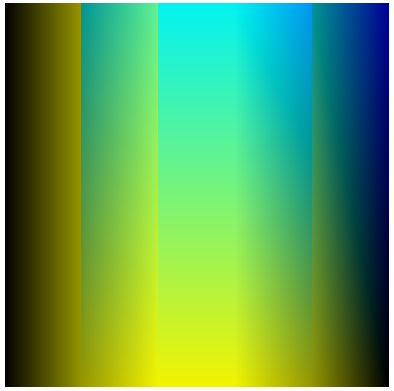
# Table of Contents

Acumen Training

# Converting Blends to Smooth Shading



Blends (also called "gradients" or "fountains") are a ubiquitous feature in graphic design. The ability to specify a blend that smoothly transitions from one color to another is an important capability of all design packages.

It is difficult to do this well. Companies doing graphics software put a lot of work into figuring out how to display and print a gradient that will look good on screen, print well, and yet not be too large. Most of the methods they settled on represent compromises of one sort or other: precalculate an image (at the printer's resolution); draw a series of very closely-spaced lines or ellipses.

Pretty much all of these methods are slow to print and add considerably to file size.

## Smooth Shading

Acrobat 4 incorporated a new PDF feature called "Smooth Shading," language-level support for gradients in PostScript and Acrobat. Smooth shading is very cool; it's compact, fast, offers precise control over the color variation, and looks incredibly good.

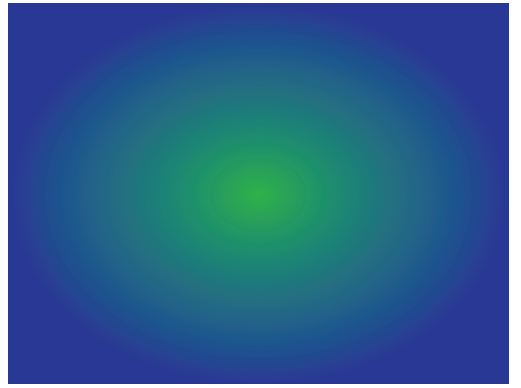
Distiller 4 and 5 have built into them a rather remarkable ability: they can detect blends created by QuarkXpress, Illustrator, Corel Draw, and Freehand and convert those blends into PDF Smooth Shading. This can dramatically reduce file size and print time and can improve print quality under many conditions.

This month, we're going to see how to turn this auto-smooth-shading feature on.

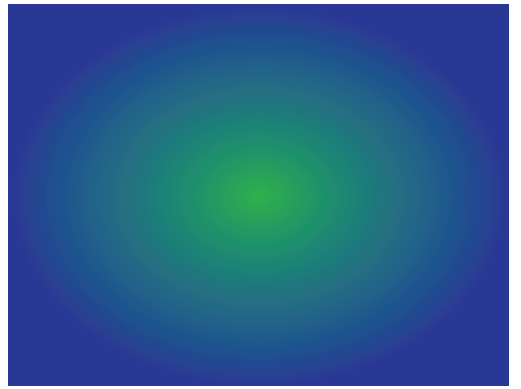
Next Page ->

**For Example**

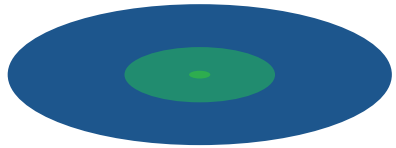
Here we have two versions of the same radial blend. The top blend was created in *QuarkXpress* and converted to PDF without blend conversion; that is, we're seeing the blend created using Quark's own method.



The lower blend was converted by Distiller to a smooth shade.



These look quite alike. However, the Quark blend consists of many concentric ellipses of varying color. These ellipses change size in small enough increments that they make up a smoothly varying patch of color. If you watch closely, you can see the ellipses being drawn when you come to this page of the *Journal.*



The lower blend is a smooth shade; it looks very much the same as the upper, but is one monolithic piece. It takes up much less than half the space of the original, will print faster, and will continue to look good even if you scale it up many hundreds of percent.

