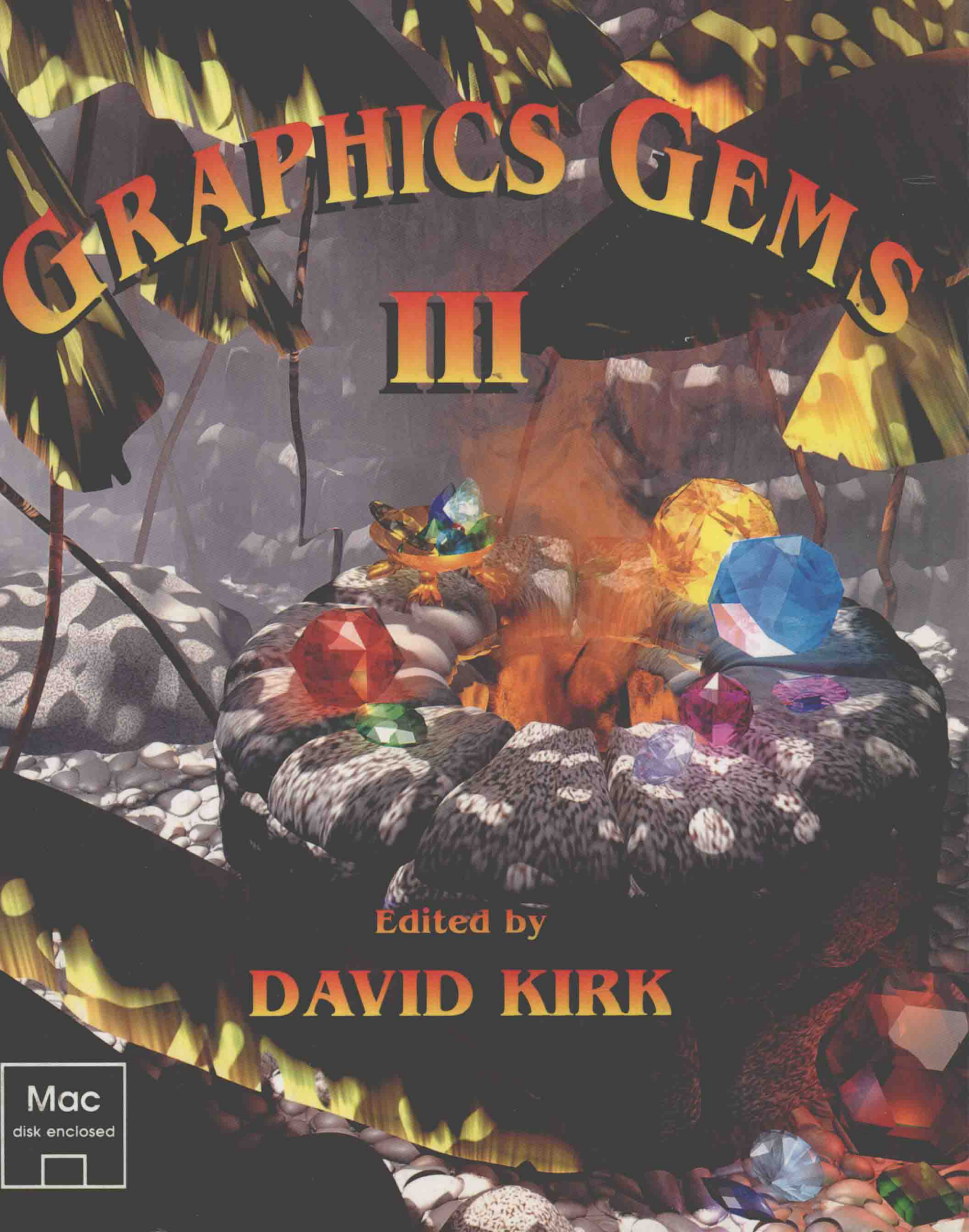

GRAPHICS GEMS
III
Edited by
DAVID KIRK
Mac
disk enclosed

GRAPHICS GEMS III

edited by

DAVID KIRK

California Institute of Technology
Computer Graphics Laboratory
Pasadena, California

ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers
Boston San Diego New York
London Sydney Tokyo Toronto

This book is printed on acid-free paper. ♾

ACADEMIC PRESS, INC.
1250 Sixth Avenue, San Diego, CA 92101-4311

United Kingdom Edition published by
ACADEMIC PRESS LIMITED
24–28 Oval Road, London NW1 7DX

