Comput. & Graphics Vol. 15, No. 2, pp. 175–184, 1991
Printed in Great Britain.

0097-8493/91 $3.00 + .00

Technical Section

SIMPLIFICATION OF OBJECTS RENDERED BY POLYGONAL APPROXIMATIONS

MICHAEL J. DEHAEMER, JR. and MICHAEL J. ZYDA*
Naval Postgraduate School, Code CS, Department of Computer Science, Monterey, CA 93943-5100

Abstract—Current technology provides a means to obtain sampled data that digitally describes three-dimensional surfaces and objects. Three-dimensional digitizing cameras can be used to obtain sampled data that maps the surface of three-dimensional figures and models. Data obtained from such sources enable accurate renderings of the original surface. However, the digitizing process often provides much more data than is needed to accurately recreate the surface or object. In order to use such data in real-time visual simulators, a significant reduction in the data needed to accurately render the sampled surfaces is required. The techniques presented were developed to drastically reduce the number of data points required to depict an object without sacrificing the detail and accuracy inherent in the digitizing process.

1. INTRODUCTION

As our technologically oriented civilization becomes increasingly more complex and sophisticated, the cost of training operators and technicians becomes more costly and time consuming. Inexpensive, three-dimensional simulators are important visualization tools that can provide an attractive alternative or supplement to traditional training methods [1]. In order to be effective, these simulators must provide a sense of realism to the trainee; a real-time three-dimensional environment is an important contributor to this sense of realism. A need exists for realistic three-dimensional models of actual real world objects—ships, aircraft, automobiles, and other vehicles—for use in these simulators. The use of a three-dimensional digitizing camera to obtain data from scale models can be used for this purpose. Such data provide an accurate depiction of objects used in the simulators [2]. However, these images can contain several hundred-thousand polygons—far too many polygons to be drawn in a real-time simulation without the use of very specialized and costly hardware. A semiautomated technique was sought for drastically reducing the number of polygons required to depict an object in real-time on relatively inexpensive hardware without sacrificing the detail and accuracy provided by the digitizing process [3].

1.1. *Background*

Many of the ongoing projects in the Graphics and Video Laboratory at the Naval Postgraduate School deal with the creation and use of low-cost, real-time visual simulators [1, 4, 5, 6]. Currently, simulators exist for land, air, sea, and undersea vehicles which run on the Silicon Graphics IRIS-4D/GT series of graphics workstations and make extensive use of the graphics capabilities of these machines. Recent work has focused on increasing the usefulness and effectiveness of the projects by incorporating real-world terrain data, including topography from digital survey databases, as well as reflectance and cultural feature information from photogrammetrically processed stereo-pair aerial photographs [7]. These simulators, for example, now allow one to pilot a ship on the waters of the Sea of Japan, or to drive a Jeep across the terrain of Fort Hunter-Liggett, California.

These efforts have been very important in adding to the visual realism of these simulators. However, there is still a need for realistic vehicles to inhabit them. Previously, vehicles were drawn up by hand on sheets of graph paper and painstakingly converted into polygonal representations for use in the programs. This technique is tedious, time-consuming, error prone, and highly dependent upon the artistic talents of the individual creating the sketches. In order to overcome these problems, a project was undertaken to provide the vehicles in these simulators the same degree of accuracy and detailing as that available from using digital topography information for the terrain models.

1.2. *Three-dimensional digitizer*

Cyberware Laboratories of Monterey, California produces a 3D digitizing camera that was used for data collection. It performs a cylindrical or linear scan of an object and produces a data file consisting of up to 1 million sample points that form a mesh that maps the surface of the object. In the case of the cylindrical scan, the mesh consists of radius values measured at uniformly spaced latitudes and longitudes around the object being digitized. The resulting mesh is identical to that produced by the lines of latitude and longitude drawn on a model globe. Borrowing from this analogy, it is convenient to visualize and discuss the data points in terms of their latitude and longitude, and to use the terms north, south, east, and west to express relationships among the sample points. The linear scans produce a mesh that resembles the grid on a sheet of graph paper; however, the terminology from the cylindrical scan is retained for consistency. It should be noted that the spacing between neighboring sample points is not necessarily the same along both latitudes and longitudes, nor are the number of samples the same in both directions.

A schematic diagram of the camera's operation is shown in Fig. 1. A low-power laser is directed through

* Contact author.