Note that this gradient takes no significant time to display when you turn to this page of the *Journal.*

Next Page ->

Acumen Journal: Acrobat User

**Printing Smooth Shadings**

Acrobat prints smooth shades in one of two ways, depending on what kind of printer you are using:

*PostScript LanguageLevel 3*

If you have a Level 3 PostScript device, Acrobat will print the gradient as a PostScript smooth shade. PDF and PostScript LanguageLevel 3 share this feature, so a converted gradient migrates from PDF to PostScript very easily. The gradient will print startlingly quickly.

*Other Printers*

If you have an earlier PostScript printer or a non-PostScript device, Acrobat will convert the gradient to a printer-resolution image. This will look very good, but will print more slowly than the LanguageLevel 3 version. Print speed should be at least as good, however, as the original, unconverted blend would have achieved.

Next Page ->

## Turning on Conversion

So how do we tell Distiller that we want it to convert blends to smooth shading? That depends upon whether you have Distiller 4 or Distiller 5.

### Distiller 5

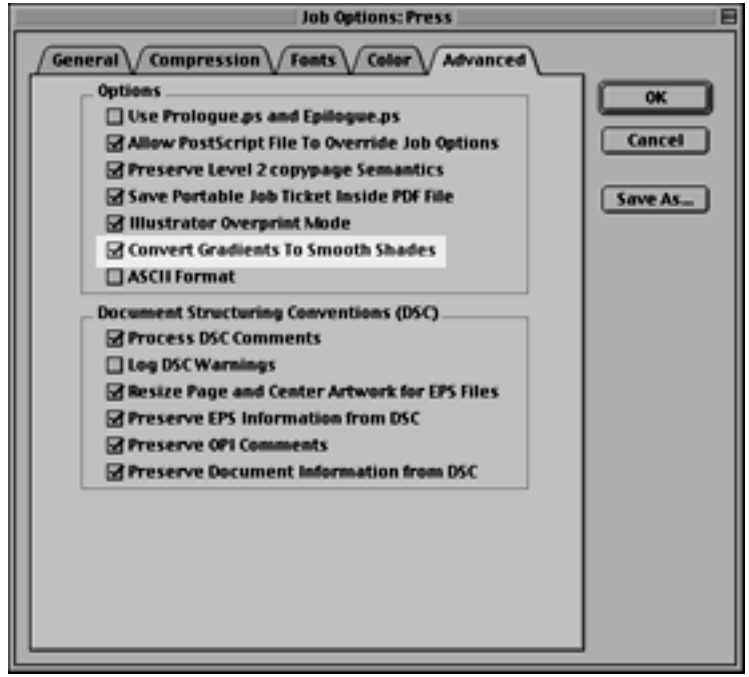
Distiller in Acrobat 5 makes it easy: there is a check box among the job option controls that toggles this feature.

Simple go to *Settings>Job Options* in Distiller and click on the Advanced tab.

Among the controls you will see one labeled *Convert Gradients to Smooth Shades.* Turn this on.

That's all there is to it. All blends in PostScript files from Quark, Illustrator, Corel Draw, Freehand, or PowerPoint will be converted to Smooth Shades.

Next Page ->



**Distiller 4**

Oddly enough, although Acrobat 4's Distiller has the blend conversion feature, there is no control in the user interface that lets you turn it on. You need to go in by the back door, using a bit of PostScript programming and the little-used *prologue.ps* and *epilogue.ps* mechanism.

*prologue.ps/epilogue.ps*

All versions of Acrobat Distiller can look in the *Data* folder in the Distiller folder for a pair of PostScript files named *prologue.ps* and *epilogue.ps*. If it finds them, Distiller will execute the contents of *prologue.ps* before each file it distills and *epilogue.ps* after each file. This allows a PostScript programmer to change the behavior of Distiller, printing watermarks, etc.

In our case, we are going to create a *prologue.ps* file that turns on the Convert-to-Smooth-Shades feature.

There will be three steps to this process:

1. Create the *prologue.ps* file.
2. Place it in the Data folder for Distiller to find.
3. Tell Distiller to look for *prologue.ps/epilogue.ps* when distilling PostScript files.

Let's do it:

Next Page ->

*1. Create prologue.ps*

First let's create our *prologue.ps* file.

1. Launch your favorite text editor and create a new file.
2. Type in the following PostScript code:

```
<< /IdiomRecognition true >> setuserparams
```

   Note that spacing is not significant here; you can have any number of spaces or tabs between the words. Case is significant, however; upper and lower case must be exactly as written.