Library of Congress Cataloging-in-Publication Data

Graphics gems III/edited by David Kirk.
p. cm.—(The Graphics gems series)
Includes bibliographical references and index.
ISBN 0-12-409670-0 (with IBM disk: alk. paper).—ISBN 0-12-409671-9 (with Macintosh disk: alk. paper)
1. Computer graphics. I. Kirk, David, 1960– . II. Series.
T385.G6973 1992
006.6′6—dc20 92-4467
CIP

Printed in the United States of America
93 94 95 MM 9 8 7 6 5 4 3 2

About the Cover

This cover image evolved out of a team effort between The VALIS Group, Andrew Glassner, Academic Press, and the generous cooperation and sponsorship of the folks at Pixar. Special thanks go to Tony Apodaca at Pixar for post-processing the gems in this picture using advanced RenderMan® techniques. The entire cover image was created using RenderMan® from Pixar.

We saw the *Graphics Gems III* cover as both an aesthetic challenge and an opportunity to demonstrate the kind of images that can be rendered with VALIS' products and Pixar's RenderMan®.

Given the time constraints, all of the geometry had to be kept as simple as possible so as not to require any complex and lengthy modeling efforts. Since RenderMan® works best when the geometric entities used in a 3-D scene are described by high order surfaces, most of the objects consisted of surfaces made up of quadric primitives and bicubic patches. Andrew's gem data and the Archimedean solids from the VALIS Prime RIB™ library are-the only polygonal objects in the picture.

Once all of the objects were defined, we used Pixar's Showplace™, a 3-D scene arranging application on the Mac, to position the objects and compose the scene. Showplace also allowed us to position the camera and the standard light sources as well as attach shaders to the objects.

In RenderMan®, shaders literally endow everything in the image with their own characteristic appearances, from the lights and the objects to the atmosphere. The shaders themselves are procedural descriptions of a material or other phenomenon written in the RenderMan® Shading Language. We did not use any scanned textures or 2-D paint retouching software to produce this picture.

Where appropriate, we used existing shaders on the surfaces of objects in this picture, taken from our commercially available VG Shaders™ + VG Looks™ libraries. For example, we used Hewn Stone Masonry (Volume 3) to create the temple wall and well; Stone Aggregate (Volume 3) for the jungle-floor, and polished metal (Volume 2) for the gold dish. In addition to these and other existing shaders, several new shaders were created for this image. The custom shaders include those for the banana leaves, the steamy jungle atmosphere, the well vapor, and the forest canopy dappled lighting effect.

Shaders also allowed us to do more with the surfaces than merely effect the way they are colored. In RenderMan®, shaders can transform simple surfaces into more complex forms by moving the surface geometry to add dimension and realistic detail. Using shaders we turned a cylinder into a stone well, spheres into boulders and rocks, and a flat horizontal plane into a jungle floor made up of stones and pebbles.

Similarly, we altered the surface opacity to create holes in surfaces. In this instance, we produced the ragged edges of the banana leaves and the well vapor by applying our custom RenderMan® shaders to flat pieces of geometry before rendering with PhotoRealistic RenderMan®.

Initially, this image was composed at a screen resolution of 450 × 600 pixels on a MacIIfx using Showplace. Rendering was done transparently over the network on a Unix® workstation using Pixar's NetRenderMan™. This configuration afforded us the convenience and flexibility of using a Mac for design and a workstation for quick rendering and preview during the picture-making process.

Once the design was complete, the final version of the image was rendered at 2250 × 3000 pixel resolution. The final rendering of this image was done on a 486 PC/DOS machine with Truevision's RenderPak™ and Horizon860™ card containing 32 MBytes of RAM.

During the rendering process, RenderMan® separates shadows into a temporary file called a shadow map. The 2k × 2k shadow map for this image was rendered in less than an hour. However, using shaders to alter the surface geometry increases rendering time and memory requirements dramatically. As a result, we had to divide the image into 64 separate pieces and render each one individually. The total rendering time for all 64 pieces was 41.7 hours. Once these were computed,

the TIFF tools from Pixar's RenderMan Toolkit™ were used to join the pieces together into a single, 33 MByte image file.

When the image was ready for output to film, we transferred the image file to a removable cartridge and sent it to a local output service. They output the electronic file onto 4×5 Ecktachrome color film and had it developed. The transparency was then sent to Academic Press in Cambridge, Massachusetts where they added the title and other elements to the final cover and had everything scanned and turned into four-color separations which were then supplied to their printer.

