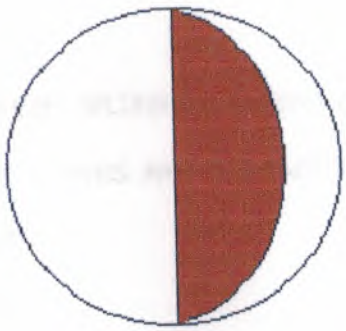
NEAR EAST



1988

UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

COM 400 Graduation Project

DISPLAY OF THREE DIMENSIONAL CURVED LINES AND SURFACES IN COMPUTER GRAPHICS

Submitted to : MEHRDAD KHALEDI

Submit by : Hakan Bozbay 940728

CONTENTS

HAKAN BOZBAY

CONTENTS

- TOPOGRAPHIC SURFACES
- SURFACE INTERSECTING AND CUTTING
- RENDERING
- VISIBLE-SURFACE DETERMINATION
- BASIC SHADING
- SMOOTH SHADING OF CURVED SURFACES
- CAST SHADOWS, TRANSPARENCY, AND REFLECTIONS
- DIFFUSE GLOBAL ILLUMINATION EFFECTS
- SURFACE DETAILS AND TEXTURES
- NATURAL PHENOMENA AND LANDSCAPE COMPOSITION
- RETOUCHING AND PAINTING SHADED IMAGES
- COMBINING SYNTHESIZED AND CAPTURED IMAGES
- OUTPUT AND PRESENTATION TECHNOLOGY
- USES AND LIMITATIONS OF SURFACE MODELING AND RENDERING
- DRAFTED LINES
- COORDINATE SYSTEMS
- POINT SPECIFICATION
- REPERTOIRES OF LINE TYPES
- CHAINS OF LINES
- BASIC OPERATIONS ON LINES
- GEOMETRIC CONSTRUCTIONS
- SELECTING, TRANSFORMING, AND DUPLICATING SUBSHAPES
- REPEATABLE STANDARD SHAPES
- PARAMETRIC VARIATION
- CONSTRAINT SOLVING
- SYNTAX-DIRECTED EDITING

HAKAN BOZBAY

CONTENTS

CONTENTS

CONTENTS

INTRODUCTION

Graphics scenes can contain many different kinds of objects: trees, flowers, clouds, rocks, water, bricks, wood paneling, rubber, paper, marble, steel, glass, plastic, and cloth, just to mention a few. So it is probably not too surprising that there is no one method that we can use to describe objects that will include all characteristics of these different materials. And to produce realistic displays of scenes, we need to use representations that accurately model object characteristics.

Polygon and quadric surfaces provide precise descriptions for simple Euclidean objects such as polyhedrons and ellipsoids; spline surfaces and construction techniques are useful for designing aircraft wings, gears, and other engineering structures with curved surfaces; procedural methods, such as fractal constructions and particle systems, allow us to give accurate representations for clouds, clumps of grass, and other natural objects; physically based modeling methods using systems of interacting forces can be used to describe the nonrigid behavior of a piece of cloth or a glob of jello ; octree encodings are used to represent internal features of objects, such as those obtained from medical CT images; and isosurface displays, volume renderings, and other visualization techniques are applied to three-dimensional discrete data sets to obtain visual representations of the data.

Representation schemes for solid objects are often divided into two broad categories, although not all representations fall neatly into one or the other of these two categories. Boundary representations (B-reps) describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. Typical examples of boundary representations are polygon facets and spline patches. Space-partitioning representations are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, nonoverlapping, contiguous solids (usually cubes). A common space-partitioning description for a three-dimensional object is an octree representation. in this chapter, we consider the features of the various representation schemes and how they are used in applications.

CONVERSION BETWEEN SPLINE REPRESENTATIONS

Sometimes it is desirable to be able to switch from one spline representation to other. For instance, a Bezier representation is the most convenient one for subdividing a spline curve, while a B-spline representation offers greater design flexibility So we might design a curve using B-spline sections, then we can convert to an equivalent Bezier representation to display the object using a recursive subdivision procedure to locate coordinate positions along the curve.

Suppose we have a spline description of an object that can be expressed with the following matrix product:

$$\mathbf{P(U) = U + Mspline1 + Mgeom1}$$

where **Mspline1** is the matrix characterizing the spline representation, and **Mgeom1** is the column matrix of geometric constraints (for example, control-point coordinates). To transform to a second representation with spline matrix ~ we need to determine the geometric constraint matrix **Mgeom2** that produces the same vector point function for the object. That is,

$$P(u) = \mathbf{U * Mspline2 * Mgeom2}$$

Or

$$\mathbf{U * MSpline2 * Mgeom2 = U * Mspline1 * Mgeom1}$$

Solving for **Mgeom2,** we have :

$$\mathbf{Mgeom2 = M\sim^{1} Sline2 * Mspline1 * Mgeom1}$$

$$\mathbf{= Ms_1 , s_2 * Mgeom1}$$

and the required transformation matrix that converts from the first spline representation to the second is then calculated as :

Ms1,s2= Mspline2 * Mspline1

A non-uniform B-spline cannot be characterized with a general spline matrix. But we can rearrange the knot sequence to change the non-uniform B-spline to a Bezier Tepresentation. Then the Bezier matrix could be converted to any other form.

DISPLAYING SPLINE CURVES AND SURFACES

To display a spline curve or surface, we must determine coordinate positions on the curve or surface that project to pixel positions on the display device. This means that we must evaluate the parametric polynomial spline functions in certain increments over the range of the functions. There are several methods we can use to calculate positions over the range of a spline curve or surface. The simplest method for evaluating a polynomial, other than a brute-force calculation of each term in succession, is *Homer's rule,* which performs the calculations by successive factoring. This requires one multiplication and one addition at each step. For a polynomial of degree *n,* there are *n* steps.

POLYGON SURFACES

The most commonly used boundary representation for a three-dimensional graphics object is a set of surface polygons that enclose the object interior. Many graphics systems store all object descriptions as sets of surface polygons. This simplifies and speeds up the surface rendering and display of objects, since all surfaces are described with linear equations. For this reason, polygon descriptions are often referred to as "standard graphics objects." In some cases, a polygonal representation is the only one available, but many packages allow objects to be described with other schemes, such as spline surfaces, that are then converted to polygonal representations for processing.

A polygon representation for a polyhedron precisely defines the surface features of the object. But for other objects, surfaces are tesselated (or tiled) to produce the polygon-mesh approximation. The surface of a cylinder is represented as a polygon mesh. Such representations are common in design and solidmQdeling applications, since the wireframe outline can be displayed quickly to give a general indication of the surface

structure. Realistic renderings are produced by interpolating shading patterns across the polygon surfaces to eliminate or reduce the presence of polygon edge boundaries. And the polygon-mesh approximation to a curved surface can be improved by dividing the surface into smaller polygon facets.

We specify a polygon surface with a set of vertex coordinates and associated attribute parameters. As information for each polygon is input, the data are placed into tables that are to be used in the subsequent processing, display, and manipulation of the objects in a scene. Polygon data tables can be organized into two groups: geometric tables and attribute tables. Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. Attribute information for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics.

A convenient organization for storing geometric data is to create three lists:

A vertex table, an edge table, and a polygon table. Coordinate values for each vertex in the object are stored in the vertex table. The edge table contains pointers back into the vertex table to identify the vertices for each polygon edge. And the polygon table contains pointers back into the edge table to identify the edges for each polygon. This scheme is illustrated two adjacent polygons on an object surface. In addition, individual objects and their component polygon faces can be assigned object and facet identifiers for easy reference.

Listing the geometric data in three tables, provides a convenient reference to the individual components (vertices, edges, and polygons) of each object. Also, the object can be displayed efficiently by using data from the edge table to draw the component lines. An alternative arrangement is to use just two tables: a vertex table and a polygon table. But this scheme is less convenient, and some edges could get drawn twice. Another possibility is to use only a polygon table, but this duplicates coordinate information, since explicit coordinate values are listed for each vertex in each polygon. Also edge information would have to be reconstructed from the vertex listings in the polygon table.

We can add extra information to the data tables of faster information extraction. For instance, we could expand the edge table to include forward pointers into the polygon table so that common edges between polygons could be identified more rapidly. This is particularly useful for the rendering procedures that must vary surface shading smoothly across the edges from one polygon to the next. Similarly, the vertex table could be expanded so that vertices are cross-referenced to corresponding edges.

Additional geometric information that is usually stored in the data tables includes the slope for each edge and the coordinate extents for each polygon. As vertices are input, we can calculate edge slopes, and we can scan the coordinate values to identify the minimum and maximum x, y, and z values for individual polygons. Edge slopes and bounding-box information for the polygons are needed in subsequent processing, for example, surface rendering. Coordinate extents are also used in some visible-surface determination algorithms.

Plane Equations

To produce a display of a three-dimensional object, we must process the input data representation for the object through several procedures. These processing steps include transformation of the modeling and world-coordinate descriptions to viewing coordinates, then to device coordinates; identification of visible surfaces; and the application of surface-rendering procedures. For some of these processes, we need information about the spatial orientation of the individual surface components of the object. This information is obtained from the vertex-coordinate values and the equations that describe the polygon planes.

When polygons are specified with more than three vertices, it is possible that the vertices may not all lie in one plane. This can be due to numerical errors or errors in selecting coordinate positions for the vertices. One way to handle this situation is simply to divide the polygons into triangles. Another approach that is sometimes taken is to approximate the plane parameters A, B, and C. We can do this with averaging methods or we can project the polygon onto the coordinate planes. Using the projection method, we take A proportional to the area of the polygon projection on the y-z plane, B proportional

to the projection area on the x-z plane, and C proportional to the projection area on the x-y plane.

High-quality graphics systems typically model objects with polygon meshes and set up a database of geometric and attribute information to facilitate processing of the polygon facets. Fast hardware-implemented polygon renderers are incorporated into such systems with the capability for displaying hundreds of thousands to one million or more shaded polygons per second (usually triangles), including the application of surface texture and special lighting effects.

CURVED LINES AND SURFACES

Displays of three-dimensional curved lines and surfaces can be generated from an input set of mathematical functions defining the objects or from a set of user specified data points. When functions are specified, a package can project the defining equations for a curve to the display plane and plot pixel positions along the path of the projected function. For surfaces, a functional description is often tesselated to produce a polygon-mesh approximation to the surface. Usually, this is done with triangular polygon patches to ensure that all vertices of any polygon are in one plane. Polygons specified with four or more vertices may not have all vertices in a single plane. Examples of display surfaces generated from functional descriptions include the quadrics and the superquadrics.

When a set of discrete coordinate points is used to specify an object shape, a functional description is obtained that best fits the designated points according to the constraints of the application. Spline representations are examples of this class of curves and surfaces. These methods are commonly used to design new object shapes, to digitize drawings, and to describe animation paths. Curve-fitting methods are also used to display graphs of data values by fitting specified curve functions to the discrete data set, using regression techniques such as the least-squares method.

Curve and surface equations can be expressed in either a parametric or a nonparametric form. Appendix A gives a -summary and comparison of parametric and nonparametric equations. For computer graphics applications, parametric representations are generally more convenient.

QUADRIC SURFACES

A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, tori , paraboloids, and hyperboloids. Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphics scenes, and they are often available in graphics packages as primitives from which more complex objects can be constructed.

Sphere

In Cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation

$$x^2 + y^2 + z^2 = r^2$$

We can also describe the spherical surface in parametric form, using latitude and longitude angles:

X = r cos 0 cos 0

Y = r cos 0 sin 0

Z = r sin0

The parametric representation provides a symmetric range for the angular parameters 0 and 4). Alternatively, we could write the parametric equations using standard spherical coordinates, where angle 0 is specified as the colatitude . Then, 0 is defined over the range O <= 0 <=pi and 0 is often taken in the range O<= 0 <=. We could also set up the representation using parameters U and V defined over the range from 0 to 1 by substituting 0 = pi(U) and 0= 2pi(u).

SPLINE REPRESENTATIONS

In drafting terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points. Several small weights are distributed along the length of the strip to hold it in position on the drafting table as the curve is drawn. The term spline

curve originally referred to a curve drawn in this manner. We can mathematically describe such a curve with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections. In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying sped-fled continuity conditions at the boundary of the pieces. A spline surface can be described with two sets of orthogonal spline curves. There are several different kinds of spline specifications that are used in graphics applications. Each individual specification simply refers to a particular type of polynomial with certain specified boundary conditions.

Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or the camera in a scene. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

Interpolation and Approximation Splines

We specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of the curve. These control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways. When polynomial sections are fitted so that the curve passes through each control point, the resulting curve is said to interpolate the set of control points. On the other hand, when the polynomials are fitted to the general control-point path without necessarily passing through any control point, the resulting curve is said to approximate the set of control points.

Interpolation curves are commonly used to digitize drawings or to specify animation paths. Approximation curves are primarily used as design tools to structure object surfaces. Approximation spline surface created for a design application. Straight lines connect the control-point positions above the surface.

A spline curve is defined, modified, and manipulated with operations on the control points. By interactively selecting spatial positions for the control points, a designer can set up an initial curve. After the polynomial fit is displayed for a given set of control points, the designer can then reposition some or all of the control points to restructure the

shape of the curve. In addition, the curve can be translated, rotated, or scaled with transformations applied to the control points. CAD packages can also insert extra control points to aid a designer adjusting the curve shapes.

The convex polygon boundary that encloses a set of control points is called the convex hull. One way to envision the shape of a convex hull is to imagine a rubber band stretched around the positions of the control points so that each control point is either on the perimeter of the hull or inside. Convex hulls provide a measure for the deviation of a curve or surface from the region bounding the control points. Some splines are bounded by the convex hull, thus ensuring that the polynomials smoothly follow the control points without erratic oscillations. Also, the polygon region inside the convex hull is useful in some algorithms as a clipping region.

A polyline connecting the sequence of control points for an approximation spline is usually displayed to remind a designer of the control-point ordering. This set of connected line segments is often referred to as the control graph of the curve. Other names for the series of straight-line sections connecting the control points in the order specified are control polygon and characteristic polygon.

Parametric Continuity Conditions

To ensure a smooth transition from one section of a piecewise parametric curve to the next, we can impose various continuity conditions at the connection points. Zero-order parametric continuity, described as C^0 continuity, means simply that the curves meet. That is, the values of x, y, and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x, y, and z evaluated at u~ for the next curve section. First-order parametric continuity, referred to as continuity, means that the first parametric derivatives (tangent lines) of the coordinate functions two successive curve sections are equal at their joining point. Second-order parametric continuity, or C^2 continuity, means that both the first and second parametric derivatives of the two curve sections are the same at the intersection. Higher-order parametric continuity conditions are defined similarly.

With second-order continuity, the rates of change of the tangent vectors for connecting sections are equal at their intersection. Thus, the tangent line transitions smoothly from one section of the curve to the next . But with first-order continuity, the rates of change of the tangent vectors for the two sections can be quite different, so that the general shapes of the two adjacent sections can change abruptly. First-order continuity is often sufficient for digitizing drawings and some design applications, while second-order continuity is useful for setting up animation paths for camera motion and for many precision CAD requirements. A camera traveling along the curve path with equal steps in parameter U would experience an abrupt change in acceleration at the boundary of the two sections, producing a discontinuity in the motion sequence. But if the camera were traveling along the path , the frame sequence for the motion would smoothly transition across the boundary.

Geometric Continuity Conditions

An alternate method for joining two successive curve sections is to specify conditions for geometric continuity. In this case, we only require parametric derivatives of the two sections to be proportional to each other at their common boundary instead of equal to each other.

Zero-order geometric continuity, described as G^0 continuity, is the same as zero-order parametric continuity. That is, the two curves sections must have the same coordinate position at the boundary point. First-order geometric continuity, or G^1 continuity, means that the parametric first derivatives are proportional at the intersection of two, successive sections. If we denote the parametric position on the curve as P(u), 'the direction of the tangent vector P (u), but not necessarily its magnitude, will be the same for two successive curve sections at their joining point under continuity. Second-order geometric continuity, or G^2 continuity, means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under G^2 continuity, curvatures of two curve sections will match at the joining position.

A curve generated with geometric continuity conditions is similar to one generated with parametric continuity, but with slight differences in curve shape. Provides

a comparison of geometric and parametric continuity. With geometric continuity, the curve is pulled toward the section with the greater tan-gent vector.

BEZIER CURVES AND SURFACES

This spline approximation method was developed by the French engineer Pierre Bezier for use in the design of Renault automobile bodies. Bezier splines have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Be zier splines are widely available in various CAD systems, in general graphics packages (such as GL on Silicon Graphics systems), and in assorted drawing and painting packages (such as Aldus SuperPaint and Cricket Draw).

Bezier Curves

In general, a Bezier curve section can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of the Be zier polynomial. As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending functions. For general Bezier curves, the blending-function specification is the most convenient.As a rule, a Be zier curve is a polynomial of degree one less than the number of control points used: Three points generate a parabola, four points a cubic curve, and so forth. Demonstrates the appearance of some Bezier curves for various selections of control points in the xy plane ($z = 0$). With certain control-point placements, however, we obtain degenerate Bezier polynomials. For example, a Bezier curve generated with three collinear control points is a straight-line segment. And a set of control points that are all at the same coordinate position produces a Bezier "curve" that is a single point.

Design Techniques Using Bezier Curves

Closed Bezier curves are generated by specifying the first and last control points at the same position. Also, specifying multiple control points at a single coordinate

position gives more weight to that position. A single coordinate position is input as two control points, and the resulting curve is pulled nearer to this position.

We can fit a Bezier curve to any number of control points, but this requires t~ calculation of polynomial functions of higher degree. When complicated curves are to be generated, they can be formed by piecing several Bezier sections of lower degree together. Piecing together smaller sections also gives us better control over the shape of the curve in small regions. Since Bezier curves pass through endpoints, it is easy to match curve sections (zero-order continuity). Also, Bezier curves have the important property that the tangent to the curve at an endpoint is along the line joining that endpoint to the adjacent control point. Therefore, to obtain first-order continuity between curve sections, we can pick control points P'o and P'i of a new section to be along the same straight line as control points Pi and p of the previous section. When the two curve sections have the same number of control points, we obtain C^1 continuity by choosing the first control point of the new section as the last control point of the previous section and by positioning the second control point of the new section at position Thus, the three control points are collinear and equally spaced.

We obtain C^2 continuity between two Bezier sections by calculating the p0sition of the third control point of a new section in terms of the positions of the last three-control points of the previous section as

Pn~2 + 4(pn - Pn-1)

Requiring second-order continuity of Bezier curve sections can be unnecessarily restrictive. This is especially true with cubic curves, which have only four control points per section. In this case, second-order continuity fixes the position of the first three control -points and leaves us only one point that we can use to adjust the shape of the curve segment.

B-SPLINE CURVES AND SURFACES

These are the most widely used class of approximating splines. B-splines have two advantages over Bezier splines: (1) the degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations), and (2) B-splines

allow local control over the shape of a spline curve or surface. The trade-off is that B-splines are more complex than Bezier splines. There are several differences between this B-spline formulation and that for Bezier splines. The range of parameter U now depends on how we choose the B-spline parameters. And the B-spline blending functions Bk,d are polynomials of degree d - 1, where parameter d can be chosen to be any integer value in the range from 2 up to the number of control points, $n + 1$. (Actually, we can also set the value of d at 1, but then our "curve" is just a point plot of the control points.) Local control for B-splines is achieved by defining the blending functions over subintervals of the total range of U.

The selected set of subinterval endpoints U is referred to as a knot vector. We can choose any values for the subinterval endpoints satisfying the relation $U_1 \sim U_{1+1}$. Values for **Umin** and **Umax** then depend on the number of control points we select, the value we choose for parameter d, and how we set up the subintervals (k not vector). Since it is possible to choose the elements of the knot vector so that the denominators in the previous calculations can have a value of 0, this formulation assumes that any terms evaluated as 0/0 are to be assigned the value 0.

Demonstrates the local-control characteristics of B-splines. In addition to local control, B-splines allow us to vary the number of control points used to design a curve without changing the degree of the polynomial. Also, any number of control points can be added or modified to manipulate curve shapes. Similarly, we can increase the number of values in the knot vector to aid in curve design. When we do this, however, we also need to add control points since the size of the knot vector depends on parameter n.

B-spline curves have the following properties:

- The polynomial curve has degree d - 1 and Cd^{-2} continuity over the range of u.
- For n + 1 control points, the curve is described with $n + 1$ blending functions.

- Each blending function Bk_4 is defined over d subintervals of the total range of U, starting at knot value Uk.

- The range of parameter U is divided into $n + d$ subintervals by the $n + d + 1$ values specified in the knot vector.

A plot of the four periodic, quadratic blending functions, which demonstrates the local feature of B-splines. The first control point is multiplied by blending function $B_{0,3}(u)$. Therefore, changing the position of the first control point only effects the shape of the curve up to $U = 3$. Similarly, the last control point influences the shape of the spline curve in the interval where B_3 is defined.

Illustrates the limits of the B-spline curve for this example. All blending functions are present in the interval from $Ud - 1 = 2$ to $U\sim+_1 = 4$. Below 2 and above 4, not all blending functions are present. This is the range of the polynomial curve, and the interval; Thus, the sum of all blending functions is 1 within this interval. Outside this interval, we cannot sum all blending functions, since they are not all defined below 2 and above 4.

Since the range of the resulting polynomial curve is from 2 to 4, we can determine the starting and ending positions of the curve by evaluating the blending functions at these points to obtain Thus, the curve starts at the midposition between the first two control points and ends at the mid-position between the last two control points. We can also determine the parametric derivatives at the starting and ending positions of the curve.

In the preceding example, we noted that the quadratic curve starts between the first two control points and ends at a position between the last two control points. This result is valid for a quadratic, periodic B-spline fitted to any number of distinct control points. In general, for higher-order polynomials, the start and end positions are each weighted averages of 4 - 1 control points. We can pull a spline curve closer to any control-point position by specifying that position multiple times.

General expressions for the boundary conditions for periodic B-splines can be obtained by reparameterizing the blending functions so that parameter U is mapped onto the unit interval from 0 to 1. Beginning and ending conditions are then obtained at $U = 0$ and $U = 1$.

Cubic, Periodic B-Splines

Since cubic, periodic B-splines are commonly used in graphics packages, we consider the formulation for this class of splines. Periodic splines are particularly useful for generating certain closed curves. For example, the closed curve can be generated in sections by cyclically specifying four of the six control

RATIONAL SPLINES

A rational function is simply the ratio of two polynomials. Thus, a rational spline is the ratio of two spline functions . Rational splines have two important advantages compared to non-rational splines. First, they provide an exact representation for quadric curves (conics), such as circles and ellipses. Non-rational splines, which are polynomials, can only approximate conics. This allows graphics packages to model all curve shapes with one representation rational splines without needing a library of curve functions to handle different design shapes. Another advantage of rational splines is that they are invariant with respect to a perspective viewing transformation . This means that we can apply a perspective viewing transformation to the control points of the rational curve, and we will obtain the correct view of the curve. Non-rational splines, on the other hand, are not invariant with respect to a perspective viewing transformation. Typically, graphics design packages use non-uniform knot-vector representations for constructing rational B-splines. These splines are referred to as NURB 's *(non-uniform rational B-splines).*

Homogeneous coordinate representations are used for rational splines, since the denominator can be treated as the homogeneous factor in a four-dimensional representation of the control points. Thus, a rational spline can be thought of as the projection of a four-dimensional non-rational spline into three-dimensional space.

Constructing a rational B-spline representation is carried out with the same procedures for constructing a non-rational representation. Given the set of control points, the degree of the polynomial, the weighting factors, and the knot vector, we apply the recurrence relations to obtain the blending functions.

Other sections of a unit circle can be obtained with different control-point positions. A complete circle can be generated using geometric transformation in the *x y* plane. For example, we can reflect the one-quarter circular arc about the x and y axes to produce the circular arcs in the other three quadrants.

In some CAD systems, we construct a conic section by specifying three points on an arc. A rational homogeneous-coordinate spline representation is then determined by computing control-point positions that would generate the selected conic type.

SURFACES AND RENDERINGS

Earlier we saw how Alberti, in his Ten Books on Architecture, defined designs as abstractions separated from physical matter, but serving to specify how the physical matter of a building was to be organized and ordered. It followed, then, that a design consisted of "lines and angles" and that the designer's task was to make "a firm and graceful pre-ordering of the lines and angles." Correspondences between lines on paper and lines in space could be established systematically through use of projection techniques--in particular, perspective. This view provides the theoretical foundation for design by line construction in the plane (drafting) and for design by wireframe modeling. In another of his works, On Painting (1435), Alberti pointed out that there was an alternative approach. "Mathematicians," he wrote, "measure with their minds alone the forms of things separated from all matter." But speaking as a painter, he then continued, "Since we wish the object to be seen, we will use a more sensate wisdom."