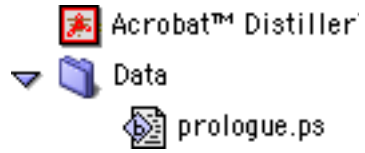
3. Save the file with the name *prologue.ps.*

   If you used a word processor (Microsoft Word, etc.), rather than a text editor to create this file, make sure you save the file as plain text.

(If you wish, you can get this file from the Acumen Training website. Go to www.acumentraining.com/resources.html and look among the Acrobat sample files.)

*2. Place the File in Data*

For Distiller to find this file, you must place it in the *Data* folder, located in the same folder as Distiller. Just drag it to the folder.

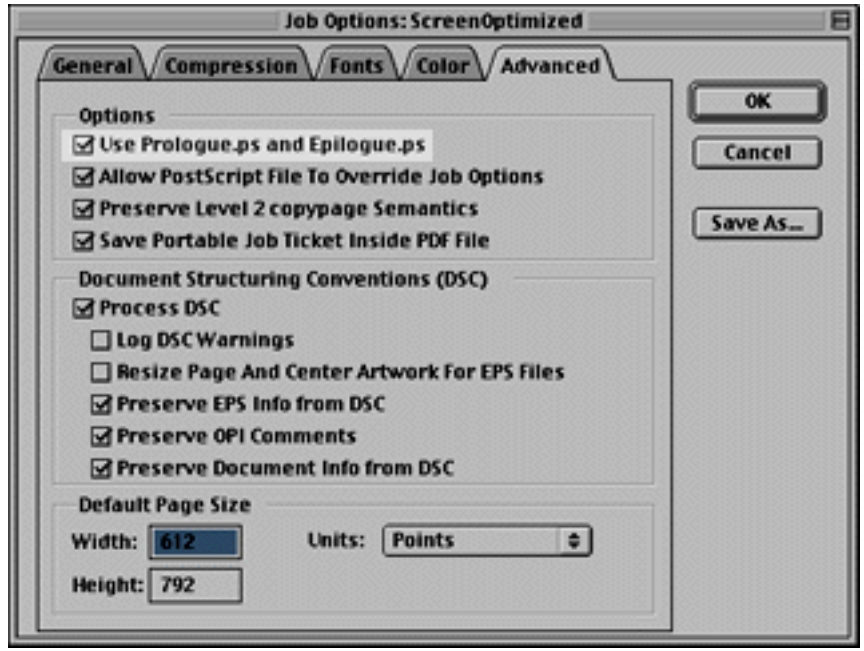



By the way, the Acrobat 4 documentation says the *prologue.ps* file should be in the same folder as Distiller. This doesn't seem to work; put it in the *Data* folder.

Next Page ->

*3. Tell Distiller* Lastly, we need to tell Distiller to look for the *prologue.ps* file.

We do this by going to *Settings>Job Options* and clicking on the Advanced tab. The topmost control is a check box labeled *Use Prologue.ps and Epilogue.ps.*




Turn this on.

From now on, Distiller will convert any blends that it knows about (QuarkXpress, Illustrator, CorelDraw, Freehand) to Smooth Shades.

## Summary

There is really no disadvantage to turning on blend conversion. With no loss of quality (indeed, often an improvement), blends will be smaller, faster, and more dependable.

Give it a try.

Return to Main Menu

# *BeginPage* and *EndPage*

Here's a common need I often see posted on the PostScript newsgroup: download a PostScript file and have something happen on each page: print a watermark, offset each page for binding, etc.

A commonly-proposed solution to these tasks is to redefine *showpage* to carry out the desired activity:

```
/showpage { ... do some stuff ... showpage } bind def
```

There is a less troublesome approach, however. PostScript LanguageLevel 2 introduced a very good mechanism to do just such page-level tasks. Using the *setpagedevice* operator, you can define two procedures, *BeginPage* and *EndPage.*

Not surprisingly, PostScript will automatically execute *BeginPage* at the beginning of each page and *EndPage* at the end of each page.

In this month's *PostScript Tech,* we'll discuss each of these procedures and then use them to implement two-up printing, that is, printing two pages from our PostScript file on each piece of paper. This example can be easily generalized to print any number of document pages on each sheet.