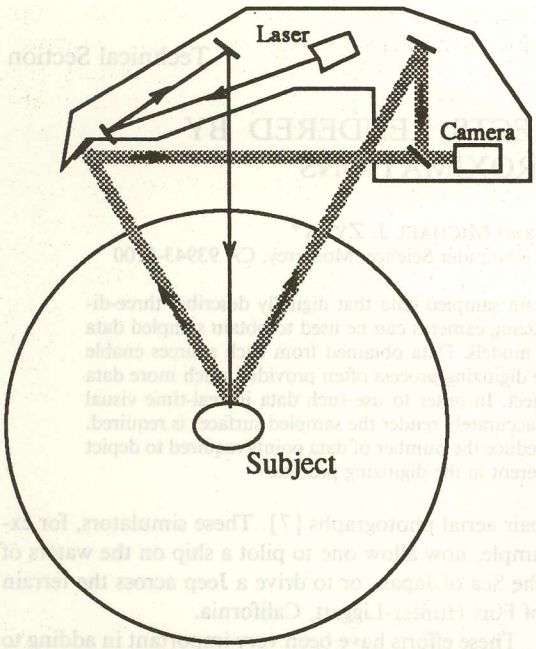


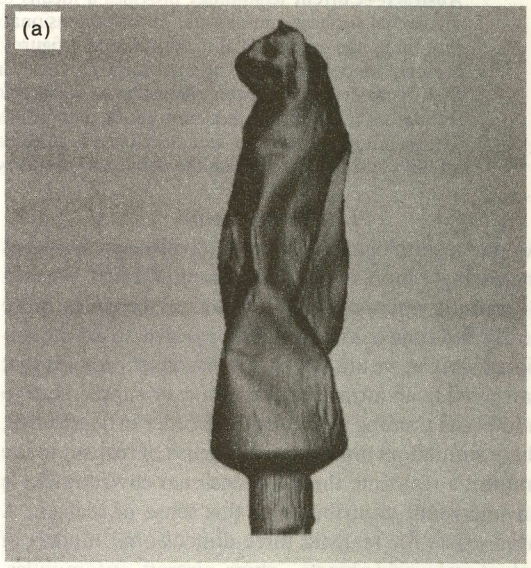
Fig. 1. Schematic view of Cyberware digitizer.

a series of lenses and mirrors to create a plane of light that is projected onto the subject. Mirrors view the reflected light from both sides of the subject in order to alleviate any problems caused by shadowing. These images of the surface contour are combined at a half-silvered mirror and reflected into the lens of a high-resolution CCD camera. The image is then processed by additional circuitry to extract the contour information and relay it to the controlling computer. The entire apparatus is mounted on a revolving framework so that it rotates around the subject creating a cylindrical scan of the surface contours. Alternately, small subjects can be moved past a stationary camera to produce linear scans of the subject. The most recent version of the camera is able to produce a 512×512 grid of sample points at a resolution of 0.7 mm in 15–20 seconds.

By using the shading and lighting models on the IRIS workstation, the renderings of these digitizing objects are very realistic. We currently have scans of objects from simple geometric shapes to complex and detailed ornamental carvings, with an average of over 110,000 data points (see Photos 1–4). However the drawing capabilities of the IRIS workstations are not sufficient to allow models of this detail to be used in real-time 3D visual simulators. Therefore, a method for reducing the number of data points required to accurately render an object was needed.

1.3. *Solution techniques*

There are two basic approaches to solving this problem, as shown in Fig. 2. The first, and probably more straightforward method, is to systematically add or delete polygons from the object being rendered until the desired realism is obtained or the system's capabilities are exceeded. The second approach is to specify how many polygons the final object is to have, and then try to find the combination of polygons that gives the *best* rendering with that number of polygons. In either case, realism of the rendering must be balanced against drawing complexity and speed. The number of polygons to be drawn is a reasonable metric to be used in determining drawing complexity; however, a similar