We hope you like this image. Producing it was fun and educational. As you might guess, many "graphic gems" were employed in the software used to produce this picture.

Mitch Prater, Senior Partner
Dr. Bill Kolomyjec, Senior Partner
RoseAnn Alspektor, Senior Partner
The VALIS Group
Pt. Richmond, CA
March, 1992

FOREWORD

by Andrew Glassner

Welcome to *Graphics Gems III*, an entirely new collection of tips, techniques, and algorithms for the practicing computer graphics programmer. Many ideas that were once passed on through personal contacts or chance conversations can now be found here, clearly explained and demonstrated. Many are illustrated with accompanying source code.

It is particularly pleasant for me to see new volumes of Gems, since each is something of a surprise. The original volume was meant to be a self-contained collection. At the last moment we included a card with contributor's information in case there might be enough interest to someday prepare a second edition. When the first volume was finished and at the printer's, I returned to my research in rendering, modeling and animation techniques.

As with the first volume, I was surprised by the quantity and quality of the contributions that came flowing in. We realized there was a demand, a need for an entirely new second volume, and *Graphics Gems II* was born. The same cycle has now repeated itself again, and we have happily arrived at the creation of a third collection of useful graphics tools.

Since the first volume of *Graphics Gems* was published, I have spoken to many readers and discovered that these books have helped people learn graphics by starting with working codes and just exploring intuitively. I didn't expect people to play with the codes so freely, but I think I now see why this helps. It is often exciting to start learning a new medium by simply messing around in it, and understanding how it flows. My piano teacher encourages new students to begin by spending time playing freely at the keyboard: keep a few scales or chord progressions in mind, but otherwise explore the spaces of melody, harmony, and rhythm. When I started to learn new mediums in art classes, I often spent time simply playing with the medium: squishing clay into odd shapes or brushing paint on paper in free and unplanned motions. Of course one often moves on to develop control and technique in order to communicate one's message better, but much creativity springs from such uncontrolled and spirited play.

It is difficult for the novice to play at programming. There is little room for simple expression or error. A simple program does not communicate with the same range and strength as a masterfully simple line drawing or haunting melody. A programmer cannot hit a few wrong notes, or tolerate an undesired ripple in a line. If the syntax isn't right, the program won't compile; if the semantics aren't right, the program won't do anything interesting. There are exceptions to the latter statement, but they are notable because of their rarity. If you're going to write a program to accomplish a task, you've got to do some things completely right, and everything else almost perfectly. That can be an intimidating realization, particularly for the beginner: if a newly constructed program doesn't work, the problem could be in a million places, anywhere from the architecture to data structures,

algorithms, or coding errors. The chance to start with something that already works removes the barrier to exploration: your program already works. If you want to change it, you can, and you will discover which new ideas work and which ones don't.

I believe that the success of the *Graphics Gems* series demonstrates something very positive about our research and practice communities. The modern view of science is an aggregate of many factors, but one popular myth depicts the researcher as a dispassionate observer who seeks evidence for some ultimate truth. The classical model is that this objective researcher piles one recorded fact upon another, continuously improving an underlying theoretical basis. This model has been eroded in recent years, but it remains potent in many engineering and scientific texts and curricula. The practical application of computer graphics is sometimes compared to a similarly theoretical commercial industry: trade secrets abound, and anything an engineer learns remains proprietary to the firm for as long as possible, to capitalize on the advantage. I do not believe that either of these attitudes are accurate or fruitful in the long run. Most researchers are biased. They believe something is true, from either experience or intuition, and seek support and verification for that truth. Sometimes there are surprises along the way, but one does not simply juggle symbols at random and hope to find a meaningful equation or program. It is our experience and insight that guide the search for new learning and limited truths, or the clear demonstration of errors of the guiding principle. I hail these prejudices, because they form our core beliefs, and allow us to choose and judge our work. There are an infinite number of interesting problems, and many ways to solve each one. Our biases help us pick useful problems to solve, and to judge the quality and elegance of the solution. By explicitly stating our beliefs, we are better able to understand them, emphasizing some and expunging others, and improve. Programmers of graphics software know that the whole is much more than the sum of the parts. A snippet of geometry can make a complex algorithm simple, or the correct, stable analytical solution can replace an expensive numerical approximation. Like an orchestral arranger, the software engineer weaves together the strengths and weaknesses of the tools available to make a new program that is more powerful than any component.

