

1 DISK(S)
ENCLOSED

Alan Watt

Third Edition

3D Computer Graphics



ADDISON-WESLEY





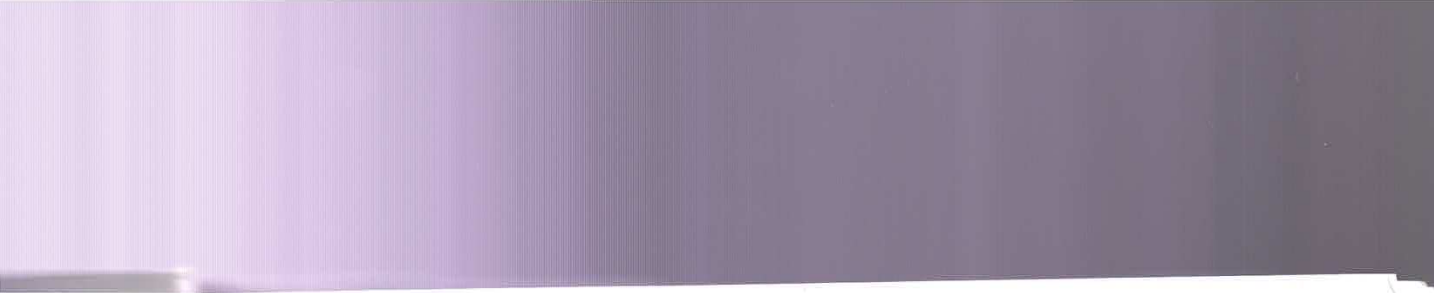
Alan Watt

Third Edition

3D Computer Graphics

T
385
W3778
2000

WENDT
ADDISON
WESLEY





3D Computer Graphics



3D Computer Graphics

THIRD EDITION

ALAN WATT



ADDISON-WESLEY

An imprint of PEARSON EDUCATION

Harlow, England · London · New York · Reading, Massachusetts · San Francisco · Toronto · Don Mills, Ontario · Sydney
Tokyo · Singapore · Hong Kong · Seoul · Taipei · Cape Town · Madrid · Mexico City · Amsterdam · Munich · Paris · Milan

Pearson Education Limited

Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies around the world

Visit us on the World Wide Web at:
www.pearsoneduc.com

First published 1989
Second edition 1993
This edition first published 2000

© 1989, 1993 Addison-Wesley Publishing Ltd, Addison-Wesley Publishing Company Inc.
© Pearson Education Limited 2000

The right of Alan Watt to be identified as author of
this work has been asserted by him in accordance with
the Copyright, Designs, and Patents Act 1988.

All rights reserved; no part of this publication may be reproduced, stored
in a retrieval system, or transmitted in any form or by any means, electronic,
mechanical, photocopying, recording, or otherwise without either the prior
written permission of the Publishers or a licence permitting restricted copying
in the United Kingdom issued by the Copyright Licensing Agency Ltd,
90 Tottenham Court Road, London W1P 0LP.

The programs in this book have been included for their instructional value.
The publisher does not offer any warranties or representation in respect of their
fitness for a particular purpose, nor does the publisher accept any liability for any
loss or damage (other than for personal injury or death) arising from their use.

Many of the designations used by manufacturers and sellers to distinguish their
products are claimed as trademarks. Pearson Education Limited has made every
attempt to supply trademark information about manufacturers and their products
mentioned in this book. A list of trademark designations and their owners
appears on page xxii.

ISBN 0 201 39855 9

British Library Cataloguing-in-Publication Data
A catalogue record for this book can be obtained from the British Library.

Library of Congress Cataloging-in-Publication Data
Available from the publisher.

Typeset by 42
Printed and bound in The United States of America

10 9 8 7 6 5 4 3 2
04 03 02 01 00

**General Library System
University of Wisconsin - Madison
728 State Street
Madison, WI 53706-1494
U.S.A.**

Contents

Colour plates appear between pages 506 and 507

Preface

xvi

Acknowledgements

xxi

1 Mathematical fundamentals of computer graphics	1
1.1 Manipulating three-dimensional structures	1
1.1.1 Three-dimensional geometry in computer graphics – affine transformations	2
1.1.2 Transformations for changing coordinate systems	8
1.2 Structure-deforming transformations	9
1.3 Vectors and computer graphics	11
1.3.1 Addition of vectors	12
1.3.2 Length of vectors	12
1.3.3 Normal vectors and cross products	12
1.3.4 Normal vectors and dot products	14
1.3.5 Vectors associated with the normal vector reflection	15
1.4 Rays and computer graphics	17
1.4.1 Ray geometry – intersections	17
1.4.2 Intersections – ray–sphere	18
1.4.3 Intersections – ray–convex polygon	19
1.4.4 Intersections – ray–box	21
1.4.5 Intersections – ray–quadric	23
1.4.6 Ray tracing geometry – reflection and refraction	23
1.5 Interpolating properties in the image plane	25
2 Representation and modelling of three-dimensional objects (1)	27
Introduction	27
2.1 Polygonal representation of three-dimensional objects	33
2.1.1 Creating polygonal objects	37

2.1.2	Manual modelling of polygonal objects	38
2.1.3	Automatic generation of polygonal objects	38
2.1.4	Mathematical generation of polygonal objects	39
2.1.5	Procedural polygon mesh objects – fractal objects	44
2.2	Constructive solid geometry (CSG) representation of objects	46
2.3	Space subdivision techniques for object representation	51
2.3.1	Octrees and polygons	53
2.3.2	BSP trees	55
2.3.3	Creating voxel objects	56
2.4	Representing objects with implicit functions	56
2.5	Scene management and object representation	58
2.5.1	Polygon mesh optimization	59
2.6	Summary	64
3	Representation and modelling of three-dimensional objects (2)	66
	Introduction	66
3.1	Bézier curves	69
3.1.1	Joining Bézier curve segments	75
3.1.2	Summary of Bézier curve properties	77
3.2	B-spline representation	78
3.2.1	B-spline curves	78
3.2.2	Uniform B-splines	80
3.2.3	Non-uniform B-splines	84
3.2.4	Summary of B-spline curve properties	90
3.3	Rational curves	90
3.3.1	Rational Bézier curves	91
3.3.2	NURBS	93
3.4	From curves to surfaces	94
3.4.1	Continuity and Bézier patches	98
3.4.2	A Bézier patch object – the Utah teapot	100
3.5	B-spline surface patches	101
3.6	Modelling or creating patch surfaces	106
3.6.1	Cross-sectional or linear axis design example	107
3.6.2	Control polyhedron design – basic technique	110
3.6.3	Creating patch objects by surface fitting	115
3.7	From patches to objects	121
4	Representation and rendering	123
	Introduction	123
4.1	Rendering polygon meshes – a brief overview	124

4.2	Rendering parametric surfaces	125
4.2.1	Rendering directly from the patch descriptions	125
4.2.2	Patch to polygon conversion	128
4.2.3	Object space subdivision	128
4.2.4	Image space subdivision	135
4.3	Rendering a CSG description	138
4.4	Rendering a voxel description	140
4.5	Rendering implicit functions	141
5	The graphics pipeline (1): geometric operations	142
	Introduction	142
5.1	Coordinate spaces in the graphics pipeline	143
5.1.1	Local or modelling coordinate systems	143
5.1.2	World coordinate systems	143
5.1.3	Camera or eye or view coordinate system	143
5.2	Operations carried out in view space	147
5.2.1	Culling or back-face elimination	147
5.2.2	The view volume	147
5.2.3	Three-dimensional screen space	149
5.2.4	View volume and depth	152
5.3	Advanced viewing systems (PHIGS and GKS)	156
5.3.1	Overview of the PHIGS viewing system	157
5.3.2	The view orientation parameters	159
5.3.3	The view mapping parameters	159
5.3.4	The view plane in more detail	162
5.3.5	Implementing a PHIGS-type viewing system	164
6	The graphics pipeline (2): rendering or algorithmic processes	167
	Introduction	167
6.1	Clipping polygons against the view volume	168
6.2	Shading pixels	171
6.2.1	Local reflection models	173
6.2.2	Local reflection models – practical points	177
6.2.3	Local reflection models – light source considerations	179
6.3	Interpolative shading techniques	179
6.3.1	Interpolative shading techniques – Gouraud shading	180
6.3.2	Interpolative shading techniques – Phong shading	181
6.3.3	Renderer shading options	182
6.3.4	Comparison of Gouraud and Phong shading	183
6.4	Rasterization	183
6.4.1	Rasterizing edges	183
6.4.2	Rasterizing polygons	185

6.5	Order of rendering	187
6.6	Hidden surface removal	189
6.6.1	The Z-buffer algorithm	189
6.6.2	Z-buffer and CSG representation	190
6.6.3	Z-buffer and compositing	191
6.6.4	Z-buffer and rendering	192
6.6.5	Scan line Z-buffer	193
6.6.6	Spanning hidden surface removal	193
6.6.7	A spanning scan line algorithm	194
6.6.8	Z-buffer and complex scenes	196
6.6.9	Z-buffer summary	198
6.6.10	BSP trees and hidden surface removal	199
6.7	Multi-pass rendering and accumulation buffers	202
7	Simulating light-object interaction: local reflection models	205
	Introduction	205
7.1	Reflection from a perfect surface	206
7.2	Reflection from an imperfect surface	207
7.3	The bi-directional reflectance distribution function	208
7.4	Diffuse and specular components	211
7.5	Perfect diffuse – empirically spread specular reflection	212
7.6	Physically based specular reflection	213
7.6.1	Modelling the micro-geometry of the surface	214
7.6.2	Shadowing and masking effects	214
7.6.3	Viewing geometry	216
7.6.4	The Fresnel term	216
7.7	Pre-computing BRDFs	219
7.8	Physically based diffuse component	221
8	Mapping techniques	223
	Introduction	223
8.1	Two-dimensional texture maps to polygon mesh objects	228
8.1.1	Inverse mapping by bilinear interpolation	229
8.1.2	Inverse mapping by using an intermediate surface	230
8.2	Two-dimensional texture domain to bi-cubic parametric patch objects	234
8.3	Billboards	235
8.4	Bump mapping	236
8.4.1	A multi-pass technique for bump mapping	238
8.4.2	A pre-calculation technique for bump mapping	239

