

What Is a Pixel?

James F. Blinn

Microsoft
Research

I am not a major sports fan. But there is one story from the folklore of football that has always intrigued me. The story goes that legendary football coach Vince Lombardi was observing his new players, recruited from the best college teams and all presumably excellent players. He was, however, not pleased with their performance. So he called them all to a meeting, which he began by holding up the essential tool of their trade and saying, "This is a football." I was sufficiently impressed by this back-to-basics attitude that, when I taught computer graphics rendering classes, I used to start the first lecture of the term by going to the blackboard (boards were black then, not white) and drawing a little dot and saying, "This is a pixel."

But was I right? Most computer graphicists would agree that the pixel is the fundamental atomic element of imaging. But what is a pixel really? As I have played with various aspects of pixel mashing, it has occurred to me that the concept of the pixel is really multifaceted (to use a weird metaphor.) Perhaps a better metaphor comes from the old story of the blind men describing an elephant and basing their description on what part they were touching: "An elephant is a ... tree (leg), a wall (side), a rope (tail), a snake (trunk), a spear (tusk)." In

that spirit I am going to list some possible meanings for a pixel that I will expand on in some later columns.

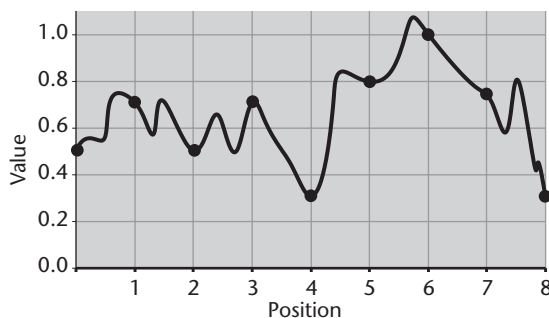
A pixel is a little square

Early 2D windowing systems considered a pixel a little square. This is perhaps the simplest possible definition, but it only works if you are mostly drawing horizontal and vertical lines and rectangles that are integral numbers of pixels in size. Anything at fractional pixel size or at an angle yields jaggies and other forms of aliasing.

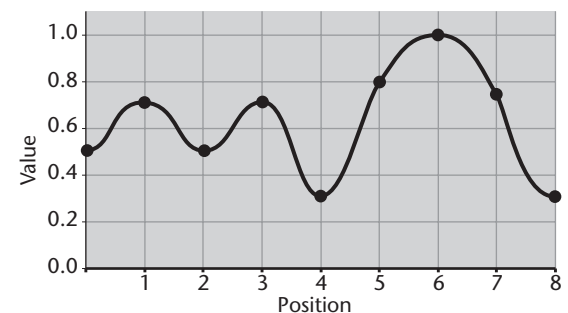
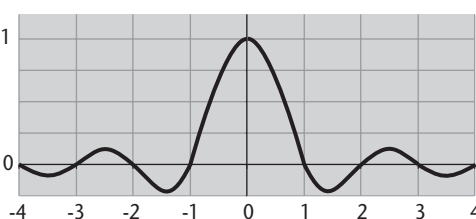
A pixel is a point sample of a continuous function

A more enlightened signal processing approach thinks of a pixel as a point sample of a continuous function (see the dots in Figure 1). (This approach was actually taken by the rendering community long before pixel displays were used for user interfaces and windowing systems.) Applying linear filtering theory to pixel manipulation gives, among other things, an approach to antialiasing. You start out with the ideal continuous function you wish to display. Then you filter out the frequencies that are too high to be represented by the given pixel spacing. A theoretically ideal antialiasing filter would be a low-pass filter with a cutoff at one-half cycle per pixel spacing. One way to accomplish this is to convolve the image function with the function sinc/x (see Figure 2) smoothing out the jaggies and giving Figure 3. Then you sample the result. This theoretical foundation is the basis for various algorithms for zooming in and out of images, warping images, and so forth.