When we share our components, we all benefit. Two products may share some basic algorithms, but that alone hardly makes them comparable. The fact that so many people have contributed to *Gems* shows that we are not afraid to demonstrate our preferences for what is interesting and what is not, what is good and what is bad, and what is appropriate to share with colleagues.

I believe that the *Graphics Gems* series demonstrates some of the best qualities in the traditional models of the researcher and engineer. Gems are written by programmers who work in the field who are motivated by the opportunity to share some interesting or useful technique with their colleagues. Thus we avoid reinventing the wheel, and by sharing this information, we help each other move towards a common goal of amassing a body of useful techniques to be shared throughout the community.

I believe computer graphics has the potential to go beyond its current preoccupation with photorealism and simple surfaces and expand into a new creative medium. The materials from which we will shape this new medium are algorithms. As our mastery of algorithms grows, so will our ability to imagine new applications and make them real, enabling new forms of creative expression. I hope that the algorithms in this book will help each of us move closer to that goal.

PREFACE

This volume attempts to continue along the path blazed by the first two volumes of this series, capturing the spirit of the creative graphics programmer. Each of the Gems represents a carefully crafted technique or idea that has proven useful for the respective author. These contributors have graciously allowed these ideas to be shared with you. The resulting collection of ideas, tricks, techniques, and tools is a rough sketch of the character of the entire graphics field. It represents the diversity of the field, containing a wide variety of approaches to solving problems, large and small. As such, it "takes the pulse" of the graphics community, and presents you with ideas that a wide variety of individuals find interesting, useful, and important. I hope that you will find them so as well.

This book can be used in many ways. It can be used as a reference, to find the solution to a specific problem that confronts you. If you are addressing the same problem as one discussed in a particular Gem, you're more than halfway to a solution, particularly if that Gem provides C or C++ code. Many of the ideas in this volume can also be used as a starting point for new work, providing a fresh point of view. However you choose to use this volume, there are many ideas contained herein.

This volume retains the overall structure and organization, mathematical notation, and style of pseudo-code as in the first and second volumes. Some of the individual chapter names have been changed to allow a partitioning that is more appropriate for the current crop of Gems. Every attempt has been made to group similar Gems in the same chapter. Many of the chapter headings appeared in the first two volumes, although some are new. Ray tracing and radiosity have been combined into one chapter, since many of the gems are applicable to either technique. Also, a chapter more generally titled "Rendering" has been added, which contains many algorithms that are applicable to a variety of techniques for making pictures.

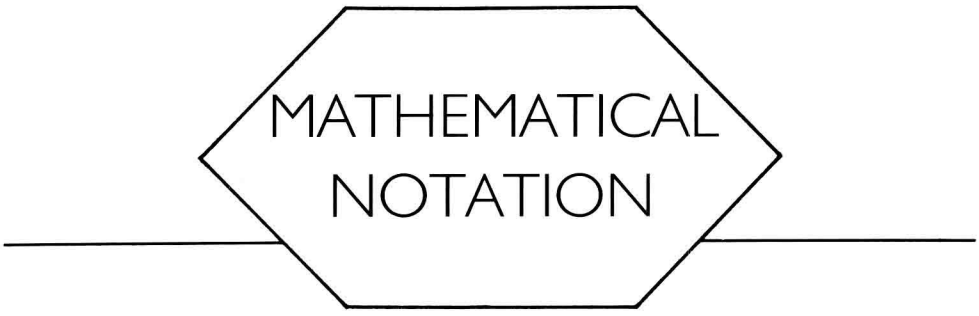
As in the second volume, we have taken some of the important sections from the first volume and included them verbatim. These sections are entitled "Mathematical Notation," "Pseudo-Code," and the listings,

"Graphics Gems C Header File," and "2-D and 3-D Vector Library," the last of which was revised in volume two of Graphics Gems.

At the end of most Gems, there are references to similar Gems whether in this volume or the previous ones, identified by volume and page number. This should be helpful in providing background for the Gems, although most of them stand quite well on their own. The only background assumed in most cases is a general knowledge of computer graphics, plus a small amount of skill in mathematics.

