified number of milliseconds before trying to read the pdflatex output.

You achieve this by creating a small text file called *ipe.conf* and placing it in the top-level Ipe directory (that is, the directory that contains *readme.txt* and *gpl.txt*). The contents of the file should be:

```
IPEWINE=1000
IPELATEXPATH=c:\windows
```

(You can define any other environment variable in the same file.)

CHAPTER

NINE

THE IPE FILE FORMAT

Ipe can store documents in two different formats. One of them is standard PDF, which can be read by any application capable of opening PDF files. (Ipe embeds its own information inside PDF files. The way this is done is not documented here, and may change between releases of Ipe.)

The second Ipe file format is a pure XML implementation. Files stored in this format can be parsed with any XML-aware application, and you can create XML files for Ipe from your own applications.

A DTD for the Ipe format is available as ipe.dtd. For instance, you can use this to validate an Ipe document using

```
xmllint --dtdvalid ipe.dtd --noout file.ipe
```

or

```
xmlstarlet val -d ipe.dtd file.ipe
```

The tags understood by Ipe are described informally in this section.

Ipe does not use a full XML-parser, in particular it does not understand namespaces and doesn't allow certain elements that have no contents to be expanded (you cannot, for instance, write <info></info> instead of <info />).

Tags in the XML file can carry attributes other than the ones documented here. Ipe ignores all attributes it doesn't understand, and they will be lost if the document is saved again from Ipe. Ipe will complain about any XML elements not described here, with the exception that you can use elements whose name starts with `x-` freely to add your own information inside an Ipe file.

An Ipe XML file must contain exactly one <ipe> element, while an Ipe stylesheet file must contain exactly one <ipestyle> element (both types of files are allowed to start with an <?xml> tag, which is simply ignored by Ipe). An Ipe file may also contain a <!DOCTYPE> tag.

All elements are documented below.

9.1 The <ipe> element

Attributes

`version` (required)

> The value (a number, e.g. 70103 for Ipelib 7.1.3) indicates the earliest Ipelib version that can interpret the document. Ipe will refuse to load documents that require a version larger than its own, and may refuse to load documents that are too old (and which will have to be converted using a separate program).

`creator` (optional)

> indicates the program that created the file. This information is not interpreted by Ipe at all.

Contents

1. An <info> element (optional),
2. a <preamble> element (optional),
3. a series of <bitmap> and <ipestyle> elements (optional),
4. a series of page elements.

The <ipestyle> elements form a *cascade*, with the *last* <ipestyle> element becoming the *top-level* style sheet. When symbolic names are looked up, the style sheets are checked from top to bottom. Ipe always appends the built-in standard style sheet at the bottom of the stack.

9.1.1 The <info> element

Attributes

title (optional) document title,

author (optional) document author,

subject (optional) document subject,

keywords (optional) document keywords,

language (optional) the document language, to be used for spell checking. E.g. *de_DE* for German spell checking. Currently has an effect only on Linux when spell checking is compiled in.

pagemode (optional) the only value understood by Ipe is fullscreen, which causes the document to be opened in full screen mode in PDF readers.

created (optional) creation time in PDF format, e.g. D:20030127204100.

modified (optional) modification time in PDF format,

numberpages (optional) if the value is yes, then Ipe will save PDF documents with visible page numbers on each page.

sequentialtext (optional) if the value is yes, then Ipe will typeset all text objects one by one, even if they are identical, in the order in which they appear in the document.

tex (optional) determines the TeX-engine used to translate your text. The possible values are pdftex, xetex, and luatex.

This element must be empty.

9.1.2 The <preamble> element

The contents of this element is LaTeX source code, to be used as the LaTeX preamble when running LaTeX to process the text objects in the document. It should *not* contain a \documentclass command, but can contain \usepackage commands and macro definitions.

9.1.3 The <bitmap> element

Each <bitmap> element defines a bitmap to be used by <image> objects.

Attributes

id (required) the value must be an integer that will define the bitmap throughout the Ipe document,

width (required) integer width in pixels,

height (required) integer height in pixels,

ColorSpace (optional) possible values are DeviceGray, DeviceGrayAlpha, DeviceRGB (default value), and DeviceRGBAlpha. The suffix Alpha indicates the presence of an alpha channel.

ColorKey (optional) an RGB color in hexadecimal, indicating the transparent color (not supported for JPEG images and for images with alpha channel),

length (required unless there is no filter and no alpha channel) the number of bytes of image data,

Filter (optional) possible values are FlateDecode or DCTDecode to indicate a compressed image (the latter is used for JPEG images),

encoding (optional) possible value is base64 to indicate that the image data is base64-encoded (not in hexadecimal),

alphaLength (optional) indicates that the alpha channel is provided separately.

The contents of the <bitmap> element is the image data, either base64-encoded or in hexadecimal format. White space between bytes is ignored. If no filter is specified, pixels are stored row by row.

If alphaLength present, then the alpha channel follows the image data. If the data is deflated, image data and alpha channel are deflated separately. If no alphaLength is present, then the alpha component is part of each pixel before the color components.

Bitmaps use 8-bit color and alpha components. Bitmaps with color maps or with a different number of bits per component are not supported, and such support is not planned. (The *Insert image* function does allow you to insert arbitrary image formats, but they are stored as 8-bit per component images. Since the data is compressed, this does not seriously increase the image data size.)

9.2 The <page> element

Attributes

title (optional) title of this page (displayed at a fixed location in a format specified by the style sheet),

section (optional) Title of document section starting with this page. If the attribute is not present, this page continues the section of the previous page. If the attribute is present, but its value is an empty string, then the contents of the title attribute is used instead.

subsection (optional) Title of document subsection starting with this page. If the attribute is not present, this page continues the subsection of the previous page. If the attribute is present, but its value is an empty string, then the contents of the title attribute is used instead.

marked (optional) The page is marked for printing unless the value of this attribute is no.