The Reception of Light

Alberti's concern as a painter was not with abstract geometry as specified by lines in space, but with the appearances of solid objects as they present themselves to the eye. Such objects, in his terminology, have "skins" composed of an outline (orlo) bounding a "plane" (superficie). The superficie is that "certain external part of a body which is known not by its depth but only by its length and breadth and by its quality." These external parts may, he says, be divided into four kinds according to their

curvatures: flat ("that which a straight ruler will touch in every part if drawn over it"), spherical ("any part of that body is equidistant from its center"), hollowed ("as in the interior of an egg shell"), and compound ("in one part flat and in another hollowed or spherical like those on the interior of reeds or on the exterior of columns"). Appearances, he notes, are determined not only by properties of outline and curvature, but also by the positions of bodies relative to the observer: "as soon as the observer changes his position these planes appear larger, of a different outline, or of a different color." And there is one more thing "which makes the plane appear to change." It is "the reception of light." Alberti elaborates: You see that spherical and concave planes have one part dark and another part bright when receiving light. Even though the distance and position of the centric line are the same, when the light is moved those parts which were first bright now become dark, and those bright which were dark. Where there are more lights, according to their number and strength, you see more spots of light and dark. Thus the painter's fundamental intellectual program, as formulated by Alberti, was to study the appearances of surfaces in light --a matter of outline and surface geometry, lighting conditions, and observer position. In one important sense the history of European painting, over the span from Alberti to the Impressionists, can be understood as an elaboration and working out of this program. Successive generations of painters observed and found accurate ways to render increasingly complex and subtle effects of surface revealed in light. Since the emergence of high-quality raster graphic display devices, which can render surfaces in light on a cathode ray tube as a painter does with pigments deployed on canvas, this intellectual program has been reconstructed in a new form. Software has been developed for modeling three-dimensional objects not just in terms of their edge lines, but as collections of surfaces described by their outlines and curvatures. Such surface-modeling systems can produce not only wireframe images, but also hidden-surface views showing opaque surfaces in light. They allow information specifying surface properties (color, specularity, texture, and so on) to be associated with surface elements and allow the properties of light sources to be specified. From the geometric database, information about surface and lighting properties, and specified values for viewing parameters, they render effects such as shading, cast shadows, highlights, and reflections. The incorporation of increasingly sophisticated rendering algorithms, taking advantage of

increasingly powerful computer resources, has enabled them to produce images that more and more closely approach photorealism. Just as drafting and wireframe-modeling software enables us to explore architectural composition as Alberti conceived it in his Ten Books on Architecture--as the organization of lines and angles that specify the essential geometry of a building--so surface modeling supports architectural composition as it was defined in another way by Le Corbusier in Vers une architecture: Architecture is the masterly, correct and magnificent play of masses brought together in light. Our eyes are made to see forms in light; light and shade reveal these forms.

Insertion of Surface Facets

Illustrates two plates from Sebastiano Serlio's Architettura, which was published just a few years after Alberti's treatise on painting. The first shows an octagonal well in wireframe perspective. In the second, opaque surfaces have been inserted into the wireframe such that polygons found within the frame have become outlines of surface elements. Serlio remarks that the solid body thus shown "is the same that is before shewed, both form and measure, but all the lines which cannot outwardly be seen are hidden." The wireframe has served as an ordering skeleton over which the solid has been constructed: Serlio notes that they who "well understand and perfectly bear in mind the hidden lines shall better understand the art than others who content themselves with the outer superficies." So it is with surface-modeling software: the user first constructs lines, then employs these to guide development and positioning of surfaces. The most basic surface-insertion operation is a generalization of the basic line-insertion operation. Just as a one-dimensional straight line segment is specified by its zero-dimensional boundaries (end points), so a two-dimensional plane polygon is specified by its one-dimensional boundaries (edge lines). Thus an obvious way to insert a plane polygon into a surface model is to indicate its boundary points in sequence. A typical database format for a simple surface-modeling system, which is closely related to the basic surface-insertion operation, is illustrated in figure 10.3. There is a vertex list, an edge list, and a facet list. The vertex list records X and Y coordinates of vertices, the edge list specifies pairs of vertices linked by edges, and the facet list specifies sequences of edges bounding surface facets.

Sweep Operations

Another way to look at a straight line (as we saw earlier) is as the path swept out by a translated point. Similarly, we can regard a rectangular surface as the shape swept out by a translated straight line. It follows that we can specify such a surface by indicating a straight line and specifying a translation. More generally, we can specify assemblages of surfaces by sweeping arbitrary chains of lines. Most surface-modeling systems provide such translational sweep operations for surface insertion. A combination of facet-bounding and translational sweep operations can be used to model an open rectangular box. First the rectangular plan is constructed and swept to create the four upright sides. Then the vertices of the plan shape are picked off in sequence to create the bottom facet. Singly curved surfaces can be constructed by performing translational sweep operations on plane curves. Sweeping a circle, for example, constructs an open-ended cylinder. More complex profiles can be swept to construct the forms of architectural moldings. In addition to translational sweeps, surface-modeling systems usually provide rotational sweep operations in which line shapes are swept along arcs of circles. Cylindrical and conical surfaces, for example, can be produced by sweeping straight lines. Both spherical and toroidal surfaces can result from rotational sweeps of arcs. The combination of translational and rotational sweep operations is very powerful. Almost all classical and Gothic architectural elements (columns, piers, entablatures, arches, moldings, etc.) can, for example, be modeled by applying translational and rotational sweep operations to profiles constructed from straight lines and arcs: this is closely analogous to the stonemason's technique of using a template to mark a profile on a block of stone, then cutting this shape through the mass. But some more general sweep operations are needed to construct certain types of surfaces and are thus provided by advanced surface-modeling software. One generalization is to allow sweeping of arbitrary curves along arbitrary curves--an ellipse along a spline. Some systems allow the two ends of a line to be swept along different curves. In particular, ruled surfaces may be specified by sweeping straight lines in this way. Two particular types of ruled surfaces are fairly common in architecture: the hyperboloid of revolution and the hyperbolic paraboloid. A cone may be constructed not only by rotationally sweeping a straight line, but also by sweeping a diminishing circle along a straight line. By analogy, some advanced surface-modeling systems provide

generalized cone operations that define complex surface shapes by sweeping arbitrarily changing curves along arbitrary curves. This operation, for example, is useful for modeling human limbs.

Faceted Approximations of Curved Surfaces

Some surface-modeling systems represent curved surfaces internally by storing parameter values. A sphere can be represented by its center coordinates and radius, for example, and a cylinder can be represented by the end-point coordinates of its axis together with a radius. These parameters can be used, in conjunction with appropriate mathematical formulae, to generate accurate images of surfaces as required. An alternative approach is to approximate curved surfaces by small planar facets just as a curved line may be approximated by small, straight segments. Often these facets are triangular, since triangles are always planar, but facets of other shapes may be used as well. This technique proves to be adequate for many practical purposes, and it simplifies many of the computational tasks that a surface-modeling system must perform, so it is widely used in contexts where precise representation of surfaces is not critical and where computational resources are limited.

Surface Patches

Where a surface is approximated by a mesh of triangles, linear interpolation between the vertices of any triangle produces points within that triangle. If a surface is approximated by a mesh of quadrilaterals, however, the vertices of a given quadrilateral do not necessarily lie in the same plane, and if they do not, linear interpolation between them will produce a bilinear curved surface. Thus quadrilateral bilinear patches provide an alternative to triangular plane facets for representation of curved surfaces. The idea of curved surface patches may be extended in various ways to provide curved surface representations that are appropriate for different practical purposes. An obvious generalization of the bilinear patch, for example, is a patch bounded by four arbitrary curves. This type of patch, which provides more precise control of slopes, is known as a Coons patch. These patches are commonly used by boat and aircraft (and occasionally architectural) designers for skinning shapes that have been specified as sequences of

profiles or ribs. Variants on these shapes can readily be produced by manipulating the parameters that control profile curvature and spacing. A specialization of the Coons patch, which has some attractive mathematical properties, is the bicubic patch: this has parametric cubic polynomials as the four boundary curves. Just as a smoothly curved line may be produced by bending a thin, elastic strip of wood or metal, so a smoothly curved surface may be produced by bending a thin, elastic sheet of material. And just as a drafting system's analogy to a physical spline is a mathematical spline curve manipulated by deploying control points on the drafting plane, so the surface-modeling system's analogy to a twisted elastic sheet is a mathematical spline surface manipulated by deploying control points in space. B-spline surfaces, which extend the idea of a B-spline curve, are especially widely used. A particular type of B-spline surface, known as a NURBS (non-uniform rational B-spline) has formal properties that make it attractive in many practical modeling applications. A typical NURBS curved-surface modeler provides a vocabulary of basic shapes modeled as NURBS surfaces with meshes of control points. These control points may be pulled and twisted (usually in real time) to sculpt the shape as required. A particular concern in using bicubic, B-spline, and NURBS modelers to design curved objects is maintaining smooth curvature--not only of the surface itself but also, sometimes, of its first and second derivatives. Discontinuities in smoothness show up as unattractive bulges, wrinkles, kinks, and dimples. Sometimes they can also cause engineering problems. Appropriate curved-surface vocabularies and modeling strategies are sometimes determined by the materials and fabrication processes that are to be used to produce the surfaces. Stiff sheet materials that are to be fabricated by cutting out facets suggest use of a faceted modeler. Flexible sheet materials such as plywood and sheet metal can be bent into cylindrical and similar singly curved (translationally swept) surfaces or twisted into ruled surfaces such as hyperbolic paraboloids. Flexible boards can be used to form lofted surfaces such as those of boat hulls. Pressed metal and injection-molded plastic can be formed into many doubly curved B-splined shapes, but this is usually an expensive industrial process.

Fractal Surfaces

Fractal surfaces are the opposite of smooth surfaces. They commonly result in nature from growth and erosion processes, and they are produced computationally by recursive subdivision of surface patches, as illustrated. A surface is first approximated by a mesh of triangles. Then, using a random-number generator, each triangle is subdivided into four smaller triangles. This process is carried out recursively until a microstructure of tiny irregular facets is produced. By controlling the parameters of this process it is possible to construct fractal surfaces that depict various types of terrain and different sorts of textured materials with varying types and levels of roughness.

Topographic Surfaces

Topography varies arbitrarily, so representation of topographic surfaces is a matter of sampling and interpolation. Frequently, for example, elevations are recorded at points on a square plan grid. The surface can then be modeled by bilinear or bicubic patches. Increasingly accurate representations can be produced by sampling elevations at finer resolutions. Relatively coarse sampling and curve interpolation produces very smooth surfaces, finer sampling allows small bumps and depressions to show up, and very fine sampling reveals that natural topographic surfaces are actually fractals. (Notice the close analogy with scanning a visual field to produce a square grid of intensity levels.) Another common technique is to record elevations of arbitrary points (spot levels) on a surface, then to employ a triangulation procedure to construct a mesh of planar facets known as a TIN (triangulated irregular network) model. This is efficient, since data points can be concentrated where necessary to record abrupt changes and fine detail, but can be sparser elsewhere. Topographic surface-modeling software usually provides for use of a variety of interpolation strategies and production of several different types of surface representations. Thus a set of spot levels supplied by a surveyor might be translated into a contour map, a drainage direction map, a grid of bilinear patches, a set of parallel section curves, and a TIN model.

Surface Intersecting and Cutting

In two-dimensional drafting it is often necessary to find line intersections and to divide or trim lines at intersection points. Earlier we saw how two-dimensional drafting software provides operations for accomplishing this. Similarly, in surface modeling it is

frequently necessary to find surface intersections and to divide or trim surfaces at intersection lines. The simplest case is intersection of a plane surface by a plane surface--which always results in a straight line. Efficient procedures to find such intersection lines are not difficult to implement, so surface-modeling systems often provide plane-cutting operations. In software that relies on faceted approximations to curved surfaces, it is easy to go a step further and provide a generalized surface-cutting operation. But where curved lines and surfaces are represented accurately, by equations and coefficient values, it is necessary to compute the equations describing intersection lines. This can be a very complex mathematical task, so software for true curved-surface manipulation is much more elaborate (and usually more expensive and demanding of computational resources) than software based on the idea of faceted approximation. Systems that represent all kinds of surfaces by means of NURBS patches can provide general surface-cutting and intersecting tools. This means that designers can use complex curved surfaces as cutting tools to sculpt other curved surfaces. Furthermore, complex surface-intersection problems (as when complex profiles meet at awkward angles) can be resolved with ease. Section drawings can be produced from surface models by cutting to a plane (or sometimes a more complex surface) to produce a collection of lines--in effect a wireframe model, which may be stored on a separate layer. Where closed facets of such wireframes indicate the interiors of solids, these facets can be fitted with surfaces to indicate poch.

Rendering

When a building or other artifact has been modeled as a collection of plane or curved surfaces in space, it can be rendered realistically in line, tone, or color. This is a three-step process. Rendering software must first generate a perspective or other projection: as in production of a wireframe view, each element in the geometric model is projected onto the viewplane and clipping is performed. Next, the visible surfaces must be determined: only those closest to the viewer will be displayed. Finally, some surface-rendering computation must be executed to determine how the visible surfaces will look. Thus the rendering pipeline, which provides the way to go from a surface model to a rendered image

Visible-surface Determination

The problem of visible-surface determination is simply stated: given an object modeled as a collection of opaque surfaces, determine which edges and surfaces are visible to an observer at a specified location or viewing from a specified direction. The solution principle is also simple: edges and surfaces in back will be obscured by opaque surfaces in front. Thus the problem is, in essence, one of sorting surfaces in depth. The practical difficulty is one of computational complexity: as the number of surfaces in a model grows, the computer time required to determine the visible surfaces also grows (perhaps exponentially), and at some point the computation becomes impractical. For several decades, then, a great deal of research effort has been devoted to development of efficient visible-edge- and surface-determination procedures, and various clever approaches have emerged. But the details of these are now mostly of little concern to designers: with increasing availability of very fast processors, increasing sophistication of visible-edge- and surface-determination algorithms, and a growing tendency to incorporate standard algorithms in hardware, quick and reliable visible- surface determination for large geometric models has become commonplace. One detail that is of practical concern is the distinction between hidden-line and hidden-surface procedures. Hidden-line procedures perform accurate floating-point arithmetic to determine where lines are cut by edges of opaque polygons. This is a relatively slow and expensive process, but it yields coordinate data that can be used to produce large, accurate, pen-plotted or laser-printed line drawings. Hidden-surface procedures are used to produce bitmapped images. One common, simple procedure of this type is known (with little respect for the intelligence of painters) as the painter's algorithm. This is just the procedure of sorting polygons by depth back from the picture plane, then drawing from the back forward so that later polygons overwrite earlier ones. Since it does not require explicit determination of intersection coordinates, it is extremely efficient. Simple depth-sorting, hidden-surface procedures are defeated by conditions such as cyclical overlap of surfaces and will produce inaccurate results when these conditions are encountered. The procedures can be elaborated to deal appropriately with these conditions, but the amount of computational work that they must do goes up accordingly, and it takes longer for them to produce results. Some rendering software allows the user to make the trade-off--to

choose a quick sorting procedure that may produce flawed results due to sorting errors, or to choose a foolproof sorting procedure that takes longer. Quick sorts often suffice for a designer's own working purposes, since minor sorting errors are unlikely to cause confusion in this context. But they should never be used for presentations, since spatial ambiguities due to sorting errors can easily make a view incomprehensible to somebody who is not already familiar with the scheme. A rendering process can be terminated following hidden-surface determination to produce an unshaded view. This highly economical sort of view represents in a way that is fundamentally different from shaded views. As Heinrich Welfflin remarked, in a famous passage in Principles of Art History, "If we wish to reduce the difference between the art of the art of Rembrandt to its most general formulation, we say that a draughtsman and Rembrandt a painter." He took these artists as representative of two "radically different modes of vision"--the linear and the painterly--and suggested that these were "two conceptions of the world, differently oriented in their taste and in their interest in the world, and yet each capable of giving a perfect picture of visible things." In summary: "linear style sees in lines, painterly in masses." In a drawing composed of edge lines, as Welfflin further commented, "the eye is led along the boundaries and induced to feel along the edges." This is a form of representation useful to a designer when "the sense and beauty of things is first sought in the outline."

Basic Shading

To produce more painterly images, light intensities at points on visible surfaces must be computed and rendered. To produce a monochrome shaded image, a single intensity is computed for each point; and to produce a colored, shaded image, intensities of red, green, and blue components are computed. In general, intensity at a point is a function of many parameters: the spatial and spectral reflectivity properties of the surface at that point, the spatial and spectral emission properties of the light sources that are to be taken into account, the location of the viewer, and the relation of the surface to other surfaces in the scene. Simple, quick intensity-determination procedures take only some of these factors into account, and so produce only approximate results. More complex and

costly procedures consider more of them to render wider ranges of optical effects and subtler nuances. The most sophisticated surface-rendering algorithm is not necessarily best for a designer's purposes. It is important to understand the basic properties of the various available algorithms, the kinds of results that they produce, and their demands on computer resources. The most important effect of surface shading is to create pictorial space. Leonardo da Vinci put the matter thus: The first business of the painter is to make a plane surface appear to be a body raised and standing out from this surface, and whoever excels the others in this matter deserves the highest praise. And this study, or rather this summit of our learning, depends on lights and shades. Leonardo made endless careful studies of the reception of different kinds of light by different kinds of surfaces. Later scientists were to produce quantitative laws describing the distribution of light reflected from surfaces, and these eventually provided the basis for surface-shading algorithms used in computer graphics. The most elementary of these laws is Lambert's cosine law, discovered by the sixteenth-century physicist and astronomer Johann Lambert. It reduces to a precise formula the fact long known to painters, and explicitly noted by Leonardo, that the intensity of reflected light from a plane surface is related to the angle at which the light is incident. If we simplify the situation by assuming a dull, matte surface (so that there are no highlights) and directional diffuse light (so that no shadows are cast), diffuse reflection will result--energy arriving from the light source will be scattered uniformly in all directions. Lambert's law states that the intensity of light reflected in this way is proportional to the cosine of the angle of incidence. Thus intensity diminishes as the surface is tipped obliquely to the light source. Light incident perpendicular to a surface is reflected with maximum intensity, while light parallel to a surface will leave the surface in darkness. The s-shape of the cosine function assures that the drop-off in intensity is rapid in the middle ranges, but it flattens out at both extremes. The amount of light reaching a viewer is independent of that viewer's position: apparent intensities of surfaces will not change as the viewpoint changes. A simple and remarkably effective way to shade plane surfaces, then, is to specify lighting direction, calculate surface normals, then use a cosine function to calculate surface intensities. Old-fashioned drawing books demonstrate how to do it by hand with tonal media like charcoal and watercolor wash; simple software can perform it efficiently to produce bitmapped images

from surface models; and special-purpose graphics hardware can now accomplish it at very high speed. The effect is very much like that achieved by painters of illusionistic architectural decoration and backgrounds at Pompeii or by a photographer who places a matte gray architectural model in diffuse light. The simplest way to perform Lambert shading is to adopt a standard convention for the direction of light and to let surface intensities vary over the full range from white to black. A common architectural convention, for example, is to let light arrive from over the viewer's shoulder, at angles of 45 degrees horizontally and vertically from the line of sight. If the shaded object is then placed on a black ground, the light faces and their profiles will be emphasized; if it is placed on a white ground, the dark faces will be emphasized; and if it is placed on a mid-gray ground, the emphasis will be evenly distributed. More control over visual effects can be gained by introducing additional parameters into the basic shading equation. Most obviously, it is useful to be able to vary the direction of the incident light to change the distribution of darks and lights on the object. Control of lighting can be elaborated by providing for multiple light sources with different directions and intensities and by allowing specification of not only light source direction, but also actual position. Where position can be controlled it becomes possible to locate sources within buildings for night views and to simulate effects of artificial light (rather than sunlight) by diminishing intensity of illumination with distance according to the inverse square law. Surfaces can be differentiated by associating a surface reflection coefficient k with each one. This coefficient varies between 0 (black) and 1 (white). Thus the basic Lambert-shading equation is written Id = Ip K cos(Theta) where Id is the intensity of the reflected light, Ip is the intensity of the incident light, K is the surface-reflection coefficient, and Theta is the angle of incidence. By varying surface-reflection coefficients the effect of a black-and-white photograph can be produced. Objects rendered by this equation seem to exist in cold and harsh light like that of a flash photograph taken outdoors at night. This is because no account is taken of effects of nondirectional ambient light, which in the real world usually softens shadows and reduces contrasts (particularly in interior spaces, where there is much interreflection of light between surfaces). It is easy, however, to elaborate the basic Lambert-shading equation as follows I = Ia Ka + Id where I represents the surface intensity resulting from both ambient light and directional light Id.

The ambient contribution is simply the product of ambient intensity Ia and the ambient reflection coefficient Ka. If the ratio of ambient to directional components Ia/Id is high, then contrasts between differently oriented surfaces will be reduced and the model flattened, but if it is low then contrast will be high and modeling will be accentuated. Where color displays are available the idea of Lambert shading can be extended, in a very straightforward way, to provide a technique for producing colored, shaded images from surface models. Surface-reflection coefficients and source intensities are specified for red, green, and blue components, then used to compute reflected intensities for these components. The result is a color, specified in the RGB system, for each surface. Relationships of lighting and surface color can be adjusted to produce different types of images. If there is relatively little variation is surface color but dramatic, directional lighting, then images that emphasize modeling and chiaroscuro (like the paintings of Rembrandt) will result. But if local color is dramatically varied while lighting is kept flat, then images that cling closer to the picture plane (like Japanese prints or the paintings of Manet) can be generated. The essential result of Lambert shading is to add to an image information about the orientations of surfaces. This clarifies angular relationships between surfaces and enhances the illusion of pictorial depth by reinforcing the depth cues provided by foreshortening. Lambert-shaded images, then, are often particularly useful for studying issues of massing: they clearly present just the information that is of interest, but they suppress irrelevant and distracting effects. A characteristic disadvantage of Lambert shading is that parallel surfaces with the same reflection coefficients will have the same intensity. Thus the shading does not, in this case, convey depth information, and if the surfaces overlap in the image their outlines will be confused. Similarly, if the incident light bisects the angle between two faces, definition of the separating edge will be lost. Even worse, if the incident light equally divides the solid angle at a vertex, definition of that vertex will be lost. These effects are particularly troublesome in one-point perspectives of buildings with parallel elevation planes and in plan and elevation projections. Often they can be avoided or minimized by careful adjustment of viewpoint or view direction or by movement of the light source. But in many cases introduction of additional graphic information is needed to disambiguate the image. An obvious expedient is to combine Lambert shading with edge outlines. This produces a particularly

crisp, clear image since it defines with precision both shapes and orientations of surfaces. However, such images direct the viewer's attention less precisely to profile and contour than do pure line images, and less precisely to tone and mass than do pure shaded images.

Smooth Shading of Curved Surfaces

When Lambert shading is applied to a faceted approximation of a curved surface, the results are reasonably satisfactory when the facets are small enough. But the results may not be acceptable to a designer who wants to study fine nuances of curvature and the resulting modulation of light. Furthermore, distracting Mach bands--perceived exaggerations of intensity changes at edges of facets--are likely to appear. (That the dark side of an edge will often appear darker, and the light side lighter, was known to painters such as Leonardo and Mantegna, but the illusion is named after the physicist Ernst Mach, who studied it in the nineteenth century.) A better approach to curved-surface shading is to distribute intensities smoothly, rather than to change them abruptly at edges of facets. In his treatise on painting Alberti formulated this task and suggested a way to proceed: Remember that on a flat plane the color remains uniform in every place; in the concave and spherical planes the color takes variations; because what is here light is there dark, in other places a median color. This alteration of colors deceives the stupid painters, who, as we have said, think the placing of the lights to be easy when they have well designed the outlines of the planes. They should work in this way. First, they should cover the plane out to the outlines as if with the lightest dew with whatever white or black they need. Then above this another and thus little by little they should proceed. Where there is more light they should use more white. The computational equivalent of this little-by-little procedure for smooth tonal distribution was first developed at the University of Utah by Henri Gouraud. His essential idea was to calculate surface normals at facet vertices, then calculate intensities at these points, and finally, linearly interpolate intensities between these points to render a smoothly shaded surface. Linear interpolation is the numerical version of grading a wash or smearing charcoal with your finger. The calculations are simple (and Gouraud shading is often now implemented in special-purpose hardware for even higher speed), but the results can be very convincing. However, Gouraud-shaded

objects never sparkle. They always look as if they are made of some dull, matte material. This follows from the fundamental assumption, made in both cosine and Gouraud shading, that light is reflected equally in all directions. But most real surfaces reflect light somewhat unequally in different directions, with the result that specular highlights--more or less definite reflections of the light source--appear. These highlights move and change as the viewpoint alters. This effect has long been known and exploited by painters: E. H. Gombrich has suggested that its use might go back to the great Greek Apelles; there certainly was extensive use by Roman and Byzantine painters; and Jan Van Eyck and the Flemish illusionists deployed it with exquisite mastery. The scientific basis for calculation of specular highlights is provided by the law of the perfect mirror: the angle of reflection is equal to the angle of incidence. Thus the viewer can see specularly reflected light from a perfect mirror only when the angle alpha zero. For imperfect reflectors the intensity of specularly reflected light gradually diminishes as alpha increases, so that the viewer sees a fuzzy highlight rather than a perfect reflection of the light source. Thus the principle of cosine and Gouraud shading can be modified to take account of specular reflection by making intensity a function not only of the angle of incidence, but also of the angle alpha between the angle of reflection and the angle of view. A popular method for shading shiny curved objects in this way, to render specular highlights, was first developed by Bui-Tuong Phong. Phong shading not only deals with specular highlights, but also uses a more accurate interpolation technique than Gouraud shading. Consequently it is a more complex and expensive process than Gouraud shading, and it is difficult to compile into silicon. More recent research has focused on the accuracy and the speed of this type of rendering. Blinn shading procedures, and other more recent refinements of the idea, provide better results in some contexts. Compares _Gouraud, Phong, and Blinn shading. For Phong shading the angle alpha is calculated from the position or direction of light and the position or direction of view. The value of a specularity coefficient must be specified for each surface in the model. High specularity values yield the effect of very shiny surfaces with intense, concentrated highlights; lower values yield more diffuse highlights; and very low values yield the effect of a matte surface with almost imperceptible highlights. The center of a highlight will appear at the point where the angle of incidence equals the angle of reflection. The color of the highlight will be that of the light source,

not that of the surface. Multiple light sources will produce multiple reflections. The overall effect is very much like that of a careful airbrush rendering with highlights. An array of Phong-shaded spheres: diffuse reflectivity varies along the vertical axis, while specularity varies along the horizontal axis. With a surface modeler and a Phong-shading system an architect can conduct parametric studies of potential building appearance by systematically varying diffuse and specular reflectivities (to approximate effects of different materials and finishes) and lighting parameters. This sort of investigation would be impossibly laborious using conventional rendering techniques, but it is quick and inexpensive with computer-aided design.