1 Point sample of a continuous function.



2 The ideal antialiasing filter.



3 Point sampling after filtering.

A pixel is a fuzzy blob

Now you have a collection of discrete spatial samples (see Figure 4), which are actually a representation of the continuous image function. To display the continuous function you need to interpolate between the samples. Fortunately, the display hardware does this for us. A properly adjusted CRT will display pixels as little fuzzy blobs that are about as wide as the spacing between pixels. This will create a smooth function out of our discontinuous samples. Unfortunately, this won't give exactly the function of Figure 3. The theoretically ideal interpolation to get back Figure 3 would consist of convolving the samples of Figure 4 with the sinc/x function of Figure 2, which is called, in this situation, a *reconstruction filter*. The fuzzy blob provided by the hardware is only an approximation to the ideal of Figure 2. It's not perfect, but it's not too terrible.

A pixel is three tiny colored rectangles

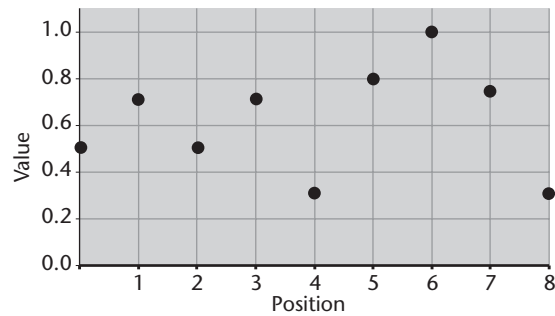
Liquid crystal displays (LCDs) and micro-mirror displays (digital light processors) generate pixels rather differently than CRTs. In this case, a (physical) pixel really is a little square. This is an even worse approximation to the ideal reconstruction filter of Figure 2 than the Gaussian bump that a CRT gives.

In the case of color displays, an LCD pixel actually consists of three little solid colored rectangles (see Figure 5). Typically a pixel consists of a tall, red rectangle on the left; a green rectangle in the middle; and a blue rectangle on the right. This means that the three primary color images are being displayed offset by one-third of a pixel spacing between the three colors. This has implications on the proper spatial sampling and filtering of an image. Briefly, the three colors of the image should be sampled at points with a corresponding offset (see Figure 6). But that is only part of the story. There are also some subtle effects of this process on the proper spatial filtering of the image. This becomes important when the image has a lot of high-frequency content, as with text. This is one of the (many) components of Microsoft's ClearType algorithm and will be the subject of a future column.

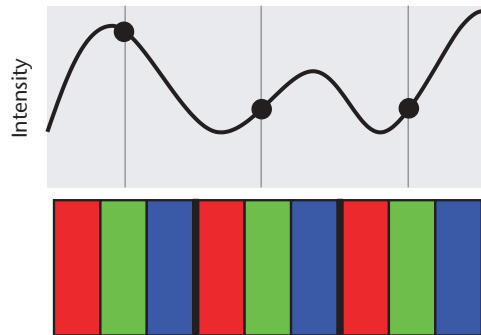
A pixel is a tradeoff between blurring and aliasing

The whole subject of sampling and reconstructing an image is a study in the tradeoffs between antialiasing and crispness of the resulting image. Classical linear sampling theory gives us much insight into this tradeoff and can help us design algorithms that perform better than seat-of-the-pants intuition. This approach, while theoretically sound, has many practical limitations:

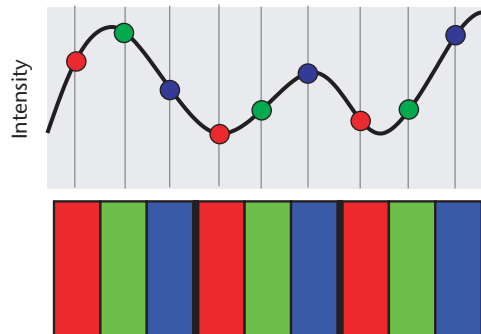
- *Finite kernel.* A perfect low-pass filter is not achievable with a finite (or even a reasonably small) piece of the sinc/x function, the so-called filter kernel. The closer you want to come to a perfect low-pass filter, the wider the kernel needs to be and the more arithmetic you need to do.
- *Negative light.* There is no such thing as negative light. A perfect filter has negative lobes that admit the possibility of negative output values.