8.5	Light maps	240
8.6	Environment or reflection mapping	243
8.6.1	Cubic mapping	245
8.6.2	Sphere mapping	247
8.6.3	Environment mapping: comparative points	248
8.6.4	Surface properties and environment mapping	249
8.7	Three-dimensional texture domain techniques	251
8.7.1	Three-dimensional noise	251
8.7.2	Simulating turbulence	252
8.7.3	Three-dimensional texture and animation	254
8.7.4	Three-dimensional light maps	256
8.8	Anti-aliasing and texture mapping	256
8.9	Interactive techniques in texture mapping	260
9	Geometric shadows	263
	Introduction	263
9.1	Properties of shadows used in computer graphics	265
9.2	Simple shadows on a ground plane	265
9.3	Shadow algorithms	267
9.3.1	Shadow algorithms: projecting polygons/scan line	267
9.3.2	Shadow algorithms: shadow volumes	268
9.3.3	Shadow algorithms: derivation of shadow polygons from light source transformations	271
9.3.4	Shadow algorithms: shadow Z-buffer	271
10	Global illumination	275
	Introduction	275
10.1	Global illumination models	276
10.1.1	The rendering equation	277
10.1.2	Radiance, irradiance and the radiance equation	278
10.1.3	Path notation	281
10.2	The evolution of global illumination algorithms	283
10.3	Established algorithms – ray tracing and radiosity	284
10.3.1	Whitted ray tracing	284
10.3.2	Radiosity	286
10.4	Monte Carlo techniques in global illumination	288
10.5	Path tracing	292
10.6	Distributed ray tracing	294
10.7	Two-pass ray tracing	297
10.8	View dependence/independence and multi-pass methods	300

10.9 Caching illumination	301
10.10 Light volumes	303
10.11 Particle tracing and density estimation	304
11 The radiosity method	306
Introduction	306
11.1 Radiosity theory	308
11.2 Form factor determination	310
11.3 The Gauss-Seidel method	314
11.4 Seeing a partial solution – progressive refinement	315
11.5 Problems with the radiosity method	318
11.6 Artefacts in radiosity images	319
11.6.1 Hemicube artefacts	319
11.6.2 Reconstruction artefacts	321
11.6.3 Meshing artefacts	323
11.7 Meshing strategies	325
11.7.1 Adaptive or <i>a posteriori</i> meshing	325
11.7.2 <i>A priori</i> meshing	332
12 Ray tracing strategies	342
Introduction – Whitted ray tracing	342
12.1 The basic algorithm	343
12.1.1 Tracing rays – initial considerations	343
12.1.2 Lighting model components	344
12.1.3 Shadows	345
12.1.4 Hidden surface removal	346
12.2 Using recursion to implement ray tracing	347
12.3 The adventures of seven rays – a ray tracing study	350
12.4 Ray tracing polygon objects – interpolation of a normal at an intersection point in a polygon	352
12.5 Efficiency measures in ray tracing	354
12.5.1 Adaptive depth control	354
12.5.2 First hit speed up	355
12.5.3 Bounding objects with simple shapes	355
12.5.4 Secondary data structures	357
12.5.5 Ray space subdivision	363
12.6 The use of ray coherence	364
12.7 A historical digression – the optics of the rainbow	367

13 Volume rendering	370
Introduction	370
13.1 Volume rendering and the visualization of volume data	373
13.2 'Semi-transparent gel' option	377
13.2.1 Voxel classification	378
13.2.2 Transforming into the viewing direction	379
13.2.3 Compositing pixels along a ray	379
13.3 Semi-transparent gel plus surfaces	380
13.3.1 Explicit extraction of isosurfaces	382
13.4 Structural considerations in volume rendering algorithms	384
13.4.1 Ray casting (untransformed data)	385
13.4.2 Ray casting (transformed data)	387
13.4.3 Voxel projection method	388
13.5 Perspective projection in volume rendering	390
13.6 Three-dimensional texture and volume rendering	391
14 Anti-aliasing theory and practice	392
Introduction	392
14.1 Aliases and sampling	393
14.2 Jagged edges	397
14.3 Sampling in computer graphics compared with sampling reality	398
14.4 Sampling and reconstruction	400
14.5 A simple comparison	401
14.6 Pre-filtering methods	402
14.7 Supersampling or post-filtering	404
14.8 Non-uniform sampling – some theoretical concepts	406
14.9 The Fourier transform of images	411
15 Colour and computer graphics	418
Introduction	418
15.1 Colour sets in computer imagery	419
15.2 Colour and three-dimensional space	420
15.2.1 RGB space	423
15.2.2 The HSV single hexcone model	424
15.2.3 YIQ space	427
15.3 Colour, information and perceptual spaces	427
15.3.1 CIE XYZ space	429
15.3.2 CIE xyY space	433
15.4 Rendering and colour spaces	435

15.5 Monitor considerations	436
15.5.1 RGB _{monitor} space and other monitor considerations	436
15.5.2 Monitor considerations – different monitors and the same colour	437
15.5.3 Monitor considerations – colour gamut mapping	439
15.5.4 Monitor considerations – gamma correction	440
16 Image-based rendering and photo-modelling	443
Introduction	443
16.1 Reuse of previously rendered imagery – two-dimensional techniques	444
16.1.1 Planar impostors or sprites	445
16.1.2 Calculating the validity of planar impostors	445
16.2 Varying rendering resources	447
16.2.1 Priority rendering	448
16.2.2 Image layering	452
16.3 Using depth information	452
16.3.1 Three-dimensional warping	456
16.3.2 Layered depth images (LDIs)	458
16.4 View interpolation	460
16.4.1 View morphing	463
16.5 Four-dimensional techniques – the Lumigraph or light field rendering approach	463
16.6 Photo-modelling and IBR	465
16.6.1 Image-based rendering using photographic panoramas	469
16.6.2 Compositing panoramas	469
16.6.3 Photo-modelling for image-based rendering	470
17 Computer animation	473
Introduction	473
17.1 A categorization and description of computer animation techniques	476
17.2 Rigid body animation	477
17.2.1 Interpolation or keyframing	477
17.2.2 Explicit scripting	479
17.2.3 Interpolation of rotation	483
17.2.4 Using quaternions to represent rotation	484
17.2.5 Interpolating quaternions	488
17.2.6 The camera as an animated object	492
17.3 Linked structures and hierarchical motion	493
17.3.1 Solving the inverse kinematics problem	500

436			
436			
437			
439			
440			
443			
443			
444			
445			
445			
447			
447			
448			
452			
452			
456			
458			
460			
463			
465			
469			
469			
470			
473			
473			
476			
477			
477			
479			
483			
484			
488			
492			
493			
500			
	17.4 Dynamics in computer animation		504
	17.4.1 Basic theory for a rigid body – particles		505
	17.4.2 The nature of forces		506
	17.4.3 Rigid bodies – extended masses		507
	17.4.4 Using dynamics in computer animation		510
	17.4.5 Simulating the dynamics of a lumped mass		511
	17.4.6 Space–time constraints		515
	17.5 Collision detection		517
	17.5.1 Broad phase/narrow phase algorithms		518
	17.5.2 Broad phase collision detection with OBBs		519
	17.5.3 Narrow phase: pairs of convex polyhedra – exact collision detection		522
	17.5.4 Single phase algorithms – object hierarchies		524
	17.6 Collision response		526
	17.7 Particle animation		529
	17.8 Behavioural animation		531
	17.9 Summary		534
	18 Comparative image study		536
	Introduction		536
	18.1 Local reflection models		537
	18.2 Texture and shadow mapping		538
	18.3 Whitted ray tracing		539
	18.4 Radiosity		541
	18.5 RADIANCE		543
	18.6 Summary		543
	References		544
	Index		553

Preface

This is the third edition of a book that deals with the processes involved in converting a mathematical or geometric description of an object – a computer graphics model – into a visualization – a two-dimensional projection – that simulates the appearance of a real object. The analogy of a synthetic camera is often used and this is a good allusion provided we bear in mind certain important limitations that are not usually available in a computer graphics camera (depth of field and motion blur are two examples) and certain computer graphics facilities that do not appear in a camera (near and far clipping planes).

Algorithms in computer graphics mostly function in a three-dimensional domain and the creations in this space are then mapped into a two-dimensional display or image plane at a late stage in the overall process. Traditionally computer graphics has created pictures by starting with a very detailed geometric description, subjecting this to a series of transformations that orient a viewer and objects in three-dimensional space, then imitating reality by making the objects look solid and real – a process known as rendering. In the early 1980s there was a coming together of research – carried out in the 1970s into reflection models, hidden surface removal and the like – that resulted in the emergence of a *de facto* approach to image synthesis of solid objects. But now this is proving insufficient for the new demands of moving computer imagery and virtual reality and much research is being carried out into how to model complex objects, where the nature and shape of the object changes dynamically and into capturing the richness of the world without having to explicitly model every detail. Such efforts are resulting in diverse synthesis methods and modelling methods but at the moment there has been no emergence of new image generation techniques that rival the pseudo-standard way of modelling and rendering solid objects – a method that has been established since the mid-1970s.

So where did it all begin? Most of the development in computer graphics as we know it today was motivated by hardware evolution and the availability of new devices. Software rapidly developed to use the image producing hardware. In this respect the most important development is the so-called raster display, a device that proliferated in the mass market shortly after the development of the PC. In this device the complete image is stored in a memory variously called a

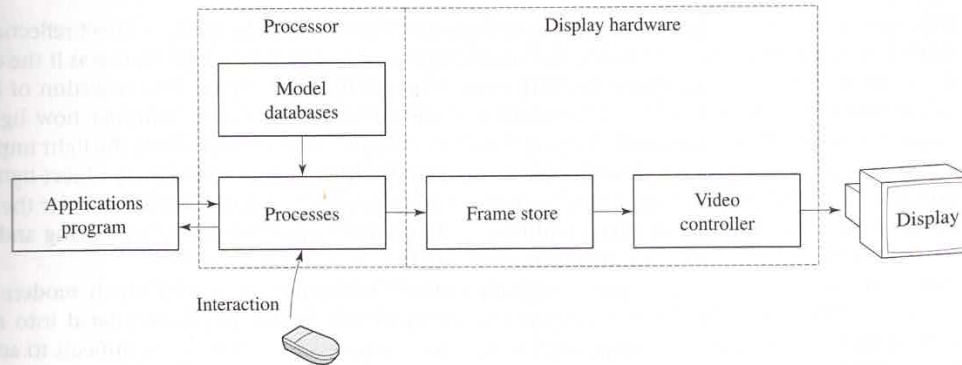


Figure P.1
The main elements of a graphics system.

frame store, a screen buffer or a refresh memory. This information – the discretized computer image – is continually converted by a video controller into a set of horizontal scan lines (a raster) which is then fed to a TV-type monitor. The image is generated by an application program which usually accesses a model or geometric description of an object or objects. The main elements in such a system are shown in Figure P.1. The display hardware to the right of the dotted line can be separate to the processor, but nowadays is usually integrated as in the case of an enhanced PC or a graphics workstation. The raster graphics device overshadows all other hardware developments in the sense that it made possible the display of shaded three-dimensional objects – the single most important theoretical development. The interaction of three-dimensional objects with a light source could be calculated and the effect projected into two-dimensional space and displayed by the device. Such shaded imagery is the foundation of modern computer graphics.