Cast Shadows, Transparency, and Reflections

The advanced chapters of traditional treatises and textbooks on perspective usually consider the topics of cast shadows, transparency, and mirror reflections. These effects can all be investigated and rendered by extending the basic idea of tracing the paths of rays to determine foreshortening and occlusion. A cast shadow, for example, can be treated as the darkness thrown on a surface by an object that intercepts light. It falls on the side opposite to the light source and can only become visible when the light direction differs from the view direction. In the simplest case there is a single, infinitely distant point light source--approximately the case of the sun shining out of a cloudless sky. This idealized situation was extensively treated in Gaspard Monge's Geometrie descriptive (1799), and Monge's projective methods developed into the traditional architectural subject of sciagraphy. From a more modern computational viewpoint, the task of determining the shapes and locations of cast shadows turns out to be very closely related to the task of producing a hidden-surface perspective, and many of the same procedures can be employed. Surfaces produce shadow volumes from a point light source in just the same way that they produce occlusion volumes from eye points. A cast shadow, then, is the intersection of a surface with a shadow volume. To put this another way, every surface that the light source "sees" is in light, while every surface occluded from the light source is in shadow. So addition of shadows to a hidden-surface scene can be accomplished by specifying a point light source and performing some extra surface-

projection and depth-sorting operations. Shadow-casting procedures based on this principle can be used to add shadows to hidden-line, cosine-shaded, Gouraud-shaded, Phong-shaded, and Blinn-shaded scenes. Transparency effects are the inverse of cast-shadow effects. Where an opaque surface casts a patch of darkness, a transparent opening like a window casts a patch of light. And where an opaque surface occludes part of a scene, a transparent surface reveals part of a scene: it creates a view volume rather than an occlusion or shadow volume. Thus surface descriptions can be extended by specifying opacity coefficients, and effects of surface transparency can be rendered through further generalization of surface-projection and depth-sorting procedures. Computations of mirror-reflection effects are based on the mirror law--that the angle of incidence equals the angle of reflection. In the simple case of a plane mirror, then, the effect is to "double" the geometric model about the mirror plane. This double can then be projected in perspective in exactly the same way as the original, unreflected part of the model. These principles of geometric optics are elegantly combined with the principles of shading that we considered earlier in a computationally intensive but increasingly popular technique known as raytracing. Raytracing procedures consider the picture plane as a fine grid of pixels placed between the viewer's eye and the scene, and they send a ray from the eye through each pixel to the scene. By computing the red, green, and blue intensities reaching the eye along each ray, the color of each pixel can be established. Clearly this can become a very large task. If the grid has a resolution of one thousand by one thousand pixels, for example, it will be necessary to trace one million rays. The color of each pixel is calculated as follows. The ray through the current pixel is traced to its first intersection with a surface in the scene. The color of the light coming from the surface at this point will be the color of the pixel. This color is due to the combined effect of three things: shading resulting from light directly incident on that point from light sources in the scene (calculated by one of the shading procedures that we have considered), reflections of other objects in the scene, and the color of any object seen through the surface at that point. To account for cast shadows, a shadow ray is fired from the point to each of the light sources in the scene (so the amount of computation goes up rapidly when multiple shadowing light sources are introduced into complex scenes). And to account for the reflection and transparency effects, the ray is split into a reflected ray and a transmitted

ray, and these two spawned rays are continued until they, in turn, intersect surfaces. The color coming from these surfaces is then calculated in the same way. Of course the process need not stop here. The two spawned rays may themselves be split in two, and so on recursively to construct an intersection tree of reflected and transmitted rays for each pixel. The intersection trees are developed to whatever depth is judged necessary to account adequately for cast shadow, transparency, and reflection effects. The computation required to render a scene grows exponentially with the depth to which spawned rays are traced. Raytracing amounts to a strategy for discretizing and point-sampling the scene. Discretizing is accomplished by dividing the picture plane into pixels, and sampling is accomplished by constructing the intersection trees. Like any such strategy raytracing has characteristics that make it effective in some contexts but not in others. It works very well for highly specular scenes with point light sources, where it renders mirror reflections, highlights, sharp cast shadows, and refraction by transparent solids with breathtaking fidelity (see color plate 8). It is least effective for rendering scenes that consist mostly of matte surfaces with spot, line, and area light sources, since evaluation of integrals rather than point sampling is needed for accurate rendition of diffuse shading and interreflection effects. Effects such as soft penumbrae and "bleeding" of colors due to diffuse interreflection between surfaces are not present in raytraced images. These limitations can be overcome, to some extent, by elaborating the basic raytracing procedure, but this tends to make the computations even more complex and expensive. Raytracing was first applied to perspective rendering of architectural scenes by Arthur Appel in the late 1960s. During the 1970s and 1980s the technique was much elaborated, but production of raytraced images was regarded as a supercomputer application, and there was little practical application in design. By the beginning of the 1990s, though, it was becoming increasingly feasible on inexpensive personal computers. It will have a growing role to play in design contexts where effects of cast shadow, transparency, or mirror reflection are important and require close investigation.

Diffuse Global Illumination Effects

The qualities of many architectural interiors depend more on diffuse interreflections and soft shadows than they do on the specular reflections and point-source

cast shadows that are rendered so effectively by raytracing. (Raytracers usually account for diffuse interreflection, in very approximate fashion, by introducing an ambient light term into shading equations.) To study such diffuse effects, radiosity-rendering procedures (which derive from thermal engineering techniques for studying radiant energy) are more effective. Radiosity procedures begin by dividing the surfaces in the scene, rather than the picture plane, into small discrete elements. Thus they are based on a different discretization strategy from that of raytracing--one that is independent of observer position. They do not distinguish between light sources and reflecting surfaces: any element of a scene may act as an emitter of light energy. This makes them very suitable for rendering scenes with area sources of light (such as the panes of a window) and scenes in which large, brightly lit surfaces contribute substantially to interreflection effects. The sampling strategy is also different from that of raytracing. The basic assumption is that light reaches a given surface patch in two ways: directly from the light sources in the scene and indirectly by reflection from other surface patches. It is assumed at the outset that all surface patches have zero intensity. A first iteration of the shading calculation establishes the amount of light reaching each patch directly from the light sources. Some of this will be absorbed, some of it will be transmitted, and some of it will be reflected. A second iteration calculates the amount of light reaching each patch on the first bounce from other patches. Again, some of this will be absorbed, some of it will be transmitted, and some will be reflected back into the environment. A third iteration calculates the effects of this second bounce, and so on. Since the light energy leaving a patch is always less than the light energy incident on a patch (unless the patch is a light emitter), the effects of additional bounces eventually become negligible. Thus successively more accurate approximations are constructed. The geometric relation between a pair of patches determines the fraction of the light leaving one that reaches the other. This fraction is dependent on the shapes and areas of the two patches, their orientations, their distance apart, and any occlusion by intervening surfaces. The fraction for a given pair of patches is known as that pair's form-factor. The form-factors for a scene can be computed independently of viewer and light-source positions. Basically, computation of form-factors establishes a network of energy transfer paths between patches to be used in the iterative calculation of their intensities. For complex scenes

computation of the form-factors is a massive task. However, this need not be repeated when viewing parameters are changed. Thus the radiosity method is expensive for producing a single view, but very efficient for producing large numbers of views. In nondiffuse environments radiosity calculations become much more complex and time-consuming to carry out, partly because the intensity equations become more complicated when directional reflection must be considered and partly because smaller surface patches must be used to achieve satisfactory results. For scenes where both diffuse and specular effects are of interest, radiosity can be used to compute diffuse effects, raytracing can be used to compute specular effects, and the results can be summed to produce the final image .

Although the principles of radiosity have been exploited in various contexts since the 1960s (in the GLIM lighting analysis program, for example), radiosity rendering has been slower than raytracing to find practical application in design--mainly because it is even more demanding of computational resources. For a designer interested in close study of effects of light and surface, however, a radiosity renderer (particularly when calibrated carefully to produce accurate results) is an exceptionally powerful simulation tool and a necessary complement to a raytracer. In many contexts, for example, a raytracer can appropriately be used to produce studies of a building's exterior in crisp sunlight, while a radiosity renderer is used to study interior spaces with diffuse artificial light, window areas that function as light sources, and light-colored walls and ceilings that produce extensive diffuse interreflection.

Surface Details and Textures

All the rendering procedures that we have considered so far make the assumption that surfaces (whether planar or curved) are uniform within their boundaries. In other words, reflectivity, specularity, and transparency values describe whole surface facets.

This assumption greatly simplifies modeling and rendering, and is entirely appropriate when a designer is interested in studying basic interrelationships of surface geometry and lighting in a scheme. At another level of consideration, though, microstructural variation within surfaces becomes important. Consider, for example, a brick wall. As a first

approximation , we might model and render it as a uniformly colored rectangle. For more accurate rendering to support closer consideration of design issues like the effects of different bonding patterns, we might next model it as a collection of smaller rectangles--the bricks themselves. Next, we might want to study the subtle but important effects of different ways of raking the mortar joints: this requires a three-dimensional model of surface relief and a rendering procedure that shows the shadow lines under individual bricks. Finally, we might recognize that individual bricks are not in fact uniform, but display variations of color and shininess across their surfaces. The appropriate level of consideration, modeling, and rendering depends on the design issue that is of current interest. Finer and finer details of a design can always, in principle, be studied by building more and more intricate surface models--but this approach requires enormous modeling effort, consumes excessive amounts of memory for model storage, and makes huge demands on rendering systems. Fortunately, it is not always necessary. Approximate techniques for specifying and rendering surface detail will often suffice, and these are frequently provided by advanced rendering software. The simplest approach is to provide for surface-detail polygons that are coplanar with base polygons in the surface model. These suffice for painted signs and decoration, and often for shallow-relief detail like brick and tile patterns or curtain wall fenestration. This is economical, since the base polygons rather than the surface-detail polygons are depth sorted in hidden-surface and shadow-casting calculations. Finer surface detail can be approximated by scanning an appropriate pattern or texture and mapping it onto a plane or curved surface. Many raytracers and radiosity renderers can produce texture-mapped renderings in this way. The process is one of selecting an image to be mapped and specifying how the four corners of the image are to be placed on the surface. This can produce very effective results, but developing libraries of scanned textures, selecting textures to apply to surfaces, and specifying mappings are all very time-consuming processes. Furthermore, great care must be taken to avoid unconvincing distortion of textures when they are mapped onto nonrectangular and curved surfaces, to keep textures in correct scale and orientation, and to avoid perceptible repetition when small texture samples are tiled over large surfaces. A far more practical approach, for many classes of textures, is to generate the required surface variation by application of procedures that are controlled by just a

few parameters. One obvious parameterization approach is to adapt from two-dimensional paint systems the idea of a gradient fill operation. Another possibility is to apply a random speckling procedure. Sine functions can be used to generate ripple effects of various kinds. Recursive fractalization procedures can generate very realistic wood-grain, marbling, clouds, and other natural patterns. Regular tessellations and other repeating patterns can be produced by procedures that instantiate and transform standard motifs. Wood-grain and many other surface textures vary characteristically from surface to surface of an object (the end grain looks different from the side grain, and so on) because they result from cutting oriented three-dimensional patterns (see color plate 10). These textures are often most effectively generated by using procedures that produce three-dimensional "solid" textures and intersect them with object surfaces to produce variant surface qualities. Such procedures can also be used by designers to study (without expensive experimentation on actual materials) effects of cutting wood, stone, and other materials in different ways. Texture maps (either captured or procedurally generated) can be used to control not only variation in local surface color, but also variation in specular reflectivity, transparency, and relief. Effects of relief, such as those of orange skin or incised patterns, are produced by perturbing surface normals to approximate the shading of microfacets . This strategy has some limitations (particularly in rendition of edges), but it can often produce very effective results. It is particularly useful for studying architectural compositions, such as those of H. H. Richardson, in which relationships of surface relief and roughness play an important role. In summary, textures are functions of spatial coordinates: a two-dimensional texture is a function of X and Y in a surface-coordinate system, and a three-dimensional texture is a function of X, Y, and Z in a three-dimensional coordinate system. Such a function may have a single number as its value at specified coordinates, or it may have a vector of numerical values. Its values may be interpreted by a shading procedure as diffuse reflectivities, specular reflectivities, transparencies, surface normal perturbations, or anything else that the procedure takes into account in computing intensities. Depending on the character of the function, it may be represented as a stored array of values (sampled texture) or as a procedure that takes coordinate values as input (procedural texture): this is basically a store-versus-compute

trade-off. Sophisticated rendering systems provide extensive libraries of textures, together with facilities for assigning textures to surfaces or solids in geometric models.

Natural Phenomena and Landscape Composition

The techniques that we have considered so far yield a powerful tool kit for modeling, rendering, and studying landforms, architectural proposals, and urban design proposals. But landscape designers (and architects who want to study buildings in natural settings) need to extend this repertoire still further with tools for modeling and rendering vegetation, water, and atmospheric effects. A simple but frequently very effective way to handle trees (particularly distant ones) is to texture map scanned photographs of trees onto transparent rectangles arranged parallel to the picture plane. For greater realism, and to provide greater flexibility in choice of viewpoint, such planes can be crossed. An alternative approach is to employ growth-simulation procedures for construction of vegetation forms. These can be designed either to produce two-dimensional images for texture mapping onto transparent planes or to produce full three-dimensional surface models that are then inserted into a scene and rendered in the usual way. (Three-dimensional vegetation models can become extremely complex, however, so rendering times can easily become excessive.) Very simple recursive procedures often suffice to produce convincing results, and adjustment of a few parameters of these procedures can yield wide ranges of variants. More sophisticated procedures can be based on the concept of a Lindenmayer system (a special type of grammar for describing growth and differentiation processes) to produce detailed, accurate models of particular plant species. Still, horizontal water surfaces can be approximated in raytraced renderings by reflective planes, and ripple effects can be added with a sinosoidal texture-generation procedure and bump mapping to perturb surface normals. Moving streams, waterfalls, and fountains are more difficult: the usual approach is to model them not as surfaces, but as procedurally generated configurations of discrete particles. The generative procedures incorporate laws of particle propagation and motion. This approach can be extended to modeling many other natural phenomena, such as swirling fog and smoke, clouds, flames and fireworks, and masses of foliage. Effects of aerial perspective, haze, and mist can be approximated in landscape scenes by attenuating surface colors according to some exponent of the distance back from the picture plane. A high exponent produces rapid attenuation, as in dense fog, while a lower exponent yields more gradual attenuation. A shift toward white

produces a fog effect, a shift toward blue produces aerial perspective, a shift toward brown produces Los Angeles smog, and a shift toward black generates gathering gloom.

Retouching and Painting Shaded Images

Images rendered from surface models are not always perfect. Sometimes there are small modeling errors, such as omission of a surface, which yield blemishes. Sometimes there are polygon sorting errors or other undesirable artifacts resulting from the limitations of or bugs in rendering procedures. Or there may be problems of contrast and color balance. Usually these sorts of problems can be fixed by correcting the model or regenerating the rendering with slightly different viewing or lighting parameters. But if the problems are minor, it is often quicker and easier to move the bitmap to a paint and retouching program for correction--exactly as a scanned photograph might be retouched. More interestingly, a synthesized image can be used as the base for hand sketching and rapid exploratory development of a design idea. One approach is to make a plot of a hidden-line view as a base for further development with colored pencil or watercolor. Another approach is to move a simple shaded image to a paint system, then to sketch over the top of it with paint tools.

Combining Synthesized and Captured Images

When two perspectives are overlaid, objects are brought together into definite relationship within the same frame of reference. This technique is sometimes used for piecewise synthesis of very large and complex perspectives. It can also be used for combining synthesized shaded images with scanned photographs. To show a building in site context, for example, a perspective is synthesized with viewing parameters that match the camera position and settings for a site photograph. Then the synthesized image is carefully positioned relative to the photograph and electronically matted. Various subtleties need attention if a convincing result is to be achieved. First, the lighting of the synthesized image must be matched as closely as possible to the lighting of the photograph. Foreground elements such as foliage must be replaced. Shadows and reflections may need to be adjusted by careful painting. And edges may need to be

blended through use of a smoothing brush. Variations on existing buildings and urban settings can also be depicted effectively in this way.

Output and Presentation Technology

For individual working purposes and small conferences, the bitmapped images produced by shading a surface model can be displayed directly on a color monitor. For presentation to larger groups, a video projector can be connected to the monitor. Alternatively, slide or print output can be generated. High-quality results can be obtained by making high-resolution 35 mm slides on a digital film recorder. If these are projected at a large enough size to extend to the edges of the viewer's visual field, and if the viewer is located at a position corresponding to the station point from which the perspective was generated, a startling effect of three-dimensional realism is produced. Adequate working prints can be generated from monochrome shaded images by halftoning and laser printing, and publication-quality halftones can be produced with a laser image setter. Inexpensive inkjet and thermal wax-transfer color printers usually produce disappointing results, since they lack the color and spatial resolution needed for accurate reproduction of the subtleties and complexities of sophisticated shaded images. Much better prints can be obtained (usually at correspondingly higher cost) with dye-sublimation printers, by making photographic negatives on a film recorder, or by making digital color separations on an image setter.

Uses and Limitations of Surface Modeling and Rendering

Surface models obviously contain substantially more information than corresponding wireframe models. The complexity of the data structure increases too, since associations must be maintained not only between vertices and edges, but also between edges and surface facets. Thus storage and manipulation of data structures for surface models makes heavier demands on computational resources. There are more commands for a user to learn, and there is usually more work to do in constructing and editing a surface model, since surface shapes and properties must be specified. Generation of images can become significantly more time-consuming and expensive--particularly if detailed surface descriptions, sophisticated lighting models, and advanced rendering

techniques are used. The advantages of surface modeling over wireframe modeling for many purposes are also considerable, however, and these frequently justify the additional effort and cost. (Furthermore, the cost difference is of decreasing significance as basic computational costs continue to drop and as hidden-surface and shading algorithms are increasingly embodied in silicon.) Much more realistic images than a wireframe view can be produced from a surface model, and subtle effects of light and shade, color, texture, transparency, and reflection can be explored. Furthermore, you can get just the level of realism that you need by applying different types of rendering procedures to a model. Shaded images generated from surface models usually play a complementary role to plans, sections, and wireframes in design investigation. A plan or section provides a highly abstracted summary of a project's essential organization, a wireframe image gives an overview of its three-dimensional geometry, but a shaded or hidden-surface view is, by contrast, partial and fragmentary: it depicts only one of a project's indefinitely many aspects. A surface-modeling system can, however, inexpensively produce many such views--so that a project can be studied as a composition of revelations and concealments unfolding as an inhabitant moves through it.

Architects of the past often looked with a painter's eye: many of the greatest Renaissance architects were also painters, and Beaux-Arts architects were adept at the use of graded watercolor wash to study qualities of shade and shadow. But architects of the twentieth century have, for both ideological and pragmatic reasons, tended to rely on line drawings that abstract away from color, texture, and shading to emphasize pure geometry. Surface modelers and renderers create the possibility of recapturing the subtle understanding of surface and light that has, as a result, been lost.

Surface modelers are often marketed to designers merely as presentation tools. But that misses the most important point about them. When they are quick and cheap enough for everyday use they support graphic problem-solving by constructing, tweaking, and intersecting surfaces and by encouraging trial-and-error exploration of potential visual qualities in a cycle of modeling, rendering, modifying the model again, and so on until the desired effect is achieved.

Drafted Lines

It may be that the most rigorous exponents of Impressionism conceived of pictures as collections of colored points corresponding to light intensities reaching the painter's eye, but this was exceptional. Painters, drafters, and designers usually conceive of pictures in a much more highly structured way--as collections of geometric entities, such as straight lines, arcs of circles, and closed polygons. The artist composes by inserting these sorts of entities into a picture and relating them to each other in appropriate ways. In the creation of representational pictures the artist deploys these two-dimensional geometric entities on the picture plane to depict physical objects in three-dimensional space.

When an artist makes a freehand sketch, the intended entities and relationships are rendered only approximately by marks on paper. In technical drafting, though, geometric entities are rendered precisely through use of instruments such as straightedges and compasses, and relationships are formed accurately through execution of geometric constructions. Similarly, computer graphics software that is designed for use in technical drafting provides tools for precise manipulation and accurate presentation of geometric entities. The databases processed by this software contain digital representations of geometric elements and their relationships. There is a point table and a line table. The entries in the point table record x and y coordinates of points, and the entries in the line table specify which pairs of points are associated to define lines. Associated procedures translate the values in this table into lines on a display screen. Raster graphic displays produced from such databases may look much like displays produced from bitmaps, but they behave very differently since operations are defined on the lines themselves rather than on the pixels that make up the display of lines.

Coordinate Systems

The idea of a rectangular, two-dimensional Cartesian coordinate system provides the foundation for all drafting software. Within such a system, points are specified by their coordinates--that is, pairs of numbers. These numbers are stored in binary format in

computer memory. Let us assume that one bit is used to specify each coordinate. This yields a 2 x 2 grid of specifiable locations, as illustrated . Within this grid, then, six different straight lines may be specified by their end points. If we take a design to be a set of lines, this elementary coordinate system allows us to construct just 26 = 64 different designs. However, this number increases very dramatically as we increase the number of bits used to encode each coordinate. With two bits per coordinate we have a 4 x 4 grid of sixteen distinct points, which provides a thousandfold increase in the number of different designs. Practical drafting systems employ eight-bit, sixteen-bit, or thirty-two-bit binary numbers to represent coordinates. Usually, however, designers do not want to think in terms of integer coordinates. They want to use, instead, normal units such as feet and inches, meters, and so on. Hence a drafting system normally provides a menu of units from which the user can select and internally converts coordinates expressed in these user-selected units into binary integer form. It is straightforward to extend this idea to provide a unit-conversion capability so that, for instance, coordinates expressed in feet and inches can be reexpressed in meters and centimeters. In some design contexts it is convenient to work in polar coordinates rather than rectangular coordinates. Many drafting systems provide this option. The translation between polar and rectangular coordinates is just a matter of some simple trigonometry and can be performed automatically. The usual way to translate from user coordinates to the internal binary representation is automatically to set the maximum extent of the equal to the full extent of the coordinate system (as determined by the number of bits used for each coordinate). This means that small objects can be represented with high precision or very large objects can be represented with lower precision. However, if very large objects (complete city plans, say) and very small objects (such as window details) are shown in the same drawing, the small objects may be represented with inadequate precision. Hence the user of a drafting system must keep in mind that coordinates are represented to finite precision and organize drawings so that precision problems do not develop. Some systems simplify the issue of extent and precision by representing coordinates internally as floating-point rather than integer numbers. This yields a coordinate system of large and indefinite extent, in which small coordinate values have many significant decimal places and large coordinate values have few. In effect, small objects located near the origin of the coordinate system are represented with high precision, while small objects located far

from the origin are represented more approximately. By contrast, an integer coordinate system provides a uniform density of spatial indexing throughout its extent.

Point Specification

The most fundamental operation that a drafting system provides is specification of a point within the coordinate system. The most precise but most cumbersome way to accomplish this is to type in numerical values. It is quicker and more convenient to place the cursor and click with a mouse, but this is much less precise. Where coordinates are to be digitized from an existing drawing, use of a digitizing tablet with a stylus or digitizing puck is the most suitable technique. Since pointing and digitizing devices are inherently imprecise, geometric guides are frequently used to constrain their input . These various numerical and graphical techniques have their strengths and weaknesses, which make them suitable for different applications, so many systems provide the means to use all of them quickly and interchangeably. The capacity to specify any point in the coordinate system is clearly essential, but it does not, in itself, provide a sufficient basis for efficient construction of drawings. In most practical contexts a designer needs to select from among a very much smaller subset of points. A drafting system should, then, provide tools for specifying such subsets, constraining choice to them, and efficiently selecting from among their members. Many drawings, for example, are constructed under the discipline of some modular grid. The grid amounts to a specified subset of the points in the coordinate system, and the end points of lines are constrained to be in this subset. In response to this, drafting systems normally allow users to define grids with grid points at specified intervals and provide for efficient selection of grid points by automatically "snapping" an indicated point to the nearest grid point . This is particularly useful with mouse or stylus input, since it allows the user to indicate points quickly and approximately without paying the penalty of lost precision. The minimal requirement is for a drafting system to provide square grids parallel to the coordinate axes. More sophisticated systems may go beyond this to provide the other regular plane systems of points. They may also allow grids to be rotated and superimposed. Polar grids are appropriate where polar rather than rectangular coordinates are used. As a drawing develops, a designer mostly needs to select significant points in the existing skeleton of

lines--end points of existing lines, center points and tangent points of arcs, line intersection points. Some of these points are represented explicitly in the data structure, and the rest are implicit but can be computed as required. Sophisticated drafting systems, then, provide for snapping to both explicit and implicit significant points. Different sorts of points may, of course, seem significant in different design contexts at different moments. A drafting system can respond to this variability by allowing the user to specify the types of points that are to be considered and a diameter of attention. Then, as the cursor moves across the drawing, points of the specified types within the specified radius of attention can be highlighted. Experienced users of drafting systems usually take extensive advantage of constraint and snapping capabilities. They begin by building up skeletons of construction lines, then use these structures to snap further lines quickly into place, then use these lines to locate still finer details, and so on until the entire drawing is complete. By following this strategy they can almost entirely eliminate the need for painstaking location of points by calculating and typing coordinates or by precise stylus work.