Next Page ->

## *BeginPage & EndPage*

You specify the *BeginPage* and *EndPage* procedures in a call to *setpagedevice:*

```
<<
     /BeginPage { ... }
     /EndPage { ... }
>> setpagedevice
```

The *BeginPage* and *EndPage* procedures will now be executed at the start and end of every page.

## *BeginPage* Arguments

*BeginPage* will be called with one argument on the stack:

```
pageNum   BeginPage   =>   ---
```

*pageNum* is the number of times that the *showpage* operator has been executed so far. This will be one less that the current page number. (If you prefer, you can think of it as a zero-based page count.)

Your *BeginPage* should not have any return values.

Next Page ->

**When is *BeginPage* Called?**

*BeginPage* will be called at the beginning of each page in the PostScript file. But what, exactly, does that mean? *BeginPage* will be called at the following times:

*showpage & copypage*

The *showpage* and *copypage* operators call *BeginPage* as their last activity.

*setpagedevice*

The *setpagedevice* operator also calls *BeginPage* as its last action. This includes the *setpagedevice* that established the current *BeginPage* procedure.

*device reactivation*

Any operator that reactivates an earlier *pagedevice* will also execute *BeginPage.*

*setpagedevice* installs a new PostScript pagedevice into the graphics state. Loosely defined, a pagedevice is a collection of data that describes the current imaging target. This includes such things as the current page dimensions, resolution, media choice, etc.

The current pagedevice is part of the PostScript graphics state. Therefore, anything that restores an earlier graphics state may also restore an earlier page device. If a *gsave-grestore* or *save-restore* pair have a call to *setpagedevice* between them:

```
gsave  << ... >> setpagedevice  ...  grestore
```

the *grestore/restore* will change the pagedevice back to its previous state. In this case, the *grestore/restore* will execute *BeginPage.*

Next Page ->

### *EndPage* Arguments

The *EndPage* procedure must take two numbers from the stack and yield a boolean return value:

```
pageNum  eventCode  EndPage  =>  bool
```

The *pageNum* means the same thing here as in *BeginPage:* it is the number of times *showpage* has been executed.

The eventCode indicates why *EndPage* is being called. It can have one of three values:

| | |
|---|---|
| *0* | *EndPage* is being called by *showpage.* |
| *1* | *EndPage* is being called by *copypage.* |
| *2* | *EndPage* is being called by a page device deactivation (such as a call to *grestore* or *restore* that changes the page device). |

The boolean return value indicates whether the current page should be printed. If *true,* PostScript will print the current page; if *false,* PostScript will suppress the printing of the current page and further marks will be added to the those already painted on the page.

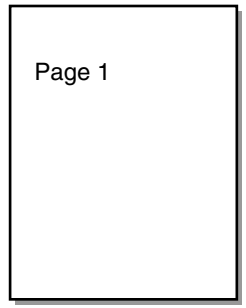
Next Page ->

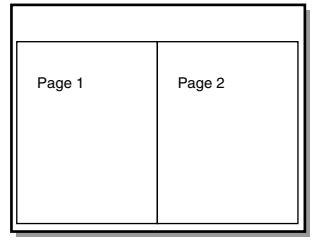
Acumen Journal: PostScript Tech

## An Example: 2-Up Printing

As an example, let's look at a definition of *BeginPage* and *EndPage* that, when placed at the beginning of a PostScript file, cause the file to print two-up, that is, two pages on each piece of paper.

This example presumes an 8½ x 11-inch page (American letter). It can be easily generalized to other page sizes or to 4-up.




### *BeginPage* Overview

To print 2-up, our *BeginPage* procedure needs to do the following:

1. Change the page to landscape orientation
2. Scale by a factor that will map a letter-size portrait page into half of a landscape page (minus a quarter-inch margin).
3. If we are printing an even-numbered page, translate to the right by the (scaled) width of the page.
4. Draw a border around the page.
5. Clip to the scaled page (so if the PostScript bleeds off the page, we don't draw off the "virtual page").

Next Page ->

### *EndPage* Overview

Our *EndPage* procedure needs to decide whether to eject paper at the end of each page. It will return a boolean that will be *true* if it's time to print; false, if page printing should be suppressed.