The two early landmark achievements that made shaded imagery possible are the algorithms developed by Gouraud in 1971 and Phong in 1975 enabling easy and fast calculation of the intensities of pixels when shading an object. The Phong technique is still in mainstream use and is undoubtedly responsible for most of the shaded images in computer graphics.

A brief history of shaded imagery

When we look at computer graphics from the viewpoint of its practitioners, we see that since the mid-1970s the developmental motivation has been photo-realism or the pursuit of techniques that make a graphics image of an object or scene indistinguishable from a TV image or photograph. A more recent strand of the application of these techniques is to display information in, for example, medicine, science and engineering.

The foundation of photo-realism is the calculation of light-object interaction and this splits neatly into two fields – the development of local reflection

models and the development of global models. Local or direct reflection models only consider the interaction of an object with a light source as if the object and light were floating in dark space. That is, only the first reflection of light from the object is considered. Global reflection models consider how light reflects from one object and travels onto another. In other words the light impinging on a point on the surface can come either from a light source (direct light) or indirect light that has first hit another object. Global interaction is for the most part an unsolved problem, although two partial solutions, ray tracing and radiosity, are now widely implemented.

Computer graphics research has gone the way of much modern scientific research – early major advances are created and consolidated into a practical technology. Later significant advances seem to be more difficult to achieve. We can say that most images are produced using the Phong local reflection model (first reported in 1975), fewer using ray tracing (first popularized in 1980) and fewer still using radiosity (first reported in 1984). Although there is still much research being carried out in light-scene interaction methodologies much of the current research in computer graphics is concerned more with applications, for example, with such general applications as animation, visualization and virtual reality. In the most important computer graphics publication (the annual SIG-GRAPH conference proceedings) there was in 1985 a total of 22 papers concerned with the production techniques of images (rendering, modelling and hardware) compared with 13 on what could loosely be called applications. A decade later in 1995 there were 37 papers on applications and 19 on image production techniques.

Modelling surface reflection with local interaction

Two early advances which went hand-in-hand were the development of hidden surface removal algorithms and shaded imagery – simulating the interaction of an object with a light source. Most of the hidden surface removal research was carried out in the 1970s and nowadays, for general-purpose use, the most common algorithm is the Z-buffer – an approach that is very easy to implement and combine with shading or rendering algorithms.

In shaded imagery the major prop is the Phong reflection model. This is an elegant but completely empirical model that usually ends up with an object reflecting more light than it receives. Its parameters are based on the grossest aspects of reflection of light from a surface. Despite this, it is the most widely used model in computer graphics – responsible for the vast majority of created images. Why is this so? Probably because users find it adequate and it is easy to implement.

Theoretically based reflection models attempt to model reflection more accurately and their parameters have physical meaning – that is they can be measured for a real surface. For example, light reflects differently from an isotropic surface, such as plastic, compared to its behaviour with a non-isotropic surface

such as brushed aluminium and such an effect can be imitated by explicitly modelling the surface characteristics. Such models attempt to imitate the behaviour of light at a 'milliscale' level (where the roughness or surface geometry is still much greater than the wavelength of light). Their purpose is to imitate the material signature – why different materials in reality look different. Alternatively, parameters of a model can be measured on a real surface and used in a simulation. The work into more elaborate or theoretical local reflection models does not seem to have gained any widespread acceptance as far as its implementation in rendering systems is concerned. This may be due to the fact that users do not perceive that the extra processing costs are worth the somewhat marginal improvement in the appearance of the shaded object.

All these models, while attending to the accurate modelling of light from a surface, are local models which means that they only consider the interaction of light with the object as if the object was floating in free space. No object-object interaction is considered and one of the main problems that immediately arises is that shadows – a phenomenon due to global interaction – are not incorporated into the model and have to be calculated by a separate 'add-on' algorithm.

The development of the Phong reflection model spawned research into add-on shadow algorithms and texture mapping, both of which enhanced the appearance of the shaded object and tempered the otherwise 'floating in free space' plastic look of the basic Phong model.

Modelling global interaction

The 1980s saw the development of two significant global models – light reflection models that attempt to evaluate the interaction between objects. Global interaction gives rise to such phenomena as the determination of the intensity of light within a shadow area, the reflection of objects in each other (specular interaction) and a subtle effect known as colour bleeding where the colour from a diffuse surface is transported to another nearby surface (diffuse interaction). The light intensity within a shadow area can only be determined from global interaction. An area in shadow, by definition, cannot receive light directly from a light source but only indirectly from light reflecting from another object. When you see shiny objects in a scene you expect to see in them reflections of other objects. A very shiny surface, such as chromium plate, behaves almost as a mirror taking all its surface detail from its surroundings and distorting this geometrically according to surface curvature.

The successful global models are ray tracing and radiosity. However, in their basic implementation both models only cater for one aspect of global illumination. Ray tracing attends to perfect specular reflection – very shiny objects reflecting in each other, and radiosity models diffuse interaction which is light reflecting off matte surfaces to illuminate other surfaces. Diffuse interaction is common in man-made interiors which tend to have carpets on the floor and matte finishes on the walls. Areas in a room that cannot see the light source are

illuminated by diffuse interaction. Mutually exclusive in the phenomena they model, images created by both methods tend to have identifying 'signatures'. Ray-traced images are notable for perfect recursive reflections and super sharp refraction. Radiosity images are usually of softly-lit interiors and do not contain specular or shiny objects.

Computer graphics is not an exact science. Much research in light-surface interaction in computer graphics proceeds by taking existing physical models and simulating then with a computer graphics algorithm. This may involve much simplification in the original mathematical model so that it can be implemented as a computer graphics algorithm. Ray tracing and radiosity are classic examples of this tendency. Simplifications, which may appear gross to a mathematician, are made by computer graphicists for practical reasons. The reason this process 'works' is that when we look at a synthesized scene we do not generally perceive the simplifications in the mathematics unless they result in visible degeneracies known as aliases. However, most people can easily distinguish a computer graphics image from a photograph. Thus computer graphics have a 'realism' of their own that is a function of the model, and the nearness of the computer graphics image to a photograph of a real scene varies widely according to the method. Photo-realism in computer graphics means the image *looks* real not that it approaches, on a pixel by pixel basis, a photograph. This subjective judgement of computer graphics images somewhat devalues the widely used adjective 'photo-realistic', but there you are. With one or two exceptions very little work has been done on comparing a human's perception of a computer graphics image with, say, a TV image of the equivalent real scene.

a Opalach; Figure
workstations and
GmbH & Co. KG
with the permis-
s, 15:3, July 1996

ght material, in a
nity to offer our
ave unwittingly

There's
is not going to
nc.

nc.
disple
nity's
and ly
the hap
the the
the were
of a confi

1

Mathematical fundamentals of computer graphics

- 1.1 Manipulating three-dimensional structures
- 1.2 Structure-deforming transformations
- 1.3 Vectors and computer graphics
- 1.4 Rays and computer graphics
- 1.5 Interpolating properties in the image plane

1.1

Manipulating three-dimensional structures

Transformations are important tools in generating three-dimensional scenes. They are used to move objects around in an environment, and also to construct a two-dimensional view of the environment for a display surface. This chapter deals with basic three-dimensional transformations, and introduces some useful shape-changing transformations and basic three-dimensional geometry that we will be using later in the text.

In computer graphics the most popular method for representing an object is the polygon mesh model. This representation is fully described in Chapter 2. We do this by representing the surface of an object as a set of connected planar polygons and each polygon is a list of (connected) points. This form of representation is either exact or an approximation depending on the nature of the object. A cube, for example, can be represented exactly by six squares. A cylinder, on the other hand can only be approximated by polygons; say six rectangles for the curved surface and two hexagons for the end faces. The number of polygons used in the approximation determines how accurately the object is represented and this has repercussions in modelling cost, storage and rendering cost and quality. The popularity of the polygon mesh modelling technique in computer graphics is undoubtedly due to its inherent simplicity and the development of inexpensive shading algorithms that work with such models.

A polygon mesh model consists of a structure of vertices, each vertex being a three-dimensional point in so-called world coordinate space. Later we will be concerned with how vertices are connected to form polygons and how polygons are structured into complete objects. But to start with we shall consider objects just as a set of three-dimensional vertices and look at how these are transformed in three-dimensional space using linear transformations.

1.1.1

Three-dimensional geometry in computer graphics – affine transformations

In this section we look at three-dimensional affine transformations. These are the transformations that effect rotation, scaling, shear and translation. Affine transformations can be represented by a matrix, and a set of affine transformations can be combined into a single overall affine transformation. Technically we say that an affine transformation is made up of any combination of linear transformations (rotation, scaling and shear) followed by translation.

Objects are defined in a world coordinate system which is conventionally a right-handed system. A right-handed and left-handed three-dimensional coordinate system is shown in Figure 1.1. Right-handed systems are the standard mathematical convention, although left-handed systems have, and still are, used in the special context of viewing systems in computer graphics. The difference between the two systems is the sense of the z axis as shown in the figure. Rotating your fingers around the z axis, from the positive x axis to the positive y axis, gives a different z direction for your thumb depending on which system is used.

It is sometimes convenient to define objects in their own local coordinate system. There are three reasons for this. When a three-dimensional object is modelled it is useful to build up the vertices with respect to some reference point in the object. In fact a complex object may have a number of local coordinate systems, one for each sub-part. It may be that the same object is to appear many times in a scene and a definition with a local origin is the only sensible way to

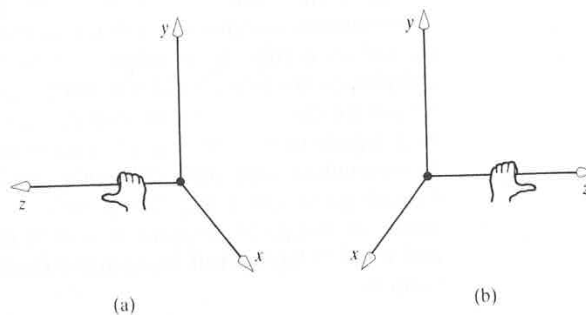


Figure 1.1
(a) Right-handed and
(b) left-handed coordinate
systems.

set this up. Instanting an object by applying a mix of translations, rotation and scaling transformations can then be seen as transforming the local coordinate system of each object to the world coordinate system. Finally when an object is to be rotated, it is easier if the rotation is defined with respect to a local reference point such as an axis of symmetry.

A set of vertices or three-dimensional points belonging to an object can be transformed into another set of points by a linear transformation. Both sets of points remain in the same coordinate system. Matrix notation is used in computer graphics to describe the transformations and the convention in computer graphics is to have the point or vector as a column matrix, preceded by the transformation matrix T .