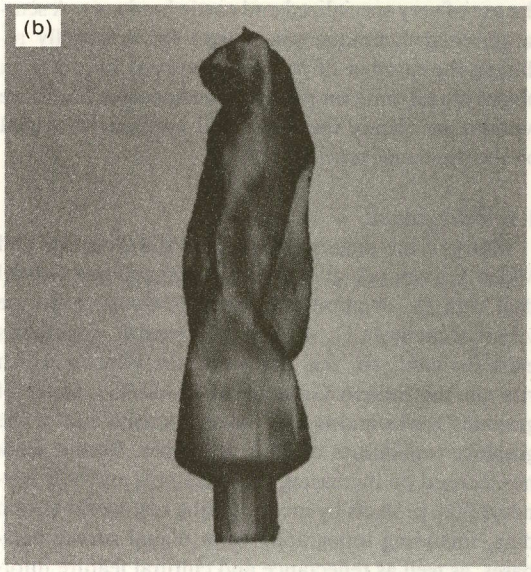


Photo 1. Tube of paint. (a) Full data set: 75264 polygons; 75776 vertices; 752640 library calls; 4229120 bytes; (b) Reduced data set using *g2e* variation: 3617 polygons; 5407 vertices; 43006 library calls; 266344 bytes; 0.030″ tolerance.

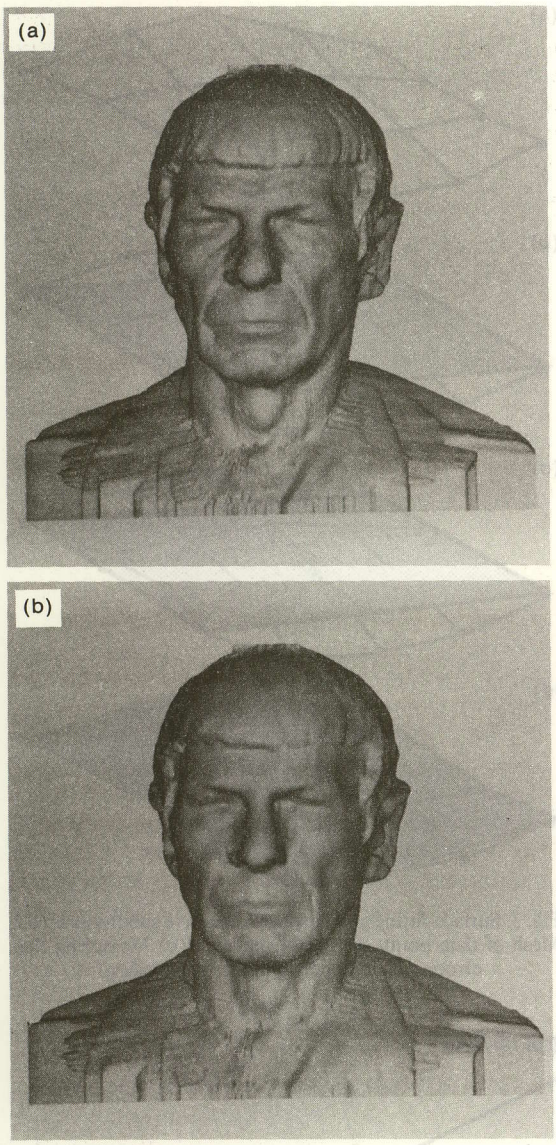


Photo 2. Spock bust. (a) Full data set: 112128 polygons; 112640 vertices; 1121280 library calls; 6293504 bytes; (b) Reduced data set using *g2e* variation: 12821 polygons; 17918 vertices; 128210 library calls; 660692 bytes; 0.033″ tolerance.

metric is not available to measure the realism of the rendering. For this reason, the techniques are not completely automated. The user is tasked with viewing the results of the data reduction and determining whether or not the realism is sufficient for the purpose at hand.

2. ADAPTIVE SUBDIVISION

In their article, Schmitt *et al.* describe an adaptive subdivision method of fitting surfaces to sampled data [8]. The technique approximates the surface represented by the sampled data points with bicubic Bernstein–Bézier surface patches. By constraining the patches to be continuous with neighboring patches, the coefficients of the patch can be determined. An accuracy metric is then used to measure the *closeness* of the approximating patch to the actual data. If the approximation is not within a user-specified tolerance, the patch is subdivided into four smaller patches. The process is then recursively performed on these subpatches until the set of patches approximates the sampled data to within the specified tolerance.