Contents

1. An optional <notes> element,
2. a possibly empty sequence of <layer> elements,
3. a possibly empty sequence of <view> elements,

4. a possibly empty sequence of Ipe object elements.

If a page contains no layer element, Ipe automatically adds a default layer named `alpha`, visible and editable.

If a page contains no view element, a single view where all layers are visible is assumed.

9.2.1 The <notes> element

This element has no attributes. Its contents is plain text, containing notes for this page.

9.2.2 The <layer> element

Attributes

name (required) Name of the layer. It must not contain white space.

edit (optional) The value should be `yes` or `no` and indicates whether the user can select and modify the contents of the layer in the Ipe user interface (of course the user can always modify the setting of the attribute).

snap (optional) The value should be `never`, `visible`, or `always`, and indicates whether snapping to this layer is enabled. The default is `visible`.

data (optional) A free-use string associated with the layer. Ipe makes no use of this string, except for layers it creates itself for group edits.

The layer element must be empty.

9.2.3 The <view> element

Attributes

layers (required) The value must be a sequence of layer names defined in this page, separated by white space.

active (required) The layer that is the active layer in this view.

effect (optional) The symbolic name of a graphics effect to be used during the PDF page transition. The effect must be defined in the style sheet.

name (optional) The name of the view.

marked (optional) The view is marked for printing if the value of this attribute is `yes`.

The view element may be empty, or it may contain a sequence of attribute mappings and layer transformations.

An attribute mapping is a <map> element with three attributes

`kind`, `from`, and `to`. The `kind` attribute must have one of the following values:

```
pen, symbolsize, arrowsize, opacity, color, dashstyle, symbol
```

The attributes `from` and `to` must both be names of symbolic attribute values. The effect is that in this view, the symbolic attribute `from` is replaced by the symbolic attribute `to` for attributes of the given kind.

A layer transformation is a <transform> element. It must have two attributes: `layer` is the name of a layer of the page, `matrix` is a transformation matrix. In the view, all objects on that layer are transformed with this matrix.

9.3 Ipe object elements

9.3.1 Common attributes

layer (optional) Only allowed on top-level objects, that is, objects directly inside a <page> element. The value indicates into which layer the object goes. If the attribute is missing, the object goes into the same layer as the preceding object. If the first object has no layer attribute, it goes into the layer defined first in the page, or the default alpha layer.

matrix (optional) A sequence of six real numbers, separated by white space, indicating a transformation matrix for all coordinates inside the element (including embedded elements if this is a <group> element). A missing matrix attribute is interpreted as the identity matrix.

pin (optional) Possible values are yes (object is fixed on the page), h (object is pinned horizontally, but can move in the vertical direction), and v (the opposite). The default is no pinning.

transformations (optional) This attribute determines how objects can be deformed by transformations. Possible values are *affine* (the default), *rigid*, and *translations*.

9.3.2 Color attribute values

A color attribute value is either a symbolic name defined in one of the style sheets of the document, one of the predefined names

black or white, a single real number between 0 (black) and 1 (white) indicating a gray level, or three real numbers in the range $[0,1]$ indicating the red, green, and blue component (in this order), separated by white space.

9.3.3 Path construction operators

Graphical shapes in Ipe are described using a series of path construction operators with arguments. This generalizes the PDF path construction syntax.

Each operator follows its arguments. The operators are

m (moveto) (1 point argument): begin new subpath.

l (lineto) (1 point argument): add straight segment to subpath.

c (cubic B-spline) (n point arguments): add a uniform cubic B-spline with $n+1$ control points (the current position plus the n arguments). If $n = 3$, this is equivalent to a single cubic Bézier spline, if $n = 2$ it is equivalent to a single quadratic Bézier spline.

q (deprecated) (2 point arguments): identical to 'c'.

e (ellipse) (1 matrix argument): add a closed subpath consisting of an ellipse, the ellipse is the image of the unit circle under the transformation described by the matrix.

a (arcto) (1 matrix argument, 1 point argument): add an elliptic arc, on the ellipse describe by the matrix, from current position to given point.

s (deprecated) (n point arguments): add an old style uniform cubic B-spline as used by Ipe up to version 7.1.6.

C (n point arguments plus one tension argument): add a cardinal cubic spline through the given points and the given tension. (The definition of the tension in the literature varies. Ipe's tangent is the factor by which the vector from previous to next point is multiplied to obtain the tangent vector at this point. So 0.5 corresponds to the Catmull-Rom spline.)

L (n point arguments): add a clothoid spline as computed by the libspiro library by Raph Levien. When Ipe writes clothoid splines, it includes the control points of the computed Bezier approximation in the path description, separated from the defining points by a *.

u (closed spline) (n point arguments): add a closed subpath consisting of a closed uniform B-spline with n control points,

h (closepath) (no arguments): close the current subpath. No more segments can be added to this subpath, so the next operator (if there is one) must start a new subpath.

Paths consisting of more than one closed loop are allowed. A subpath can consist of any mix of straight segments, elliptic arcs, and B-splines.

9.3.4 The <group> element

The <group> element allows to group objects together, so that they appear as one in the user interface.

Attributes

clip (optional) The value is a sequence of path construction operators, forming a clipping path for the objects inside the group.

url (optional) The value is a link action (and the attribute name is somewhat of a misnomer, as actions do not need to be URLs—see the description of group objects).

decoration (optional) The name of a decoration symbol. The default is *normal*, meaning no decoration.

The contents of the <group> element is a series of Ipe object elements.

9.3.5 The <image> element

Attributes

bitmap (required) Value is an integer referring to a bitmap defined in a <bitmap> element in the document,

rect (required) Four real coordinates separated by white space, in the order x_{1}, y_{1}, x_{2}, y_{2}, indicating two opposite corners of the image in Ipe coordinates).

opacity (optional) Opacity of the image. This must be a symbolic name. The default is normal, meaning fully opaque.