Using matrix notation, a point V is transformed under translation, scaling and rotation as:

$$V' = V + D$$

$$V' = S V$$

$$V' = R V$$

where D is a translation vector and S and R are scaling and rotation matrices.

These three operations are the most commonly used transformations in computer graphics. In animation a rigid body can undergo only rotation and translation, and scaling is used in object modelling. To enable the above transformations to be treated in the same way and combined, we use a system called homogeneous coordinates which increase the dimensionality of the space. The practical reason for this in computer graphics is to enable us to include translation as matrix multiplication (rather than addition) and thus have a unified scheme for linear transformations. In a homogeneous system a vertex:

$$V(x, y, z)$$

is represented as

$$V(w \cdot X, w \cdot Y, w \cdot Z, w)$$

for any scale factor $w \neq 0$. The three-dimensional Cartesian coordinate representation is then:

$$x = X/w$$

$$y = Y/w$$

$$z = Z/w$$

If we consider w to have the value 1 then the matrix representation of a point is:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Translation can now be treated as matrix multiplication, like the other two transformations and becomes:

$$\mathbf{V}' = \mathbf{T} \mathbf{V}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

This specification implies that the object is translated in three dimensions by applying a displacement T_x , T_y and T_z to each vertex that defines the object. The matrix notation is a convenient and elegant way of writing the transformation as a set of three equations:

$$x' = x + T_x$$

$$y' = y + T_y$$

$$z' = z + T_z$$

The set of transformations is completed by scaling and rotation. First scaling:

$$\mathbf{V}' = \mathbf{S} \mathbf{V}$$

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here S_x , S_y and S_z are scaling factors. For uniform scaling $S_x = S_y = S_z$, otherwise scaling occurs along these axes for which the scaling factor is non-unity. Again the process can be expressed less succinctly by a set of three equations:

$$x' = x \cdot S_x$$

$$y' = y \cdot S_y$$

$$z' = z \cdot S_z$$

applied to every vertex in the object.

To rotate an object in three-dimensional space we need to specify an axis of rotation. This can have any spatial orientation in three-dimensional space, but it is easiest to consider rotations that are parallel to one of the coordinate axes. The transformation matrices for anti-clockwise (looking along each axis towards the origin) rotation about the x , y and z axes respectively are:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

other two trans-

$$\mathbf{R}_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dimensions by
the object. The
transformation

The z axis matrix specification is equivalent to the following set of three equations:

$$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \\ z' &= z \end{aligned}$$

Figure 1.2 shows examples of these transformations.

The inverse of these transformations is often required. $\mathbf{T}^{-1}$ is obtained by negating T_x, T_y and T_z . Replacing S_x, S_y and S_z by their reciprocals gives $\mathbf{S}^{-1}$ and negating the angle of rotation gives $\mathbf{R}^{-1}$.

Any set of rotations, scaling and translations can be multiplied or concatenated together to give a net transformation matrix. For example if:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{M}_1 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \mathbf{M}_2 \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

then the transformation matrices can be concatenated:

$$\mathbf{M}_3 = \mathbf{M}_2 \mathbf{M}_1$$

and

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \mathbf{M}_3 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

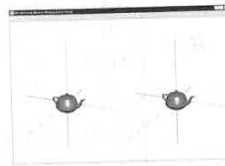
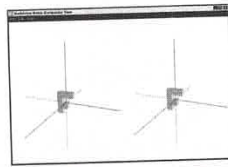
Note the order: in the product $\mathbf{M}_2 \mathbf{M}_1$ the first transformation to be applied is $\mathbf{M}_1$. Although translations are commutative, rotations are not and

$$\mathbf{R}_1 \mathbf{R}_2 \neq \mathbf{R}_2 \mathbf{R}_1$$

This is demonstrated in Figure 1.2(e) and 1.2(f).

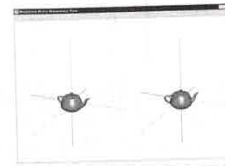
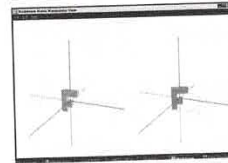
(a) Identity

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



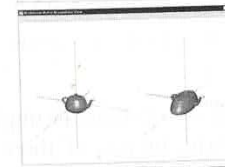
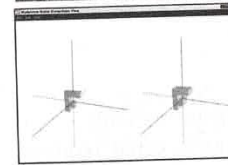
(b) Z-axis rotation

$$\begin{bmatrix} 0.866 & 0.5 & 0 & 0 \\ -0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



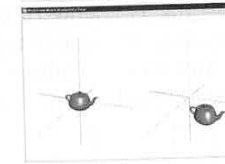
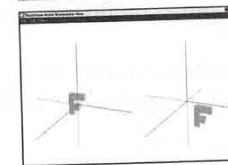
(c) X-scale

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



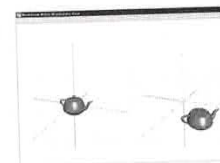
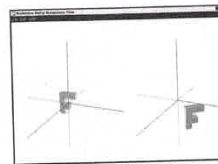
(d) Translation

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(e) Rotation followed by translation

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & 0.5 & 0 & 0 \\ -0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 & 0 & 2 \\ -0.5 & 0.866 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(f) Translation followed by rotation

$$\begin{bmatrix} 0.866 & 0.5 & 0 & 0 \\ -0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 & 0 & 2.732 \\ -0.5 & 0.866 & 0 & 0.732 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

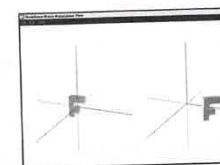
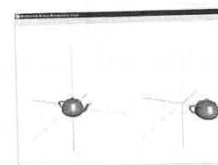


Figure 1.2
Examples of linear
transformations.

A general transformation matrix will be of the form:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & T_x \\ A_{21} & A_{22} & A_{23} & T_y \\ A_{31} & A_{32} & A_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The 3×3 upper-left sub-matrix **A** is the net rotation and scaling while **T** gives the net translation.

The ability to concatenate transformations to form a net transformation matrix is useful because it gives a single matrix specification for any linear transformation. For example, consider rotating a body about a line parallel to the z axis which passes through the point $(T_x, T_y, 0)$ and also passes through one of the vertices of the object. Here we are implying that the object is not at the origin and we wish to apply rotation about a reference point in the object itself. In other words we want to rotate the object with respect to its own coordinate system known as a local coordinate system (see also Section 1.1.2). We cannot simply apply a rotation matrix because this is defined with respect to the origin and an object not positioned at the origin would rotate and translate – not usually the desired effect. Instead we have to derive a net transformation matrix as follows:

- (1) Translate the object to the origin,
- (2) Apply the desired rotation, and,
- (3) Translate the object back to its original position.

The net transformation matrix is:

$$\begin{aligned} T_z R T_1 &= \begin{bmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta & 0 & (-T_x \cos \theta + T_y \sin \theta + T_x) \\ \sin \theta & \cos \theta & 0 & (-T_x \sin \theta - T_y \cos \theta + T_y) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

This process is shown in Figure 1.3 where θ is 30° .

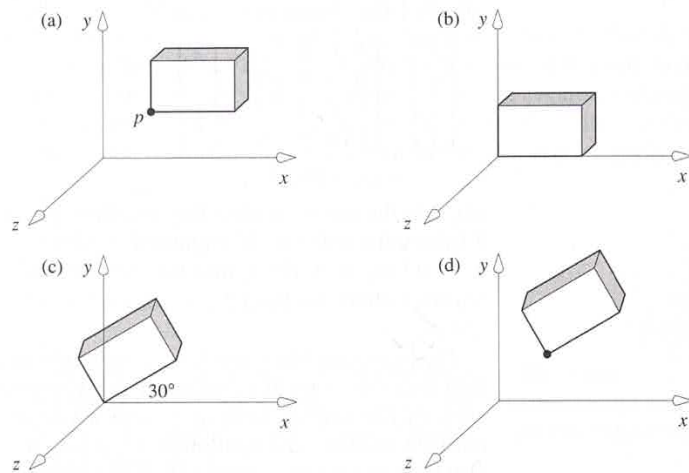


Figure 1.3
Two stages in building up
the rotation of an object
about one of its own
vertices. The rotation is
about an axis parallel to the
 z axis at point $(T_x, T_y, 0)$. A
two-dimensional projection
(with the z axis coming out
of the paper) is shown for
clarity. (a) Original object at
 $(T_x, T_y, 0)$. (b) Translate to
the origin. (c) Rotate about
the origin. (d) Translate to
at $(T_x, T_y, 0)$.

scaling while T gives

5

The graphics pipeline (1): geometric operations

5.1 Coordinate spaces in the graphics pipeline

5.2 Operations carried out in view space

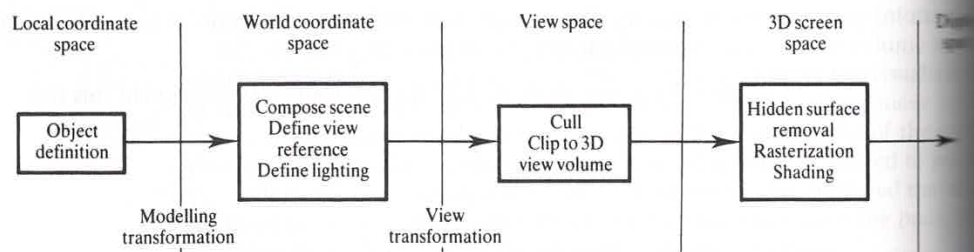
5.3 Advanced viewing systems (PHIGS and GKS)

Introduction

The purpose of a graphics pipeline is to take a description of a scene in three-dimensional space and map it into a two-dimensional projection on the view surface – the monitor screen. Although various three-dimensional display devices exist, most computer graphics view surfaces are two-dimensional, the most common being the TV-type monitor. In most VR applications a pair of projections is produced and displayed on small monitors encased in a helmet – a head-mounted display, or HMD. The only difference in this case is that we now have a pair of two-dimensional projections instead of one – the operations remain the same.

In what follows we will consider the case of polygon mesh models. We can loosely classify the various processes involved in the graphics pipeline by putting them into one of two categories – geometric (this chapter) and algorithmic (Chapter 6). Geometric processes involve operations on the vertices of polygons – transforming the vertices from one coordinate space to another or discarding

Figure 5.1
A three-dimensional rendering pipeline.



polygons that cannot be seen from the view point, for example. Rendering processes involve operations like shading and texture mapping and are considerably more costly than the geometric operations most of which involve matrix multiplication.

A diagram representing the consecutive process in a graphics pipeline is shown in Figure 5.1. From this it can be seen that the overall process is a progression through various three-dimensional spaces – we transform the object representation from space to space. In the final space, which we have called screen space, the rendering operations take place. This space is also three-dimensional for reasons that we will shortly discover.