4 Pixel samples before reconstruction.



5 Pixels from an LCD showing conventional spatial sampling.

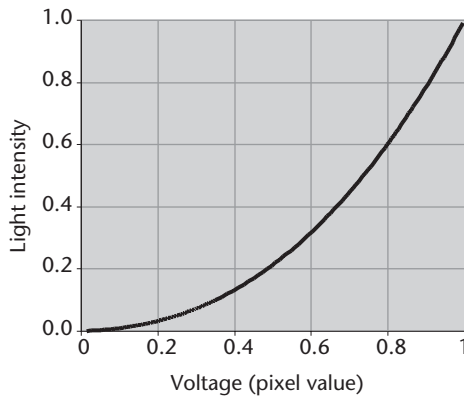


6 Pixels from an LCD showing correct spatial sampling.

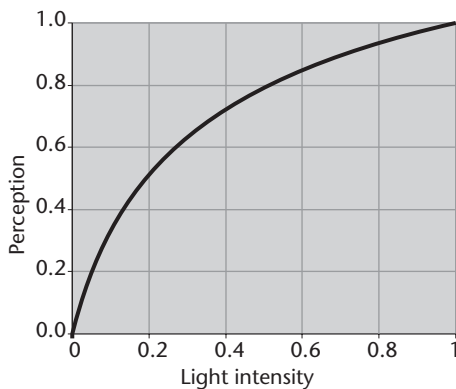
- *Convolution with continuous function.* Antialiasing requires filtering a continuous function. This is often computationally complex. Most real-world algorithms, such as filtered subsampling, are only crude approximations.
- *Info beyond screen edges.* You need to image information outside the screen region if the filter kernel is wider than \pm half a pixel.
- *Bad hardware reconstruction.* You usually have no control over the reconstruction filter; the hardware display device dictates it. And it's never a sinc/x function. Since you do not have a perfect reconstruction filter, your theoretical antialiasing efforts cannot achieve their potential. You sometimes must sacrifice some perfectly good, representable frequencies to compensate for imperfect reconstruction.

Also, a variety of nonlinear sampling techniques are being researched that can do better, but they present their own sets of tradeoffs.

7 Relationship between pixel value and light intensity emitted by a CRT.



8 Relationship between light intensity and human eye perception.



A pixel is a quantization of intensity

Now let's leave the spatial domain and consider the intensity domain. Encoding a pixel intensity digitally implies some quantization of the pixel intensity. Hardware pixels are usually quantized into 8 bits per color, but as we'll see in future columns, that isn't really enough. It's much better, if we can afford it (and we usually can nowadays), to use 16 or even 32 bits per pixel per color. The trick is in figuring out the most effective encoding to use these 16 or 32 bits. Three encoding schemes are the most typical: gamma corrected, linear fixed point, and floating point.

Virtually all 8-bit encodings are gamma corrected. This is motivated by a combination of the physics of CRTs and a modeling of the intensity sensitivity of the human eye. First, consider CRT physics. Passing an 8-bit quantity through a digital/analog converter and applying the net voltage to a CRT doesn't generate a linear relationship between the 8-bit value and the displayed intensity. The intensity follows the gamma curve $I = V^\gamma$ (see Figure 7). The effective value of gamma varies from CRT to CRT and as the adjustments of a particular CRT are modified, but is generally in the range from 1 to 3. The computer and video industry have various slightly different standards for the exact function that maps an 8-bit quantity into a physical intensity, but the important point is that none of the mapping functions are linear. They all look roughly like Figure 7. In fact, even though the native physics of LCDs differs from the curve of Figure 7, LCDs will usually

have internal circuitry to mimic a CRT's gamma curve for legacy compatibility.

An important implication of this gamma function is that when a pixel intensity is quantized into 8 bits (on the horizontal axis of Figure 7), the brightness range of the quantization bins (along the vertical axis of Figure 7) are not all the same size. They are smaller for the dimmer end of the range than they are for the brighter end of the range. And that's a good thing because of how the eye works. The eye happens to respond to light intensity in the roughly logarithmic fashion of Figure 8. This is pretty close to the inverse of the gamma function. This means that the nonequal spacing of the quantization bins in intensity translates into approximately equal spacing of the bins in perceptual space. How convenient.