The image element is normally empty. However, it is allowed to omit the bitmap attribute. In this case, the <image> must carry all the attributes of the <bitmap> element, with the exception of id. The element contents is then the bitmap data, as described for <bitmap>.

9.3.6 The <use> element

The <use> element refers to a symbol (an Ipe object) defined in the style sheet. The attributes stroke, fill, pen, and size make sense only when the symbol accepts these parameters.

Attributes

name (required) The name of a symbol defined in a style sheet of the document.

pos (optional) Position of the symbol on the page (two real numbers, separated by white space). This is the location of the origin of the symbol coordinate system. The default is the origin.

stroke (optional) A stroke color (used whereever the symbol uses the symbolic color sym-stroke). The default is black.

fill (optional) A fill color (used whereever the symbol uses the symbolic color `sym-fill`). The default is white.

pen (optional) A line width (used whereever the symbol uses the symbolic value `sym-pen`). The default is `normal`.

size (optional) The size of the symbol, either a symbolic size (of type `symbol size`), or an absolute scaling factor. The default is 1.0.

The <use> element must be empty.

9.3.7 The <text> element

Attributes

stroke (optional) The stroke color. If the attribute is missing, black will be used.

type (optional) Possible values are *label* (the default) and *minipage*.

size (optional) The font size—either a symbolic name defined in a style sheet, or a real number. The default is `normal`.

pos (required) Two real numbers separated by white space, defining the position of the text on the paper.

width (required for minipage objects, optional for label objects) The width of the object in points.

height (optional) The total height of the object in points.

depth (optional) The depth of the object in points.

valign (optional) Possible values are *top* (default for a minipage object), *bottom* (default for a label object), *center*, and *baseline*.

halign (optional) Possible values are *left*, *right*, and *center*. *left* is the default. This determines the position of the reference point with respect to the text box.

style (optional) Selects a LaTeX `style` to be used for formatting the text, and must be a symbolic name defined in a style sheet. There are separate definitions for minipages and for labels. For minipages, the standard style sheet defines the styles `normal`, `center`, `itemize`, and `item`. If the attribute is not present, the `normal` style is applied.

opacity (optional) Opacity of the element. This must be a symbolic name. The default is `normal`, meaning fully opaque.

The dimensions are recomputed by Ipe when running LaTeX, with the exception of `width` for minipage objects whose width is fixed.

The contents of the <text> element must be a legal LaTeX fragment that can be interpreted by LaTeX inside `\hbox`, possibly using the macros or packages defined in the preamble.

9.3.8 The <path> element

Attributes

stroke (optional) The stroke color. If the attribute is missing, the shape will not be stroked.

fill (optional) The fill color. If the attribute is missing, the shape will not be filled.

dash (optional) Either a symbolic name defined in a style sheet, or a dash pattern in PDF format, such as `[3 1] 0` for `three pixels on, one off, starting with the first pixel`. If the attribute is missing, a solid line is drawn.

pen (optional) The line width, either symbolic (defined in a style sheet), or as a single real number. The default value is `normal`.

cap (optional) The *line cap* setting of PDF as an integer. If the argument is missing, the setting from the style sheet is used.

join (optional) The *line join* setting of PDF as an integer. If the argument is missing, the setting from the style sheet is used.

fillrule (optional) Possible values are `wind` and `eofill`, selecting one of two algorithms for determining whether a point lies inside a filled object. If the argument is missing, the setting from the style sheet is used.

arrow (optional) The value consists of a symbolic name, say `triangle` for an arrow type (a symbol with name `arrow/triangle(spx)`), followed by a slash and the size of the arrow. The size is either a symbolic name (of type `arrowsize`) defined in a style sheet, or a real number. If the attribute is missing, no arrow is drawn.

rarrow (optional) Same for an arrow in the reverse direction (at the beginning of the first subpath).

opacity (optional) Opacity of the element. This must be a symbolic name. The default is `normal`, meaning fully opaque.

stroke-opacity (optional) Opacity of the stroked part of the element. This must be a symbolic name. The default is to use the opacity attribute.

tiling (optional) A tiling pattern to be used to fill the element. The default is not to tile the element. If the element is not filled, then the tiling pattern is ignored.

gradient (optional) A gradient pattern to be used to fill the element. If the element is not filled, then the gradient pattern is ignored. If `gradient` is set, then `tiling` is ignored.

The contents of the <path> element is a sequence of path construction operators. The entire shape will be stroked and/or filled with a single stroke and fill operation.

9.4 The <ipestyle> element

Attributes

name (optional) The name serves to identify the style sheet informally, and can be used to automatically update the style sheet from a file with the matching name.

The contents of the <ipestyle> element is a series of style definition elements, in no particular order. These elements are described below.

9.4.1 The <symbol> element

Attributes

name (required) The name identifies the symbol and must be unique in the style sheet. For parameterized symbols, the name must end with the pattern `(s?f?p?x?)`, where `s` stands for stroke, `f` for fill, `p` for pen, and `x` for size.

transformations (optional) As for objects.

xform (optional) If this attribute is set, a PDF XForm will be created for this symbol when saving or exporting to PDF. It implies `transformations="translations"`, and will be ignored if any of the symbol parameters (that is, stroke, fill, pen, or size) are used. Setting this attribute will cause the PDF output to be significantly smaller for a complicated symbol that is used often (for instance, a complicated background used on every page).

snap (optional) A list of space-separated coordinates for the snap positions of the symbol.

The contents of the <symbol> element is a single Ipe object.

9.4.2 The <preamble> element

See the <preamble> elements inside <ipe> elements.

9.4.3 The <textstyle> element

Attributes

name (required) The symbolic name (to be used in the style attribute of <text> elements),

begin (required) LaTeX code to be placed before the text of the object when it is formatted,

end (required) LaTeX code to be placed after the text of the object when it is formatted.

type (optional) Either label or minipage (the default).