This technique has a useful property with respect to the simulator work being done in this laboratory—the method yields a reduction in the number of data points required to approximate the surface of a 3D object. By adjusting the method's tolerance, the user can make the approximation as accurate as necessary (limited by the precision of the digitizing process). However, the IRIS-4D/GT series of graphics workstations does

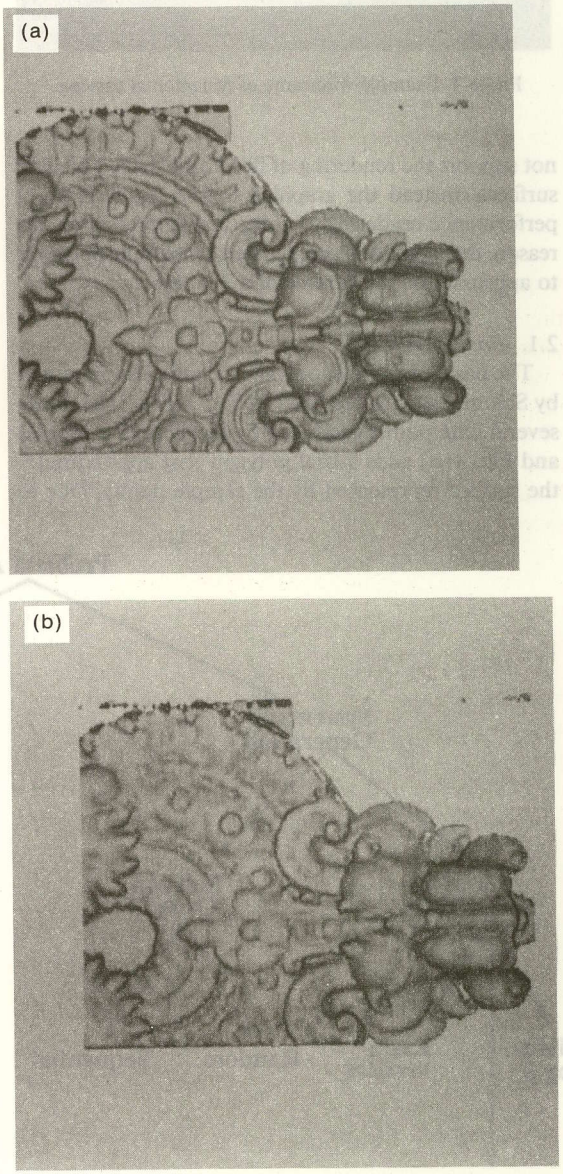


Photo 3. Ornamental carving. (a) Full data set: 130560 polygons; 131022 vertices; 1305600 library calls; 7325695 bytes; (b) Reduced data set using *ce* variation: 15964 polygons; 19447 vertices; 173078 library calls; 1018240 bytes; 0.032″ tolerance.

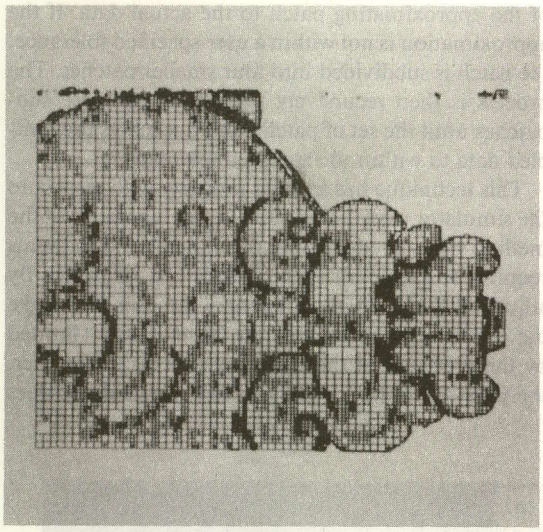


Photo 4. Example wireframe of ornamental carving.

not support the rendering of filled and shaded bicubic surfaces, instead the graphics engine supports high-performance rendering of polygonal surfaces. For this reason, the methods presented here use simple polygons to approximate the surface of a 3D object.