Repertoires of Line Types

The capacity to specify lines is built from the capacity to specify points. Any two points define a straight line, for example, so all drafting systems allow a user to specify straight line segments by indicating their end points. When a mouse or stylus is used for this purpose, indication of the first end point fixes one end of the line and a "rubber-band" line then stretches from that location to the current location of the cursor and follows the cursor until the second end point is selected. This graphically demonstrates the mathematical equivalence of a second basic way to describe a straight line numerically--by its origin, direction, and length. An indefinitely long straight can be specified by slope and intercept. Curved lines of various types can be specified by giving their end points plus sufficiently many additional points or parameter values to define the shape between the two ends. Any three points lie on a circular arc, for example. Mathematical equivalences multiply in this case: drafting systems normally provide for specification of arcs by end points and a center point, by end points and a point on the circumference, by end points and an included angle, by an end point, center point, and swept angle. Any of

these ways may be needed, depending upon the context into which the arc is to be snapped. Notice, incidentally, that a computer drafting system has no difficulty with handling arcs of large radius--a type of graphic element that causes problems in traditional drafting, where compasses are of physically limited radius and the drawing surface for location of center points is of limited extent. Any straight line is the radius of a circle, so complete circles are usually input by means of a variant of the rubber-band line. Furthermore, any arc specifies a complete circle, so any of the methods used for inserting arcs may be modified slightly to serve for inserting circles. Any straight line may also be interpreted in yet another way, as the defining diagonal of a rectangle that bounds an ellipse. So the rubber-band line technique can also be adapted to serve for convenient specification of ellipses. Whereas ellipses are difficult to handle with traditional manual drafting instruments, and so have often been approximated by ovals (constructions of tangentially connected circular arcs), they present no problems when even a very simple computer drafting system is used. Straight lines, circular arcs, and elliptical arcs are all subclasses of conic section curves. The repertoire of a drafting system can be expanded to the conic sections in general by providing for hyperbolic and parabolic arcs as well. Since any curve can be represented by a sufficiently complicated polynomial expression, and other instances of the same type can be specified by varying the coefficients of that polynomial, the repertoire of different types of curved lines provided by a drafting system can be extended almost indefinitely. There is little practical need for this, however. Addition of a few basic types of spline curves to the conics suffices for most design purposes. A spline curve, much like the pinned wooden splines used in manual drafting, is specified by an arbitrary number of control points, which (depending on the particular type of spline) may or may not lie on the curve. A drawing from a drafting system, then, is essentially a two-dimensional arrangement of instances of the line types that the system provides (as a text is a one-dimensional arrangement of instances of characters from a character set, and a melody is a one-dimensional arrangement of sounds). The wider a system's repertoire of line types, the more versatile it will be in its representational capabilities. Notice that, under this representational scheme, some lines in a drawing are explicitly represented in the underlying data structure but others are only implicit.This means that the three sides of the triangle are only implicitly represented. Alternatively, the

figure might be input and stored as six shorter lines. Thus the three sides of the triangle are represented explicitly, but now the longer lines are only implicit. (There are many other possibilities as well.) This has important practical consequences, since most drafting systems (at least among those commercially available at the time of writing) allow you to select and operate only on lines that are explicitly represented in the data structure. However, this is not a necessary restriction: it is feasible (although computationally more expensive) to extend the principle of constraint-based editing and allow a user to specify not only the types of points that are to be selectable, but also the types of lines--independently of whether they happen to be represented explicitly in the data structure.

Chains of Lines

Most drafting systems provide for representation of irregular lines, in piecewise fashion, as connected sequences of short line segments. In the simplest case, points on an arbitrary curve are connected by straight lines to produce a faceted approximation the corresponding input technique is to specify these points in sequence. In general, the segments might be of any line type provided by the system. Such a chain, often called a polyline, is useful for describing a complex object or path, such as a boundary or contour line in a site plan, that is to be understood as a single entity. If a chain's last point is connected back to its first point, it forms a closed circuit--a structure of particular interest that will be discussed in detail in the following chapter. Another kind of chain operation constructs irregular lines from straight lines by executing procedures to fractalize them. The degree of irregularity that results can be controlled by a parameter. Some drafting systems include this type of line in their repertoires.

Basic Operations on Lines

Just as a text-processing system provides basic tools for inserting, selecting, and deleting characters, a drafting system always provides basic tools for inserting, selecting, and deleting lines--the visible elements of a drawing. These suffice for constructing and editing drawings, but it is convenient to have a wider range of manipulative capabilities at one's disposal. Sophisticated drafting systems are distinguished from simpler ones by the

wider ranges of specialized tools they provide. Among the most commonly provided additional tools are break, extend, and trim operations. A break operation separates a specified line into two lines at a specified point . (Notice that this does not change the appearance of the drawing, but it does change the underlying digital representation, and this can later cause the drawing to behave differently when you perform other operations.) An extend operation lengthens a selected line by a specified amount or to meet some specified line .Conversely, a trim operation shortens a selected line by a specified amount or cuts it back to some specified line.

Geometric Constructions

Traditionally, facility in technical drafting has depended upon knowledge of constructive procedures executed with traditional drawing instruments--procedures for constructing parallels and perpendicular bisectors to lines, tangents to circles, and so on. (The famous secrets of the medieval masons mostly consisted, in fact, of step-by-step descriptions of these procedures.) They can be replicated with the basic drafting system functions of snapping (equivalent, for example, to placing the point of compasses at the intersection of two lines) and insertion, deletion, division, extension, and trimming of primitives. There is no need, however, to burden the user of a drafting system with remembering all the steps of all these procedures. It is better to provide for automatic execution of commonly used constructive procedures, thus extending the tool kit with some higher-level operations. The computer versions of these procedures depend not on finding points by intersecting lines or arcs (as in Euclid) but on evaluating formulae. Consider, for example, the problem of finding the midpoint of a straight line. The beautiful construction given by Euclid involves striking two arcs to locate a pair of points, then constructing a line through those points to locate the required midpoint. In the computer version, the X-coordinate of the midpoint is calculated by taking the average of the X-coordinates of the ends, and the Y-coordinate of the midpoint is calculated by taking the average of the Y-coordinates of the ends. The generalization to division into any specified number of equal parts is obvious. Although the arithmetic becomes more complicated, further generalization to subdivision of arcs and splines is also straightforward. In addition to procedures for subdividing existing lines, a draftsperson

needs to know procedures for inserting new lines in specified orientations to existing lines. To show wall thicknesses on a plan, for example, an architect often needs to insert parallels to existing straight lines, arcs concentric with existing arcs, and even splines appropriately offset from existing splines. Other artifact geometries may be defined in terms of perpendiculars to straight lines, and normals and tangents to curves. Sophisticated drafting systems provide convenient tools for locating crucial construction points or for inserting lines directly in these relationships. These may be used for graphic problem-solving, in the same way that the mathematical functions provided by electronic calculators, spreadsheets, and symbolic mathematics systems may be used for numerical problem-solving. The larger its repertoire of useful geometric construction tools, the more effectively a drafting system can be used as a problem-solving rather than mere decision-recording medium. Designers often want to avoid discontinuities in curved profiles, for example, and this necessitates locating curves in tangential relationships: a straight line may be tangent to a circular arc, an arc may be tangent to another arc, an arc may form a fillet between two straight lines, and so on. Unless your geometric knowledge is extensive, construction of these figures can present considerable difficulty. Thus many drafting systems emphasize tools for continuous connection of straight lines and curves. This game can be elaborated endlessly, since entire construction procedures can always be used as steps in still more elaborate construction procedures--just as very complex mathematical functions can be built up from simple ones. In this way a whole hierarchy of drawing construction tools can be implemented. At the bottom of the hierarchy are a very few simple, basic operations and at the top there are potentially many powerful, specialized procedures. Skilled users of drafting systems achieve maximum efficiency by avoiding low-level operations wherever possible and exploiting the power of whatever high-level operations are available. It is worth noting, parenthetically, that traditional drafting (and hence design) practice is based almost entirely on the relatively simple constructions of lines and arcs that can be executed efficiently with straightedges and compasses and a basic knowledge of Euclid's Elements. But a computer can rapidly execute very complex constructions, so designers' geometric explorations no longer need be constrained by these ancient limitations. Some architects have begun to exploit this freedom.

Selecting, Transforming, and Duplicating Subshapes

Lines are put together (whether by hand or by using a computer drafting system) to construct more complex shapes such as rectangles, triangles, or complete building elevations. Any part of such a shape that can be traced is a subshape. Computer drafting systems provide not only for selecting and operating on points and lines, but also for selecting and operating on arbitrary subshapes. This greatly enhances their power. Most computer drafting systems formalize the concept of subshape by treating the complete drawing as a set of instances of line types, and subshapes therefore as subsets. These subsets form a Boolean lattice under the relation of shape inclusion, and the nodes in that lattice define all the possible subshapes. It follows that subshapes can be selected either by selecting constituent lines one by one or by specifying an area of drawing surface and counting all lines falling within that area as members of the subshape (or selection set, as it is often called). More complex subshapes may be selected by adding lines to a selection set or deleting lines from it. This approach is not entirely satisfactory, however, since many of the subshapes that may be of interest to a designer are not subsets of line primitive and cannot directly be selected in these ways. The alternative is to extend the idea of constraint-based editing one step further and provide a facility for specifying the subshape types that are currently of interest and for automatically recognizing emergent instances of them. Designers most frequently pick out subshapes because they want to transform them in some way . Drafting systems provide tools for translating, rotating, reflecting, scaling, and distorting specified subshapes, and use of these tools to manipulate entire subshapes usually proves to be a much more efficient way of modifying a drawing than modifying lines one by one. Since values for transformation parameters are frequently defined by existing geometry, it is often very efficient to specify transformations by snapping or by pointing to existing geometric entities to describe distances and angles. Simple systems allow translation, rotation, reflection, and scaling operations to be performed individually. More sophisticated systems allow arbitrary sequences of these transformations (concatenated linear transformations) to be specified, named, and recalled and applied. Internally, the task of performing geometric transformations is essentially one of multiplying the coordinate pairs that define the original shapes by transformation matrices to produce transformed coordinate pairs that

define the transformed shapes. This highly standardized and repetitive task can be performed by software or, for greater speed, by specialized graphics hardware. Drafting systems also provide tools for replicating (and deleting) complete shapes. It is not very useful to replicate a shape directly on top of itself, so replication is usually combined with some transformation. Combination of replication with translation provides an extremely efficient way to generate rows and grids of shapes. Similarly, combination of replication with rotation and reflection is an effective way to generate figures with symmetry about points and axes .

Repeatable Standard Shapes

Some types of subshapes are used very frequently in drawings, so drafting systems are often elaborated to provide for instantiation of complete subshapes selected from a standard menu--much as a drafter working by hand might trace stencils or cut and paste photocopies to produce instances of common shapes. Some simple shapes, such as rectangles, are used in many different contexts. But others are more specialized: doorswing symbols are commonly used in architectural plans, resistor symbols are commonly used in electrical diagrams, boxes and arrows are commonly used in flowcharts, and so on. So drafting systems usually provide small menus of repeatable standard shapes for general use, plus much more extensive custom menus for particular applications. In the more sophisticated systems, users can extend and customize these menus themselves. Documentation for different drafting systems refers to these standard shapes variously as cells, components, instances, and groups. However the IGES (International Graphic Exchange Standard) standard for graphic data transfer calls them segments, and this terminology is now widely used. Segments are defined in their own local coordinate systems and are instantiated in drawings by specifying values for location parameters: X-coordinate, Y-coordinate, orientation, and handedness. The handedness parameter allows an element such as a doorswing to be instantiated in both right-handed and left-handed versions. Where a design is built up from repeating elements, use of segments provides a very efficient drawing construction method--particularly when it is combined with some of the other techniques that we have considered. Segments can be snapped into place on grids and skeletons of construction

lines, and arrangements of segments can be replicated and transformed as higher-level subsystems. As J-N-L Durand demonstrated in his Precis and other texts, complex designs can often be parsed into hierarchies of repeating subsystems. Skeletons of construction lines define the geometric relationships between subsystems. At the lowest level of the hierarchy there are very basic standard elements. A computer drafting strategy based on use of segments, snapping to constructed points and lines, and replication and transformation of increasingly complex subsystems is directly in the tradition of Durand.

Parametric Variation

The usefulness of a standard shape to a designer is much enhanced if instances of that shape can be varied appropriately to fit many different contexts: a doorswing that can fit to an opening of any width is more useful than one of fixed width. So standard shapes are usually parameterized for adaptability. Consider, for example, a semicircular arch. Obvious parameters are span, thickness, and number of voussoirs. By varying these parameters, while preserving the essential form of the arch, we can produce a wide range of instances. Drafting systems that provide vocabularies of parameterized shapes allow the user to specify parameter values and have the capacity to produce the corresponding instance automatically--a process known as parametric variation. This is accomplished by treating some of the entries in the data structure as variables rather than constants and by specifying that the values of some variables are dependent on the values of others. The form of the dependency is described by a numerical function. To instantiate or modify a parameterized shape, the designer simply specifies values for the parameters by typing them in, by moving a slider, or by selecting and moving a point on the shape itself. Values of dependent variables are then calculated, and the corresponding instance is displayed. Drafting systems with simple data structures typically use parametric shape procedures merely as data-input tools--"adjustable stencils"--that create sets of lines and add them to the data structure. Under this arrangement, relationships between shape variables are not recorded as part of the design, and shapes cannot be varied parametrically after they are inserted. Systems with more sophisticated data structures store parameter values in the data structure and execute parametric procedures whenever instances of parametric shapes need to be displayed. Thus relationships between shape

variables are permanently recorded, and shapes can be varied parametrically at any time. The most comprehensive and rigorous approach to implementation of parametric variation capability provides for storage of relationships both within and between parametric shapes. In other words, the user can specify functions that make the parameters of one shape dependent on the parameters of another shape and can vary the parameters of this function to create different instances of the relationship. For example, a parameterized rectangle and a parameterized ellipse might be put into a coaxial relationship. Thus the coordinates and orientation of the ellipse are made dependent on the coordinates and orientation of the rectangle, and the parameter of the relationship is the center-to-center distance along the common axis. The relationship can be depicted by drawing the axis and showing a dimension arrow. An interface might allow for selecting and dragging the tips of the dimension arrow to vary the relationship. Many of the shape-to-shape relationships that designers need to define are quite simple and can easily be shown graphically by analogy with common mechanisms. If a point on one shape is "pinned" to a point on the other, then the shapes can rotate in relation to each other. If a straight line on one shape is "hinged" with a straight line on the other, then the two shapes can reflect in relation to each other. If a straight line on one shape can "slide" along a straight line on the other, then the two shapes can translate in relation to each other. And an adjustable pair of lines diverging from a center of scaling can be used to vary a proportion relationship. Tick marks can be placed on these regulating lines to show the limits of variation. The idea can be extended indefinitely by allowing locations and orientations of some regulating lines to control locations and orientations of other regulating lines. (The idea of regulating lines is, of course, an old one: both Durand and Le Corbusier relied heavily on it. But it becomes much more useful and interesting when lines are used not just as a rigid construction skeleton, but to regulate the behavior of a drawing and maintain its essential structure as parts are manipulated.) Some drafting systems provide fixed vocabularies of parameterized standard shapes and regulating lines which you can "click together." Others allow you to extend the vocabulary as required by programming your own through spreadsheet interfaces or by providing a general-purpose programming language (such as Pascal or Lisp) that can access the data structure.

Constraint Solving

Now consider a relation of the form a + b = c between three shape parameters. Such relations are called constraints, since assignment of a value to any variable restricts or determines the values that can consistently be taken by the others. In this particular case, assignment of values to any two of the variables determines the value of the third, as we can see by transposing the expression to read a - c = b or b - c = a. In general, any kind of arithmetic expression can appear in a constraint, and the relational operator linking the two sides may specify equality, that one side is less than the other, or that one side is greater than the other. The most general and flexible way to provide for parametric shape variation in a drafting system is to allow the user to specify arbitrary constraints on the locations of line end points (and additional defining points in the case of curves) and automatically to adjust end-point locations as necessary to maintain consistency whenever the user selects and moves any one of the constrained points. Software systems that perform the necessary calculations are known as constraint solvers, and these are an increasingly standard feature of advanced drafting systems. Depending on the nature of the specified constraints, a shape may be underconstrained so that there are many ways to adjust it to maintain consistency, it may be uniquely constrained so that there is just one way to adjust it, or it may be overconstrained so that there is no way to adjust it. You can discover how much freedom there is in the shape by experimentally attempting to move points and letting the constraint solver adjust it in response. General, efficient constraint solvers capable of handling realistic design problems are exceptionally difficult to implement: they can become formidably complex, and the computations required to find solutions to constraint problems can be extensive. So, although integration of constraint solvers into drafting systems was first proposed in the early 1960s, it was not until the late 1980s that robust, practical constraint solvers for drafting systems became commercially available.

Syntax-directed Editing

Just as the idea of constraint-based point selection and constraint-based line selection can be extended to constraint-based subshape selection, so the idea of geometric

construction (putting lines in specified types of relationships with other lines) can be generalized to putting subshapes into specified types of relationships with other subshapes. Consider the possible relationships of a square to another square, for example. These include face adjacency, vertex adjacency, parallel-sided and concentric, rotated and concentric, and face to diagonal. A simple syntax-directed drafting system that provided a square as the vocabulary element and tools for inserting squares in these relationships would support very efficient construction of the types of compositions. (The term "syntax directed" is used because such relationships define the syntax of compositions made from vocabulary elements.) Deletion of subshapes may also be syntax directed. In other words, the system may provide operations for deleting specified parts of certain types of subshapes. In general, a syntax-directed editing operation may be described as a rule for replacing one type of subshape with another. The rule can be applied to any subshape that matches the left-hand side. The result of application may be insertion of a new subshape, deletion of part or all of the existing subshape, or some combination of insertion and deletion. Syntax-directed editing rules customize a drafting system for very efficient construction of designs of a certain type. (They amount to shape grammars that specify languages of line compositions.) There may be rules for laying out floor plans, developing window details, and so on.

Interface Dynamics

Traditional drawings are static: lines remain fixed in place on the paper and can be altered only by laboriously erasing and redrawing. But the line structures maintained and displayed by a drafting system are dynamic: they can be varied rapidly and continuously. Increasingly, as computers have become faster and display technology has become more sophisticated, drafting system software has reduced its reliance on metaphors carried over from traditional drafting and has exploited the potentials of dynamic interaction. Use of a highly dynamic drafting system not only is quicker than traditional drafting (or use of the older style of computer drafting system), but also provides a qualitatively different way to explore shape, dimension, and geometric organization. The most elementary use of dynamic interaction is in line-insertion operations. Most systems provide rubber-band straight lines and variants of the same

technique for inserting circular arcs, elliptical arcs, and rectangular boxes. More sophisticated systems also provide for real-time manipulation of spline curves by selecting and tweaking control points. The most sophisticated systems evaluate dependent variables or even solve constraint systems with sufficient speed to support real-time parametric variation of complex shapes. In older systems geometric transformations of shapes were first specified by a typed command and then executed--often with significant delay. But in newer systems translations, rotations, and resizings of complex shapes are accomplished in real time by selecting the shape with a mouse or stylus then continuously dragging into position. Even reflection operations can be performed continuously (although they actually amount to rotating a shape out of the plane) by dragging the corner of a selection rectangle back across one of that rectangle's edges. This lets a designer observe a continuum of possibilities. But the real potential of dynamic interaction emerges when it is combined with snapping, geometric construction, and syntax-directed editing. In the simplest case, a rubber-band line or dragged shape snaps from grid location to grid location instead of varying continuously. In a more sophisticated syntax-directed system, the user specifies the spatial relationships that are of interest, and a line or shape snaps to the nearest one as the cursor moves around. As a rubber-band line moves across a complex drawing, for example, it might snap to the end points of existing lines, to the center and tangent points of existing arcs, and into parallel and perpendicular relationships with existing lines. In traditional drafting, lines are fitted into place one by one--like the stones of a pyramid. But in dynamic drafting, a designer specifies the elements and constraints of mechanisms, then explores adjustments of those mechanisms to join them together and appropriately embed them in particular contexts. A drawing is not a fixed structure of lines, but the current state of a line mechanism that has been organized to behave in a particular way.

Structuring Drawings

Lengthy texts are usually subdivided into named parts, such as chapters, then these parts may be further subdivided into subsections, and so on. (This book, for example, has a two-level hierarchy described by a table of contents.) Similarly, it is convenient to subdivide a large and complex drawing into named parts that have related

content. Each of these parts is a subset of the set of points and lines constituting the complete drawing. One way to represent the assignment of entities to subsets is to store each subset of graphic entities in a separate, appropriately named file. Another possibility is to store all the entities in a single file, but to add a field for a subset name to each entity record. In either case, the effect is to assign each entity to one and only one subset. By analogy (though an imperfect one) with the traditional use in technical drafting of overlaid sheets of transparent paper, subsets of graphic entities are usually referred to as drawing "layers". Assigning an entity to a layer, then, is like drawing it on a particular sheet. Drafting systems provide mechanisms (which range from elementary to very elaborate and sophisticated) for organizing drawings in this way. At a minimum, they provide for creating and naming layers, working in specified layers or combinations of layers, shifting elements and shapes from layer to layer, and selecting layers or combinations of layers for display, editing, or printing. Flexible layer-selection capabilities become particularly important when large, complex drawings must be displayed and edited on small computer screens. More sophisticated systems provide for partitioning of layers into sublayers, then further partitioning of sublayers, and so on to produce a hierarchy of graphic parts. This provides the user with an efficient way to retrieve parts of a large drawing, in the same way that the hierarchical organization of a large library according to the Dewey Decimal System provides an efficient way to retrieve books. However, the simple "transparent paper" metaphor begins to break down when layers are organized in this fashion. In general, a drafting system's drawing organization capabilities need not be constrained by the old-fashioned and inadequate "transparent paper layer" metaphor. A more flexible strategy is to allow association of multiple nongeometric attribute fields with each graphic entity. Some of these fields may contain values needed to control graphic display and access. Others may be defined by the user for the purpose of associating identifiers--floor number, room number, subsystem, supplier, and so on--with each entity record. Entities may then be selected, sorted, and displayed by any specified combination of these identifiers. In other words, the drawing is treated as a graphic database from which different types of reports are generated for different purposes. Drawings organized in this way are content addressable rather than location addressable: you retrieve and display drawing elements according to what they

are, rather than according to where you put them. The idea of a drawing as graphic database can be taken one useful step further by allowing association with graphic entities of data fields for costs, specification clauses, text notes, and the like. This facilitates coordination of drawings, specifications, and other documents, and (with appropriate data-extraction software) allows the graphic database to serve as a source of input for costing, schedule production, and other programs. The approach is of limited utility, however, since drafted drawings typically do not provide a sufficiently complete, nonredundant, consistent description of a project. (Later we shall see that appropriately structured three-dimensional geometric models can do so.) Different design tasks demand different organizations of a drawing. A simple diagram might be drafted on a single layer. An elementary organization into a small number of layers usually suffices for development of architectural plans and elevations. But very large, complex, technical drawings usually benefit from careful organization into sophisticated graphic databases. The drawing-organizational tools provided by drafting software are correspondingly varied. Inexpensive systems, designed for performing simple tasks on personal computers, usually provide only basic layering schemes. But systems designed for handling large and complex projects need to provide general, flexible database-management tools equal to such tasks.

Formatting Drawings

Recall the way that, in text processing, the stored text may be given particular typographic form and layout for display and printing. Similarly, with drafting systems, a file of abstract geometric information may be given specific graphic form. Shapes may be laid out at particular scales on surfaces of specified dimensions, lines may be given particular weights and styles, and details such as title blocks may be added. In traditional drafting a sheet size and scale must be selected before you begin drawing, and you cannot change the weight or style of a line once it has been drawn (except by laborious erasing and redrawing); but a computer drafting system allows you to make or change formatting decisions at any time and to generate displays and drawings at different scales and in different formats from the same file. This yields large efficiencies. It also facilitates making graphic design decisions, since you can easily try out different line weights and so

on, and judge their effect on the appearance of the complete drawing. One important consequence of this separation between specification of geometric information and formatting of drawings is that you do not work (as you must, in traditional drafting) at a specific, fixed scale. Instead, you specify actual (full-scale) dimensions and coordinates, and map this information at any appropriate scale whenever you need to produce a display or plotted drawing. Display software treats the window in which the drawing is displayed as the viewfinder of a camera with which you can track and zoom freely across the surface of the drawing. You can zoom in to work on a detail, or you can zoom out to see the whole drawing. You may want to have several windows open at once--one providing an overview of the entire drawing, and the others zoomed in to various details. This separation of dimensional definition from scaling for display allows some useful freedoms in the handling of dimensional information. You can, for example, work in a dimensionless conceptual unit such as a module or bay and only later give a design definite dimensions by assigning a value to this unit. Or you can design a building in feet and inches but produce the drawings at a metric scale. Another difference from traditional drafting is that you can precisely control the kind and amount of graphic information displayed at a particular moment or shown on a print that is made for a particular purpose. Where a layering scheme is used you accomplish this simply by switching layers on and off to yield different combinations: if your drawing is subdivided into n layers you will be able to display or print 2n different combinations of layers. Where more sophisticated database-management techniques are used, you specify exactly what kinds of things you want to see and the software responds by searching the database to produce an appropriate graphic report. Selective display is not just a convenience, it can also be an important design-analysis tool. If you want to study the relationship between, say, the circulation system and the structure of a building, you can generate a display or print showing just the circulation and structural elements.