9.4.4 The <layout> element

It defines the layout of the frame on the paper and the paper size.

Attributes

paper (required) The size of the paper.

origin (required) The lower left corner of the frame in the paper coordinate system.

frame (required) The size of the frame.

skip (optional) The default paragraph skip between textboxes.

crop (optional) If the value of crop is yes, Ipe will create a CropBox attribute when saving to PDF.

9.4.5 The <titlestyle> element

It defines the appearance of the page title on the page.

Attributes

pos (required) The position of the title reference point in the frame coordinate system.

color (required) The color of the title.

size (required) The title font size (same as for <text> elements).

halign (optional) The horizontal alignment (same as for <text> elements).

valign (optional) The vertical alignment (same as for <text> elements).

9.4.6 The <pagenumberstyle> element

It defines the appearance of page numbers on the page. The contents of the element is a *template for a text object*.

Attributes

pos (required) The position of the page number on the page.

color (optional) The color of the page number. The default is black.

size (optional) The font size (same as for <text> elements). The default is normal.

halign (optional) The horizontal alignment (same as for <text> elements).

valign (optional) The vertical alignment (same as for <text> elements).

9.4.7 The <textpad> element

It defines padding around text objects for the computation of bounding boxes. The four required attributes are `left`, `right`, `top`, and `bottom`.

9.4.8 The <pathstyle> element

It defines the default setting for path objects.

Attributes

cap (optional) Same as for <path> elements.

join (optional) Same as for <path> elements.

fillrule (optional) Same as for <path> elements.

9.4.9 The <opacity> element

The `opacity` element defines a possible opacity value (also known as an alpha-value). All opacity values used in a document must be defined in the style sheet.

Attributes

name (required) A symbolic name, to be used in the `opacity` attribute of a `text` or `path` element.

value (required) An absolute value for the opacity, between 0.001 and 1.000. A value of 1.0 implies that the element is fully opaque.

9.4.10 The <gradient> element

The `gradient` element defines a gradient pattern.

Attributes of <gradient>

name (required) The symbolic name (to be used in the `gradient` attribute of <path> elements).

type (required) Possible values are `axial` and `radial`.

extend (optional) `yes` or `no` (the default). Indicates whether the gradient is extended beyond the boundaries.

coords (required) For axial shading: the coordinates of the endpoints of the axis (in the order `x1 y1 x2 y2`). For radial shading: the center and radius of both circles (in the order `cx1 cy1 r1 cx2 cy2 r2`).

matrix (optional) A transformation that transforms the gradient coordinate system into the coordinate system of the path object using the gradient. The default is the identity matrix.

The contents of the <gradient> element are <stop> elements defining the color stops of the gradient. There must be at least two stops. Stops must be defined in increasing offset order. It is not necessary that the first offset is 0.0 and the last one is 1.0.

Attributes of <stop>

offset (required) Offset of the color stop (a number between 0.0 and 1.0).

color (required) Color at this color stop (three numbers). Symbolic names are not allowed.

9.4.11 The <tiling> element

The `tiling` element defines a tiling pattern. Only very simple patterns that hatch the area with a line are supported.

Attributes

`name` (required) The symbolic name (to be used in the `tiling` attribute of <path> elements).

`angle` (required) Slope of the hatching line in degrees, between -90 and +90 degrees.

`width` (required) Width of the hatching line.

`step` (required) Distance from one hatching line to the next.

Here, `width` and `step` are measured in the y-direction if the absolute value of `angle` is less than 45 degrees, and in the x-direction otherwise.

9.4.12 The <effect> element

The `effect` element defines a graphic effect to be used during a PDF page transition. Acrobat Reader supports these effects, but not all PDF viewers do.

Attributes

`name` (required) The symbolic name (to be used in the `effect` attribute of <view> elements).

`duration` (optional) Value must be a real number, indicating the duration of display in seconds.

`transition` (optional) Value must be a real number, indicating the duration of the transition effect in seconds.

`effect` (optional) a number indicated the desired effect. The value must be an integer between 0 and 27 (see `ipe::Effect::TEffect` for the exact meaning).

9.4.13 Other style definition elements

The remaining style definition elements are:

<color> Defines a symbolic color. The value must be an absolute color, that is either a single gray value (between 0 and 1), or three components (red, green, blue) separated by space.

<dashstyle> Defines a symbolic dashstyle. The value must be a correct dashstyle description, e.g. `[3 5 2 5] 0`.

<pen> Defines a symbolic pen width. The value is a single real number.

<textsize> Defines a symbolic text size. The value is a piece of LaTeX source code selecting the desired font size.

<textstretch> Defines a symbolic text stretch factor. The symbolic name is shared with <textsize> elements. The value is a single real number.

<symbolsize> Defines a symbolic size for symbols. The value is a single real number, and indicates the scaling factor used for the symbol.

<arrowsize> Defines a symbolic size for arrows. The value is a single real number.

<gridsize> Defines a grid size. The symbolic name cannot actually be used by objects in the document — it is only used to fill the grid size selector in the user interface.

<anglesize> Defines an angular snap angle. The symbolic name cannot actually be used by objects in the document — it is only used to fill the angle selector in the user interface.

Common attributes

`name` (required) A symbolic name, which must start with a letter “a” to “z” or “A” to “Z”.

`value` (required) A legal absolute value for the type of attribute.

CHAPTER

TEN

USING IPE FIGURES IN LATEX

If—like many Latex users nowadays—you are a user of Pdflatex you can include Ipe figures in PDF format in your Latex documents directly.