2.1. *Basic method*

The basic adaptive subdivision method as presented by Schmitt *et al.* is sketched in Fig. 3. Fig. 3(a) shows several data points produced by the sampling process and Fig. 3(b) adds a trial polygon that approximates the surface represented by the sample points. Due to

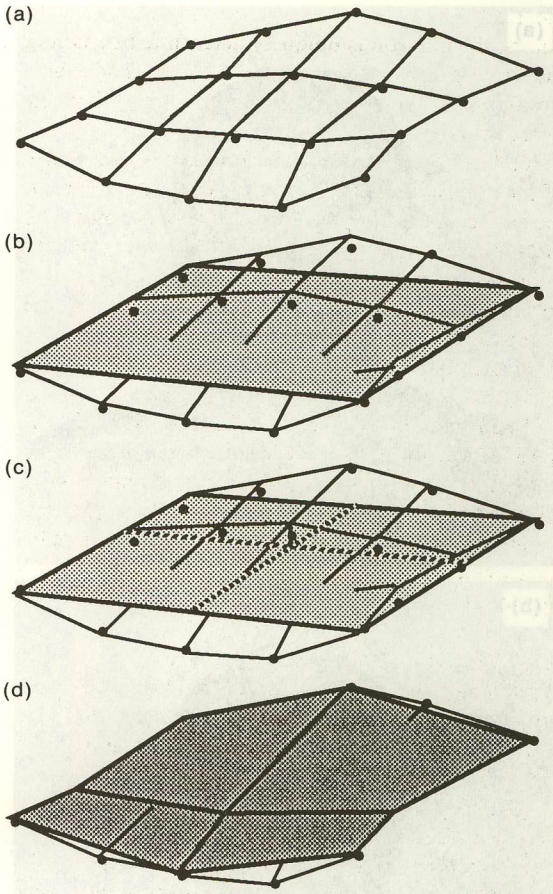


Fig. 3. Surface-fitting polygons with adaptive subdivision. (a) Mesh of data points, (b) Trial polygon, (c) Measuring the errors, (d) Trial polygon after subdivision.

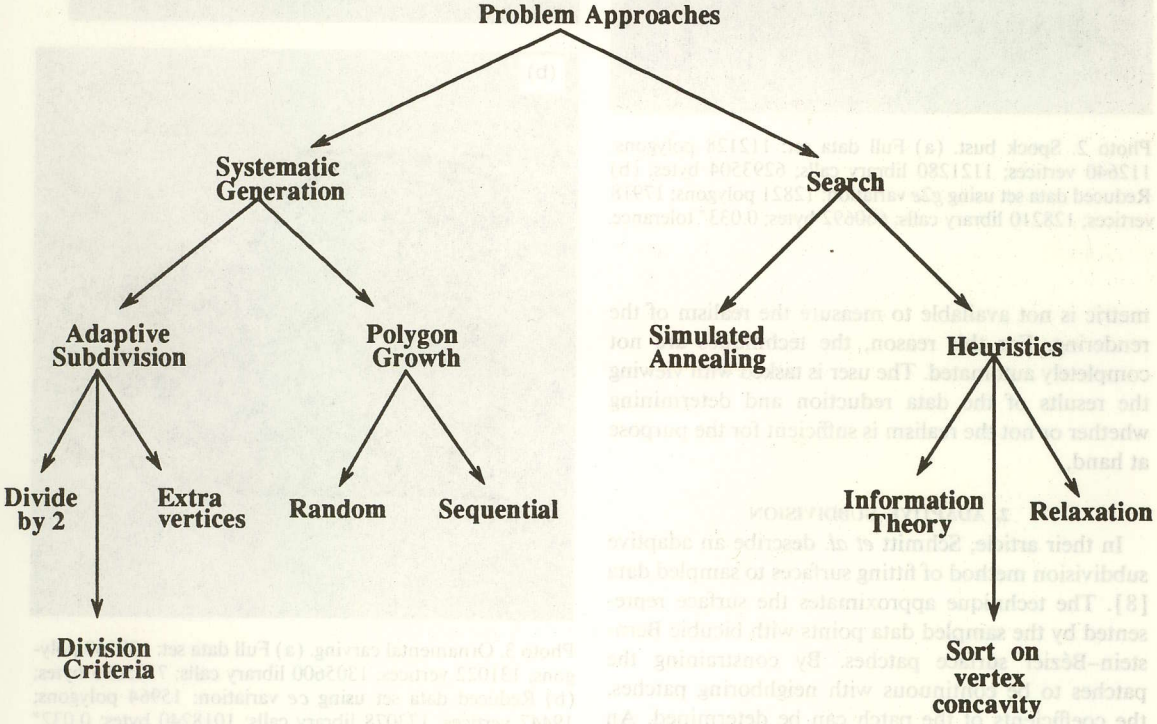


Fig. 2. General approaches to data reduction.

the regularity of the mesh represented by the data points, the polygon is uniquely determined by noting the latitude and longitude of the northeast and southwest corners; these latitudes and longitudes form an index into the array of data points.