Printing and Plotting

Printers and plotters are, like traditional devices such as ruling, Graphos, and Rapidograph pens, devices for precisely placing ink on paper. The difference is that they are controlled by streams of commands sent from a computer rather than by a human

hand and eye. They vary widely in cost, speed, reliability, sheet size, line quality, and the precision of control that they offer. Electromechanical pen plotters, which emerged in the earliest days of computer graphics, provided for many years the standard way to produce output from drafting systems, and they still have their uses. They literally embody the idea that a line is the visible trajectory of a moving point, since they all work by moving a pen or marker across a drawing surface, though there is considerable variation in the specific arrangements of gantries, drums, and so on used to achieve this. The pen is moved in a raised position to locate it at the start of a line, then moved in a lowered position to draw the line. Different line weights and colors are produced through some arrangement for selecting different pens. Pen plotters can produce large, precisely drafted ink drawings on standard drafting film, so they fit very easily into design offices that still make extensive use of traditional drafting and reprographic techniques. The plotted drawings can be stored and printed in the usual ways and can be finished or corrected by hand if this is desired. Since pen plotters have many mechanical components, however, they cannot be produced very inexpensively, and they are intrinsically limited in both speed and reliability. This means that they are increasingly being displaced as more fully electronic raster printing devices develop in capacity and sophistication and drop in cost. Some people find the precise, mechanical quality of pen plotters unattractive and prefer the slightly wobbly line work and imprecise endings of hand-drawn lines. Pen-plotted drawings can be "humanized," in a disconcertingly convincing way, by mounting a pen loosely so that it shakes a little as it moves. Raster printers and plotters work by depositing tiny dots of ink to build up lines, characters, and halftone screens. Many different transfer mechanisms are used: impact, thermal, electrostatic, inkjet, and laser. Resolution varies (depending on the technology) from less than one hundred dots per inch to several thousand dots per inch, and paper formats range from letter size up to the sizes handled by the largest pen plotters. In the past the use of raster printers and plotters was limited by the need to execute a slow and expensive process of rasterizing line data (converting it to patterns of dots) before printing. Large, high-resolution drawings presented a particular problem, since they translated into huge quantities of raster data. As processor and memory costs have dropped, however, rasterization processes have become quicker and cheaper--making raster printing and

plotting increasingly attractive. Raster printers and plotters are much more versatile than pen plotters, since lines, characters, and halftone screens can all be built up from dots. There is no need to use crosshatching to produce toned areas on drawings, and the characters used in annotation text can be in standard fonts (not constructed from small numbers of pen strokes). So many of the old conventions of technical drafting, which derive from the physical limitations of technical pens and the need to minimize penstrokes, can be abandoned when raster printers and plotters are used for drawing production.

Automated Measurement and Analysis

Designers do not just construct and produce drawings, they also measure and analyze them. (Indeed, one of the fundamental purposes of precise construction is to allow accurate measurement.) They measure lengths, angles, and areas, and they count instances of things. Traditionally, the have used tools such as graduated scale rules, protractors, and planimeters to accomplish these tasks, together with slide rules or electronic calculators to derive additional quantities from these basic measurements.

Drafting systems provide measurement capabilities through application of arithmetic procedures to values stored in the data structure. The most elementary capability is that of reporting the numerical coordinates of a selected point. The capability of a scale rule can be provided by a simple procedure that calculates and reports the distance between two selected points and that of a protractor by a procedure that calculates and reports the angle between two straight lines. More complex procedures can be implemented to calculate distances along the various types of curved lines provided by a system. In systems that provide parametric shapes, parameter values can be processed not only to produce shape instances, but also to yield properties of those instances. If you know the length and width of a rectangle, for example, you can calculate its area, perimeter, proportion, area/perimeter ratio, and so on. The data structure can also be scanned to produce counts of the instances of specified types of subshapes. Note, however, that counting procedures that recognize emergent instances will, in general, produce different results from procedures that only take account of instances that are explicitly represented in the data structure. The simplest measurement facilities merely

report measured values on the screen. More ambitious systems provide interfaces to spreadsheets or programming languages so that further analyses can be developed from the extracted values. A spreadsheet might be used, for example, to apply cost coefficients to measured plan areas and report rough cost estimates.

Uses and Limitations of Two-dimensional Drawings

All models are abstractions from the full complexity of reality: they specify some properties and relationships of real objects completely and accurately, but distort others or leave them out entirely. An appropriate model for some particular purpose conveniently represents just those properties and relationships that are relevant to that purpose, but at the same time achieves clarity and economy by leaving out everything that is not. A model that consists of a set of two-dimensional line drawings of a building or other design is a particularly highly abstracted representation, and this is the source of both its major strengths and its most obvious weaknesses. When it explicitly presents and appropriately structures precisely the information that matters to an architect or engineer at a particular stage in the building design process, it is particularly convenient and economical. But because it deals with only a few aspects of a very complex reality, there are many important design activities that it cannot support. Since the data structure of a two-dimensional drafting system represents surfaces and solids only by their edge lines, and most edge lines only by their end points, and since it represents three-dimensional objects only in two-dimensional projection, it has the virtues of parsimony. It is relatively quick and easy to construct (by comparison with the more complete three-dimensional representations that we will consider later), economical in use of memory, and rapidly manipulable. Since coordinates are stored with high precision it allows sizes, shapes, and locations of elements in plan, section, and elevation to be specified with great accuracy. (This is not possible in a paint system, where precision is severely restricted by the limited size of the raster grid.) The main disadvantage of such an abstract and economical representation is that the viewer must "fill in" a great deal of information to interpret two-dimensional shapes as projections of three-dimensional objects and lines as boundaries of surfaces and solids. Misinterpretation is possible (especially if the delineator and viewer do not share a common understanding of architectural forms and construction processes),

and there is considerable danger that ambiguities and inconsistencies will escape notice until it is too late to avoid damaging consequences. Furthermore, where a complete and consistent geometric description of a design is needed as input to a procedure that performs some design analysis or synthesis task, a drafting system model usually cannot provide it. The value of models that have greater geometric completeness should not be overstated, however. Plan, section, and elevation drawn precisely in line are not just deficient representations of three-dimensional form (as proponents of more elaborate three-dimensional modeling systems sometimes like to suggest): they are, when properly constructed and used, powerful abstractions that allow the designer to focus on issues of central importance. As Le Corbusier remarked, in Vers une architecture: To make a plan is to determine and fix ideas. It is to have had ideas. . . . A plan is to some extent a summary like an analytical contents table. In a form so condensed that it seems as clear as crystal and like a geometric figure, it contains an enormous quantity of ideas and the impulse of an intention. But, as he also emphasized, exploration of architectural ideas does not stop at this level of abstraction: The plan is the generator, "the plan is the determination of everything; it is an austere abstraction, and cold of aspect." It is a plan of battle. The battle follows and that is the great moment. The battle is composed of the impact of masses in space. Thus a two-dimensional drafting system is most appropriately used in design at a relatively early stage when, after some initial unstructured exploration (perhaps by freehand sketching on paper or with a paint system), it is time to "determine and fix ideas" with some precision. Then, to engage the "battle" that follows, it becomes more useful to employ less abstract models--models that represent lines, surfaces, and volumes in three-dimensional space and that show the effects of surfaces in light. Later, when a design has been completed, the precise, parsimonious character of drafted plans, sections, and elevations again becomes appropriate for expression of definitive construction information.

It would be a serious mistake to think that use of a computer for drafting such two-dimensional line representations merely results in quicker production of finished drawings. Indeed, efficiency in drawing production is no more than a useful byproduct. The real significance of computer use for drafting is that static, location-addressable, fixed-format, non-machine-analyzable design representations give way to dynamic,

content-addressable, variable-format, machine-analyzable representations. This provides more effective support of design exploration, graphic problem-solving, and analysis.

Geometry of Curved Space

The large scale geometry of the universe is governed by Einstein's General Theory of Relativity. Einstein showed that gravity curves three-dimensional space, and that space in turn moves matter. For the universe as a whole, the shape of the curvature depends on the average density of the matter.

If the average density of matter in the universe is greater than the critical density, the force of gravity will eventually rein in expansion and cause the universe to collapse upon itself. In this case, the universe is said to be positively curved, and Omega, the ratio of the average density to the critical density, is greater than 1.

Conversely, if the average density of matter in the universe is less than the critical density, gravity will lose its grip on matter and the universe will expand forever. This negatively curved universe is defined by an Omega less than one.

If Omega is exactly one--that is, if the average density of the universe is equal to the critical density--then the universe will expand to a maximum density and remain there for eternity. This universe is flat; it has zero curvature.

Cosmos Curvatures

Now three dimensional curved space is difficult to visualize, but we can illustrate the curvature in two dimensions. A positively curved universe is like the surface of a sphere; a negatively curved universe, like a saddle. A universe with zero curvature is, not surprisingly, like a plane. If you could draw a triangle by reaching far enough into space to draw lines connecting three far-flung galaxies, you could determine the curvature of the universe. The angles of a triangle in a negatively curved universe would add to greater than 180 degrees; those of a triangle in a positively curved universe, to less than 180 degrees. In

a flat universe, familiar Euclidean geometry applies, and the angles of the triangle add up to exactly 180 degrees.

Can We Measure the Curvature?

In principle, cosmologists can determine the curvature of three dimensional space by similar means using volumes rather than areas. Unfortunately, geometric methods have proven impractical because they rely on a uniform distribution of galaxies throughout the universe which does not evolve with time--assumptions which are false since observations indicate that the galaxies have a finite age and have changed over the eons.

Lines (and planes)

So far, we have discussed the advantage of using lines in our object manipulations; namely, that one only needs to move (or project) the endpoints and the line will shift as well. This has an advantage over manipulations of curves, in which every point on the curve needs to be manipulated. Lines are relatively simple objects, but they still possess an amount of complexity. It is no small task, for example, to code a program that will plot a line segment on a computer screen. However, I have the advantage of using a programming language that has a built-in function for plotting lines. There are two ways of expressing the formula for a line. The first is easier to use, but the second will help us to better understand higher dimensions. In two dimensions, the formula for a line is often expressed in the explicit form `y=mx+b`. No explicit equation can define all three variables of a line in three dimensions. Instead, we use implicit equations which describe the line in terms of each variable, x, y, and z, in relation to a particular value, t, that defines the position along the line. (The line represents a one-dimensional array and t expresses the position in that dimension.) The orientation of the line (or one-dimensional array) is expressed in three dimensions as a vector with A, B, and C values which correspond to the x, y, and z axes, in relation to a given point on the line `(x0,y0,z0)`.

A three dimensional line is represented by the following parametric equations:

```
x=x0+At
```

$y=y0+Bt$

$z=z0+Ct$

To more fully understand the equations of a line and a plane, we will consider two forms of the explicit function for a line in two dimensions. A line can be represented by a vector which is a scalar of the line (in other words, points in exactly the same direction/is parallel). Any point on the line $(x1,y1)$ can be expressed in relation to another given point $(x0,y0)$ on the line and to the vector $AI+BJ=K$. The equation would be $y=B/A\ x+b$. Variable b, the y-intercept, is found $b=y0-B/A\ x0$.

A line represented by a point on the line and a vector scalar to the line. The formula can be written $y= mx+b$ or $y= B/A\ x+y0-B/A\ x0$. Parametric form is $x=x0+At,\ y=y0+Bt$.

Another way of specifying the equation in two dimensions would be to use the normal vector, or a vector which is perpendicular to the line. Again, a starting point on the line $(x0,y0)$ is used as a reference. This time, the formula comes out :

$y=-A/B\ x\ +\ y0\ +\ A/B\ x0.$

By introducing the quantity $C=-Ax0-By0$, and substituting into the above equation, you get $y=-A/Bx-C/B$. Multiplying the equation by B (mult. property of equalities), and placing all variables on one side of the equation results in the formula $Ax+By+C=0$. This is very similar to the formula for a plane in three dimensions, often written in the form $Ax+By+Cz+D=0$. It is easy to see from the above demonstration that the above plane has a normal vector $AI+BJ+CK=L$. In the equation, $D=-Ax0-By0-Cz0$. From this it is easy to see that the equation for a hyperplane in four dimensions is derived from a normal vector $AI+BJ+CK+DL=M$ and a starting point $(x0,y0,z0,w0)$. Naming $E=-Ax0-By0-Cz0-Dw0$, the equation for the hyperplane would be $Ax+By+Cz+Dw+E=0$. Extrapolating further dimensions is a snap.

Intersection of a line and a (hyper)plane

The equations for lines and planes have one important application in this project, as I will shortly demonstrate. We have already arrived at a way to project images from another dimension, the first principal way of rendering higher-dimensional forms. We can make four-dimensional models visible on a two-dimensional computer screen by first projecting them into three dimensions, and then projecting three dimensions into two. This loses some of the effect we would have if we could somehow display the model in three dimensions. However, the ability to rotate the model and view it from different angles mostly makes up for this.

As I mentioned at the outset, I also intended to show the representation of a four-dimensional model by slicing the portion that appears in one hyperplane, or three-space. This slice is then displayed using projection, as a two-dimensional rendering. Again, this rendering can be rotated from any angle, giving the illusion of viewing a three-dimensional object.

Calculations for the intersection of a line and a plane (or hyperplane) in 3 and 4 dimensions are crucial to the logic of the above process. This is because the slice, or intersection, is found by detecting the point(s) of intersection between each polygon and a given plane. In order to do this, we must analyze each line of each polygon, and record the point(s) where it intersects the plane.

After we have found the intersection points of a polygon with the plane, we can plot a line or lines between them, demonstrating the line(s) of intersection. When each polygon is thus processed, all of the lines of intersection of the object with the plane will be displayed, often creating a new polygon or polygons. If we use the plane z=0 for clipping, with normal vector `0I+0J+1K`, with `(0,0,0)` as the reference point, we can simplify the above equation to `t=-z1/(z2-z1)`. A value of `0<t<1` means that the intersection occurs between the two endpoints of the line segment. This is very useful for our polygon-clipping routine, because we are not interested in any intersections that take place outside the polygon.

To find the actual intersection points, we substitute the above-simplified t value back into our original equation for line :

`x=x1+(x2-x1)t, y=y1+(y2-y1)t`, and `z=z1+(z2-z1)t`.

This method has been used in the accompanying computer program, and the results are, in my humble opinion, rather spectacular.

Visualisation data and its Representation

Graphics and visualisation is about transformation and mapping - normally mapping information into some graphics primitives. It is from one data representation into another. What are the common forms of data one encounters in visualisation?

Characterising data

The representation schemes require the design of efficient data structures with efficient access methods. By understanding the character of the data we need to visualise, we (hopefully) avoid designing inflexible and limited visualisation systems.

Visualisation data is discrete: A consequence of digital computing. This means our data must be *sampled* information, and at a finite number of points. So invariably continuous data (in the real world) is represented by discrete sampling. We can then join the sampled data values with straight lines, or curved segments to obtain the more appealing However we have made strong presumptions about the relationship between between neighbouring data points, to apply the interpolation scheme we have.

Under discrete sampling we have measured no data in the regions between the data points. However a primary purpose of visualisation is to "see" or "measure" data values at arbitrary positions. Obviously the solution is to interpolate - but we need to have a good understanding of the general behaviour of the underlying function our data represents, *eg.* is it monotonically increasing (decreasing), is it approximately linear over sufficiently small intervals.

The data is either **regular** or **irregular.**

Synonymous terms are *structured* and *unstructured*. *Regular* data has an inherent relationship between the data points. In the previous example the function was sampled

over equally spaced intervals. It does not need to be equally spaced to be *regular*, there just needs to be a known relationship between the data points. For instance, sequentially sampling at an interval twice as wide as the previous, will be regular.

The advantage of regular data is that one does not need to store all the point coordinates. The initial coordinate, the interval (or relationship if it is not equally spaced) and the number of points is sufficient. We can store regular data more compactly, and many visualisation algorithms work more efficiently with regular data.

For irregular data there is no simple, known relationship between the data points. The advantage of this type of data is that one can represent information more densely where it is more rapidly varying (improving the interpolation quality) and less dense where the change is not as large.

The above are the most common cell types used to represent the geometry to the graphics subsystem. The first three categories form the basis of hardware acceleration, that is they are drawn without having to worry about the details of setting pixel colors. General polygon hardware acceleration can be found in higher end graphics systems.

Triangle strips (adjacent triangles share an edge) are a very efficient representation, requiring n + 2 points to represent n triangles. They can also represent non-planar surfaces (OpenGL also supports a triangle-fan cell, where adjacent triangles share an edge and all triangles share a common vertex).

Other cell types to consider :

- Quadrilateral
- Pixel
- Tetrahedron
- Hexahedron
- Voxel

Such topology types are generally not hardware accelerated, but they may be a better description of the data you have. Support for them would be in software.

Attribute types

Attribute data is usually associated with the data set points or cells, but can be assigned to the cell components, such as the edges or the faces.

Examples of attribute types are, the temperature or velocity at a point, mass of a cell or the heat flux across the face of a cell. The categories of attribute types are,

Lines in Space

Renaissance texts on perspective frequently depicted buildings as "wireframes"-collections of lines in three-dimensional space that had been projected in perspective onto a two-dimensional picture plane. Similarly, in computer-aided design, the idea of a two-dimensional drafting system can readily be generalized to that of a three-dimensional wireframe-modeling system. This is accomplished by representing lines within a three-dimensional rather than a two-dimensional Cartesian coordinate system, providing corresponding editing operations, and providing software for producing perspective and other projections on the display screen from geometric information stored in the three-dimensional database.

Construction Planes

Techniques for specification of points and lines in a three-dimensional Cartesian coordinate system are straightforward extensions of those used in two-dimensional drafting. Points can be specified by entering numerical X, Y, and Z values from a keyboard or by snapping to previously established points. A special three-dimensional digitizing device can be used in place of a two-dimensional digitizing tablet, or (more commonly) some scheme can be used to adapt a standard mouse or digitizing tablet for specification of points in three-dimensional space. The usual way to adapt a mouse or tablet for this purpose is to introduce the idea of construction planes. Imagine a large sheet of glass located arbitrarily in space. You could locate points and lines in space by two-dimensional

drafting on the surface of that sheet. Then, by inserting additional sheets at different locations, you could locate points and lines in other planes. Through use of enough of these construction planes you could build up a three-dimensional arrangement of lines specifying the geometry of a building. Three-dimensional geometric-modeling systems usually provide a set of commands for specifying a construction plane--by giving the coordinates of three points on it or by tilting up from another plane, for example. Using these commands, you can make a construction plane coincide with any desired surface in a design's geometry . Once a plane has been defined, you can name it and save it for future recall. By defining several construction planes, and moving back and forth among them, you can manipulate a three-dimensional model with ease. Technically, a construction plane is simply a local two-dimensional coordinate system. Two-dimensional coordinates from a mouse or tablet are interpreted by the software as points in the current construction plane and are automatically converted into three-dimensional global coordinates. In other words, coordinates are transformed as follows:

[X_local, Y_local] --> [X_global, Y_global, Z_global]

It is usually most convenient to organize the graphic interface so that the currently active construction plane coincides with the plane of the display screen.

Glass-sheet Models

Dimensions and construction lines can be carried over from one construction plane to another. Consider, for example, a horizontal plan-construction plane intersected by a vertical section-construction plane as illustrated. When the section-construction plane is being used, new lines can be snapped to the intersections of existing plan lines with the section plane. Similarly, when the plan-construction plane is being used, new lines can be snapped to the intersections of existing section lines with the plan plane. This makes it very easy to work back and forth between plan, section, and elevation to produce a "glass-sheet" model in which a building's essential geometric organization is defined by shaping and positioning profiles and contours in intersecting horizontal and vertical planes.

Three-dimensional Geometric Transformations

Just as a drafting system provides operations for copying, translating, rotating, reflecting, and scaling two-dimensional shapes in the plane, so a three-dimensional wireframe-modeling system provides these operations for constructing and arranging three-dimensional line shapes in three-dimensional space. They can be used to move shapes out of their original construction planes--to develop a glass-sheet model into a complete wireframe. In particular, a plan arrangement is often developed in the third dimension by translating shapes various distances in a direction normal to a horizontal plan-construction plane. Similarly, elevations and sections may be developed by translating shapes in directions normal to vertical construction planes. In these cases the transformations that are applied record the designer's decisions about height, depth, and breadth. Another common move is to rotate elements about axes in their construction planes. By setting up appropriate construction planes, by constructing plan, section, and elevation profiles and contours, by copying and transforming, and by inserting lines between established points, you can quickly delineate the boundaries of three-dimensional forms. For example, a vertical prism might be constructed by first setting out a polygon in a horizontal construction plane to define the base, then copying and translating to define the top, and finally inserting connecting lines to define the vertical faces. Variants such as pyramidal forms, wedge-shaped forms, twisted forms, and antiprisms can be produced by simple variations of the transformation and connecting operations. Reflection operations in three dimensions take place across specified planes, rather than across lines as in two-dimensions. Scale, stretch, shear, and perspective-distortion operations can also be applied to wireframe shapes to complete the repertoire of linear transformations

Sweeping Points

When a straight line is translated out of a construction plane, its end points sweep out two new straight lines .Thus the original line, the translated line, and these two new lines together describe the edges of a rectangle. This idea can be generalized: a translated polyline sweeps out the edges of a fence-shaped object, and a translated polygon sweeps out the edges of a prism. Many wireframe-modeling systems, then, provide the translational sweep operation for quick construction of "fences" and the edges of prismatic objects. (This operation eliminates the task of explicitly inserting connecting lines between

the original and translated shapes.) Similarly, when a straight line is rotated out of a construction plane, its end points sweep out two arcs .The original line, the translated line, and the two arcs together describe the edges of a cylindrical surface. This rotational sweep operation is also commonly provided by wireframe modelers. If a profile consisting of connected lines and arcs is rotationally swept in increments, a surface mesh consisting of patches bounded by arcs and straight lines is the result. A patch may bound a surface fragment of a cylinder, a sphere, a torus, or a cone. When a three-dimensional object is scaled, it is sometimes useful to connect automatically the end points of the original and the resized lines. Thus, for example, the stones of a hemispherical dome can be constructed by first rotationally sweeping an arc to construct a mesh describing the inner surface, then scaling and connecting to construct the outer surface and the radial edges of the stones

Space Curves

Most wireframe modelers provide only translational and rotational sweeps, but the idea can, in fact, be generalized endlessly. If a straight line is simultaneously translated and rotated, for example, each of its end points sweeps out a helix. Whereas straight lines and arcs are plane curves (you can always fit construction planes through them), a helix is a space curve--one that exists only in three-dimensional space. A limitless variety of space curves can be generated by procedures that calculate the coordinates of a point as some function of a parameter T. By incrementing the value of T through some range, a sequence of points on the curve is produced. Thus a procedure of this type is a tool for producing a space curve, in the same way that a straightedge is a tool for producing a straight line or a pair of compasses is a tool for producing an arc. If a wireframe-modeling system has an interface to a programming language, you can build your own tool kit of these procedures. In the past it has been extremely difficult for architects to compose with space curves, since traditional drawing instruments work on planar surfaces to produce plane curves. Wireframe modeling removes this limitation.

Structuring Wireframe Models

The ideas of grouping and layering generalize in a straightforward way from two-dimensional drafting to three-dimensional wireframe modeling. The metaphor of a "layer"

is less clear, however, since wireframes on different layers may intertwine in three-dimensional space. The idea of treating collections of lines as reusable vocabulary elements also generalizes. Lines may be grouped and copied, just as in two-dimensional drafting. And many wireframe modelers provide built-in vocabularies of basic shapes such as boxes, regular prisms, pyramids, wedges, spheres, cylinders, and cones.

Viewing

A wireframe model might be realized in three-dimensional form--literally as a set of wires in space, for example--but it is usually more convenient to work with two-dimensional projections in which lines of the model are projected onto a flat surface for display. Geometric-modeling systems provide software for displaying and plotting specified projections. Whereas a wireframe model is essentially a set of lines in three-dimensional Cartesian coordinate space, a drawing projected from that model is a corresponding set of lines in a two-dimensional Cartesian picture plane. Thus a projection method (such as orthographic, axonometric, or perspective) is simply a consistent rule for converting coordinate triples [X********** ********** ********** ********** ********** ********** ********** *** accepted as standard conventions. All the common projection methods produce linear transformations of the wireframe model: that is, they convert straight lines into straight lines--never into curves of some other kind. These projections can be visualized by imagining a picture plane located somewhere in the three-dimensional coordinate system and straight projection rays passing through the picture plane to connect end points in the three-dimensional model to end points of lines on the picture plane . To produce a display on the computer screen, this picture plane is brought into coincidence with the plane of the screen to produce the effect of looking through a "window" into a three-dimensional world.