We'll do the following:

1. Examine the event code. If it's anything other than *0* (*showpage* execution), then return return *false.*

2. If we *are* doing a *showpage,* examine the *showpage* count. If we're printing an even-numbered page, return *true;* otherwise, return *false.*

Next Page ->

**The Code** The following can be inserted at the beginning of any PostScript code to print 2-up.

```
<<
     /BeginPage      % stack: pageNum => ---
     {
          612 0 translate 90 rotate      % Landscape orientation
          18 18 translate                % Indent by an 18-pt margin
          .618 dup scale                 % Scale to fit
          2 mod 1 eq                     % Are we on an even page?
          { 612 0 translate } if         % If so, translate to the right
          [ 0 0 612 792 ] dup
          rectstroke rectclip            % Stroke and clip to page boundary
     } bind

     /EndPage        % stack: pageNum eventCode => prtPageBool
     {
          0 ne            % Is this something other than a showpage?
          { false }       % Yes (it's not a showpage): Don't print page
          { 2 mod 1 eq } % No (it's a showpage): Print page if even page
          ifelse
     } bind
>> setpagedevice
```

Next Page ->

*BeginPage Details*

```
612 0 translate 90 rotate      % Landscape orientation
18 18 translate                % Indent by an 18-pt margin
.618 dup scale                 % Scale to fit
```

This is straightforward: change the the page to landscape, indent by an 18-point margin, and then scale by .618 (which maps the 612-pt wide page into half of a 792-point space, minus the 18 point margin).

```
2 mod 1 eq                     % Are we on an even page?
{ 612 0 translate } if         % If so, translate to the right
```

The only really counter-intuitive part in *EndPage* is how we determine whether we're on an even-numbered page. The argument handed to *BeginPage* is the number of times *showpage* has been executed; this is one less than the current page number. Thus, if this number is odd, then we're on an even page.

The test we perform is to execute *2 mod* on the showpage count and see if the result is *1.* If so, we're on an even page and will want to translate to the right by the page width (612, in our case). Note that the page width will be scaled by .618.

```
[ 0 0 612 792 ] dup
rectstroke rectclip            % Stroke and clip to page boundary
```

Finally, we construct a rectangle containing the page's lower left corner (0,0) and width and height (612, 792) and hand it to *rectstroke* and *rectclip.*

Next Page ->

*EndPage Details*

```
0 ne
{ false }
```

*EndPage* first examines the event code to see if it is zero, indicating a *showpage* execution. If not, we return *false,* since we don't want to eject a page unless we're at a *showpage.*

```
{ 2 mod 1 eq }      % No (it's a showpage): Print page if even page
```

If we are doing a *showpage,* we want to return a *true* if we're currently printing an even-numbered page, that is, if the *showpage* count is odd. The easiest way to do this is to perform our *2 mod 1 eq* and leave the result on the stack as the *EndPage* return value.

Next Page ->

**A Minor Problem**

This *BeginPage-EndPage* pair work very nicely, but for one thing: if the PostScript document has an odd number of pages, you will never get the final page printed out. (We don't eject paper for odd numbered pages.)

The simple work-around is to add an extra *showpage* at the end of the PostScript file. If the original PostScript file produced an even number of pages, this new *showpage* will have no visible effect; it will add an extra, odd numbered extra page that will not print. On the other hand, if the PostScript file normally would print an odd number of pages, this extra *showpage* will boost the page count to an even number, provoking a final page ejection.

Thus, to a print PostScript file 2-up, you must do two things:

1. Paste the call to *setpagedevice* that defines *BeginPage* and *EndPage* at the beginning of the PostScript file.

2. Put an extra *showpage* at the very end of the PostScript file.

It works very well and is much more effective than some of the complicated *showpage* redefinitions I've seen.

**Generalization**

It should be relatively easy to modify this example to do 4-up or other *n*-up printing. You may also want to support other page sizes.

These are left as an exercise for the reader. Ya'll have fun.

Return to Main Menu

# Schedule of Classes, June 2001 - August 2001

Following are the dates and locations of Acumen Training's PostScript and Acrobat classes. Clicking on a class name below will take you to the description of that class on the Acumen training website.