A bilinear interpolation is performed over the surface of the approximating polygon to locate the point on the surface "directly below" a sample data point. The distance (or error) between the surface and the data point is then calculated, as shown in Fig. 3(c), and is compared to the user-specified tolerance. If any of the errors exceeds the tolerance value, then the polygon is divided into four smaller polygons, as shown in Fig. 3(d), and the process is repeated recursively.

When the approximation to the data is within the specified tolerance, the recursion is terminated and the polygon is saved in a linked list. Upon completion of the subdivision algorithm, the resulting polygons are displayed for the user to judge the results. Generally, the user runs the program repeatedly while varying the maximum tolerance until the results are suitable.

2.2. *Shortcomings of the basic method*

Although the basic adaptive subdivision method provides a good starting point for this work, it has several shortcomings when polygons are used in place of bicubic patches in approximating the surface.

2.2.1. *Edge Gaps.* In [8] the authors use bicubic Bernstein–Bézier surface patches to model the data. The transitions across patch edges to neighboring patches are guaranteed to be smooth and continuous due to the constraints applied when solving for the patch control points.

This is not the case when using polygons to approximate the data. As shown in Fig. 4, it is very likely that the edges of neighboring polygons will not coincide. The result is that when the object is rendered using the polygonal approximation against a background of contrasting color, holes can appear where these edge gaps exist.

2.2.2. *Simplistic subdivision.* The algorithm presented by Schmitt *et al.* always subdivides the surface patch into four smaller patches along the latitude and longitude passing closest to the center of the patch. Such subdivision proved sufficient for their purposes and results in a simple and fast procedure. However, when used for surface-fitting with polygons, it can leave artifacts in the final rendering. Additionally, the characteristics of the sampled data are not used to intelligently guide or optimize the subdivision process.

The artifacts left by the basic subdivision method are similar to the creasing problem encountered with some methods for creating fractal mountains [9, 10]. Due to the regularity of choosing the division point of the polygons, the edges and corners of adjacent polygons tend to line up with each other. This can create "creases" in the surface of the final rendered object, causing the surface to appear unnatural.

2.2.3. *Aliasing.* There are two forms of aliasing that can occur when rendering the reduced data. The first results when an edge gap is viewed tangentially. This causes the outline of the rendered object to have a stair-step appearance. Data sets generated with large accuracy tolerances are much more susceptible to this effect than those with small tolerances.

The second form of aliasing results from the use of the IRIS's lighting models. The raw data produced by the Cyberware digitizer represents a mesh of sampled data points in a 512×512 grid. However, the volume scanned by the digitizer is usually taller than the subject. This causes the highest and lowest latitudes of the data grid to contain zero or meaningless values. Such latitudes are usually trimmed off when the raw data is edited and the data mesh is no longer square. In such a case, many of the approximating polygons become very long and narrow. The shading routines essentially calculate the correct shade at the vertices of the polygon and then use interpolation to determine the shading of the interior of the polygon. When several of these strip-like polygons are adjacent to a larger polygon, there can be a noticeable step in shading between the small polygons and the larger polygon. This problem can be minimized by using material definitions with low values of specular reflectance [11] and by using small tolerances.

2.3. VARIATIONS ON ADAPTIVE SUBDIVISION

As described above, there are several weaknesses in using the basic adaptive subdivision method for building polygonal approximations to sampled surface data. In order to overcome these weaknesses and to investigate alternative strategies, several variations were developed. Each variation is coded as a separate module, and the modules can be combined in many ways as shown in Fig. 5—one module from each level. There are three basic ways in which the subdivision procedure

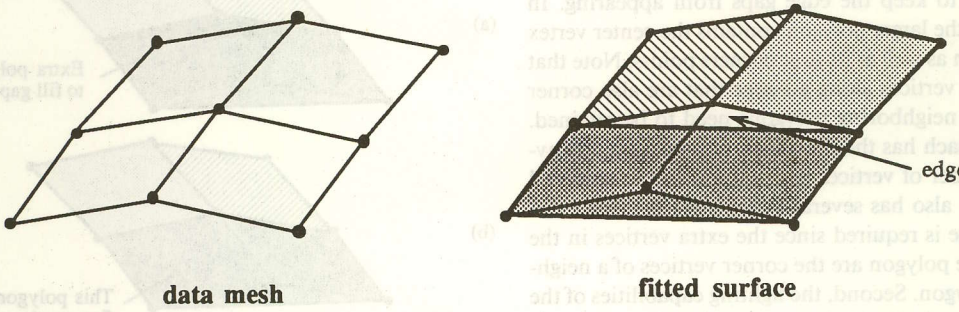


Fig. 4. Edge gaps.