To specify the location of the picture plane in the three-dimensional coordinate system, it is often convenient to take a specified point as the origin of an auxiliary polar coordinate system. The location of the picture plane can then be described in terms of azimuth, altitude, and distance. This is like moving a camera around a fixed object to produce views from different sides. Alternatively, the center of the picture plane can be taken as the fixed origin of a polar coordinate system, and translations and rotations of the three-dimensional model can be specified in this system to create the required relationship of plane and model. This is like holding the model in your hand and turning it around to look at it from different sides. The two ways of specifying the relationship between model and picture plane are obviously mathematically equivalent. Some wireframe modelers provide one, some provide the other, and some provide both. Viewer-centered viewing is natural when you want to think of a design as an object, as is often the case in mechanical part or product design. But object-centered viewing is natural when you want to think of a design as an environment through which you move, as is usually the case in architectural, landscape, and urban design. In either case the internal operation is always one of moving the picture plane rather than performing a three-dimensional transformation of the model itself.

Orthographic Projections

The simplest possible rule for converting three-dimensional coordinates into two-dimensional coordinates is to throw away the Z values in the [X,Y,Z] coordinate triples while leaving the x and y values unchanged. This rule may be expressed: [X,Y,Z] --> [X,Y] This produces an orthographic projection onto an xy viewplane .Similar rules can be used to project onto XZ and YZ viewplanes. The most salient property of such an orthographic projection is that the image of a projected straight line will be the same length as that of the original line when the original is parallel to the viewplane. This means that contours, profiles, and faces parallel to the viewplane will be undistorted--the same sizes and shapes as the originals. But lines not parallel to the viewplane will be shortened when projected. The ratio of the projected length to the actual length is called the foreshortening ratio. By changing view direction you can make different planes of a modeled building parallel to the viewplane--and thus present them in undistorted fashion. Furthermore,

through multiplication by appropriate scale factors, you can produce views to any desired scale. In particular, by taking a top view with the viewplane parallel to a building's floor planes and applying an appropriate scale factor, you can produce a scaled plan projection. (Use of this convention in computer-aided design continues a tradition that goes back at least to 2150 bc and the reign of King Gudea of Lagash. Most buildings also have many vertical wall planes, so it is also useful to produce elevations by setting the viewplane parallel to various important wall planes. (But this is not always the case: consider I. M. Pei's glass pyramid in the courtyard of the Louvre.)

Axonometric Projections

Often a designer needs to see the relationships between plans and elevations. The most obvious way to show these is to make an orthographic projection from an oblique view direction--such that horizontal and vertical faces are seen at once. This yields an axonometric orthographic projection. Such projections can be subclassified according to the foreshortening ratios of the faces in the three principal directions: the projection is isometric if the three ratios are equal, dimetric if two of the three are equal, and trimetric if they are all different. When a cube is drawn in isometric, the vertex angles in the drawing become 60 degrees and 120 degrees: this is the most common architectural convention. The choice among viewing directions for axonometrics depends upon what is to be shown and emphasized. A worm's-eye axonometric shows how a plan organization develops into three-dimensional interior space, and a bird's-eye illustrates the relationships between elevations and roof forms. Dimetrics are often appropriate for showing corner details. Trimetrics place the emphasis on just one of the faces. Clearly the strengths and weaknesses of plans, elevations, and different kinds of axonometrics are complementary, so it is useful to have simultaneous orthographic views. Whenever one view is altered in the course of a design process, then, the rest must correspondingly be updated. In manual drawing practice this process is laborious and a source of errors and inconsistencies. But a computer-aided design system can simultaneously update all the views whenever a change is made. Orthographic-projection software is normally controlled by two viewing parameters. A direction-of-view parameter controls the distribution of foreshortening--that

is, whether a plan, elevation, isometric, dimetric, or trimetric will result. And a scale (or zoom) factor determines the size of the projected image on the screen.

Oblique Projections

Sometimes an architect needs to keep one elevation of a building parallel to the viewplane (and thus undistorted) while showing something of the plan and faces of other elevations. This is impossible in orthographic projection, but it can be accomplished.

The rule here is:

[X,Y,Z] --> [(X + Z/A), (Y + Z/B)]

The constants A and B control the foreshortening of the receding faces. In effect, depth information is encoded in the image by shifting points diagonally according to their depth back into the scene. Whereas the orthographic projections discussed earlier result from taking parallel projectors perpendicularly through the viewplane, this new type of projection results from using oblique parallel projectors. Chinese and Japanese painters have traditionally employed oblique parallel projection in architectural scenes, but it has been less popular in the West. A variant of the same idea, which allows the plan of a building to remain undistorted, is sometimes preferred to the worm's-eye axonometric for showing how a plan develops into a three-dimensional interior space. It was popularized by Auguste Choisy in the nineteenth century, then widely used by twentieth-century modernists. It also enables a roof plan to be developed downward. Software to produce oblique parallel projections requires specification of one more parameter than for standard parallel projections: the angle at which receding faces are to be taken back. This controls the foreshortening of the receding faces.

Perspective Projections

If X and Ycoordinates are divided by an amount dependent on depth (rather than added to an amount dependent on depth as in oblique projections), then perspective projections are produced. The rule here is:

[X,Y,Z] --> [X/ (C*Z), Y/ (C*Z)]

The constant C controls rate of diminishment with depth back into the scene. This is simply another convention for encoding depth information in an image by distorting coordinates with depth. It is, however, related particularly closely to our optical experience, since the visual angle subtended at the eye (or a camera) by an object decreases with distance. As a famous illustration long-ago showed, perspective projection can be understood as the result of taking projectors, diverging from the viewer's eye, through the picture plane. The associated viewing parameters can readily be understood in these terms.

First, the viewer's station point and direction of view establish the principal line of sight through the picture plane to the object. By varying these parameters we can change the number and positions of vanishing points in an image and give emphasis to different faces of an object. A cube, for example, can be shown in one-point, two-point, or three-point perspective. The horizontal and vertical angles subtended at the apex of the viewing pyramid can also be varied. The effect is like that of manipulating a zoom lens. A narrow viewing angle produces an effect like that of a telephoto lens, and a wide viewing angle produces an effect like that of a wide-angle lens.

Clipping and Sectioning

When a parallel projection is displayed, lines must be clipped to the edges of the viewing box; in the case of a perspective projection they must be clipped to the edges of the viewing pyramid. The effect is to terminate lines where they intersect the rectangular boundaries of the viewing window. This is called lateral clipping. Sometimes, as well, it is useful to clip lines to specified front (or hither) and back (or yon) planes. This is called depth clipping or z-clipping. It is analogous to the standard architectural technique of taking a plan or section slice through a building. Efficient procedures for clipping lines are well known and are standardly incorporated in wireframe-modeling systems. It is a straightforward extension of z-clipping (though commercial wireframe-modeling systems rarely provide it) to clip lines to an arbitrary plane as a starting point for construction of a plan or section. Lines need to be broken at the section plane, and the break points highlighted. These can then quickly be connected to produce the required plan or section drawing. Usually it is best to assign the plan or section to its own layer, so that it can be shown in a different color, shown by itself, or switched off entirely.

Spatial Ambiguity and Depth Cues

A common problem with wireframe images is the existence of spatial ambiguities. A cube in axonometric, for example, has two consistent spatial readings. Selection of points and lines in projected wireframe views is inherently fraught with ambiguity, since a selected point on the picture plane actually specifies an infinite ray passing through the three-dimensional model .These selection rays are parallel in orthographic projections but diverge in perspective projections--making accurate selection in perspective an exceptionally difficult task. Any ray may pass through multiple selectable entities. Good snapping capability, to compensate for selection inaccuracy, is thus essential in wireframe modeling. Problems such as these can be mitigated or eliminated, however, through use of display techniques that provide additional depth cues to disambiguate an image. In hand drawing depth cues are often provided by breaking back lines. This is not very practical in computer graphics (since it requires time-consuming calculation of intersection points), so alternative conventions are usually employed. Depending on the capabilities of the available display technology, line weight, intensity, or color may be varied to indicate depth. The most common approach to removing spatial ambiguity, however, is to provide multiple, simultaneous projections. This is a straightforward computational task: the screen is subdivided into viewing windows, projections are performed, and projected views are mapped to windows. Typically, one of these windows will provide a picture plane that is coincident with the current construction plane, and there will be at least one other projection to relieve ambiguity. Operations, such as drawing a line, may be begun in one view and ended in another. Multiple views also provide a designer with a way to keep multiple design issues in mind as a model is constructed and edited. One common technique is to set up several perspective views from crucial points in the approaches to a building from different directions. Then, whenever a design operation is performed in one view, the implications for other views can immediately be seen. This provides a way to escape from the myopic vision of a project that commonly results from working exclusively in plan or in section or in perspective from a particular viewpoint. And it completely eliminates the tedious task of constructing new views to see what a change made in plan or section means in perspective. Fast computers allow real-time variation of viewing parameters to resolve spatial ambiguities, to reveal different aspects of an object's

geometry, and to select telling viewpoints. This was rare in the past, but as sufficiently powerful computers have become increasingly available, it has come to be regarded as an essential feature of a good wireframe modeler. Illustrates how azimuth and altitude of the picture plane may be varied continuously. Where available computer power is insufficient to handle a complete wireframe model in this way, a simplified version containing just enough information for orientation may be used for real-time viewing operations. Yet another disambiguation technique (sometimes combined with real-time variation of viewing parameters in advanced systems) is to provide stereoscopic wireframe views. This requires computation of perspectives from slightly differing station points, together with use of a viewing system that recombines these views in some way. An ordinary color display can be used to display superimposed complementary-colored images which are viewed through complementary-colored pairs of filters. More advanced systems use cross-polarization, alternating views, synchronized shuttered goggles, and so on. With a stereo display, the cursor can be freed from the picture plane and can move in three-dimensional space to select points and lines directly. Stereo displays and three-dimensional cursors can also be used to construct wireframe models by digitizing from photographs. The starting point is a stereo pair of photographs. These are scanned and displayed on a stereo display screen. The three-dimensional cursor is then used to digitize points, just as a two-dimensional cursor can be used to digitize points from a two-dimensional bitmapped underlay. A variant on this idea, which does not require use of a stereo display system, is to use two photographs, to digitize each point in each view, then to use a special reconstruction transformation of the form:

[[X1,Y1], [X2,Y2]] --> [X,Y,Z]

Producing Drawings from Wireframe Models

It might be thought (and it was once commonly suggested) that a plan could automatically be generated from a wireframe model by projecting it onto the ground plane and that a section could be produced by projecting onto an appropriate vertical plane. But a plan or section is a more subtle notation of design intention than a mere projection of lines from three-dimensional space. First, not all of the lines in a three-dimensional model are relevant in a plan drawing made for a particular purpose: some will have to be culled out

by careful depth-clipping or sectioning operations or by explicit line-deletion operations. (This is particularly troublesome when distinct lines in space become coincident in plan projection.) Second, some lines in the plan will need to be given emphasis (by assignment of heavier line weight, or dashing, for example), and it is difficult to specify foolproof procedures for accomplishing this automatically. Finally, many plan notations are not, in fact, projections of three-dimensional shapes but conventional symbols: these must be inserted in place of the corresponding projected shapes. The projection of a carefully layered and sectioned wireframe model is, however, useful as a starting point for construction of plans, sections, or elevations. It is best used as a reference layer that defines the "raw" geometry of a design and that serves as an underlay over which the two-dimensional expression is developed. One way to approach the task of further graphic development is to use drafting-system operations to edit the projected wireframe, define appropriate line weights, and lay out sheets. Anot**********
****
****
****
****
****
****
*****of the design and in fabrication and construction. A carefully constructed wireframe can provide a three-dimensional skeleton of construction lines that establishes key dimensions and defines key locations as constructed end points and intersection points. This skeleton can be used as a basis for extraction of definitive coordinate and dimension values and as a framework for snapping design elements quickly and accurately into place. This generalizes the old idea (as developed, for example, in J- N-L Durand's famous series of architectural textbooks) of controlling development of a design by means of a two-dimensional skeleton of axes, arcs, and grids. To snap a rigid wireframe element into position in space, three fixed points must be specified. (If one point on the element is snapped to a point on the existing wireframe, then the element can rotate in three axes about the connection point. If two points are snapped, then the element can rotate about the

axis passing through them. But if three points are snapped, no rotation is possible.) A system that supports point-to-point, point-to-line, and line-to-line snapping, together with real-time translation and rotation of elements (so that the remaining degrees of freedom for an element are always evident from its motion) makes accurate snapping together of wireframe elements particularly quick and easy. Syntax-directed editing capabilities are also very useful. Rules for snapping three-dimensional wireframe elements together in commonly used spatial relationships can eliminate explicit performance of a lot of tedious and error-prone selection, translation, and rotation operations.

Uses and Limitations of Wireframe Models and Views

A wireframe model takes its place in the image-production pipeline of a computer graphics system. Two-dimensional points are converted (through use of construction planes and so on) into a permanent three-dimensional geometric model consisting of points and lines. Projected views of that model become temporary display files that are used to drive display devices. And, when a raster display is used, the two-dimensional line data in the display files must be converted into bitmapped images.

A wireframe model provides a more complete representation of building geometry than a collection of two-dimensional drafted views--which are much like display files without a well-defined common reference. It is less of an abstraction away from three-dimensional physical reality. Both its advantages and disadvantages follow from this shift in emphasis.

Data structures used to store wireframes tend to require more memory than those used to store two-dimensional drawings, since there will usually be more lines in the model (compare a wireframe of a cube with its plan or elevation) and since each coordinate is expressed by three numbers instead of two. More computational work must be done in manipulating these larger data structures, and considerable additional computation is required to produce perspective and parallel projections for viewing. There is more geometric information for the user to input, and many input and editing operations become more complex--both computationally and for the user. In general, construction of a wireframe model takes longer, and costs more, than construction of corresponding two-

dimensional line models. The additional trouble and expense can be justified when there is a genuine need for a higher level of geometric completeness and more systematic coordination of views--when the three-dimensional form is not readily evident from plans and elevations, for instance, or when views from many different directions are needed, or where animated movement through and around the building is required. Development of plans, elevations, and sections into a complete wireframe model can also provide the occasion for further resolution of a building's geometry and the basis for further elaboration into a surface or solid model--as discussed in the next two chapters. Most importantly, a wireframe model can support forms of design exploration, geometric problem-solving, and measurement and analysis that are very difficult or impossible with two-dimensional representations. A designer can, for example, execute geometric constructions in planes that are not horizontal or vertical, perform constructions with space curves, and accurately measure shapes that would be foreshortened in plan or section.

Assemblies of Solids

Sometimes designers want to conceive of three-dimensional compositions not abstractly in terms of lines in space, nor more visually as collections of surfaces in light, but spatially, as arrangements of volumes--both solids and enclosed voids. Indeed, as Le Corbusier pointed out in Vers une architecture, this characterizes some architectural styles. "Egyptian, Greek or Roman architecture," he wrote, "is an architecture of prisms, cubes and cylinders, pyramids or spheres: the Pyramids, the Temple of Luxor, the Parthenon, the Coliseum, Hadrian's Villa." By shaping and arranging blocks of wood or polystyrene an architect can compose directly in volumes, but this is slow and cumbersome. An increasingly attractive alternative is to employ solid-modeling software that provides prisms, cubes, cylinders, spheres, and so on as geometric primitives, together with tools for inserting, deleting, transforming, and combining these. The displays produced by solid-modeling systems look much like the displays produced by wireframe- or surface-modeling systems (depending upon the way that solids are rendered), but the underlying geometric databases are very different. As a result, there are powerful geometry-editing operations not available in wireframe or surface systems, there are additional data

extraction and analysis possibilities, and the process of design exploration with a solid modeler tends to evolve in different ways.

Voxel Representation

Just as sounds can be represented by one-dimensional arrays of data points and images can be represented as two-dimensional arrays of data points, so compositions of solids can be represented as three-dimensional arrays of data points. For this purpose we employ a cuboid subdivided into cubic voxels (volumetric elements) rather than a rectangle subdivided into square pixels . We can then represent the forms of solid objects, using one bit of information per voxel, by coding a voxel outside the solid as zero and a voxel inside the solid as one. As with sampling sounds and sampling images, we need a sufficiently high density of samples to avoid unacceptable aliasing effects. But high sample densities are particularly hard to achieve in this case: whereas halving the interval between sound samples doubles the total number of data points and halving the distance between image samples quadruples the number of data points, halving the distance between solid samples produces an eightfold increase in the number of data points. However, voxel representations can usually be compressed effectively through use of a technique known as octree encoding . This is a three-dimensional version of the quadtree technique for compressing bitmapped images, which was discussed . An octree is constructed by first subdividing the voxel array into octants, then further subdividing any nonuniform octants, and so on until there is no need for further subdivision or the level of individual voxels is reached. Each terminal node of the octree can then be labeled with the value of the corresponding volume. We can generate output from voxel representations in several ways. Horizontal or vertical slices through the voxel array are one-bit bitmaps that can be displayed as sections. We can also produce hidden-line and shaded images by interpreting the faces of voxels as opaque square surfaces. Raytracing techniques can be adapted to render solids as transparent volumes--a particularly popular technique in medical-imaging applications. And we can employ special devices to produce actual three-dimensional solids. A stereolithography device, for example, operates on a tank of liquid to produce laser-induced solidification at locations corresponding to occupied voxels. Use of one bit per voxel suffices to distinguish between solid and void (which is enough for many design

purposes), but we can introduce more distinctions if we wish. Use of two bits per voxel, for instance, provides for distinction between four different occupancy conditions--different materials, say, or different densities of material. This is useful when we need to represent the internal structures of solids, as geologists do when they represent geological structures, as oceanographers and atmospheric scientists do in their domains, and as medical imagers do when they investigate the internal structure of the human body. As sensing and sampling techniques develop, and as the growing availability of inexpensive memory and processing power increases the feasibility of handling large, high-resolution voxel representations, processing of voxel data for scientific visualization purposes is becoming an increasingly important field.

Boundary Representation

For designers' purposes, however, voxel representations suffer from the same sorts of limitations as the bitmapped images that we considered they are low-level, unstructured, imprecise, and inefficient in use of available computational resources. We saw that, for greater precision and economy and to provide for higher-level design operations, we could use sparser and more structured representations in terms of lines--the boundaries of things. An analogous approach can be taken to solid modeling. The basic idea here is to generalize and extend the techniques of surface representation that we considered . Connected pairs of zero-dimensional points (vertices) define finite one-dimensional lines, connected sequences of three or more one-dimensional lines can be used to define two-dimensional closed polygons, and connected assemblies of four or more closed polygons can be used to define closed polyhedral solids. Thus data structures for boundary representation of polyhedral solids can be structured as illustrated . These can be generalized, if desired, to provide for curved as well as planar faces. These sorts of data structures consume more memory and are more cumbersome to manipulate than data structures for wireframe or even surface models of the same forms. This is partly because they must maintain a richer network of associations between geometric elements and partly because the associated operations for transforming and combining solids (which are, of course, implemented as operations on values in the data structure) must be prevented from

producing invalid solids--self-intersecting ones, ones with "missing" faces, and the like .The data structures and associated repertoires of operations of solid modelers are usually organized to maintain the topological properties of closed polyhedral solids as specified by Euler's theorem.

Vocabularies of Solid Building Blocks

Drafting systems and wireframe-modeling systems provide operations for inserting various types of lines; surface-modeling systems provide operations for inserting various types of surfaces; and, as we might expect, solid-modeling systems provide operations for inserting various types of closed solids. A very simple system might, for example, provide operations for inserting, selecting, and deleting rectangular boxes, of specified dimensions at specified locations and oriented parallel to the axes of the coordinate system. This structures a useful but very restricted domain of formal possibilities. An obvious and simply implemented generalization is to allow placement of boxes in any orientation. Vocabularies of polyhedra can be extended indefinitely--just as the Froebel blocks that Frank Lloyd Wright played with as a child came in sets of increasing variety, opening up increasingly extensive compositional possibilities. It is common, for example, to provide simple "pitched roof" forms such as appropriately parameterized triangular wedges, gables, hips, and pyramids. And, just as drafting systems customarily provide circles and circular arcs as primitives, so solid-modeling systems frequently provide the basic solid derivatives of circles: cylinders, spheres, cones, and doughnuts. Solid-modeling systems that rely entirely on creating and locating instances of vocabulary elements are known as primitive instancing systems. They are very effective in contexts where the kit of parts that a designer deploys is, in fact, strictly limited (as is sometimes the case in the manufacturing industry). In most design contexts, though, it is necessary to extend the repertoire of possibilities by providing operations for constructing solids from points, edges, and surfaces, and for combining simple solids to make more complex solids.

Sweep Operations

Sweep operations are very commonly employed in solid-modeling systems to create new solids. When a closed polygon is translated along an axis, its zero-dimensional vertices sweep out one-dimensional edge lines, its one-dimensional edges sweep out two-dimensional facets, and its two-dimensional surface sweeps out a three-dimensional volume . Solids can also be constructed by means of hierarchical sweep operations: a point might be swept to generate a straight line, that line might be swept to generate a square, and that square might be swept to generate a cube. The usual approach to sweep operations is to provide for drafting profiles in a construction plane, then translational sweeping of closed shapes to produce prismatic forms, and rotational sweeping of open profiles to produce solids of revolution. More sophisticated systems provide for sweeping closed shapes along arbitrary curves. These operations have their counterparts in fabrication operations, so they often serve well for modeling physical construction components: extruded, planed, and rolled elements such as steel sections and wooden moldings can be modeled by translational sweeping; lathed elements and turned pottery can be modeled by rotational sweeping; and bent elements can be modeled by sweeping along curves. Sweep operations can also be used to model the envelopes swept out by moving cutting tools, and hence the volumes of material that they will remove.

Skinning and Tweaking Operations

Closed solids can also be constructed in surface-by-surface fashion, as illustrated. This operation is known as skinning. The user must select the surfaces that are to be assembled into a solid. The software then checks that the specified surfaces do indeed enclose a volume (that they are "watertight"), and if so the surfaces are appropriately associated in the data structure. This method of solid definition is well suited to describing cast construction elements: indeed, the operation of constructing and positioning surfaces, checking for watertightness, and converting the hollow shell into a solid is very closely analogous to building and positioning form work and filling it with concrete. It is also good for describing the exterior volumes of buildings (since these are bounded by waterproof assemblies of surfaces) and the interior volumes of rooms (since these are often bounded by surfaces closed to keep in the warmth). A closely related operation is known as tweaking--selecting and moving a vertex, control point, edge, or face to adjust the shape of a solid . This operation must be controlled very carefully (either by the user or by the

software), since it can produce inadvertent conversion of planar faces into nonplanar ones and conversion of closed solids into self-intersecting objects. Tweaking vertices and control points is particularly effective for describing shapes such as those of tents, which are controlled at various points by poles and ropes. Objects that are formed by bending, twisting, and other such distortion operations can sometimes be modeled by taking a simple shape and tweaking it.

Features and Geometric Constructions

You can instantiate solids and locate them in space by specifying parameter and coordinate values, much as you can insert lines by specifying their end-point coordinates. But, just as drafting is more efficient when we take advantage of capabilities to snap to existing points and perform geometric constructions, so assembly of solids is more effectively performed by snapping new solids into specified relationships with existing one**********
****
****
****
****
****
****
****le, end points and midpoints of edges, center points of arcs, and centers of symmetry. Potentially significant lines include edges of faces, axes of symmetry, and surface normals. Potentially significant surfaces include not only faces, but also tangent planes to curves such as cylinders and spheres, and planes of reflective symmetry. Points may be snapped together; lines may be snapped into collinear, parallel, or perpendicular relationships; the bottom surface of one solid may be snapped into a coplanar relationship with the top surface of another one; and so on. It is, in fact, a nontrivial exercise to enumerate all the definite spatial relationships of design interest that can be formed between one type of solid and another. The realities of construction assembly often determine the potential spatial relationships of solid elements. If they are to be glued together, for example, they need face-to-face connection of sufficient area. If they are to be welded, they need edge-to-

edge or edge-to-face connection of sufficient length. If a column is to support a beam without use of some sort of shear connection, then the column must go underneath the bottom face of the beam, and there must be sufficient bearing area. If you want to make a recessed joint (such as those common in timber construction), you must cut out a housing so that you can intersect one element with the other. Similar practical constraints apply to the spatial relationships of closed volumetric elements such as rooms. If doors are needed between them, they usually need face-to-face connection of sufficient area. Alternatively (and less commonly), one might fit inside the other to form an aedicule. In classical composition, rooms were related concentrically, with coaxial axes of symmetry, or with coplanar planes of symmetry. But Frank Lloyd Wright overlapped interior volumes in ways that carefully avoided these classical relationships. And many of Frank Gehry's compositions juxtapose volumes in ways that conspicuously avoid concentric, coaxial, coplanar, parallel, and perpendicular relationships while still achieving the basic functional connections that are needed. The simplest solid modelers avoid the issue of geometric construction altogether and merely provide for direct location of solids in the coordinate system. Some more ambitiously provide snapping and geometric constructions in construction planes, but not in three-dimensional space. Only the most sophisticated so far provide generalized capabilities for selecting features of solids and executing constructions in terms of these features. However, feature-based editing capabilities allow a designer to specify not only a vocabulary within which to work, but also a rudimentary syntax. As solid modelers are used more extensively in design exploration, this capability will be regarded as increasingly essential.