Coordinate spaces in the graphics pipeline

Local or modelling coordinate systems

For ease of modelling it makes sense to store the vertices of a polygon mesh object with respect to some point located in or near the object. For example, we would almost certainly want to locate the origin of a cube at one of the cube vertices, or we would want to make the axis of symmetry of an object generated as a solid of revolution, coincident with the z axis. As well as storing the polygon vertices in a coordinate system that is local to the object, we would also store the polygon normal and the vertex normals. When local transformations are applied to the vertices of an object, the corresponding transformations are applied to the associated normals.

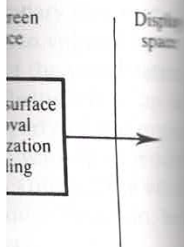
World coordinate systems

Once an object has been modelled the next stage is to place it in the scene that we wish to render. All objects that together constitute a scene have their separate local coordinate systems. The global coordinate system of the scene is known as the 'world coordinate system'. All objects have to be transformed into this common space in order that their relative spatial relationships may be defined. The act of placing an object in a scene defines the transformation required to take the object from local space to world space. If the object is being animated, then the animation system provides a time-varying transformation that takes the object into world space on a frame by frame basis.

The scene is lit in world space. Light sources are specified, and if the shaders within the renderer function are in world space then this is the final transformation that the normals of the object have to undergo. The surface attributes of an object – texture, colour and so on – are specified and tuned in this space.

Camera or eye or view coordinate system

The eye, camera or view coordinate system is a space that is used to establish viewing parameters (view point, viewing direction) and a view volume. (A



virtual camera is often used as the analogy in viewing systems, but if such an allusion is made we must be careful to distinguish between external camera parameters – its position and the direction it is pointing in – and internal camera parameters or those that affect the nature and size of the image on the film plane. Most rendering systems imitate a camera which in practice would be a perfect pinhole (or lensless) device with a film plane that can be positioned at any distance with respect to the pinhole. However, there are other facilities in computer graphics that cannot be imitated by a camera and because of this the analogy is of limited utility.)

We will now deal with a basic view coordinate system and the transformation from world space to view coordinate space. The reasons that this space exists after all we could go directly from world space to screen space, is that certain operations (and specifications) are most conveniently carried out in view space. Standard viewing systems like that defined in the PHIGS graphics standard are more complicated in the sense that they allow the user to specify more facilities and we will deal with these in Section 5.3.

We define a viewing system as being the combination of a view coordinate system together with the specification of certain facilities such as a view volume. The simplest or minimum system would consist of the following:

- A view point which establishes the viewer's position in world space; this may either be the origin of the view coordinate system or the centre of projection together with a view direction $\mathbf{N}$.
- A view coordinate system defined with respect to the view point.
- A view plane onto which the two-dimensional image of the scene is projected.
- A view frustum or volume which defines the field of view.

These entities are shown in Figure 5.2. The view coordinate system, UVN , has $\mathbf{N}$ coincident with the viewing direction and $\mathbf{V}$ and $\mathbf{U}$ lying in a plane parallel to the view plane. We can consider the origin of the system to be the view point. The view plane containing $\mathbf{U}$ and $\mathbf{V}$ is of infinite extent and we specify a view volume or frustum which defines a window in the view plane. It is the content of this window – the projection of that part of the scene that is contained within the view volume – that finally appears on the screen.

Thus, using the virtual camera analogue we have a camera that can be positioned anywhere in world coordinate space, pointed in any direction and rotated about the viewing direction $\mathbf{N}$.

To transform points in world coordinate space we invoke a change of coordinate system transformation and this splits into two components: a translation one and a rotational one (see Chapter 1). Thus:

$$\begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} = \mathbf{T}_{\text{view}} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

ng systems, but these
between external camera
g in - and internal
of the image on the
ich in practice would
: that can be produced
there are other facilities
era and because of this

em and the transform
ons that this space
reen space, is that
carried out in view
IGS graphics standard
er to specify more

tion of a view coordi
ies such as a view
e following:

on in world space
or the centre of projection

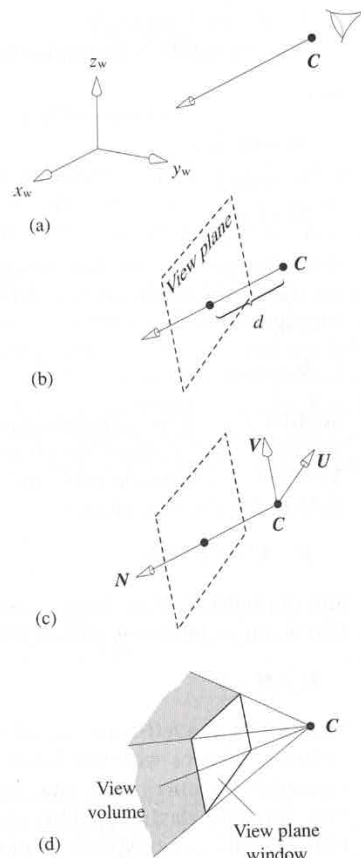
the view point.
l image of the scene

of view.

nate system, CTV , and
ing in a plane parallel
m to be the view point
nt and we specify
v plane. It is the centre
: that is contained within

a camera that can be
ed in any direction

voke a change of coordi
ponents: a translation



where:

$$T_{\text{view}} = RT$$

and:

$$T = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} U_x & U_y & U_z & 0 \\ V_x & V_y & V_z & 0 \\ N_x & N_y & N_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The only problem now is specifying a user interface for the system and mapping whatever parameters are used by the interface into U , V and N . A user needs to specify C , N and V . C is easy enough. N , the viewing direction or view plane normal, can be entered say, using two angles in a spherical coordinate system - this seems reasonably intuitive:

θ the azimuth angle
 ϕ the colatitude or elevation angle

where:

$$N_x = \sin \phi \cos \theta$$

$$N_y = \sin \phi \sin \theta$$

$$N_z = \cos \phi$$

V is more problematic. For example, a user may require 'up' to be the same sense as 'up' in the world coordinate system. However, this cannot be achieved by setting:

$$V = (0, 0, 1)$$

because V must be perpendicular to N . A sensible strategy is to allow a user to specify an approximate orientation for V , say V' and have the system calculate V . Figure 5.3 demonstrates this. V' is the user-specified up vector. This is projected onto the view plane:

$$V = V' - (V' \cdot N)N$$

and normalized. U can be specified or not depending on the user's requirements. If U is unspecified, it is obtained from:

$$U = N \times V$$

This results in a left-hand coordinate system, which although somewhat inconsistent, conforms with our intuition of a practical viewing system, which has increasing distances from the view point as increasing values along the view direction axis. Having established the viewing transformation using UVN notation, we will in subsequent sections use (x_v, y_v, z_v) to specify points in the view coordinate system.

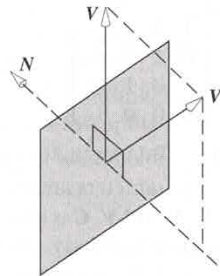


Figure 5.3
 The up vector V can be calculated from an 'indication' given by V' .

52

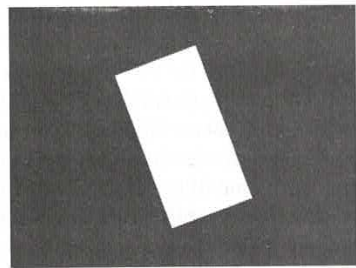
521

522

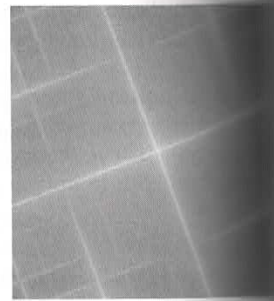
Figure 5.4
 Culling and hidden space removal. (a) Culling removes complete polygons that cannot be seen. (b) Hidden surface removal deals with the general problem: polygons will partially obscure others.

Figure 14.5

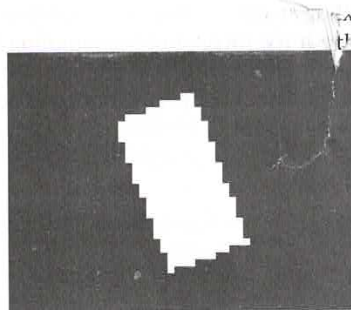
The effect of jaggies is to rotate high energy components onto the horizontal and vertical axes in the Fourier domain.



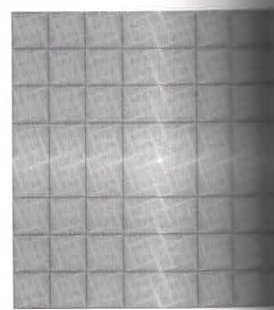
(a) Simulation of a perfect line



(b) Fourier transform of (a)



(c) Simulation of a jagged line



(d) Fourier transform of (c)

14.3**Sampling in computer graphics compared with sampling reality**

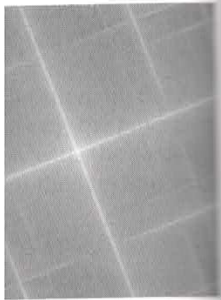
Let us now return in more detail to the notion of sampling in the image plane. In image synthesis what we are doing is performing, for each pixel, a number of (sometimes very complicated) operations that eventually calculate, for the pixel, a constant value. Usually we calculate a value at the centre of the pixel and 'spread' that value over the pixel extent.

We assume that, in principle, this is no different from having a continuous image in the view plane and sampling this with a discrete two-dimensional array of sample points (one for each pixel). We say that this assumption is valid because we can approach such an image by increasing and increasing the sample resolution and calculating a value for the image at more and more points in the image plane. However, it is important to bear in mind that we do not have access to a continuous image in computer graphics and this limits and conditions our approaches to anti-aliasing measures.

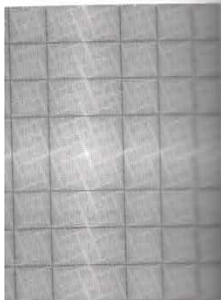
In fact, both the terms 'sampling' and 'reconstruction' – another term borrowed from digital signal processing – are used indiscriminately and, we feel, somewhat confusingly in computer graphics, and we will now emphasize the

Figure 14.6

Sampling, reconstruction and anti-aliasing in image processing and image synthesis. (a) Image capture and processing. (b) Image synthesis. (c) Anti-aliasing image capture.



Fourier transform of (a)



Fourier transform of (c)

sampling reality

ing in the image plane. Each pixel, a number of samples are calculated, for the centre of the pixel and

m having a continuous two-dimensional area. This assumption is valid and increasing the sample rate and more points in the image will now emphasize the

rection' – another term. Primarily and, we feel, will now emphasize the

Figure 14.6 Sampling, reconstruction and anti-aliasing in image processing and image synthesis. (a) Image capture and processing. (b) Image synthesis. (c) Anti-aliasing in image capture.