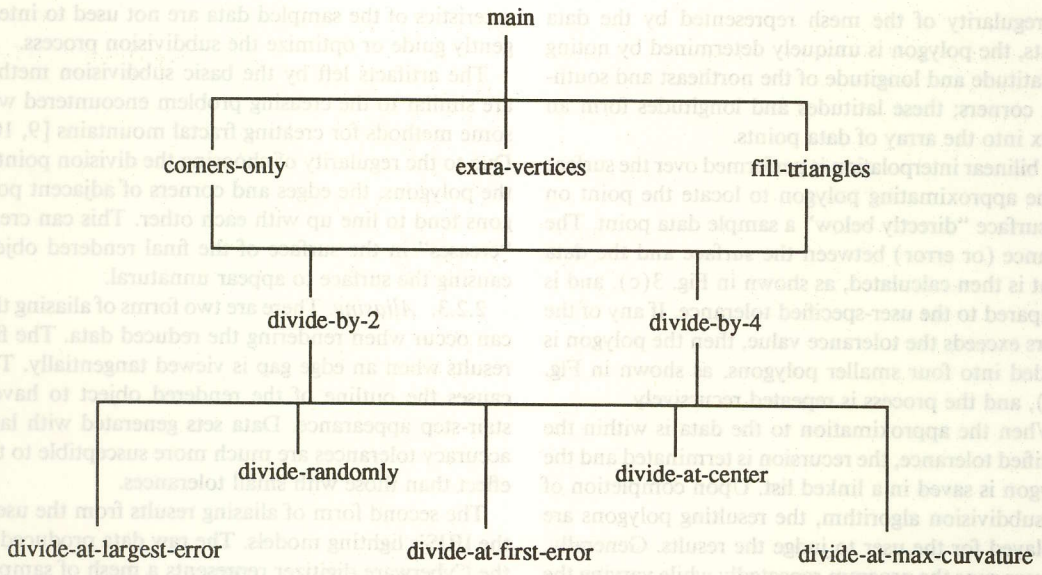


Fig. 5. Adaptive subdivision variation hierarchy.

can be modified—keep extra vertices or extra polygons to fill the edge gap, divide a polygon into two parts instead of into four parts, or change the way in which a polygon is actually subdivided.

2.3.1. *Filling the edge gap.* The first level of variation deals with the problem of edge gaps appearing in the reduced-data renderings. These gaps can be dealt with in several different ways. First, depending upon the accuracy required of the final rendering and the texture of the subject's surface, the edge gaps may not be visible and, therefore, no corrective action is necessary. Thus the *corners_only* module makes no attempt to fill the edge gaps, and was used in the original program that pointed out the edge gap problem. However, most of the test cases have demonstrated the need for filling these edge gaps.

One possible solution is to create additional polygons to fill these gaps as illustrated in Fig. 6(a). These extra polygons are created by connecting vertices lying along a common latitude or longitude with vertices that are also the corners of an adjacent polygon. The disadvantages of this solution are that more polygons are created which raises drawing complexity and these fill polygons lie in a plane perpendicular to the object's surface.

The alternative solution to the problem of edge gaps is to retain extra vertices along the edge of a polygon as needed to keep the edge gaps from appearing. In Fig. 6(b), the larger polygon contains the center vertex in the mesh as well as its four corners points. Note that only those vertices along an edge that are also corner vertices of neighboring polygons need to be retained. This approach has the drawback of increasing the average number of vertices per polygon in the rendered object, but also has several benefits. First, little extra data storage is required since the extra vertices in the edge of one polygon are the corner vertices of a neighboring polygon. Second, the lighting capabilities of the IRIS automatically fill the gaps with the correct shade depending on material, light(s), etc. And third, because vertex normals are used with the lighting model, the shading appears smooth across polygon boundaries.