The PostScript classes are taught in Orange County, California, near the Orange County airport, and in London at Adobe Systems' offices near Heathrow.

## PostScript Classes

| | | | | |
|---|---|---|---|---|
| **PostScript Foundations** | Orange Co., CA | August 6 - 10 | Orange Co., CA | October 15 - 19 |
| **Advanced PostScript** | Orange Co., CA | July 16 - 20 | Orange Co., CA | October 22 - 26 |
| **PostScript for Support Engineers** | Orange Co., CA | July 23 -27 | Orange Co., CA | October 1 - 5 |
| **Jaws Development** | Orange Co., CA | July 30 - August 2 | | |

For more classes, go to www.acumentraining.com/schedule.html

**PostScript Course Fees**

PostScript classes cost $1,750 per student until September and $2,000 thereafter. These classes may also be taught on your organization's site.

Registration →

Acrobat Classes →

# Acrobat Class Schedule

Acumen training teaches three users' classes in Adobe Acrobat (the links below will take you to the Acumen website's complete description). These are all taught with Acrobat 5, although Acrobat 4 versions may be taught if this is what your site uses.

**Acrobat Essentials** — This class teaches the student how to make perfect PDF files. It includes complete coverage of the meaning and proper settings of all of the Distiller Job Options.

**Interactive Acrobat** — Here we show you how to add bookmarks, links, buttons, sounds, movies, form fields, and other interactive features to an Acrobat file.

**Troubleshooting with Enfocus' PitStop** — This class shows the student how to use all of the capabilities of this popular editing and preflight software.

**On-site Only** — The Acrobat classes are taught only on corporate sites. If you have an interest in any of these classes for your group, please see the Acumen Training website regarding arranging an on-site class.

Back to PostScript Classes

Return to First Page

# Contacting John Deubert at Acumen Training

**For more information**

For class descriptions, on-site arrangements or any other information about Acumen's classes:

**Web site:** http://www.acumentraining.com **email:** john@acumentraining.com

**telephone:** 949-248-1241

**mail:** 25142 Danalaurel, Dana Point, CA 92629

**Registering for Classes**

To register for an Acumen Training class, contact us any of the following ways:

**Register On-line:** http://www.acumentraining.com/registration.html

**email:** registration@acumentraining.com

**telephone:** 949-248-1241

**mail:** 25142 Danalaurel, Dana Point, CA 92629

**Back issues**

Back issues of the Acumen Journal are available at the Acumen Training website: www.acumenjournal.com/AcumenJournal.html

Return to First Page

# What's New at Acumen Training?

## New Class at Seybold

Acumen Training will be teaching a new class at the September Seybold conference in San Francisco.

*Creating Acrobat Forms* will be a beginning-to-intermediate level class on creating forms in Adobe Acrobat. This class will show the student how to add form fields to a PDF file. If you need to create Acrobat-based forms with pop-up menus, check boxes, number fields that automatically calculate themselves, etc., this class will show you how.

Along the way, we'll see how to embed movies, sounds, links to web pages, and a host of other useful items in your Acrobat files.

Acumen Training also teaches a slightly broader version of this class on corporate sites as "Interactive Acrobat."

For information about *Creating Acrobat Forms,* see the Seybold Seminars website.

For information about *Interactive Acrobat,* see the Acumen Training website.

Return to First Page

# Journal Feedback

If you have any comments regarding the *Acumen Journal,* please let me know. In particular, we are looking for three types of information:

**Comments on usefulness.** Does the Journal provide you with worthwhile information? Was it well written and understandable? Did you like it, hate it, or did it make you want to eat brussels sprouts? How could we make it better? Do you like the PDF format?