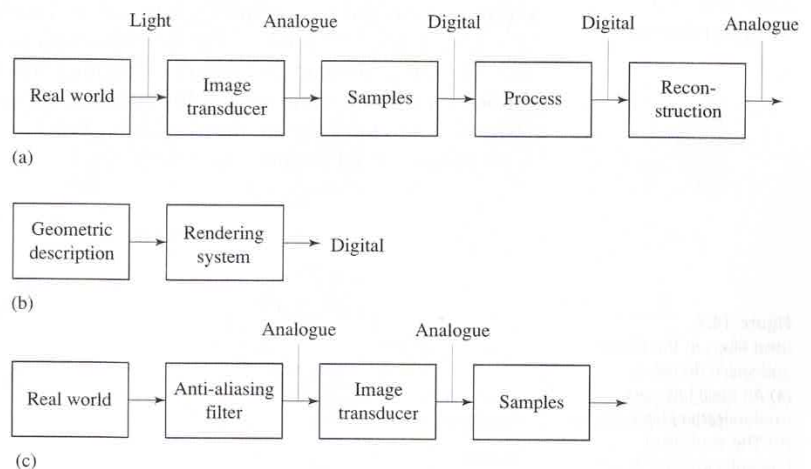
difference between an image processing system, where their usage is wholly appropriate, and their somewhat artificial use in computer graphics.

Consider Figure 14.6 which shows a schematic diagram for an image processor and a computer graphics system. In the image processor a sampler converts a two-dimensional continuous image into an array of samples. Some operations are then performed on the digital image and a reconstruction filter converts the processed samples back into an analogue signal.

Not so in image synthesis. Sampling does not exist in the same sense – the operations involved in assigning a value to a pixel depend on the rendering algorithm used and we can only ever calculate the value of an image function at these points.

Reconstruction, in image synthesis does not mean generating a continuous image from a digital one but may mean, for example, generating a low (pixel) resolution image from an image stored at a higher (undisplayable) resolution. We are not reconstructing an image since a continuous image never existed in the first place. An appreciation of these differences will avoid confusion. (In reality we do reconstruct a continuous image for display on a computer graphics monitor, but this is done by fixed electronics that operate on the image produced in the framestore by a graphics program. A comprehensive approach to anti-aliasing would need to take the transfer characteristics of the conversion electronics into account but we will not do so in this text.)

To return to the problem of aliasing artefacts. Fourier theory tells us that aliasing occurs because we sample a continuous image (or the equivalent operation in computer graphics) and we do not do this at a high enough resolution to capture the high spatial frequencies or detail in the image. The sampling theorem states that if we wish to sample an image function without loss of information then our (two-dimensional) sampling frequency must be at least twice as high as the highest frequency component in the image.



So what does this mean in terms of practical computer graphics? Just this: if we consider we are sampling a continuous image in the view plane with a grid of square pixels, then the highest frequency that can appear along a scan line is

$$f = 1/2d$$

where d is the distance between pixel centres.

Having fixed these concepts it is easy to see why anti-aliasing is so difficult in computer graphics. The problem stems from two surprising facts. There is no limit to the value of the high frequencies in computer graphics – we have already discussed this using the example of the infinite chequerboard – and there is no direct way to limit (the technical term is band-limit) these spatial frequencies.

This is easily seen by comparing image synthesis with image capture through a device like a TV camera (Figure 14.6c). Prior to sampling a continuous image we can pass it through a band limiting filter (or an anti-aliasing filter). Higher frequencies that cannot be displayed are simply eliminated from the image before it is sampled. We say that the image is pre-filtered. In such systems aliasing problems are simply not allowed to occur.

In image synthesis our scene database exists as a mathematical description or as a set of points connected by edges. Our notion of sampling is inextricably entwined with rendering. We sample by evaluating the projection of the scene at discrete points. We cannot band limit the image because no image exists – we can only define its existence at the chosen points.

14.4

Sampling and reconstruction

In Figure 14.3 we saw that provided the sampling theory is obeyed then reconstruction of the information from the samples is obtained by using a reconstruction filter in the shape of a box. However, this is a Fourier domain representation and in computer graphics all our operations have to take place in the space or image domain. Therefore the reconstruction process is convolution in the space or image domain. In computer graphics this implies (usually) filtering a rendered image in some way. If the rendered image was continuous then our reconstruction filter would consist of a sinc function $h(x, y)$ – which is the transform of the Fourier domain equivalent of a circular step function (Figure 14.7).

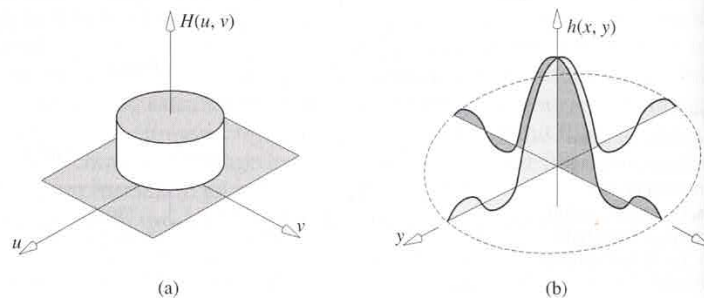


Figure 14.7
Ideal filters in the Fourier and space domains.
(a) An ideal low pass (multiplicative) filter $H(u, v)$.
(b) The equivalent (convolving) filter $h(x, y)$.

Figure 14.8
A comparison of four approaches to calculating a single value for a pixel.

l green components

es unchanged and
curs either because
naccurate value of
a wrong image with

16

Image-based rendering and photo-modelling

- 16.1 Reuse of previously rendered imagery – two-dimensional techniques
- 16.2 Varying rendering resources
- 16.3 Using depth information
- 16.4 View interpolation
- 16.5 Four-dimensional techniques – the Lumigraph or light field rendering approach
- 16.6 Photo-modelling and IBR

Introduction

A new field with many diverse approaches, image-based rendering (IBR) is difficult to categorize. The motivation for the name is that most of the techniques are based on two-dimensional imagery, but this is not always the case and the way in which the imagery is used varies widely amongst methods. A more accurate common thread that runs through all the methods is pre-calculation. All methods make cost gains by pre-calculating a representation of the scene from which images are derived at run-time. IBR has mostly been studied for the common case of static scenes and a moving view point, but applications for dynamic scenes have been developed.

There is, however, no debate concerning the goal of IBR which is to decouple rendering time from scene complexity so that the quality of imagery, for a given frame time constraint, in applications like computer games and virtual reality can be improved over conventionally rendered scenes where all the geometry is reinserted into the graphics pipeline whenever a change is made to the view point. It has emerged, simultaneously with LOD approaches (see Chapter 2) and scene management techniques, as an effective means of tackling the dependency of rendering time on scene complexity.

We will also deal with photo-modelling in this chapter. This is related to image-based rendering because many image-based rendering schemes were designed to operate with photo-modelling. The idea of photo-modelling is to capture the real-world complexity and at the same time retain the flexibility advantages of three-dimensional graphics.

16.1

Reuse of previously rendered imagery – two-dimensional techniques

We begin by considering methods that rely on the concept of frame coherence and reuse of already rendered imagery in some way. Also, as the title of the section implies, we are going to consider techniques that are essentially two-dimensional. Although the general topic of image-based rendering, of course, itself implies two-dimensional techniques there has been some use of the depth information associated with the image, as we shall see in future sections. The distinction is that with techniques which we categorize as two-dimensional we do not operate with detailed depth values, for example, a value per pixel. We may only have a single depth value associated with the image entity as is implied by visibility ordering in image layers (see Section 16.2.2).

A useful model of an image-based renderer is to consider a required image being generated from a source or reference image – rendered in the normal way – by warping the reference image in image space (Figure 16.1). In this section we shall consider simple techniques based on texture mapping that can exploit the hardware facilities available on current 3D graphics cards. The novel approach here is that we consider rendered objects in the scene as texture maps, consider a texture map as a three-dimensional entity and pass it through the graphics pipeline. The common application of such techniques is in systems where a viewer moves through a static environment.

To a greater or lesser extent all such techniques involve some approximation compared with the projections that are computed using conventional techniques and an important part of such methods is determining when it is valid to reuse previously generated imagery and when new images must be generated.

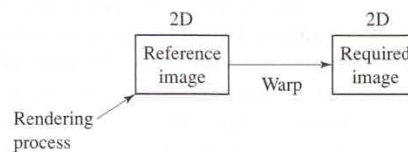


Figure 16.1
Planar imposters and image warping.

IBR as a process that produces an image by warping a reference image

16.1.1

16.1.2

16.3

Using depth information

16.3.1

Three-dimensional warping

As we have already mentioned, the main disadvantage of planar sprites is that they cannot produce motion parallax and they produce a warp that is constrained by a threshold beyond which their planar nature is perceived.

We now come to consider the use of depth information which is, of course, readily available in synthetic imagery. Although the techniques are now going to use the third dimension we still regard them as image-based techniques in the sense that we are still going to use, as source or reference, rendered images albeit augmented with depth information.

Consider, first, what depth information we might employ. The three commonest forms in order of their storage requirements are: using layers or sprites with depth information (previous section), using a complete (unsegmented) image with the associated Z-buffer (in other words one depth value per pixel) and a layered depth image or LDI. An LDI is a single view of a scene with multiple pixels along each line of sight. The amount of storage that LDIs require is a function of the depth complexity of the average number of surfaces that project onto a pixel.

We begin by considering images complete with depth information per pixel – the normal state of affairs for conventionally synthesized imagery. It is intuitively obvious that we should be able to generate or extrapolate an image at a new view point from the reference image providing that the new view point is close to the reference view point. We can define the pixel motion in image space as the warp:

$$I(x, y) \rightarrow I(x', y')$$

which implies a reference pixel will move to a new destination. (This is a simple statement of the problem which ignores important practical problems that we shall address later.) If we assume that the change in the view point is specified by a rotation $\mathbf{R} = [r_{ij}]$ followed by a translation $\mathbf{T} = (\Delta x, \Delta y, \Delta z)^T$ of the view coordinate system (in world coordinate space) and that the internal parameters of the viewing system/camera do not change – the focal length is set to unity – then the warp is specified by:

$$\begin{aligned} x' &= \frac{(r_{11}x + r_{12}y + r_{13})Z(x, y) + \Delta x}{(r_{31}x + r_{32}y + r_{33})Z(x, y) + \Delta z} \\ y' &= \frac{(r_{21}x + r_{22}y + r_{23})Z(x, y) + \Delta y}{(r_{31}x + r_{32}y + r_{33})Z(x, y) + \Delta z} \end{aligned} \quad [16.1]$$

where:

$Z(x, y)$ is the depth of the point $\mathbf{P}$ of which (x, y) is the projection.