The Spatial Set Operations

Since lines, surfaces, and solids can be regarded as sets of points, we can define the set operations of union, intersection, and subtraction (relative complement) on them their effects when applied to pairs of elements of increasing dimensionality. They can be implemented in line-, surface-, and solid-modeling systems, but they are of greatest utility in solid modeling because they provide a very elegant, powerful way to construct complex volumes from simple ones. They provide the necessary path from the simplicity of a vocabulary of elementary closed solids to the complexity of many real three-dimensional

solid objects. Some interesting design strategies follow directly from combining these operations with specific geometric constructions. Perhaps the most obvious is to locate solids so that their faces are coplanar, then to perform union operations to build complex solids from simple ones. This is closely analogous to gluing wooden blocks together or assembling pieces of cut stone. But these logical solids, unlike actual pieces of wood, can overlap in space. Unioning overlapping solids can produce surprising. If a third cube is unioned in the same relation to the second as the second is to the first, and so on recursively, a complex symmetrical polyhedron soon emerges. When the subtraction operation is used, one shape becomes a "cutting tool" on the other--much as a drill bit is a tool for subtracting a cylindrical solid from a piece of material. In general, subtraction can be used effectively to model construction components that are produced by material-removal operations such as drilling, sawing, carving, planing, and milling. At a larger scale, architectural elements that are conceptually subtracted (even though they may actually be formed in some other way) can appropriately be modeled by means of subtraction operations. Consider, for example, a rectangular solid representing a wall. By locating translationally swept solids so that their sweep axes are perpendicular to the vertical faces, then subtracting, you can quickly cut out the usual sorts of simple door and window openings. The principle can be generalized, to produce a much richer variety of openings, by locating the subtracted solids in nonperpendicular positions, by subtracting nonprismatic solids such as cones and spheres, and by subtracting from more complex wall solids. Thus the famous window openings at Ronchamp, for example, can quickly be produced by subtracting angled pyramids from a wedge-shaped wall. Subtraction can also be used to hollow out interiors. For example, locate a rectangular box in plan, locate a smaller box inside it, and subtract to produce an interior room. What remains of the original box becomes the wall poche. Or subtract a smaller hemisphere from a concentric larger one to generate a hollow dome. If you take a strict modernist attitude you will probably want the subtracted interior form to be similar to the exterior form, so that the exterior reveals the interior. But if you think more like a baroque architect you will probably want to subtract a dissimilar interior form--leaving a complex poche to mediate the differences between interior and exterior. Sir Christopher Wren's dome for Saint Paul's Cathedral in London, for example, has one shape on the exterior, a very different shape on

the interior, and a vast poche volume taking up the difference . Wren, the great mathematician, would certainly have enjoyed the possibility of exploring geometric possibilities with a solid modeler. Now imagine cutting a prismatic shape from the center of a rectangular block of polystyrene with a hot wire, cutting another prismatic shape from another direction, and removing the resulting core from the interior of the block. This core is formed by the intersection of the two prismatic shapes. The stonemason's strategy of projecting prisms from profiles drawn on the surfaces of a block, then removing everything except the intersection, illustrates the same idea. Strategies for volumetric design by intersection generalize this idea. You can construct the basic form of Helmut Jahn's State of Illinois Center in Chicago, for example, by fitting a cone (resulting from setback requirements) into the corner of a rectangular box (defined by the Chicago street grid) and then intersecting. Similarly, you can construct the form of pendentives making the transition between a square plan and a hemispherical dome by fitting a rectangular box into the equator circle of a sphere and then intersecting. This construction can be generalized for production of column capitals and many other kinds of transitional solids: the plan shape can be not just a square but any polygon, and the intersected solid can be almost anything that is nonprismatic--an elliptical spheroid, a cone, a pyramid, or whatever.

Regularizing the Spatial Set Operations

A difficulty with the spatial set operations on solids is that they are not closed--they do not necessarily yield solids. In general, the intersection of two solids may yield a solid, a face, an edge, a vertex, or the empty set. Some solid-modeling systems leave it to the user to make sure that operations are specified so as to produce nondegenerate solids, but a better approach is to eliminate the problem by regularizing the spatial set operations. The basic idea is to partition a solid (considered as a point set) into interior points and boundary points. Boundary points are defined as those whose distance from the solid and the solid's complement are zero. In general, boundaries may include dangling and floating edges and the like--the undesirable sorts of things that can result from ill-specified spatial set operations. These blemishes can be removed by the operation of regularization, which amounts to removal of every boundary point that is not adjacent to at least one interior

point. The regularized union, intersection, and subtraction operations can then be defined so that they apply to regularized solids, and are closed in the regularized solids. A related problem results from the round-off errors inherent in floating-point arithmetic. Faces that appear to be coplanar may overlap to produce intersection slivers and the like. Careful dimensional control, by snapping to grids and so on, is the best way to eliminate this possibility.

Constructive Solid-geometry Representations

Spatial set operations can be applied to the results of spatial set operations to produce trees of derived shapes. These are known as constructive solid-geometry (CSG) trees. At the terminal nodes of a CSG tree are instances of solids in the basic vocabulary of the solid-modeling system. Each higher node is a union, intersection, or difference of two lower nodes. At the root node is the complete three-dimensional composition. Solid-modeling software evaluates specified CSG trees by converting them into boundary or voxel representations that can be used to generate displays and for other computational purposes. There are two basic ways to handle this, and in choosing between them the software designer must evaluate a trade-off between making demands on memory and making demands on processor capacity. The first approach is to evaluate each union, intersection, or subtraction as soon as it is specified and then store the resulting boundary representation. This is profligate in use of memory, but has the advantage that the tree is always fully evaluated: it is not necessary to reevaluate in order to display a node or perform some other computation that requires explicit boundary information. The converse approach is to store only the parameters of the lowest-level solids and the sequence of combination operations, and to reevaluate the tree to a specified node whenever the boundary information is needed. This is extremely economical in use of memory, but makes much greater processing demands and can result in very slow generation of displays. The first approach is usually appropriate when processor speed constrains performance more than memory limitations, and the second tends to be appropriate when the converse is true. From a designer's viewpoint, however, the evaluate-as-needed approach has an additional advantage. It permits fluid variation of a design by manipulating the parameters of the lowest-level solids, by substituting different types of solids at the lowest level, and by pruning off whole branches of the tree and replacing them

with new branches. These capabilities turn a solid-modeling system into a particularly powerful tool for design exploration.

Power Sets of Solids

How many different solids can you produce by locating some elementary solids in space and performing union, intersection, and difference operations? The answer follows from the observation that overlapping closed solids always divide space into distinct closed regions. Each of these regions may, as a result of spatial set operations, become either solid or void. The set of all subsets of the set of regions (its power set), then, is the set of all possible solids. The power set forms a lattice under the relation of inclusion, as illustrated. The spatial set operations provide a convenient way to explore this lattice.

Volumetric and Engineering Analysis

Just as polygon-modeling systems are particularly useful for urban design, space planning, and other work that requires careful analysis of areas, solid models are correspondingly useful for design work that requires careful analysis of three-dimensional forms. In particular, closed solids have definite volumes, surface areas, centers of gravity, and moments of inertia. These properties are tedious to calculate by hand for any but the simplest cases, but solid-modeling systems can be equipped with efficient, general algorithms for deriving them from voxel or boundary representations. Thus you can use a solid-modeling system to measure the amount of material to be cast in a form, to measure the volume of an auditorium for heating and cooling or acoustic analysis, or to measure the volume of a building for urban design analysis. Volumetric analysis can be combined with spatial set operations to provide some very powerful problem-solving capabilities. Imagine, for example, that you need to design a room with a specified volume. Instead of choosing some simple shape (such as a rectangular box) to make the volume calculations easy, you might write a procedure to push two complex solids together along an axis, generate their intersection at each step, and calculate the volume at each step. If you did not like the shapes of appropriate volumes that resulted from this, you could try pushing them together along another axis. Furthermore, the data structures of solid-modeling systems can be extended to provide for association of material properties such as density

with solids. Associated algorithms can then derive additional properties. From volume and density, for example, the mass of a solid can be calculated. And if you know the location of the center of gravity, you know where this mass acts. For detailed analysis of engineering properties, solids may be broken up into small pieces, known as finite elements, as illustrated. Advanced solid-modeling systems provide algorithms for automatically constructing finite-element meshes from boundary models. Once this has been accomplished, finite-element analysis procedures can be used to produce detailed and accurate analyses of structural properties, thermal properties, and so on.

Assemblies

In the same way that a three-dimensional physical model can be assembled from wooden or plastic components, a digital model of a building can be assembled from discrete solids. Such a model is the three-dimensional equivalent of the polygon maps that we considered : it exhaustively and unambiguously describes the occupancy of space by subdividing space into bounded pieces. Before constructing a solid model of a building you must decide how to use solids to represent architectural elements. If you are making an exterior massing model, for example, the solids in the model will stand for major volumetric elements--much as blocks of wood or polystyrene stand for these elements in physical massing models. You may want to use such models not only to generate images, but also to analyze basic geometric properties of massing alternatives: you can compute volumes and surface areas, cut horizontal slices to reveal floor shapes, and section vertically to study profiles. For urban design purposes you can assemble simple exterior massing models of buildings into three-dimensional models of urban fabric. You can include in the model not only actual building volumes, but also notional volumes such as allowable building envelopes, air-rights volumes, and view pyramids. These models are quicker to build, easier to modify and update, and much more compactly stored than the physical models that have traditionally been used for this purpose. With appropriate associated software you can use them to generate aerial, skyline, and street-level views, to analyze sightlines, and to study the shadowing effects of buildings. You can use spatial set operations to combine height, setback, and other constraints into allowable building envelopes for sites, and you can check for spatial clashes between proposed building forms

and these envelopes. And by associating space use and ownership information with volumes, you can develop three-dimensional versions of the polygon-based geographic information systems that we discussed . At the individual building scale you can use solids to***lding as a collection of solid construction elements. Such models are particularly useful for exploring not only ways to combine physical elements in space, but also ways to sequence their assembly in time on the construction site. By taking account of mass properties, tectonic models can be used to calculate dead loads for input to structural-analysis programs. One of the hazards in design of a tectonic assembly is that you may inadvertently position solid elements such that they intersect. A duct may pass through space already occupied by a beam, for example. Fortunately, however, the procedures used to perform spatial set operations can be adapted to provide an efficient way of automatically checking for such spatial clashes. Essentially, a spatial clash checker looks for pairs of solids that have nonempty intersections. The task of modifying an assembly of solids to reflect design changes can become laborious and frustrating, since changes in the position or dimensions of one solid element may propagate long chains of necessary adjustments to other elements. If columns are moved further apart, then you need to lengthen the beam that they support, then the slab supported by the beam must be correspondingly lengthened, and so on. This process of adjustment can be automated if component solids are defined as parametric objects and their relationships are described by formulae that relate parameters so that, in effect, the whole assembly is programmed to behave appropriately in response to changes in dimensions or locations of parts. Some advanced modelers provide for this. Specifying an appropriate structure of relationships for a complex three-dimensional assembly may, however, prove to be a very difficult problem.

Nonmanifold Assemblies

For some purposes, solid-assembly models may be too realistic. A designer might, for example, want to include freestanding wireframes and floating surfaces (as well as closed solids) in an assembly model to serve as a construction skeleton. Furthermore, it may be useful to have operations that combine elements of different types--slicing solids with surfaces, extracting medial axes from solids, and so on. The data structure of a typical solid-assembly modeler is, unfortunately, not designed to allow this. More technically, solid modelers are usually based on the assumption that solids have enclosing shells of surfaces and that these enclosing surfaces are two-manifolds.

Such shells always obey Euler's polyhedron law, which may be stated:

$$V - E + F - R = 2(S - H)$$

where V, E, and F are, respectively, the numbers of vertices, edges, and faces, H is the number of holes, and R is the number of rings. This means, it turns out, that each edge must be incident at exactly two vertices, and each face must be incident at exactly two edges. The data structures of solid modelers are designed to accommodate objects that obey this law, and operators that manipulate those data structures are designed to preserve consistency with it. This excludes stand-alone and dangling faces and edges (the explicit intention of the regularized spatial set operators)--an appropriate exclusion if the intention is to model a world of solid objects, but not if the intention is to model a designer's nonmanifold world in which such abstractions play an important role. Nonmanifold geometric modelers, then, are systems that provide for assemblies of vertices, edges, faces, and solids into configurations that do not satisfy Euler's law. They include in their repertoires not only the usual operators for transforming and combining elements in each of these classes, but also operators that are not closed in one or another of the classes. These include operators that assemble edges to produce faces, that assemble faces to produce solids (the skinning operation discussed earlier), and that reduce solids to more abstract representations (which do not obey Euler's law) by pulling off faces, performing medial axis transforms, and so on. Thus they provide environments for incremental transformation of an abstract three-dimensional parti--a skeleton of lines and freestanding

faces--into a complete, consistent solid-assembly model. A full-featured, design-oriented geometric modeler might support four submodels: a point model, a wireframe model, a surface model, and a solid model. In the data structure, entities of lower-level models are associated to define entities of higher-level models: points bound lines, lines bound surfaces, and surfaces bound solids. Models at each level are regularized: the wireframe has no isolated points, the surface model has no isolated or dangling lines, the solid model has no isolated or dangling faces, and regularized union, intersection, and subtraction operators are used at each level. However, there may be points in the point model that do not bound lines in the wireframe, lines in the wireframe that do not bound surfaces in the surface model, and surfaces in the surface model that do not bound solids in the solid model. Association operations are used to create higher-level entities: points are connected to make lines, lines are connected to make surface facets, and solids are skinned by surfaces. Conversely, cutting operations are used to create lower-level entities: solids are cut by surfaces to make surfaces or by lines to make lines, surfaces are cut by surfaces to make lines or by lines to make points, and lines are cut by lines to make points.

Producing Graphic Output

Solid-assembly models implicitly embody surface and wireframe models, so all of the types of renderings that can be produced from these less spatially complete models can be produced from solid-assembly models as well. Some additional types of graphic output are also made possible by the additional spatial information that a solid-assembly model contains. First, you can cut arbitrary sections--at any location and angle and to complex section surfaces as well as to planes. Sectioning to a vertical plane to produce a traditional architectural section, to a horizontal plane to produce a plan, or to an inclined plane, is accomplished by subtracting a half space or a very large box. A thin slice can be cut out by intersecting the model with a slab-shaped object. Sometimes it is useful to intersect with more complex shapes to produce cylindrical cores, and so on. Sectioning by intersection with a plane, rather than by subtraction or by intersection with a thin solid, is a good example of the use of a non-manifold modeler. The result of sectioning a solid-assembly model by intersection with a plane is a set of two-dimensional poche polygons. These might be kept on a separate layer, since they are not really part of the basic geometric

model, or they might be transferred to a two-dimensional drafting system for use in production of a section drawing. Sectioned three-dimensional models may be shown in perspective or axonometric projection, or they may be projected orthographically onto planes parallel to the section planes to produce more traditional plan and section drawings. Section-cutting software can be extended to keep track of the new faces generated by the cut so that these faces can be shaded or outlined to show the poche. Many different rendering techniques may be used, depending on what you want to emphasize: among the options are wireframe with poche, hidden line with poche, hidden line with cast shadows, shaded, and shaded with cast shadows.

Automated Production of Physical Models

The CAD/CAM techniques that have been developed for use in the manufacturing industry can sometimes be adapted effectively for production of physical topographic and architectural models from numerical data describing solids. If a model is broken down into planar surface facets, for example, then a computer-controlled laser cutter can be used to cut the facets from thin sheet material: finely detailed wooden models of buildings and contoured topographic surfaces can be produced in this way. Alternatively, the computer-controlled milling machines that now find wide application in the manufacturing industry can be employed to produce complex solid parts in metal or high-density foam. Stereolithography is perhaps the most versatile technique, and despite its technical complexity and high cost it has rapidly found a niche in medical imaging and mechanical parts design. A stereolithography system passes computer-controlled lasers through a tank of polymer solution so that laser-induced polymerization occurs at specified locations. All of these techniques make use of complex, expensive machinery that most designers are unlikely to have in-house. They will increasingly be made available by model-making and prototyping service bureaus, however. There are also some inexpensive alternatives. One effective way to streamline model production is to print facet shapes with a laser printer, mount them on cardboard, and cut them out with a matte knife.

Uses and Limitations of Solid Models

Solid models of parts and solid-assembly models have a higher level of geometric completeness than corresponding bitmapped images, drafted drawings, wireframe models, and surface models. This is the source of both their advantages and disadvantages. In contexts where completeness and consistency of geometric representation are crucial, where it is necessary to integrate a wide range of applications around a geometric model, or where designers want to work in a directly sculptural way (rather than rely on abstractions like plans and sections), solid and solid-assembly models are particularly appropriate. But, since they must store more coordinate information and keep track of more topological relationships, solid models of artifacts tend to be much larger and more complex than less complete types of representations. This means that they make heavier demands on memory, computational capacity, and software engineering technique. A designer must consider whether the advantages of greater completeness justify the higher associated costs.

There is also a more subtle issue of representational economy. At an early stage in a design process a designer is usually interested in rapid, unencumbered exploration of ideas. Ambiguities do not cause major problems and may even become sources of creative ideas. Many inconsistencies can safely be ignored on the assumption that they will be dealt with later if the idea turns out to be a good one, but they are not worth attention if the idea is to be abandoned anyway. In this context sparse, economical representations that are easy to manipulate and do not mire the designer in demands for detail usually work better than representations that emphasize completeness and consistency. Later, when the focus shifts to resolving problems, working out details, analyzing cost and performance precisely, and producing complete documentation, abstract representations become less appropriate and techniques like solid modeling become more attractive. The practical usefulness of solid-modeling technology has grown with the availability of computing power and the sophistication of available software engineering techniques, and this trend will continue. Prototype solid modelers emerged in the 1970s, but the commercially available systems that followed in the 1970s and 1980s were limited, slow, expensive, and often unreliable. By the end of the 1980s, however, robust and effective solid modelers were available on

inexpensive personal computers and workstations, and they were becoming increasingly popular. As solid-modeling software exploits the capacities of increasingly powerful computers, it will be bound by fewer limitations on the topologies and geometries that it can process, it will be capable of handling larger and more complex projects, and it will increasingly emphasize real-time geometric transformation and spatial set operations, stereo and virtual reality interfaces, and other features that support swift, fluid manipulation of designs.

Motion Models

Consider a generalization of the idea of spatial and temporal sampling that was introduced in the discussion of digital sound, then developed further in our explorations of bitmapped images and solid models. You will recall that, in a digital sound recording, each data point has one time coordinate. In a bitmapped image each data point (pixel) has two space coordinates, and in a voxel representation of a solid each data point has three space coordinates. In an analogous digital model of a three-dimensional solid in motion over some time interval, then, each data point (hypervoxel) will have three space coordinates and one time coordinate .

Assume, now, that we use one bit per hypervoxel to specify whether points in space are occupied at moments in time. The result is a description of a four-dimensional hypersolid. Such four-dimensional objects are very difficult to visualize directly, but, just as we can collapse a spatial dimension to produce a two-dimensional image of a three-dimensional scene, so we can collapse a hypersolid to a three-dimensional voxel model. One way to do this is to collapse the time dimension, so that versions of the solid overlap in the three-dimensional spatial coordinate system: the effect is much like that of the famous multiple-exposure photographs made by Harold Edgerton to study motion. A second approach is to select a plane, then collapse the three-dimensional scene onto that plane at successive moments: this produces a sequence of two-dimensional bitmapped images--frames of a digital movie. If we display these frames side by side, we can obtain a sequence like the well-known photographic ones made by Eadweard Muybridge. And if we show them one after the other in rapid succession, we will see, from a particular

viewpoint, the solid in motion. Both of these visualization techniques have their uses. A mechanical engineer might be interested in the three-dimensional volume swept out by a moving part, since this volume must be kept free of obstructions. But if we want to see a pattern of motion unfolding over time (at the expense of some spatial information), we will prefer the sequence of frames. As you might expect, adequate sampling rate is as important in digital representation of motion as it is in digital sound recordings, bitmapped images, or voxel models of solids. If the spatial sample rate is inadequate, then the familiar effects of spatial aliasing will show up in digital movies in particularly objectionable form. Sawtooth patterns will appear to crawl along profiles, for example. If the temporal sample rate is inadequate, then temporal aliasing effects--visual equivalents of frequency aliasing in sound recording--will also appear. Motion will appear jerky, spoked wheels may seem to revolve at spurious rates (or even to revolve backward), fine details of gesture may be lost, and so on.

Keyframes

Just as we can generalize the idea of pixel and voxel representation to hypervoxel representation of solids in motion, so we can also generalize the idea of boundary representation to four dimensions. Recall that, in a simple boundary model of a solid, zero-dimensional points with specified coordinates are associated to define one-dimensional lines, one-dimensional lines at specified positions are associated to define two-dimensional polygonal faces, and two-dimensional polygonal faces at specified positions are associated to define three-dimensional solids. Three-dimensional solids at specified time coordinates, then, form the boundaries of four-dimensional hypersolids (which may be depicted in two or three dimensions in the ways that we have considered). In practice, software for modeling solids in motion typically provides the operation of keyframing for specifying such hypersolids. A pair of keyframes shows a three-dimensional solid at two moments in time--its states at the beginning and end of a motion sequence. Just as the translational sweep operation moves a two-dimensional shape through the third dimension to define a solid, the keyframe operation moves a three-dimensional solid through the fourth (time) dimension to sweep out a hypersolid. To visualize how this works for a simple case, consider a rectangular box at a specified position in space and time T = 0: this is the first

keyframe. Now imagine the box translated and rotated to a different position at time T = 1: this is the second keyframe. When two keyframes have been specified, and we assume motion over the time interval T = 0 to T = 1, then it is straightforward to calculate a frame for any intermediate value of T. By incrementing values of T in discrete steps through the range 0 to 1, we can interpolate as many intermediate frames as we may wish to see. When the box is moved in this way, each vertex sweeps out a line in four-dimensional space/time, each edge sweeps out a surface, and each surface sweeps out a solid. Thus we obtain a complete hierarchy of bounding elements defining a hypersolid. (If a wireframe or surface box instead of a solid box is keyframed, we obtain a correspondingly less complete space/time model.) Any property of a three-dimensional solid may vary between keyframes. If only position varies, then pure translational motion results. If only orientation varies, then the solid rotates in place. When position and orientation both vary, the solid tumbles along a path like a baseball. Scale may also vary, so that the transformation from one keyframe to the next is a general similarity transformation--describable by a 4 x 4 transformation matrix. Finally, there may be parametric variation of shape and variation in surface properties such as color, specularity, transparency, and roughness.

Translational Motion Paths

The simplest translational motion path between two keyframes is a straight line in the three-dimensional spatial coordinate system. This can be interpreted as the locus of the origin of a local coordinate system (sometimes called a pivot point) in which the shape of the moving solid and its rotational motions are described. More complex translational motion paths can be described by specifying not just pairs but sequences of keyframes, just as polylines can be specified by sequences of points. Then intermediate frames may be interpolated linearly (to produce jerky, segmented motion) or along arcs or splines to produce smooth motion . Many motion-choreography and computer-animation systems rely heavily on the idea of splined interpolation of translational motion paths between keyframes.

Rates of Change

If positions along a translational motion path are interpolated by dividing the path into equal intervals, then translation at a uniform rate results. But, if the path is divided into unequal intervals, then an object will speed up and slow down (like a roller-coaster car) as it moves along the path. Another way to show this variation is to plot displacement along the path against time. The first derivative of this curve shows the variation of velocity with time; the second derivative shows acceleration. Rates of change in size, color, transparency, and other keyframed variables can be plotted in exactly the same way. The interfaces of motion-modeling and computer-animation systems typically display all of these curves and allow for graphic editing by selecting and moving control points. When any curve (for example, an acceleration curve) for a variable is changed, all of the other curves for that variable are automatically adjusted. Linear interpolation (sometimes called lerping) can be used to show movement of an automobile at constant speed, uniform rotation of a wheel, or light fading at a uniform rate. To achieve smooth initiation and termination of changes, s-shaped (slow-in / slow-out) curves are often used: these have zero derivatives at their end points and constant derivatives in the middle. Other types of curves describe constantly accelerating motion (a rocket taking off), constantly decelerating motion (a rolling ball coming to rest), and sharp discontinuities (a ball struck by a baseball bat).

Motion Vocabularies and Compositions

These techniques extend the fundamental idea, which has evolved throughout this book, of a designer's vocabulary. A writer's vocabulary is a set of words, a musician's vocabulary is a set of sound types, a draftsperson's vocabulary is a set of line types, a sculptor's or architect's vocabulary is a set of surface types or a set of solid types, and a choreographer's or robot programmer's vocabulary consists of forms in motion. Each element of such a vocabulary combines a three-dimensional solid object with a translational motion path and rate curves for rotations, size changes, color changes, and so on. The element can be instantiated by locating its defining keyframes in a four-dimensional coordinate system and by specifying values for other parameters. Motion

compositions can be produced by instantiating elements within the same four-dimensional coordinate system. Variations on motion themes can be produced by giving the same motions to different forms: thus a uniformly moving sphere might be substituted for a uniformly moving cube in a composition. Alternatively, the same form might be moved along different paths,or along the same path at different rates. Most importantly, the synchronization of individual motions may be varied so that different three-dimensional figures develop at different moments .