The C programming language has been used for most of the program listings in the Appendix, although several of the Gems have C++ implementations. Both languages are widely used, and the choice of which language to use was left to the individual authors. As in the first two volumes, all of the C and C++ code in this book is in the public domain, and is yours to study, modify, and use. As of this writing, all of the code listings are available via anonymous ftp transfer from the machines 'weedeater.math.yale.edu' (internet address 128.36.23.17), and 'princeton.edu' (internet address 128.112.128.1). 'princeton.edu' is the preferred site. When you connect to either of these machines using ftp, log in as 'anonymous' giving your full e-mail address as the password. Then use the 'cd' command to move to the directory 'pub/GraphicsGems' on 'weedeater', or the directory 'pub/Graphics/GraphicsGems' on 'princeton'. Code for *Graphics Gems I*, *II*, and *III* is kept in directories named 'Gems', 'GemsII', and 'GemsIII', respectively. Download and read the file called 'README' to learn about where the code is kept, and how to report bugs. In addition to the anonymous ftp site, the source listings of the gems are available on the enclosed diskette in either IBM PC format or Apple Macintosh format.

Finally, I'd like to thank all of the people who have helped along the way to make this volume possible. First and foremost, I'd like to thank Andrew Glassner for seeing the need for this type of book and starting the series, and to Jim Arvo for providing the next link in the chain. I'd also like to thank all of the contributors, who really comprise the heart of the book. Certainly, without their cleverness and initiative, this book would not exist. I owe Jenifer Swetland and her assistant Lynne Gagnon a great deal for their magical abilities applied toward making the whole production process go smoothly. Special thanks go to my diligent and thoughtful reviewers—Terry Lindgren, Jim Arvo, Andrew Glassner, Eric Haines, Douglas Voorhies, Devendra Kalra, Ronen Barzel and John Snyder. Without their carefully rendered opinions, my job would have been a lot harder. Finally, thank you to Craig Kolb for providing a safe place to keep the public domain C code.



Geometric Objects

0	the number 0, the zero vector, the point (0, 0), the point (0, 0, 0)
a, b, c	real numbers (lower-case italics)
P, Q	points (upper-case italics)
l, m	lines (lower-case bold)
A, B	vectors (upper-case bold) (components A_i)
M	matrix (upper-case bold)
θ, φ	angles (lower-case greek)

Derived Objects

$\mathbf{A}^{\perp}$	the vector perpendicular to **A** (valid only in 2D, where $\mathbf{A}^{\perp} = (-A_y, A_x)$)
$\mathbf{M}^{-1}$	the inverse of matrix **M**
$\mathbf{M}^{T}$	the transpose of matrix **M**
$\mathbf{M}^*$	the adjoint of matrix **M** $\left(\mathbf{M}^{-1} = \frac{\mathbf{M}^*}{\det(\mathbf{M})}\right)$
$\lvert\mathbf{M}\rvert$	determinant of **M**
$\det(\mathbf{M})$	same as above
$\mathbf{M}_{i,j}$	element from row i, column j of matrix **M** (top-left is (0, 0))
$\mathbf{M}_{i,}$	all of row i of matrix **M**

$\mathbf{M}_{,j}$	all of column j of matrix $\mathbf{M}$
$\triangle ABC$	triangle formed by points A, B, C
$\angle ABC$	angle formed by points A, B, C with vertex at B

Basic Operators

$+ , - , / , *$	standard math operators
$\cdot$	the dot (or inner or scalar) product
$\times$	the cross (or outer or vector) product

Basic Expressions and Functions

$\lfloor x \rfloor$	floor of x (largest integer not greater than x)
$\lceil x \rceil$	ceiling of x (smallest integer not smaller than x)
a|b	modulo arithmetic; remainder of a ÷ b
a mod b	same as above
$B_i^n(t)$	Bernstein polynomial = $\binom{n}{i} t^i (1-t)^{n-i}$, $i = 0 \cdots n$
$\binom{n}{i}$	binomial coefficient $\dfrac{n!}{(n-i)!i!}$

Declarations (not required)

name: TYPE ← initialValue;
examples:
π:**real** ← 3.14159;
v: **array** [0..3] **of integer** ← [0, 1, 2, 3];

Primitive Data Types

array [lowerBound..upperBound] **of** TYPE;
boolean
char
integer
real
double
point
vector

matrix3
equivalent to:
matrix3: record [array [0..2] of array [0..2] of real;];
example: m: Matrix3 ← [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]];
m[2][1] is 8.0
m[0][2] ← 3.3; *assigns 3.3 to upper-right corner of matrix*

matrix4

equivalent to:
matrix4: record [array [0..3] of array [0..3] of real;];
example: m: Matrix4 ← [
[1.0, 2.0, 3.0, 4.0],
[5.0, 6.0, 7.0, 8.0],
[9.0, 10.0, 11.0, 12.0],
[13.0, 14.0, 15.0, 16.0]];
m[3][1] is 14.0
m[0][3] ← 3.3; *assigns 3.3 to upper-right corner of matrix*