Figure 16.7

A three-dimensional warp is calculated from rotation $\mathbf{R}$ and translation $\mathbf{T}$ applied to the view coordinate system

This follows from:

$$x' = \frac{x_v}{z_v} \quad y' = \frac{y_v}{z_v}$$

where (x_v, y_v, z_v) are the coordinates of the point P in the new viewing system. A visualization of this process is shown in Figure 16.7.

We now consider the problems that occur with this process. The first is called image folding or topological folding and occurs when more than one pixel in the reference image maps into position (x', y') in the extrapolated image (Figure 16.8(a)). The straightforward way to resolve this problem is to calculate $Z(x', y')$ from $Z(x, y)$ but this requires an additional rational expression and an extra Z-buffer to store the results.

McMillan (1995) has developed an algorithm that specifies a unique evaluation order for computing the warp function such that surfaces are drawn in a back-to-front order thus enabling a simple painter's algorithm to resolve this visibility problem. The intuitive justification for this algorithm can be seen by considering a simple special case shown in Figure 16.9. In this case the view point has moved to the left so that its projection in the image plane of the reference view coordinate system is outside and to the left of the reference view window. This fact tells us that the order in which we need to access pixels in the reference image is from right to left. This then resolves the problem of the leftmost pixel in the reference image overwriting the right pixel in the warped image. McMillan shows that the accessing or enumeration order of the reference image can be reduced to nine cases depending on the position of the projection of the new view point in the reference coordinate system. These are shown in Figure 16.10. The general case, where the new view point stays within the reference view window divides the image into quadrants. An algorithm structure that utilizes this method to resolve depth problems in the many-to-one case is thus:

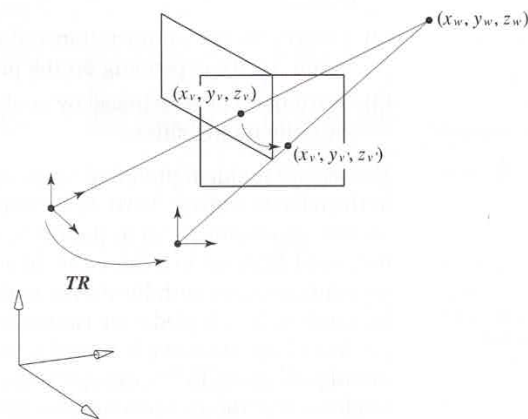
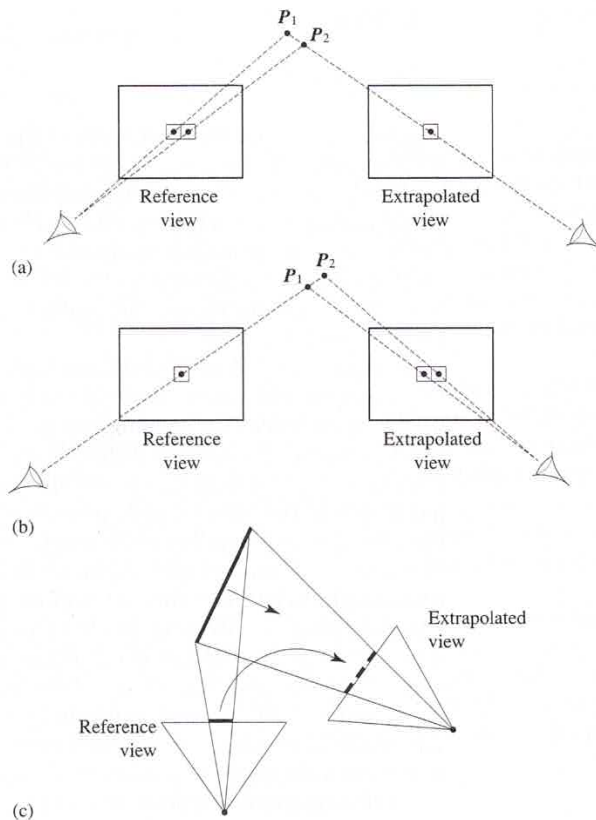


Figure 16.7
A three-dimensional warp is
calculated from rotation R
and translation T applied to
the view coordinate system.

Figure 16.8

Problems in image warping.

- (a) Image folding: more than one pixel in the reference view maps into a single pixel in the extrapolated view. (b) Holes: information occluded in the reference view is required in the extrapolated view. (c) Holes: the projected area of a surface increases in the extrapolated view because its normal rotates towards the viewing direction. (d) See Colour Plate section.



- (1) Calculate the projection of the new view point in the reference coordinate system.
- (2) Determine the enumeration order (one out of the nine cases shown in Figure 16.10) depending on the projected point.
- (3) Warp the reference image by applying Equation 16.1 and writing the result into the frame buffer.

The second problem produced by image warping is caused when occluded area in the reference image 'need' to become visible in the extrapolated image (Figure 16.8(b)) producing holes in the extrapolated image. As the figure demonstrates, holes and folds are in a sense the inverse of each other, but where a deterministic solution exists for folds no theoretical solution exists for holes and a heuristic needs to be adopted – we cannot recover information that was not there in the first place. However, it is easy to detect where holes occur. They are simply unassigned pixels in the extrapolated image and this enables the problem to be localized and the most common solution is to fill them in with colours from

Figure 16.9

The view point translates to the left so that the projection of the new view point in the image plane of the reference view coordinate system is to the left of the reference view window. The correct processing order of the reference pixels is from right to left.

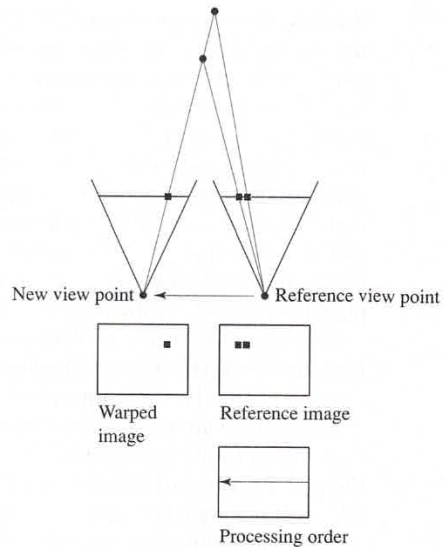
Figure 16.9 shows a reference view and an extrapolated view. The view point of the extrapolated view is to the left of the reference view window. The projection of the new view point in the image plane of the reference view coordinate system is to the left of the reference view window. The correct processing order of the reference pixels is from right to left.

Figure 16.10 shows a visualization of McMillan's priority algorithm indicating the correct processing order as a function of view point motion for nine cases (after McMillan (1995)).

Figure 16.10

A visualization of McMillan's priority algorithm indicating the correct processing order as a function of view point motion for nine cases (after McMillan (1995)).

Figure 16.9
The view point translates to the left so that the projection of the new view point in the image plane of the reference view coordinate system is to the left of the reference view window. The correct processing order of the reference pixels is from right to left.



neighbouring pixels. The extent of the holes problem depends on the difference between the reference and extrapolated view points and it can be ameliorated by considering more than one reference image, calculating an extrapolated image from each and compositing the result. Clearly if a sufficient number of reference images are used then the hole problem will be eliminated and there is no need for a local solution which may insert erroneous information.

A more subtle reason for unassigned pixels in the extrapolated image is apparent if we consider surfaces whose normal rotates towards the view direction in

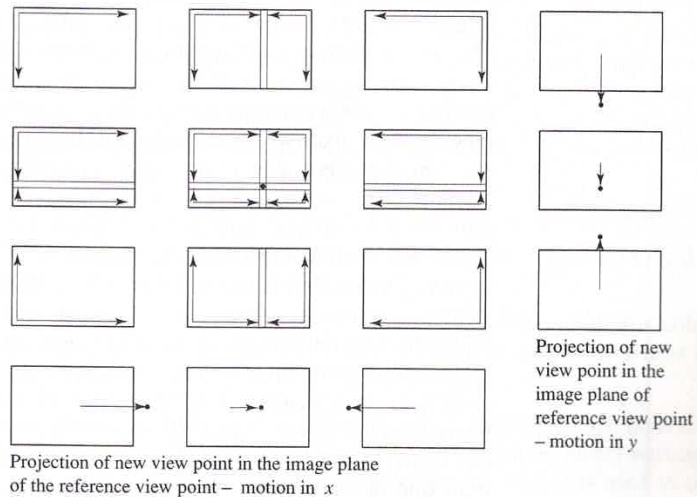


Figure 16.10
A visualization of McMillan's priority algorithm indicating the correct processing order as a function of view point motion for nine cases (after McMillan (1995)).

the new view system (Figure 16.8(c)). The projected area of such a surface in the extrapolated image plane will be greater than its projection in the reference image plane and for a one-to-one forward mapping holes will be produced. This suggests that we must take a rigorous approach to reconstruction in the interpolated image. Mark *et al.* (1997) suggest calculating the appropriate dimension of a reconstruction kernel, for each reference pixel as a function of the view point motion but they point out that this leads to a cost per pixel that is greater than the warp cost. This metric is commonly known as splat size (Chapter 13) and its calculation is not straightforward for a single reference image with a depth only for visible pixels. (A method that stores multiple depth values for a pixel is dealt with in the next section.)

The effects of these problems on an image are shown in Figure 16.8(d) (Colour Plate). The first two images show a simple scene and the corresponding Z-buffer image. The next image shows the artefacts due to translation (only). In this case these are holes caused by missing information and image folding. The next image shows artefacts due to rotation (only) – holes caused by increasing the projected area of surfaces. Note how these form coherent patterns. The final image shows artefacts caused by both rotation and translation.

Finally, we note that view-dependent illumination effects will not in general be handled correctly with this simple approach. This, however, is a problem that is more serious in image-based modelling methods (Section 16.6). As we have already noted in image warping we must have reference images whose view point is close to the required view point.

16.3.2

Layered depth images (LDIs)

Many of the problems encountered in the previous section disappear if our source imagery is in the form of an LDI (Shade *et al.* 1998). In particular we can resolve the problem of holes where we require information in the extrapolated image in areas occluded in the source or reference image. An LDI is a three-dimensional data structure that relates to a particular view point and which samples, for each pixel, all the surfaces and their depth values intersected by the ray through that pixel (Figure 16.11). (In practice, we require a number of LDIs to represent a scene and so can consider a scene representation to be four-dimensional – or the same dimensionality as the light field in Section 16.5.) Thus, each pixel is associated with an array of information with a number of elements or layers that is determined by the number of surfaces intersected. Each element contains a colour, surface normal and depth for surface. Clearly this representation requires much more storage than an image plus Z-buffer but this requirement grows only linearly with depth complexity.