The nonlinear relationship between pixel value and intensity gives rise to another problem. Many image-processing programs that operate on 8-bit pixel values don't take this nonlinearity into account, simply performing arithmetic on the 8-bit quantities as though they were linear representations of intensity. For an investigation of the errors this can generate, see an earlier column.¹ The correct processing would be to convert the 8-bit quantity into a linear intensity (presumably having more bits), perform the calculation, and then re-encode that linear value into the proper gamma-corrected 8-bit quantity. This calculation has historically been considered too slow for many programs, but that argument is going away as processors (both CPU and GPU) become faster.

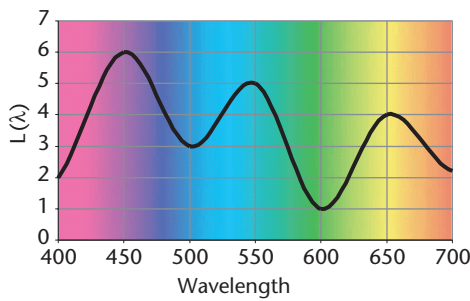
But, as previously stated, we are now moving into an era where 16 and 32 bits per pixel per color is practical. The question is how we choose between gamma-corrected, linear fixed point, and floating point representations. I will present some observations on this question in a future column.

A pixel is a projection of the color spectrum

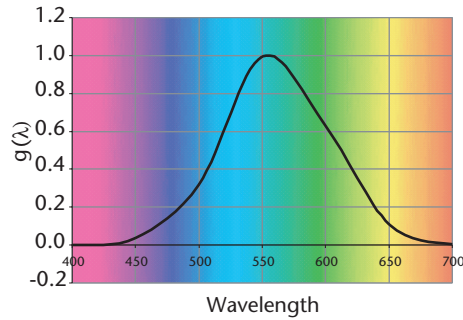
Color pixels are generally thought to have red, green, and blue components. This, however, is a vast oversimplification of the real physical world. A physically accurate color pixel model would require a complete spectral distribution $L(\lambda)$ at each pixel (see Figure 9). Fortunately, the color perception mechanism of the human eye usually makes this unnecessary. The eye and brain transform an entire spectral distribution into just three numbers. Each number comes from a different sensor in the retina. Each sensor type has a continuous spectral sensitivity as a function of wavelength. For example, the green sensor has the sensitivity $g(\lambda)$ of Figure 10. The green value that the brain senses is then the integral

$$G = \int g(\lambda) L(\lambda)$$

with similar calculations for red and blue. This information-collapsing mechanism means that many different spectra might generate the same three color-perception values. (Such colors are called *metamers*.) It's merely necessary to construct a spectrum at a pixel that's a



9 The complete (hypothetical) spectrum of a pixel.



10 One of the three spectral sensitivity functions of the human eye.

metamer of the desired color. This is possible by a weighed sum of just three primary spectra—for example, the familiar red, green, and blue display primaries. It's important to not get carried away by the approximation, because it has the following problems.

First, the primary weights (that is, display RGB values) implied by one display device aren't the same as the primary weights implied by another device. This is the familiar color correction problem that printer manufacturers have addressed most carefully. The manufacturers of CRT or LCD display devices have paid less attention to color correction, even though their color primaries aren't exactly the same either. Carefully constructed standards can at least ensure that enough information is present in a stored image to convert it for a desired output device. This gets into the topic of color management systems.

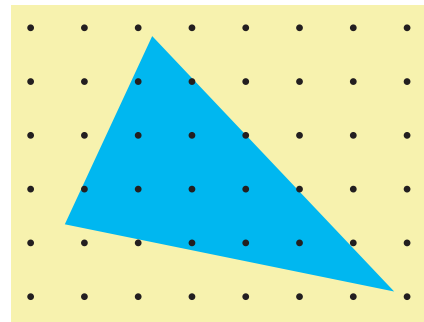
Second, sometimes a desired display color requires a negative weight for one or more display primaries. This color can't be displayed by a real-world display using those particular display primaries; it's too saturated. In fact, for any three given display primaries, many physical colors will exist that those primaries can't display. The printing industry addresses this problem by using more than three colored inks. Again, the electronic display community has paid less attention to this problem, perhaps because printers are more varied in their output responses, or perhaps because printed output is more permanent.