Records (Structures)

Record definition:
Box: **record** [
left, right, top, bottom: **integer**;
];

newBox: Box ← **new**[Box];
dynamically allocate a new instance of Box and return a pointer to it

newBox.left ← 10;
this same notation is appropriate whether newBox is a pointer or structure

Arrays

v: **array** [0..3] **of integer** ← [0, 1, 2, 3]; *v is a four-element array of integers*

v[2] ← 5; *assign to third element of v*

Comments

A comment may appear anywhere – it is indicated by italics

Blocks

```
begin
        Statement;
        Statement;
        ...
        end;
```

Conditionals and Selections

```
if Test
        then Statement;
        [else Statement];          else clause is optional

result = select Item from
        instance: Statement;
        endcase: Statement;
```

Flow Control

```
for ControlVariable: Type ← InitialExpr, NextExpr do
        Statement;
        endloop;

until Test do
        Statement;
        endloop;

while Test do
        Statement;
        endloop;

loop; go directly to the next endloop

exit; go directly to the first statement after the next endloop

return[value]          return value as the result of this function call
```

Logical Connectives

or, and, not, xor

Bitwise Operators

bit-or, bit-and, bit-xor

Relations

$=, \neq, >, \geq, <, \leq$

Assignment Symbol

$\leftarrow$
(note: the test for equality is $=$)

Available Functions

These functions are defined on all data types

min(a, b)	*returns minimum of a and b*
max(a, b)	*returns maximum of a and b*
abs(a)	*returns absolute value of a*
sin(x)	*sin(x)*
cos(x)	*cos(x)*
tan(x)	*tan(x)*
arctan(y)	*arctan(y)*
arctan2(y, x)	*arctan(y / x), defined for all values of x and y*
arcsin(y)	*arcsin(y)*
arccos(y)	*arccos(y)*
rshift(x, b)	*shift x right b bits*
lshift(x, b)	*shift x left b bits*
swap(a, b)	*swap a and b*
lerp(α, l, h)	*linear interpolation:* $((1 - \alpha)*l) + (\alpha*h) = l + (\alpha*(h - l))$

clamp(v, l, h)	*return l if v < l, else h if v > h, else v: min(h, max(l, v))*
floor(x) or ⌊x⌋	*round x towards 0 to first integer*
ceiling(x) or ⌈x⌉	*round x away from 0 to first integer*
round(x)	*round x to nearest integer, if frac(x) = .5, round towards 0*
frac(x)	*fractional part of x*

CONTRIBUTORS

Numbers in parentheses indicate pages on which authors' gems begin.

Michael J. Allison (318), *Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, California 94450*

Franklin Antonio (199), *Qualcomm, Inc., 2765 Cordoba Cove, Del Mar, California 92014*

James Arvo (117, 383), *Program of Computer Graphics, Cornell University, Ithaca, New York 14853*

Didier Badouel (89), *Computer Systems Research Institute, University of Toronto, 10 King's College Road, Toronto, Ontario M5S 1A4, Canada, badouel@dgp.toronto.edu*

A. H. Barr (137), *Computer Graphics Laboratory, California Institute of Technology, Pasadena, California 91125*

Ronen Barzel (34, 374), *Computer Graphics, California Institute of Technology, 350-74, Pasadena, California 91125, ronen@gg.caltech.edu*

Jeffrey C. Beran-Koehn (324), *Department of Computer Science, North Dakota State University, 300 Minard Hall, SU Station, P.O. Box 5075, Fargo, North Dakota 58105-5075, beran-ko@plains,nodak.edu*

Buming Bian (314), *UT System Center for High Performance Computing, 10100 Burnet Rd., Austin, Texas 78758-4497, buming@chpc.utexas.edu*

Dennis Bragg (20), *Graphics Software, Inc., 23428 Deer Run, Bullard, Texas 75757*

Russell C. H. Cheng (343, 349), *School of Mathematics, University of Wales, College of Cardiff, Senghynnydd Road, P.O. Box 915, Cardiff CF2 4AG, United Kingdom, cheng@cardiff.ac.uk*