Hierarchies of Motions

In an assembly of solids, some or all of the solids may move at once. The solar system, for example, is an assembly of spheres that spin on their own axes and also revolve around each other. We could specify these motions, in a model of the solar system, by constructing a sequence of keyframes for the entire system, but this would be very cumbersome. It is both clearer and more concise to specify movements of some objects relative to others. The obvious place to begin is with the sun, which we can locate at the origin of the global coordinate system. Then we can specify the orbit of each planet in the sun's coordinate system. Next, we can take the center of each planet as a local coordinate system in which the orbits of moons are specified. Finally, if we want to specify orbits of satellites around each moon, we can center lower-level local coordinate systems on the moons. Thus we obtain a hierarchy of nested coordinate systems, together with motions described in terms of paths and rates in each one. You can construct such hierarchies and motion descriptions in different ways, but (as Copernicus noticed) some ways are better than others. You might, for example, center the global coordinate system on the earth instead of the sun--a natural enough choice, since the earth provides the usual frame of reference for describing small-scale motions. Then, however, you would have to specify in this system the orbit of the sun around the earth and the epicycles of the planets. Very complex motions can be choreographed by concatenating simple motions in nested coordinate systems. Consider, for example, a horse on a carousel. Relative to the carousel, the horse translates up and down along a straight path . Relative to the ground, the carousel rotates about a single axis. Concatenating these two simple motions yields the more complex path of the horse relative to the ground. To support this sort of motion

choreography, many motion-choreography and computer-animation systems organize three-dimensional elements and subsystems in hierarchies instead of layers. Each subsystem in the hierarchy is selectable and has its own local coordinate system. Motions of lower-level subsystems can be specified and synchronized at any level, within any of the local coordinate systems.

Articulated Motion of the Human Body

As dancers, choreographers, and robot designers know, the movements of the human body are organized in a hierarchy similar to that of the solar system. These movements may be described schematically as follows. We center the origin of a body coordinate system on the lower torso. The upper torso moves relative to the lower torso. Legs move relative to the lower torso, and arms and head move relative to the upper torso. The movements of each arm are organized in a hierarchy from the shoulder to the elbow to the wrist and so on down to the tips of the fingers. Similarly, the movements of the legs are organized in a hierarchy from the hip joint down to the tips of the toes. Relative to the head, there is rolling of the eyes, wagging of the tongue, and wiggling of the ears. Although the motion of each body segment is relatively simple, the path of a point relative to the stage can become extraordinarily complex: consider, for example, the path of a Balinese dancer's fingertip. An obvious difference between the human body and the solar system, however, is that the parts of the body are connected by joints. These joints, by their particular physical natures, constrain the motions of the parts that they attach to other parts. The middle joints of fingers, for example, are essentially hinges allowing rotation in one axis through a limited angular interval. But the shoulder is a ball joint allowing rotation in three axes. The notation systems that choreographers have developed for specifying the articulated motion of limbs connected by joints can be adapted for use in programming computer models of the human body. One of the best known of these is the Labanotation system. This employs simple two-dimensional shapes to specify horizontal movement directions, combined with shading to specify vertical movement. Some simple dance motions are shown, in this notation

Mechanical Joints and Kinematic Chains

Motions of solids connected by joints that constrain (but do not, in general, prohibit) those motions are the particular concern of designers of mechanisms. As Franz Reuleaux systematically showed in his famous nineteenth-century textbook on kinematics, mechanical joints can be classified into types according to the ways in which they constrain motion. Each type of joint can be described more precisely as a transformation (usually in 4 x 4 matrix form) specifying the constraints on relative motion of the parts that it connects. The joint matrix has variables, so particular possible spatial relationships of the parts can be specified by assigning values to these variables, and values can be incremented with time to simulate possible motions . Complete mechanisms can thus be described schematically as kinematic chains--stick-figure diagrams in which nodes stand for solid parts and connecting lines stand for joints. (For theoretical reasons related to techniques of motion analysis, mechanical engineers draw a basic distinction between open-loop mechanisms, in which the chain has no closed cycles, and closed-loop mechanisms, in which closed cycles do occur.) The motion choreography of a complete mechanism can, then, be explored by defining the geometry of each part in the kinematic chain and the transformation matrix for each joint, by choosing some of the variables as independent variables and incrementing their values with time, and by computing the values of dependent variables. Results can be shown either as sequences of animation frames or as diagrams of motion envelopes. Advanced computer-aided mechanical design systems extend the idea of solid-assembly modeling by providing not only for modeling part geometry, but also for specifying joint types and parameter values. Motion of the mechanism can then be simulated by incrementing the joint variables at specified rates through specified ranges. Spatial clash checking can be generalized to collision checking for moving parts. The joints and mechanisms found in the built environment are generally fairly simple and can readily be described and simulated in this way. Doors and windows pivot on hinges or slide on tracks. Elevators and escalators also slide on tracks. Drawbridges and seesaws pivot vertically, swing bridges pivot horizontally, and lift bridges slide vertically. Cranes combine pivoting and sliding motions in various ways. Stadium and auditorium roofs sometimes have sections that slide or pivot to open and close. Folding awning frames, roof structures, bleachers, and chairs may be simple

multibar linkages. The kinetic sculptures of Alexander Calder are articulated and jointed to produce hierarchies of pivoting motions.

Simulation of Physical Behavior

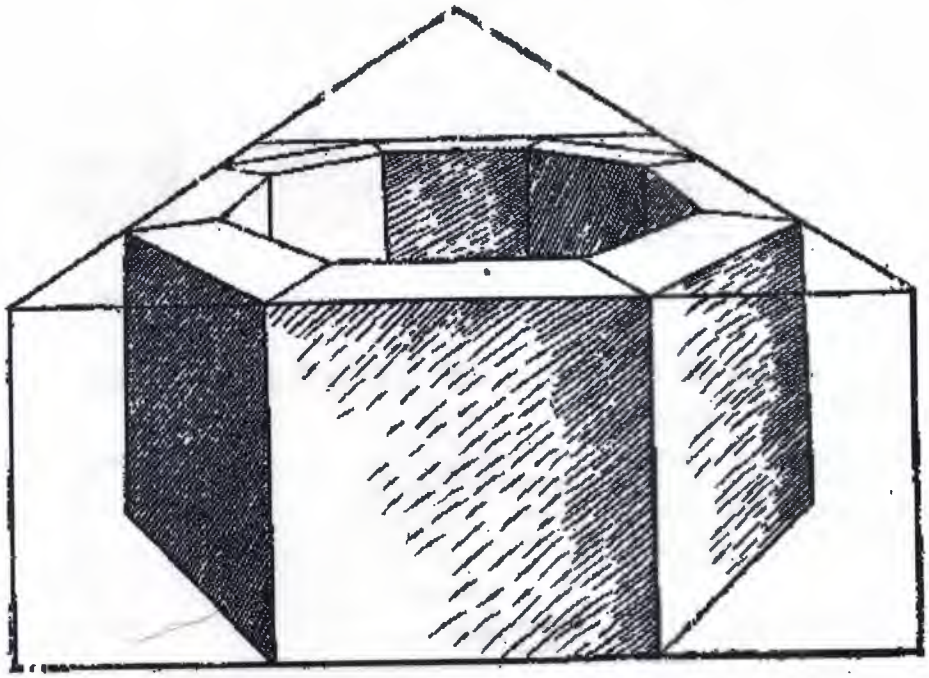
The idea of detailed physical modeling of assemblies by describing solids and the interfaces between them can be developed still further by introducing laws of dynamics into motion simulation. In this sort of simulation, solids have mass and elastic properties; initial conditions of position, velocity, and acceleration are specified; and the laws of dynamics are used to work out physically possible sequences of events. In 1986 Pixar Corporation demonstrated this possibility with an animated film (called Luxo Jr.) in which an articulated drafting lamp jumps from one position to another. Joint descriptions of the same kind as those used in dynamic simulations of mechanisms provide the basis for static and dynamic structural analysis of assemblies that are not mechanisms--those that are of particular interest to architects and civil engineers. The possibilities of joint constraint in a composition of solids actually define a continuum of mechanism and structure types. At one extreme, all the solids in a composition can be isolated free bodies--each with three degrees of translational freedom and three degrees of rotational freedom. Next, the solids can be connected together in a minimal way (by a few wires, for example, as in a Calder mobile) to produce a mechanism with many degrees of freedom. Joint constraints can be added to produce a much more constrained mechanism in which movements are strictly limited. When exactly the right amount of constraint is added to the system, a statically determinate rigid structure results. If yet more constraint is added, the structure becomes statically redundant. Finally, the components may be completely fused together to produce a monolithic structure.

Uses and Limitations of Motion Models

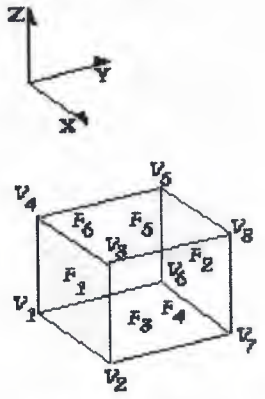
Motion models of three-dimensional assemblies are relatively costly to build, modify, and maintain, so designers must consider whether the time and cost expended on them will be justified by the value of the visualization and analysis results obtained from them. Certainly they are not always necessary: adequate structural and kinematic analyses

can often be produced from much more abstract network representations, for example. They are most likely to be justified at a late stage in a design process (when details of geometry, materials, and connections have largely been resolved) rather than at an early one, when the organization and behaviors of an assembly are particularly difficult to understand (as in a complex piece of machinery) and when the penalties for design inadequacies are particularly severe (as in nuclear power plants). The costs of building and maintaining motion models are likely to drop as the technology advances, however. And at the same time, demands for more thorough evaluation of designs are likely to grow. So we will probably see much more widespread use of such models by designers in the future. Increasingly, they will take the place of physical prototypes.

INSERTION OF SURFACE FACETS



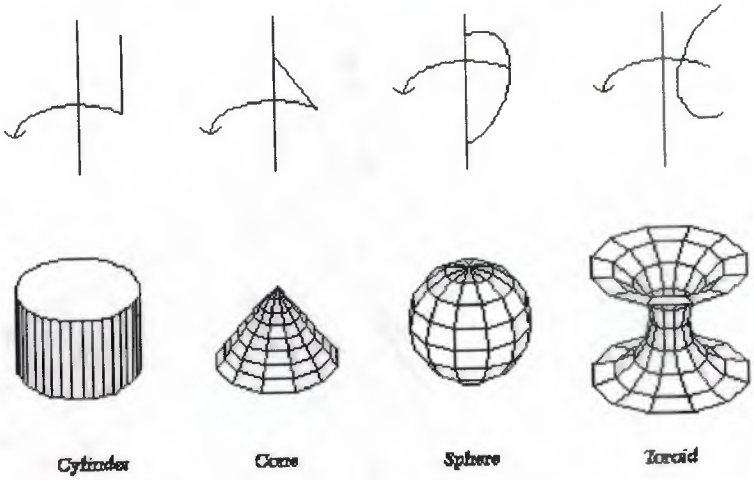
A TYPICAL DATABASE FORMAT



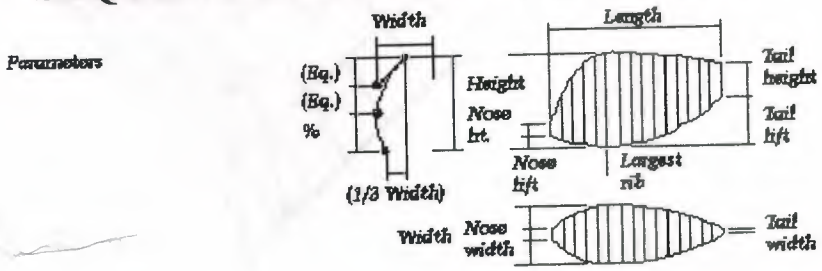
V_1	(0,0,0)	E_1	(V_1,V_2)
V_2	(1,0,0)	E_2	(V_2,V_3)
V_3	(1,0,1)	E_3	(V_3,V_4)
V_4	(0,0,1)	E_4	(V_4,V_1)
V_5	(0,1,1)	E_5	(V_5,V_6)
V_6	(0,1,0)	E_6	(V_6,V_7)
V_7	(1,1,0)	E_7	(V_7,V_8)
V_8	(1,1,1)	E_8	(V_8,V_5)
		E_9	(V_1,V_6)
		E_{10}	(V_2,V_7)
		E_{11}	(V_3,V_8)
		E_{12}	(V_4,V_5)

F_1	(E_1, E_2, E_3, E_4)
F_2	(E_5, E_6, E_7, E_8)

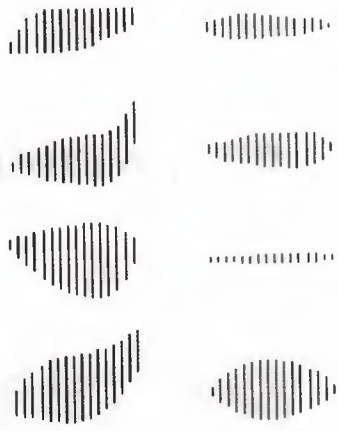
ROTATIONAL SWEEP OPERATIONS



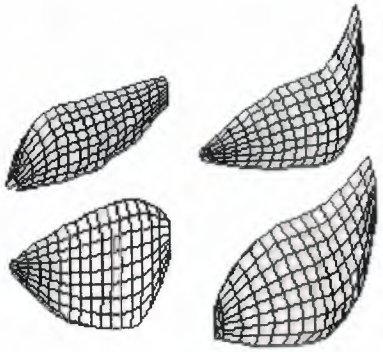
SEQUENCES OF PROFILES OR RIBS



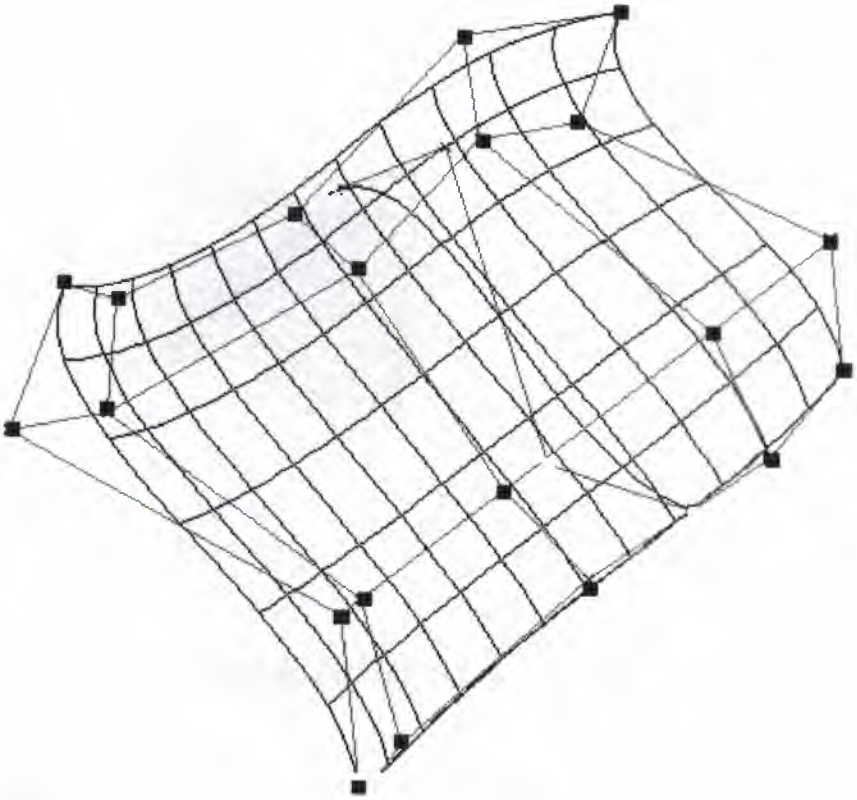
Parametric variations showing ribs in plan and section



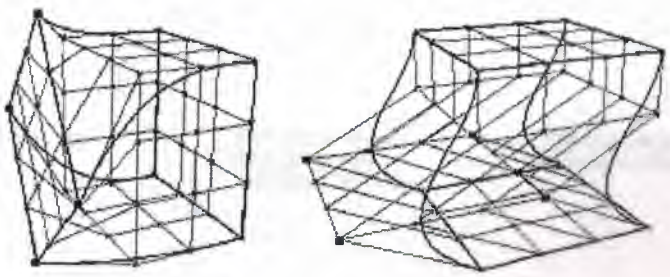
Shapes skinned with bilinear patches



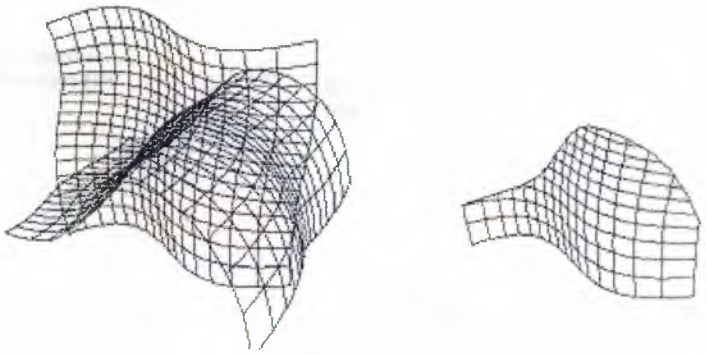
MESHES OF CONTROL POINTS



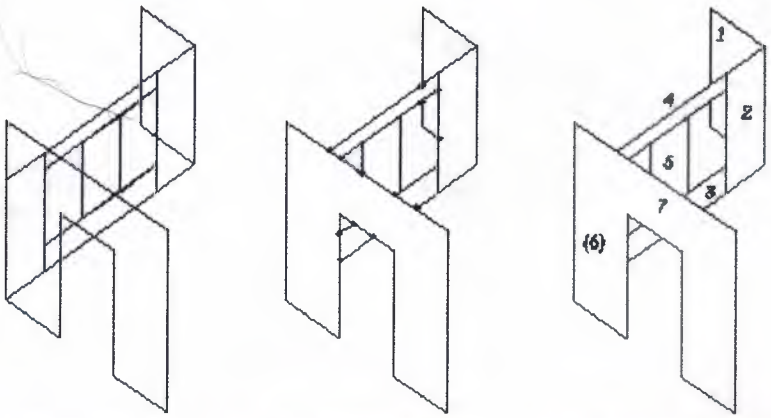
PULLED AND TWISTED



SURFACES AS CUTTING TOOLS



VISIBLE SURFACE DETERMINATION

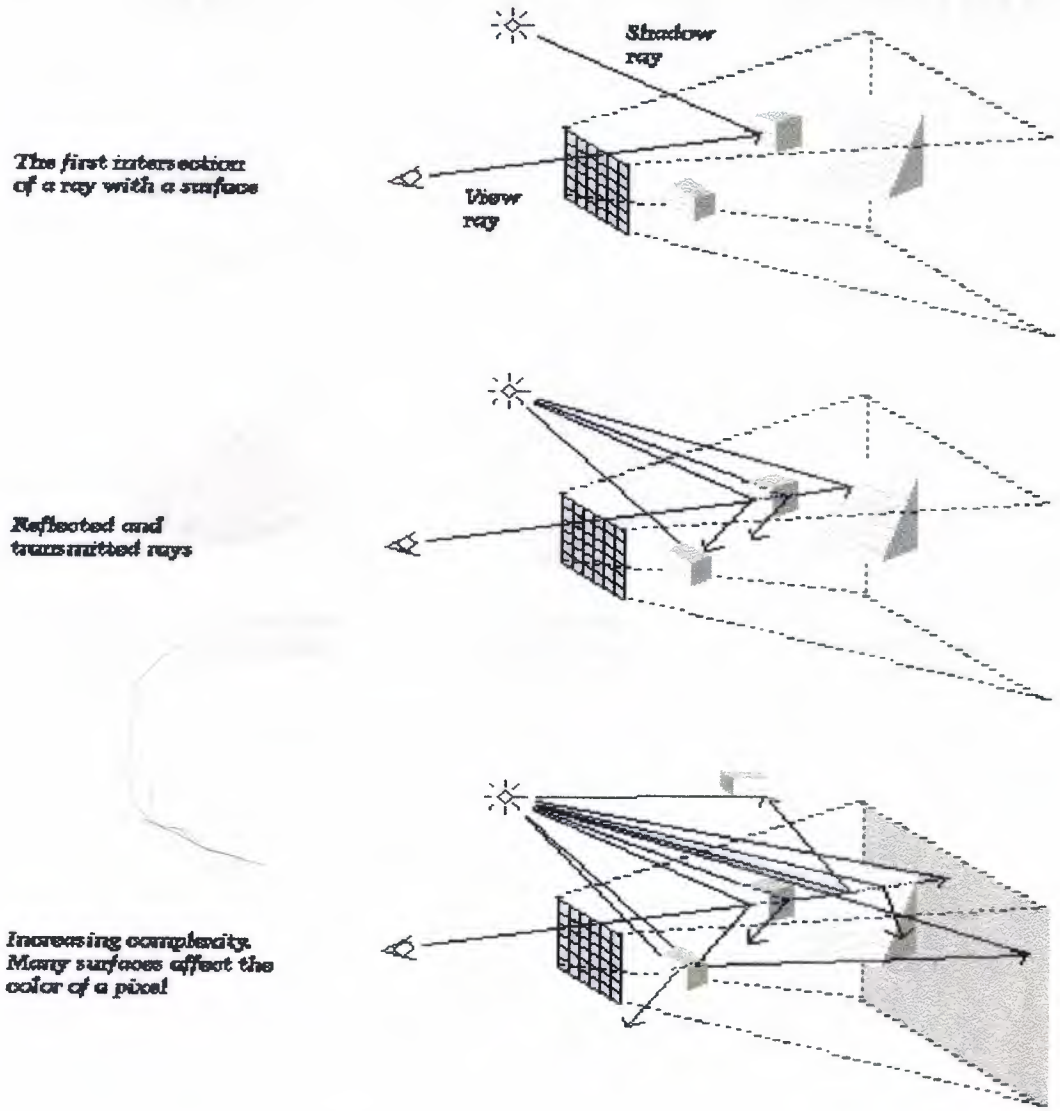


Wireframe view

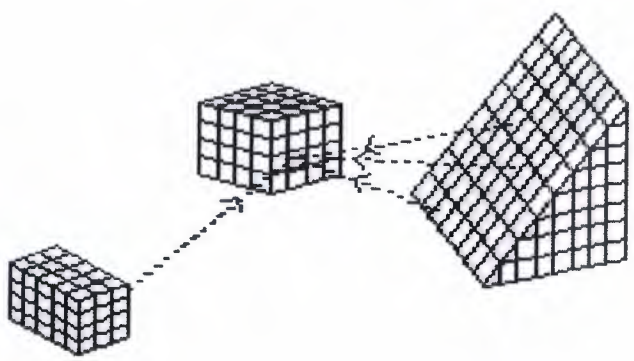
Hidden-line removal:
64 line segments
calculated, 20 hidden

Hidden-surface removal:
7 surfaces depth sorted

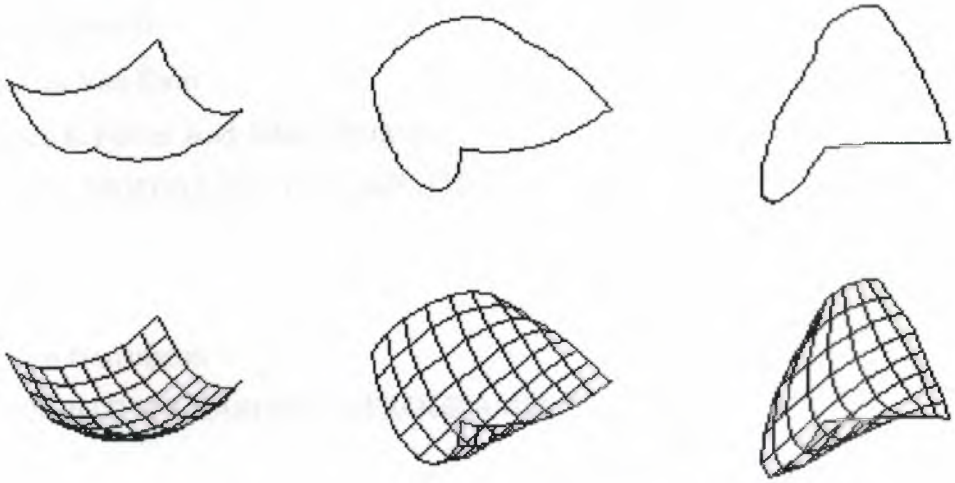
RAY



DIFFUSE GLOBAL ILLUMINATION EFFECTS



EDGES OF EQUAL QND ARBITRARY SECOND ORDER CURVERS VERTICES REGULAR AND ARBITRARY



Edges of equal 2nd-order curves, vertices regular

Edges of arbitrary 2nd-order curvature, vertices regular

Edges of arbitrary 2nd-order curves, vertices arbitrary

SOURCES

1 - Foley,james D.
Andries Van Dam
Steven K.Feiner And John F.Hughes
" SOLID MODELLING IN COMPUTER GRAPHICS "

2 - Steven Harrington
" COMPUTER GRAPHICS A PROGRAMMING APPROACH "

3 – Ray A.Plastack
Gordon Kalley
"THEORY AND PROBLEMS OF COMPUTER GRAPHICS "