Finally, there's a temptation to do image processing and rendering calculations simply using three color values. The light of a particular color \mathbf{L} is modified by paint of a particular color \mathbf{S} by multiplying the light's RGB values by the paint's RGB values:

$$\begin{aligned} \text{Light: } & \begin{pmatrix} R_L & G_L & B_L \end{pmatrix} \\ \text{Surface: } & \begin{pmatrix} R_S & G_S & B_S \end{pmatrix} \\ \text{Net: } & \begin{pmatrix} R_S \times R_L & G_S \times G_L & B_S \times B_L \end{pmatrix} \end{aligned}$$

where

$$\begin{aligned} G_L &= \int g(\lambda)L(\lambda) \\ G_S &= \int g(\lambda)S(\lambda) \text{ and so on} \end{aligned}$$



11 An ideal blue triangle placed on a yellow background.

This is a very crude approximation to the physics. A true physical simulation would go back to the complete spectra and multiply them together. The collapse of the spectra to three values only happens at the very end of the process when display values are calculated:

$$G = \int g(\lambda)S(\lambda)L(\lambda)$$

It is, in fact, remarkable that the three-color approximation works as well as it does. Nevertheless, attempts to model the physics of light more correctly have yielded some interesting results.^{2,3}

A pixel is a transparency

The common operations of compositing and blending of images have motivated the creation of an extra value for each pixel: a transparency. This contains the amount of geometric coverage that the pixel represents and is typically named alpha. For example, if the image is of a geometric shape (see Figure 11), the pixels outside the boundaries of that shape would have an alpha value of 0, those inside the shape would have an alpha value of 1, and those on the edges would have some fractional value for alpha.

When this image is overlaid on top of a background, the alpha value at each pixel will indicate how much of the pixel value should appear while one minus the alpha value will indicate how much background should show

12 Geometric coverage of the blue triangle for one pixel. Final color = F over B.
 $B \equiv \alpha F + (1 - \alpha)B.$

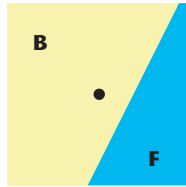


Table 1. Non-premultiplied versus pre-multiplied alpha.

Original Form	Premultiplied Form
$F = (F_R, F_G, F_B, \alpha)$	$\tilde{F} \equiv (\alpha F_R, \alpha F_G, \alpha F_B, \alpha)$
$F \text{ over } B \equiv \alpha F + (1 - \alpha)B$	$\tilde{F} \text{ over } \tilde{B} = \tilde{F} + (1 - F\alpha)\tilde{B}$

through. This operation is called the *over operator* (see Figure 12).

We store the alpha value as an extra component of the pixel color:

$$F = (F_R, F_G, F_B, \alpha)$$

A more complete analysis of the compositing operation^{4,5} shows that it's convenient to store pixel colors with the alpha value already pre-multiplied into the color values (see Table 1).

The various compositing operations become more mathematically regular when expressed in terms of the pre-multiplied form. In particular, the alpha component resulting from an over operator has the same formulation as the RGB components. This form also simplifies association between multiple layers of objects since

$$(\tilde{F} \text{ over } \tilde{G}) \text{ over } \tilde{B} = \tilde{F} \text{ over } (\tilde{G} \text{ over } \tilde{B})$$

This means that, when represented in this form, a stack of images can easily be composited either front to back, or back to front. (Various considerations indicate that the front-to-back ordering is preferable.)

There are, however, situations where the original non-premultiplied form appears more natural. (Boy, I wish there was a better name for non-premultiplied—what a mouthful.) These situations tend to come from the graphics design community whose application is to crop or extract an irregularly shaped piece out of an existing image.

Each of these forms (pre-multiplied and non-pre-multiplied) has a vocal constituency and a collection of situations where their form works best. Filtering operations on pixels seem to be more natural on the pre-multiplied version of the color. But some polygon color interpolations make more sense with non-pre-multiplied alpha. But many synthetic image-generating algorithms generate the pre-multiplied \tilde{F} directly. In a future column I hope to bring some peace to the argument. Basically we can go back to the original Porter and Duff paper⁴ and look at the Dissolve operator, which is essentially a scalar product with the pixel:

$$\text{dissolve}(\delta, \tilde{F}) = \delta(F_R, F_G, F_B, F_\alpha)$$

The pre-multiplied situation is where delta equals 1:

$$(F_R, F_G, F_B, F_\alpha) = \tilde{F}$$

The non-pre-multiplied situation is where alpha equals 1. And then you store the delta in the redundant alpha slot:

$$\delta(F_R, F_G, F_B, 1) = (F, \delta)$$

Using this viewpoint, the dichotomy between pre- and non-pre-multiplied pixels can be seen as two different special cases of a more general construct. A Grand Unified Theory of pixels. Yay!