The standard way of including PDF figures is using the `graphicx` package. If you are not familiar with it, here is a quick overview. In the preamble of your document, add the declaration:

```
\usepackage{graphicx}
```

One useful attribute to this declaration is `draft`, which stops LaTeX from actually including the figures—instead, a rectangle with the figure filename is shown:

```
\usepackage[draft]{graphicx}
```

To include the figure `figure1.pdf`, you use the command:

```
\includegraphics{figs/figure1}
```

Note that it is common *not* to specify the file extension `.pdf`. The command `\includegraphics` has various options to scale and rotate the figure. For instance, to scale the same figure to 50%, use:

```
\includegraphics[scale=0.5]{figs/figure1}
```

To scale such that the width of the figure becomes 5 cm:

```
\includegraphics[width=5cm]{figs/figure1}
```

Instead, one can specify the required height with `height`.

Here is an example that scales a figure to 200% and rotates it by 45 degrees counter-clockwise. Note that the scale argument should be given *before* the `angle` argument.

```
\includegraphics[scale=2,angle=45]{figs/figure1}
```

Let's stress once again that these commands are the standard commands for including PDF figures in a LaTeX document. Ipe files neither require nor support any special treatment. If you want to know more about the LaTeX packages for including graphics and producing colour, check the `grfguide.tex` document that is probably somewhere in your TeX installation.

10.1 Bounding boxes

There is a slight complication here: Each page of a PDF document can carry several *bounding boxes*, such as the *MediaBox* (which indicates the paper size), the *CropBox* (which indicates how the paper will by cut), or the *ArtBox* (which indicates the extent of the actual contents of the page). Ipe automatically saves, for each page, the paper size in the *MediaBox*, and a bounding box for the drawing in the *ArtBox*. Ipe also puts the bounding box in the *CropBox* unless this has been turned off by the stylesheet.

Now, when including a PDF figure, Pdflatex will (by default) first look at the CropBox, and, if that is not set, fall back on the MediaBox. It does not inspect the ArtBox, and so it is important that you use the correct stylesheet for the kind of figure you are making—with cropping for figures to be included, without cropping for presentations (as otherwise Acrobat Reader will not display full pages—Acrobat Reader actually crops each page to the CropBox).

If you have a recent version of Pdflatex (1.40 or higher), you can actually ask Pdflatex to inspect the ArtBox by saying `\pdfpagebox5` in your Latex file's preamble.

10.2 Classic LaTeX and EPS

If you are still using the "original" Latex, which compiles documents to DVI format, you need figures in Encapsulated Postscript (EPS) format (the "Encapsulated" means that there is only a single Postscript page and that it contains a bounding box of the figure). Some publishers may also require that you submit figures in EPS format, rather than in PDF.

Ipe allows you to export your figure in EPS format, either from the Ipe program (*File* menu, *Export as EPS*), or by using the command line tool *iperender* with the `-eps` option. Remember to keep a copy of your original Ipe figure! Ipe cannot read the exported EPS figure, you will not be able to edit them any further.

Including EPS figures works exactly like for PDF figures, using `\includegraphics`. In fact you can save all your figures in both EPS and PDF format, so that you can run both Latex and Pdflatex on your document—when including figures, Latex will look for the EPS variant, while Pdflatex will look for the PDF variant. (Here it comes in handy that you didn't specify the file extension in the `\includegraphics` command.)

It would be cumbersome to have to export to EPS every time you modify and save an Ipe figure in PDF format. What you should do instead is to write a shell script or batch file that calls *iperender* to export to EPS.

10.3 All figures in one Ipe document

On the other hand, if you *only* use Pdflatex, you might opt to exploit a feature of Pdflatex: You can keep all the figures for a document in a single, multi-page Ipe document, with one figure per page. You can then include the figures one by one into your document by using the `page` argument of `\includegraphics`.

For example, to include page 3 from the PDF file `figures.pdf` containing several figures, you could use

```
\includegraphics[page=3]{figures}
```

It's a bit annoying that one has to refer to the page by its page number. Ipe comes with a useful script that will allow you to use *symbolic names* instead.

CHAPTER

ELEVEN

THE COMMAND LINE PROGRAMS

11.1 Ipe

Ipe supports the following command line options:

-sheet *style_sheet_name* Adds the designated style sheet to any newly created documents.

-show-configuration With this option, Ipe will display the current configuration options on stdout, and terminate.

In addition, you can specify the name of an Ipe file to open on the command line.

11.2 Ipetoipe: converting Ipe file formats

The auxiliary program **ipetoipe** converts between the different Ipe file formats:

```
ipetoipe ( -xml | -pdf ) { <options> } infile [ outfile ]
```

The first argument determines the format of the output file. If no output filename is provided, Ipe will try to guess it by appending one of the extensions `ipe` or `pdf` to the input file's basename.

For example, the command line syntax

```
ipetoipe -pdf figure1.ipe
```

converts `figure1.ipe` to `figure1.pdf`.

Ipetoipe understands the following options:

-export No Ipe markup is included in the resulting output file. Ipe will not be able to open a file created that way, so make sure you keep your original!

-markedview (PDF only) Only the marked views of marked Ipe pages will be created in PDF format. If all views of a marked page are unmarked, the last view is exported. This is convenient to make handouts for slides.

-pages *from-to* (PDF only) Restrict exporting to PDF to this page range. This implies the `-export` option.

-view *page-view* Only export this single view from the document. This implies the `-export` option.

-runlatex Run Latex even for XML output. This has the effect of including the dimensions of each text object in the XML file.

-nozip Do not compress streams in PDF output.

11.3 Iperender: exporting to a bitmap, EPS, or SVG

The program **iperender** exports a page of the document to a bitmap in PNG format, to a figure in Encapsulated Postscript (EPS), or to scalable vector graphics in SVG format. (Of course the result contains no Ipe markup, so make sure you keep your original!) For instance, the following command line

```
iperender -png -page 3 -resolution 150 presentation.pdf pres3.png
```

converts page 3 of the Ipe document `presentation.pdf` to a bitmap, with resolution 150 pixels per inch.