In their work Shade *et al.* (1998) suggest two methods for pre-calculating LDIs for synthetic imagery. First, they suggest warping n images rendered from different view points into a single view point. During the warping process if more than one pixel maps into a single LDI pixel then the depth values associated

Figure 16.11
Representation of a
layered depth image (LDI).

16.4.1

View morphing

Up to now we have considered techniques that deal with a moving view point and static scenes. In a development that they call view morphing Seitz and Dyer (1996) address the problem of generating in-between images where non-rigid transformations have occurred. They do this by addressing the approximation implicit in the previous section and distinguish between 'valid' and 'non-valid' in-between views.

View interpolation by warping a reference image into an extrapolated image proceeds in two-dimensional image plane space. A warping operation is just that – it changes the shape of the two-dimensional projection of objects. Clearly the interpolation should proceed so that the projected shape of the objects in the reference projection is consistent with their real three-dimensional shape. In other words, the interpolated view must be equivalent to a view that would be generated in the normal way (either using a camera or a conventional graphics pipeline) by changing the view point from the reference view point to that of the interpolated view. A 'non-valid' view means that the interpolated view does not preserve the object shape. If this condition does not hold then the interpolated views will correspond to an object whose shape is distorting in real three-dimensional space. This is exactly what happens in conventional image morphing between two shapes. 'Impossible', non-existent or arbitrary shapes occur in in-between images because the motivation here is to appear to change one object into an entirely different one. The distinction between valid and invalid view interpolation is shown in Figure 16.14.

An example where linear interpolation of images produces valid interpolated views is the case where the image planes remain parallel (Figure 16.15). Physically, this situation would occur if a camera was allowed to move parallel to its image plane (and optionally zoom in and out). If we let the combined viewing and perspective transformations (see Chapter 5) be $\mathbf{V}_0$ and $\mathbf{V}_1$ for the two reference images then the transformation for an in-between image can be obtained by linear interpolation:

$$\mathbf{V}_i = (1 - s) \mathbf{V}_0 + s \mathbf{V}_1$$

If we consider a pair of corresponding points in the reference images $\mathbf{P}_0$ and $\mathbf{P}_1$ which are projections of world space point $\mathbf{P}$, then it is easily shown (see Seitz and Dyer (1996)) that the projection of point $\mathbf{P}$ from the intermediate (interpolated) view point is given by linear interpolation:

$$\begin{aligned} \mathbf{P}_i &= \mathbf{P}_0(1 - s) + \mathbf{P}_1s \\ &= \mathbf{V}_i\mathbf{P} \end{aligned}$$

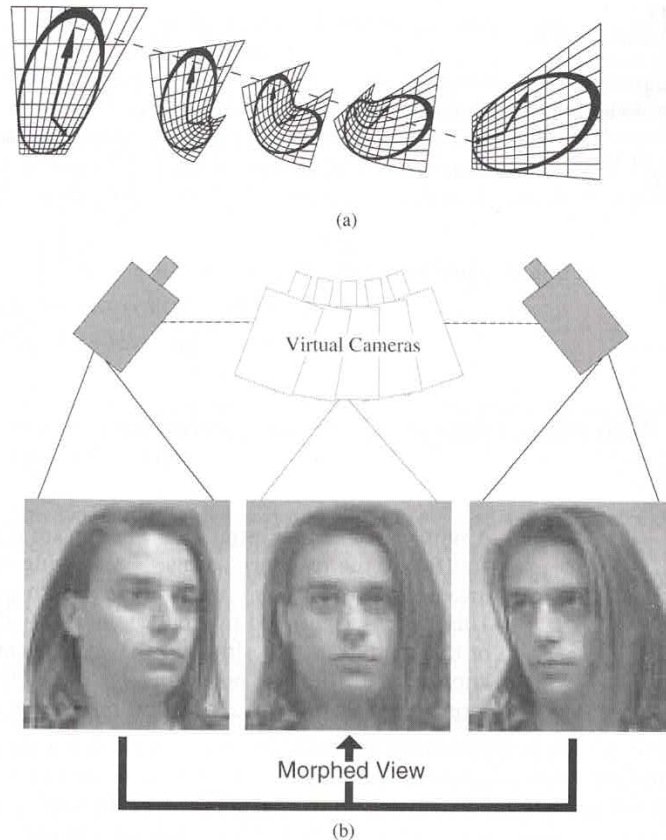
In other words linear interpolation of pixels along a path determined by pixel correspondence in two reference images is exactly equivalent to projecting the scene point that resulted in these pixels through a viewing and projective transformation given by an intermediate camera position, provided parallel views are maintained, in other words using the transformation $\mathbf{V}_i$, which would

Figure 16.14

Distinguishing between valid and invalid view interpolation. In (a), using standard (morphing) approach of linear interpolation produces gross shape deformation. This does not matter if we are morphing between two different objects – it becomes part of the effect. In (b) The interpolated (or warped) view is consistent with object shape. (Courtesy of Steven Seitz.)

Figure 16.14

Distinguishing between valid and invalid view interpolation. In (a), using a standard (morphing) approach of linear interpolation produces gross shape deformation (this does not matter if we are morphing between two different objects – it becomes part of the effect). (b) The interpolated (or morphed) view is consistent with object shape. (Courtesy of Steven Seitz.)



be obtained if $\mathbf{V}_0$ and $\mathbf{V}_1$ were linearly interpolated. Note also that we are interpolating views that would correspond to those obtained if we had moved the camera in a straight line from $\mathbf{C}_0$ to $\mathbf{C}_1$. In other words the interpolated view corresponds to the camera position:

$$\mathbf{C}_i = (sC_x, sC_y, 0)$$

If we have reference views that are not related in this way then the interpolation has to be preceded (and followed) by an extra transformation. This is the general situation where the image planes of the reference views and the image plane of the required or interpolated view have no parallel relationship. The first transformation, which Seitz and Dyer call a 'prewarp', warps the reference images so that they appear to have been taken by a camera moving in a plane parallel to its image plane. The pixel interpolation, or morphing, can then be performed as in the previous paragraph and the result of this is postwarped to form the final interpolated view, which is the view required from the virtual camera position.

Figure 16.15

Moving the camera from C_0 to C_1 (and zooming) means that the image planes remain parallel and P_i can be linearly interpolated from P_0 and P_1 (after Seitz and Dyer (1996)).

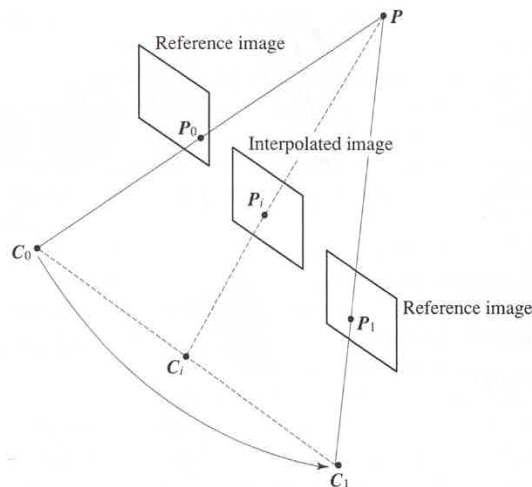
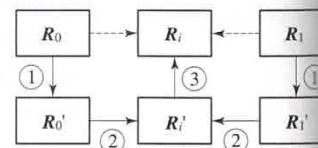
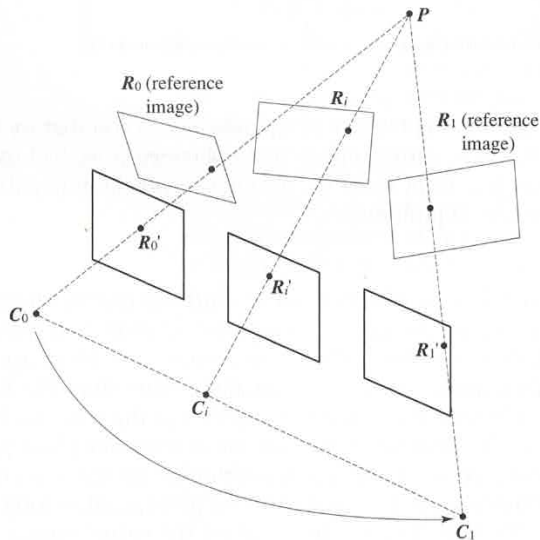


Figure 16.16

Prewarping reference images, interpolating and postwarping in view interpolation (after Seitz and Dyer (1996)).



Three-step transformation sequence

used, implying that in the case of photographic imagery we have to record or recover the camera view points.

The prewarping and postwarping transformations are derived as follows. First, it can be shown that any two perspective views that share the same centre of projection are related by a planar projective transformation – a 3×3 matrix obtained from the combined viewing perspective transformation V . Thus R_0 and R_1 are related to R_0' and R_1' by two such matrices T_0 and T_1 . The procedure is thus as follows:

- (1) Prewarp R_0 and R_1 using T_0^{-1} and T_1^{-1} to produce R_0' and R_1' .
- (2) Interpolate to calculate R_i' , C_i and T_i .
- (3) Apply T_i to R_i' to give image R_i .

16.5

Four-dimensional techniques – the Lumigraph or light field rendering approach

Up to now we have considered systems that have used a single image or a small number of reference images from which a required image is generated. We have looked at two-dimensional techniques and methods where depth information has been used – three-dimensional warping. Some of these methods involve pre-calculation of a special form of rendered imagery (LDIs) others post-process a conventionally rendered image. We now come to a method that is an almost total pre-calculation technique. It is an approach that bears some relationship to environment mapping. An environment map caches all the light rays that arrive at a single point in the scene – the source or reference point for the environment map. By placing an object at that point we can (approximately) determine those light rays that arrive at the surface of the object by indexing into the map. This scheme can be extended so that we store in effect an environment map for every sampled point in the scene. That is, for each point in the scene we have knowledge of all light rays arriving at that point. We can now place an object at any point in the scene and calculate the reflected light. The advantage of this approach is that we now minimize most of the problems related to three-dimensional warping at the cost of storing a vast amount of data.

A light field is a similar approach. For each and every point of a region in the scene in which we wish to reconstruct a view we pre-calculate and store or cache the radiance in every direction at that point. This representation is called a light field or Lumigraph (Levoy and Hanrahan 1996; Gortler *et al.* 1996) and we construct for a region of free space by which is meant a region free of occluders. The importance of free space is that it reduces the light field from a five-dimensional to a four-dimensional function. In general, for every point (x, y, z) in scene space we have light rays travelling in every direction (parametrized by two angles) giving a five-dimensional function. In occluder free space we can assume (unless there is atmospheric interaction) that the radiance along a ray is constant. The two 'free space scenes' of interest to us are: viewing an object from

image

figure 16.16. Here R_0 and R_1 respectively images to produce R_i . Important consequence image based we require and post-warp transformation which the method is