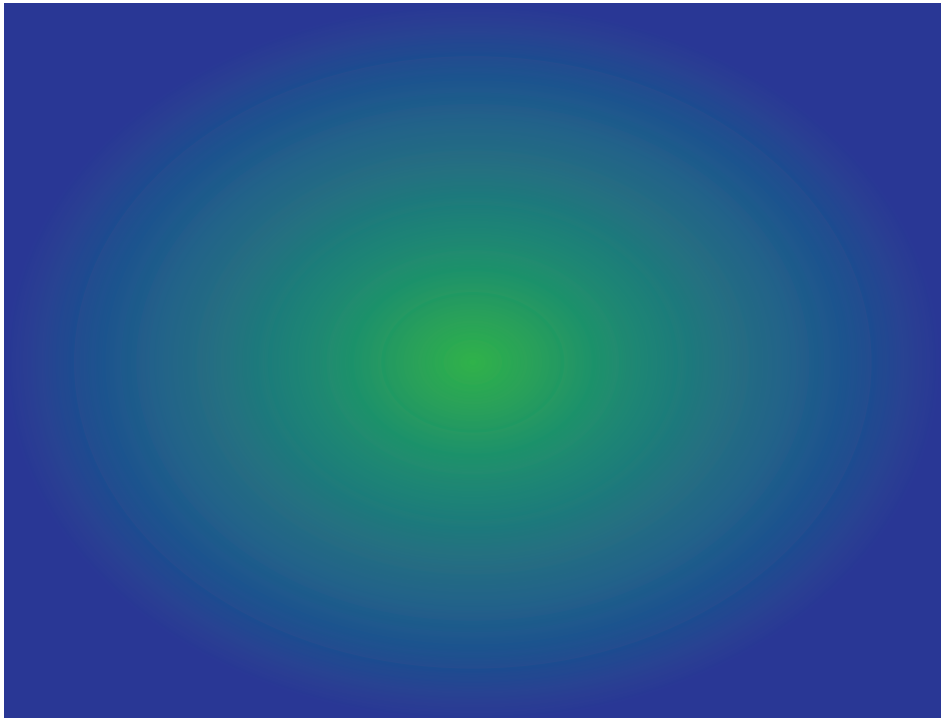
**Suggestions for articles.** Each Journal issue contains one article each on PostScript and Acrobat. What topics would you like us to address?

**Questions and Answers.** We are planning a Q&A section for future issues. Do you have any questions about Acrobat, PDF or PostScript?

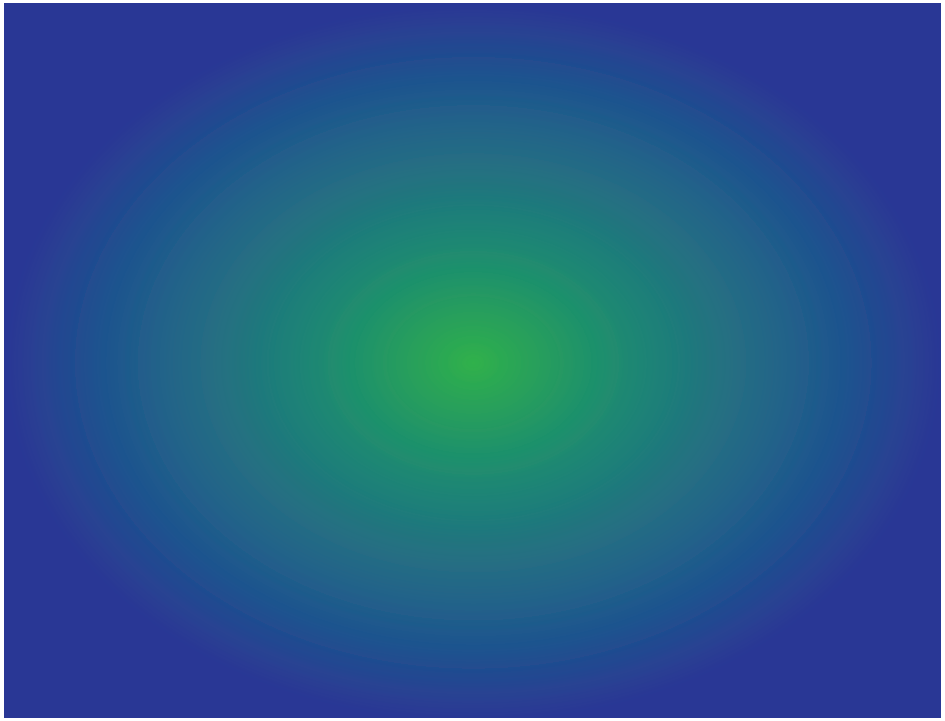
Please send any comments, questions, or problems to:

journal@acumentraining.com

Return to Menu



*QuarkXpress blend, no conversion*



*QuarkXpress blend, converted*