Iperender understands the following options:

-page *page* The page to export. This can be a page number, or a page *name* (which you can set as the *Section name* in the page properties).

-view *view* The view to save. This can be a view number, or a view *name*, which you can set with *Views* → *Edit view*.

-resolution *resolution* The bitmap resolution in pixels per inch when exporting to png format. The default is 72ppi.

-tolerance *tolerance* The tolerance determines the precision used when rendering curves using straight line segments. The default is 0.1. Use a smaller value for higher precision.

-transparent Use transparent background when exporting to png format.

-nocrop Do not crop page to the page bounding box.

11.4 Ipescript: running Ipe scripts

Ipescript runs an Ipe script written in the Lua language with bindings for the Ipe objects, such as the script *update-master*. Ipescript automatically finds the script in Ipe's script directories. On Unix, you can place your own scripts in `~/.ipe/scripts`.

The Ipe distribution contains the following scripts:

- *update-master*, explained *earlier*;
- *page-labels*, explained *earlier*;
- *onepage* collapses pages of an Ipe document into layers of a single page;
- *add-style* to add a stylesheet to Ipe figures;
- *update-styles* to update the stylesheets in Ipe figures (in the same way that Ipe does it using the *Update stylesheets* function).

11.5 Ipeextract: extract XML stream from Ipe file

Ipeextract extracts the XML stream from an PDF or EPS file made by Ipe 6 or 7 and saves it in a file. It will work even if Ipe cannot actually parse the file, so you can use this tool to debug problems where Ipe fails to open your document.

```
ipeextract infile [ outfile ]
```

If not provided, the outfile is guessed by appending `xml` to the infile's basename.

11.6 Ipe6upgrade: convert Ipe 6 files to Ipe 7 file format

Ipe6upgrade takes as input a file created by any version of Ipe 6, and saves in the format of Ipe 7.0.0.

```
ipe6upgrade infile [ outfile ]
```

If not provided, the outfile is guessed by adding the extension `ipe` to the infile's basename.

To reuse an Ipe 6 document in EPS or PDF format, you first run **ipeextract**, which extracts the XML stream inside the document and saves it as an XML file. The Ipe 6 XML document can then be converted to Ipe 7 format using **ipe6upgrade**.

If your old figure is `figure.pdf`, then the command

```
ipeextract figure.pdf
```

will save the XML stream as `figure.xml`. Then run

```
ipe6upgrade figure.xml
```

which will save your document in Ipe 7 format as `figure.ipe`. All contents of the original document should have been preserved.

11.7 Importing other formats

11.7.1 Svgtoipe: Importing SVG figures

The auxiliary program **svgtoipe** converts an SVG figure to Ipe format. It cannot handle all SVG features (many SVG features are not supported by Ipe anyway), but it works for gradients.

svgtoipe is not part of the Ipe source distribution. You can download it separately.

11.7.2 Pdftoipe: Importing Postscript and PDF

You can convert arbitrary Postscript or PDF files into Ipe documents, making them editable. The auxiliary program **pdftoipe** converts (pages from) a PDF file into an Ipe XML-file. (If your source is Postscript, you have to first convert it to PDF using Acrobat Distiller or **ps2pdf**.) Once converted to XML, the file can be opened from Ipe as usual.

The conversion process handles many kinds of graphics in the PDF file fine, but doesn't do very well on text—Ipe's text model is just too different.

pdftoipe is not part of the Ipe source distribution. You can download and build it separately.

11.7.3 Ipe5toxml: convert Ipe 5 files to Ipe 6 file format

If you still have figures that were created with Ipe 5, you can use **ipe5toxml** to convert them to Ipe 6 format. You can then use **ipe6upgrade** to convert them to Ipe 7 format.

ipe5toxml is not part of the Ipe distribution, but available as a separate download.

11.7.4 Figtoipe: Importing FIG figures

The auxiliary program **figtoipe** converts a figure in FIG format into an Ipe XML-file. This is useful if you used to make figures with Xfig before discovering Ipe, of if your co-authors made figures for your article with Xfig (converting them will have the added benefit of forcing your co-authors to learn to use Ipe). Finally, there are quite a number of programs that can export to FIG format, and **figtoipe** effectively turns that into the possibility of exporting to Ipe.

However, **figtoipe** is not quite complete. The drawing models of FIG and Ipe are also somewhat different, which makes it impossible to properly render some FIG files in Ipe. In particular, Ipe does not support FIG's interpolating splines, depth ordering independent of grouping, pattern fill, and Postscript fonts.

You may therefore have to edit the file after conversion.

figtoipe is not part of the Ipe distribution. You can download and build it separately. **figtoipe** is now maintained by Alexander Bürger.

CHAPTER

TWELVE

FREQUENTLY ASKED QUESTIONS

12.1 Where can I get support for Ipe?

You are already looking at the FAQ - good!

Have you looked at the manual?

If you need help compiling or using Ipe, please post on the Ipe discussion list.

If you have found a bug in Ipe, please go to Ipe issues. Verify that the bug has not been reported before, and file a bug report. (If the bug has been reported before but is not fixed yet, you can add additional information to the existing bug report that would help solve the problem.)

If you want to say how much you love Ipe, you can send email to the author. But please note that he hardly ever replies directly to Email about Ipe (he has a day job).

12.2 MacOS says "the developer cannot be verified"

This is the normal behaviour of macOS when you install an app by a developer who is not a registered Apple developer. Apple shamelessly asks for $99 per year just so you can sign your application, and this is simply not realistic for an open-source application that is distributed for free.

Apple explains how to start an application anyway at https://support.apple.com/en-au/guide/mac-help/mh40616/mac

The other alternative is to compile Ipe yourself.

12.3 I want more/other choices for colors!

The colors available in the drop-down box are taken from your document's *stylesheets* (like any other *symbolic attribute*).