A pixel is a piece of colored glass

While the alpha channel was originally conceived to represent an antialiased value for geometric coverage, we can also use it to represent object transparency. This works properly, however, only for objects with monochrome transmissivity. To represent colored glass, for example, the most obvious step would be to have a separate alpha channel for each color primary:

$$\tilde{F} = (F_R, F_G, F_B, \alpha_R, \alpha_G, \alpha_B)$$

This has become a common technique in current rendering algorithms. Some thought, though, will reveal that this situation is the same as for color reflectivity discussed in the previous section. A physically correct simulation of color transparency would require the storage and manipulation of a complete spectral curve for transparency as well as one for reflectivity:

$$\tilde{F} = (F(\lambda), \alpha(\lambda))$$

Experiments on the visual effects of typical approximations to this ideal have probably been done, but I don't know of any at this time. If any of you have information, please let me know.

A pixel is a bunch of other stuff

Work in image-based rendering has added yet more components to a pixel. In this case, pixels have one or more depth components as well as color and transparency components. I can imagine any number of other things to add that can be useful in more exotic rendering algorithms.

A pixel is an information-destroying technique

Ultimately a pixel must be viewed. The total list of processing required to view a pixel then includes (but is not limited to)

- antialiasing,
- offset sampling,

- color space projection,
- reconstruction filter compensation,
- compositing,
- gamma correction, and
- quantization and dithering.

If you look at all these operations you might see a pattern: Almost all of them throw away information. You filter out high frequencies, quantize intensities into bins, project a continuous color spectrum into three numbers, and represent geometric edges with a single transparency value. Seen in this way we can see that an ordinary hardware pixel, either refreshed on the screen or stored in a file, is simply a bad data compression technique.

Any rendering algorithm or image processing operation that converts data to pixels generally loses information about the original data that it uses as input. A few polygons become thousands of pixels; a high-resolution image becomes a low-resolution image. If we are interested in preserving that data we are best advised to store it in its original form. Conversion to pixels for viewing purposes used to be a slow operation, but with faster processors we no longer need to do the image generation

offline for speed purposes. We can recalculate the image whenever we need to look at it. Turning data into pixels is, quite literally, the last thing you should do. ■

References

1. J.F. Blinn, "A Ghost in a Snowstorm," *IEEE Computer Graphics and Applications*, vol. 18, no. 1, 1998, pp. 79-84.
2. R. Hall, "Comparing Spectral Color Computation Methods," *IEEE Computer Graphics and Applications*, vol. 19, no. 4, 1999, pp. 36-46.
3. G. Johnson, M. Garrett, and M.D. Fairchild, "Full-Spectral Color Calculations in Realistic Image Synthesis," *IEEE Computer Graphics and Applications*, vol. 19, no. 4, 1999, pp. 47-53.
4. T. Porter and T. Duff, "Compositing Digital Images," *Proc. Siggraph 84*, ACM Press, 1984, pp. 253-259.
5. J.F. Blinn, "Compositing, Part 1: Theory," *IEEE Computer Graphics and Applications*, vol. 14, no. 5, 1994, pp. 83-87.

Readers may contact Jim Blinn at blinn@microsoft.com.

PURPOSE The IEEE Computer Society is the world's largest association of computing professionals, and is the leading provider of technical information in the field.

MEMBERSHIP Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEB SITE

The IEEE Computer Society's Web site, at www.computer.org, offers information and samples from the society's publications and conferences, as well as a broad range of information about technical committees, standards, student activities, and more.