To have more colors available, you can either add a new stylesheet, or modify the basic stylesheet added to new documents by Ipe.

12.4 Attributes do not update

When you click on a line, the line attributes (thickness, arrow head etc.) don't update in the Properties - they remain as they were set.

In Microsoft software like Word the user interface automatically picks up the settings from the current spot in the document. I don't like this behavior, and Ipe won't adopt this. It would cause users to inadvertently reuse the properties of objects they had previously selected, instead of the standard settings they started with (and wanted).

Note that this is not a problem in, say, Microsoft Word: When you click somewhere to start typing text, the toolbar will revert to the settings active at this spot of the document. This 'reverting' wouldn't be possible in Ipe.

You can, however, copy object properties from an object to the user interface and from there to other objects with the operations *Edit → Pick properties* and *Edit → Apply properties*.

12.5 How do I use Ipe figures in beamer?

If you make an animated figure in Ipe (that is, an Ipe page with several views), you can use it in beamer like this:

```
\includegraphics<1>[page=1]{ipefig}
\includegraphics<2>[page=2]{ipefig}
```

You may also like Suresh's tip on this topic.

12.6 When Ipe refuses to open a PDF file you created with Ipe

This usually happens because you have been previewing the PDF document, and the previewer took the liberty to overwrite the original file.

Ipe PDF files contain Ipe markup hidden inside the PDF structure. When the file is modified by any other tool, this markup is lost, and Ipe will no longer be able to read it.

So please be careful with PDF viewers that have the capability to overwrite the file being viewed, in particular the Preview.app on recent version of MacOS.

Fortunately, the Preview.app has inbuilt versioning. So from within Preview.app, try *File → Revert To → Browse All Versions*, select the earliest available version, hit "Restore" and close the document. And voila, Ipe should be able to open the file again.

12.7 Online Latex-compilation does not work

If you get authentication errors for the online Latex compilation service, try using the HTTP protocol instead of HTTPS. You would need change the preference

```
prefs.latex_service_url = "http://latexonline.cc"
```

(So change https to http - see *customizing Ipe*.)

You will need to disable and again enable online compilation from the Help menu for the change to take effect.)

CHAPTER

THIRTEEN

HISTORY AND ACKNOWLEDGMENTS

The name "Ipe" is older than the program itself. When I made figures for my papers using Idraw in 1992, I was annoyed that I had to store two versions of each figure: one with Latex text, one with Postscript information. I came up with a file format that I called "Ipe," for "Integrated Picture Environment," and which was at the same time legal Latex source code and a legal Postscript file.

When I wrote the first version of Ipe at Utrecht University in the summer of 1993, it created this file format directly, and inherited the name. The first versions of Ipe (Ipe 2.0 up to 4.1) were based on my experiences with Idraw, XFig, and Jean-Pierre Merlet's JPDraw, used IRIS-GL and Mark Overmars' FORMS library, and run on SGI workstations only.

Due to popular demand, I spent two weeks in the summer of 1994 to teach myself Motif and to rewrite Ipe to run under the X window system. Unfortunately, two weeks were really not enough, and the 1994 X-version of Ipe was somewhat of a hack. I didn't have time to port the code that displayed bitmaps on the screen, it crashed on both monochrome and truecolor (24-bit) displays, and was in general quite unmaintainable.

These versions of Ipe were supported by the Netherlands" Organization for Scientific Research (NWO), and I would never have started working on it without Geert-Jan Giezeman's PLAGEO library. For testing, support, and inspiration in that original period, I'm grateful to Mark de Berg, Maarten Pennings, Jules Vleugels, Vincenzo Ferrucci, and Anil Rao. Many students of the department at Utrecht University served as alpha-testers (who apparently referred to Ipe as "the cute little core-dumper").

I gave a presentation about Ipe at the Dagstuhl Workshop on Computational Geometry in 1995, and made a poster presentation at the ACM Symposium on Computational Geometry in Vancouver in the same year. Both served to create a small but faithful community of Ipe addicts within the Computational Geometry community.

Ipe proved itself invaluable to me over the years, especially when we used it to make all the illustrations in our book *Computational Geometry: Algorithms and Applications* (Springer 1997, with Mark de Berg, Marc van Kreveld, and Mark Overmars). Nevertheless, the problems were undeniable: It was hard to compile Ipe on other C++ compilers and it only worked on 8-bit displays. It was only due to the efforts of Ipe fans such as Tycho Strijk, Robert-Paul Berretty, Alexander Wolff, and Sariel Har-Peled that the 1994 version of Ipe continued to be used until 2003.

I was teaching myself C++ while writing the first version of Ipe, and it showed—Ipe 5 was full of elementary object-oriented design mistakes. When teaching C++ to second-year students at Postech in 1996 I started to think about a clean rewrite of Ipe. My first notes on such a rewrite stem from evenings spent at a hotel in Machida, close to IBM Tokyo in July 1996 (the idea at that time was to embed Ipe into Emacs!). It proved impossible, though, to do a full rewrite next to teaching and research, and nothing really happened until the Dagstuhl Workshop on Computational Geometry in 2001, where Christian Knauer explained to me how to use Pdflatex to create presentations. I realized that PDF was ideally suited for a new version of Ipe.

Ipe 5 figures were at the same time Latex and Postscript files, and required special handling to be included into Latex documents, which sometimes required a bit of explaining when talking to co-authors or publishers. While editing a figure, Ipe 5 kept a Ghostscript window open that would show what the figure looked like after processing by Latex.

Several developments that had happened between 1993 and 2001 allowed me to use a completely new approach: First, Hàn Thê Thàn's Pdflatex takes Latex source and directly produces a PDF file with a PDF representation of the text and

all necessary fonts. Second, Derek Noonburg's Xpdf contained an open-source PDF parser that I could use to parse this PDF representation and to extract the processed text and fonts. Third, all relevant Latex fonts are now available as scalable Type1 fonts, and so it is possible to embed Latex text and formulas in figures that may still need to be scaled later. Finally, the Ghostscript window was no longer necessary as Ipe could use the beautiful Freetype library to directly display the text on-screen as it will appear on paper.

Directly after the Dagstuhl workshop I implemented a proof-of-concept: I defined the Ipe XML format (there was no question that Ipe 6 would have to be able to communicate in XML, of course), wrote **`ipe5toxml`** (reusing my old Ipe parsing code) and a program that runs Pdflatex, parses its PDF output, extracts text objects and font data, and creates a PDF file for the whole Ipe figure.

All that remained to be done was to rewrite the user interface. Mark de Berg and the TU Eindhoven made it possible for me to take some time off from teaching and research. The final design changes were made during the Second McGill-INRIA Workshop on Computational Geometry in Computer Graphics at McGill's Bellairs Research Institute in February 2003, and much inspiration was due to the atmosphere at the workshop and the magnificient cooking by Gwen, Bellair's chef. An early preview of Ipe 6.0 was "formally" released at the Dagstuhl Workshop on Computational Geometry in March 2003, to celebrate the Dagstuhl influence on Ipe.

Other than the file format, there weren't really that many changes to Ipe's functionality between Ipe 5 and Ipe 6. René van Oostrum insisted that no self-respecting drawing program can do without style sheets and layers. Views allow you to incrementally build up a page in a PDF presentation.

I also revised the interface to ipelets (which used to be called "Iums" in the good old days when people still thought that "applets" were small apples)—it is now based on dynamically loaded libraries (a technology that was still somewhat poorly understood in the early nineties, at least by me).

And, of course, there was a Windows version of Ipe 6. Who would have thought that ten years earlier!

There were many releases of Ipe 6.0, all of them called "previews," because I never considered that I had reached a stable state. A number of experimental features were tried and either built into Ipe or discarded. Ipe 6 migrated from Qt 2 and Qt 3 to Qt 4, a somewhat painful process due to a number of annyoing Qt bugs that cost me a lot of time.

When in 2007 I discovered the fantastic Cairo library for rendering, I immediately decided to switch Ipe to use this: a small dedicated library with a nice API to do the rendering, instead of the buggy monster that was Qt. The Cairo API fit Ipe so well that I could write a Cairo painter for Ipe in an hour or so. Cairo supports Freetype directly, instead of Ipe having to render each glyph into a bitmap that is then blit onto the canvas.

I made the huge mistake of announcing on the Ipe discussion list that Ipe 6.0 preview 28 was the last version of Ipe 6, and that there would soon be a new version, Ipe 7. I should have known that this was impossible during a time where I advised several graduate students, taught several new courses, and went through the tenure process. I had to release several bugfix releases of Ipe 6 while really wanting to work on Ipe 7.

However, the delay left me with enough time to carefully think about another change I wanted to make: It would be nice if Ipe embedded a scripting language that could be used to write simple ipelets without compilation. I looked at Scheme/Guile, Python, and Lua, and finally decided for Lua: a small, elegant, stable language with a tiny footprint, easily embedded with a very nice C interface.

In 2009, I had my first sabbatical ever, which I spent in the group of Ulrik Brandes at the University of Konstanz. Here I finally had the time to work on Ipe 7, and I'm very grateful to Ulrik and all members of his group for the wonderful time I had in Konstanz. Next to the two big changes mentioned above, Ipe 7 introduced tiling patterns, gradients, clipping paths, transparency, user-definable arrows and marks, and SVG output.

I wanted to avoid Qt in Ipe 7 as it had caused me quite a bit of pain during the life of Ipe 6, but it was hard to find a good replacement that would allow Ipe to run on Linux, Windows, and Macs. During the Korean Workshop on Computational Geometry organized by Tetsuo Asano at Hakusan seminar house in June 2009, I discussed using Ipe on tablet PCs with Vida Dujmovic, Jit Bose, and Stefan Langerman. It is their fault that Ipe 7 comes with a tablet input tool, and finally Stefan and Sébastien Collette convinced me that there isn't really an alternative to Qt that has the same support for tablets and Macs. So Ipe 7 is still using Qt, but in a much more restricted way than before—it turned out that this part of Qt is quite stable and not prone to new bugs.

Even though I was only using a small part of Qt to create the UI, I still had to bundle the huge Qt library, ten times the size of Ipe itself, with Ipe's binary builds for Windows and OSX. This bothered me, since of course Windows and OSX already provide all the functionality for building UIs natively. After some experimentation with building simple UIs in Windows and GTK, I settled on a new architecture where the UI library would only be used in the Ipe program itself, in the Ipe canvas, and in the small Lua library ipeui that can be used to build dialogs from Lua (and which is entirely independent of Ipe). The native Windows UI was finished while I was visiting Emo Welzl in Zurich in early 2015, on a small Windows-tablet, and was first released in Ipe 7.1.7. During my sabbatical at Bayreuth University in 2016 Christian Knauer and his group provided me with a Mac in my office, providing both the opportunity and the time to learn Objective C and Cocoa. Ipe 7.2.1 was released in Bayreuth with the first native OSX UI.

CHAPTER

FOURTEEN

COPYRIGHT

Ipe is "free," this means that everyone is free to use it and free to redistribute it on certain conditions. Ipe is not in the public domain; it is copyrighted and there are restrictions on its distribution as follows:

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

As a special exception, you have permission to link Ipe with the CGAL library and distribute executables, as long as you follow the requirements of the Gnu General Public License in regard to all of the software in the executable aside from CGAL.

This program is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU General Public License for more details.

A copy of the GNU General Public License is available on the World Wide web at http://www.gnu.org/copyleft/gpl.html. You can also obtain it by writing to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

INDEX

O

P

R

S

T

U

V

W

Z