BOARD OF GOVERNORS

Term Expiring 2005: Oscar N. Garcia, Mark A. Grant, Michel Israel, Robit Kapur, Stephen B. Seidman, Kathleen M. Swigger, Makoto Takizawa

Term Expiring 2006: Mark Christensen, Alan Clements, Annie Combelles, Ann Q. Gates, James D. Isaak, Susan A. Mengel, Bill N. Schilit

Term Expiring 2007: Jean M. Bacon, George V. Cybenko, Richard A. Kemmerer, Susan K. (Kathy) Land, Itaru Mimura, Brian M. O'Connell, Christina M. Schober

Next Board Meeting: 4 Nov. 2005, Philadelphia

IEEE OFFICERS

President and CEO: W. CLEON ANDERSON

President-Elect: MICHAEL R. LIGHTNER

Past President: ARTHUR W. WINSTON

Executive Director: TBD

Secretary: MOHAMED EL-HAWARY

Treasurer: JOSEPH V. LILLIE

VP, Educational Activities: MOSHE KAM

VP, Pub. Services & Products: LEAH H. JAMIESON

VP, Regional Activities: MARC T. APTER

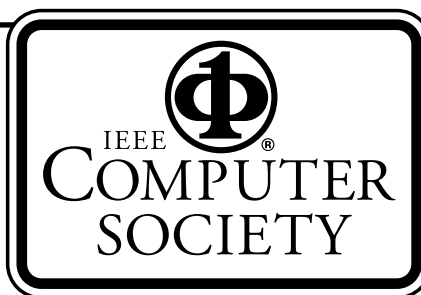
VP, Standards Association: JAMES T. CARLO

VP, Technical Activities: RALPH W. WYNDRUM JR.

IEEE Division V Director: GENE F. HOFFNAGLE

IEEE Division VIII Director: STEPHEN L. DIAMOND

President, IEEE-USA: GERARD A. ALPHONSE



COMPUTER SOCIETY OFFICES

Headquarters Office

1730 Massachusetts Ave. NW

Washington, DC 20036-1992

Phone: +1 202 371 0101

Fax: +1 202 728 9614

E-mail: bq.ofc@computer.org

Publications Office

10662 Los Vaqueros Cir., PO Box 3014

Los Alamitos, CA 90720-1314

Phone: +1 714 821 8380

E-mail: help@computer.org

Membership and Publication Orders:

Phone: +1 800 272 6657

Fax: +1 714 821 4641

E-mail: help@computer.org

Asia/Pacific Office

Watanabe Building

1-4-2 Minami-Aoyama, Minato-ku

Tokyo 107-0062, Japan

Phone: +81 3 3408 3118

Fax: +81 3 3408 3553

E-mail: tokyo.ofc@computer.org



EXECUTIVE COMMITTEE

President:

GERALD L. ENGEL*

Computer Science & Engineering

Univ. of Connecticut, Stamford

1 University Place

Stamford, CT 06901-2315

Phone: +1 203 251 8431

Fax: +1 203 251 8592

g.engel@computer.org

President-Elect: DEBORAH M. COOPER*

Past President: CARL K. CHANG*

VP, Educational Activities: MURALI VARANASIT

VP, Electronic Products and Services:

JAMES W. MOORE (2ND VP)*

VP, Conferences and Tutorials:

YERVANT ZORIAN†

VP, Chapters Activities:

CHRISTINA M. SCHOBER*

VP, Publications: MICHAEL R. WILLIAMS (1ST VP)*

VP, Standards Activities: SUSAN K. (KATHY) LAND*

VP, Technical Activities: STEPHANIE M.

WHITE†

Secretary: STEPHEN B. SEIDMAN*

Treasurer: RANGACHAR KASTURIT

2004-2005 IEEE Division V Director:

GENE F. HOFFNAGLE†

2005-2006 IEEE Division VIII Director:

STEPHEN L. DIAMOND†

2005 IEEE Division V Director-Elect:

OSCAR N. GARCIA*

Computer Editor in Chief: DORIS L. CARVER†

Executive Director: DAVID W. HENNAGE†

* voting member of the Board of Governors

† nonvoting member of the Board of Governors

EXECUTIVE STAFF

Executive Director: DAVID W. HENNAGE

Assoc. Executive Director: ANNE MARIE KELLY

Publisher: ANGELA BURGESS

Associate Publisher: DICK PRICE

Director, Administration: VIOLET S. DOAN

Director, Information Technology & Services:

ROBERT CARE

Director, Business & Product Development:

PETER TURNER