2.3.2. *Number of Subpolygons.* In the original version of the polygon-surface-fitting procedure, a polygon was always subdivided into four smaller parts. This often resulted in long thin polygons that degraded the appearance of the rendered image. As a means to alleviate this problem, a variation of the algorithm was programmed to divided a polygon into two subpolygons. In this case, the decision to divide along a latitude or a longitude could be based on which dimension of the polygon is greater—north–south or east–west.

2.3.3. *Subdivision locations.* The remaining variations of the adaptive subdivision technique all concern the choice of how to subdivide a polygon that does not meet the accuracy specified by the user. This accuracy is measured as the distance from the surface of the approximating polygon to each sample data point being approximated. In each case, the accuracy of all the data points "over" the polygon are checked; as soon as one distance exceeds the tolerance, the subdivision method is invoked to determine where the polygon should be split.

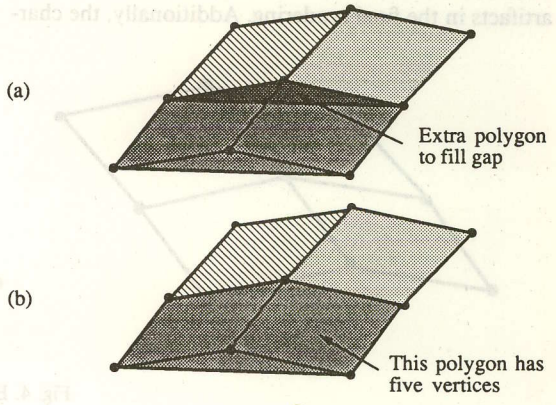


Fig. 6. Filling the edge gap.

2.3.3.1. *Center subdivision.* This is the subdivision method used by the original adaptive subdivision technique. No effort is made to analyze the data points to intelligently choose a point through which to subdivide a polygon. Instead, the algorithm simply finds the center latitude and longitude of the polygon and divides it along those lines.

2.3.3.2. *First error.* In determining the accuracy with which a polygon represents a group of sampled data points, the distance from the polygon's surface to each data point is measured and compared to the tolerance. Depending upon the dimensions of the polygon, these points are checked in a north-to-south/west-to-east or a west-to-east/north-to-south manner. The checking of the data points terminates when a measurement exceeds the tolerance, and the latitude and longitude of the offending point are used to divide the polygon.

2.3.3.3. *Greatest error.* This version is similar to *first_error*, except that accuracy measurement does not end when an error measurement exceeds the user specification. Instead, the process is continued so that the point of maximum error can be found. Upon completing the accuracy check, the latitude and longitude of the point of maximum error are used to subdivide the original polygon. Note that this slows down the procedure due to the large number of comparisons which must be made.

2.3.3.4. *Maximum curvature.* In attempting to intelligently choose a point to divide a polygon, consider how it might be done manually. An obvious place to put an edge of a polygon is along an apparent edge on the surface—such as along the edge of a cube. For a computer algorithm, detecting these edges is not a simple task. However, for the adaptive subdivision to work, the exact edges are not needed.

Consider the mesh of data points resulting from digitizing the surface of a cube and picture the normal vectors associated with each such data point. The data points that lie on the top side of the cube will have normal vectors that are all parallel to each other (or nearly parallel). As shown in Fig. 7, this will not be the case for the normal vectors of data points that lie on opposite sides of an edge. Therefore, the angle formed between the normals of nearby data points gives an indication of how sharply the surface is curving.

In this method, before the subdivision begins, each

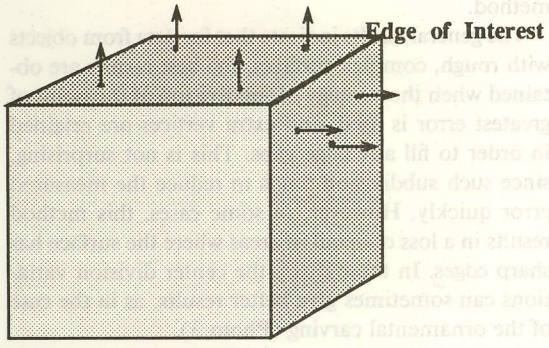


Fig. 7. Normal vectors on opposite sides of an edge.

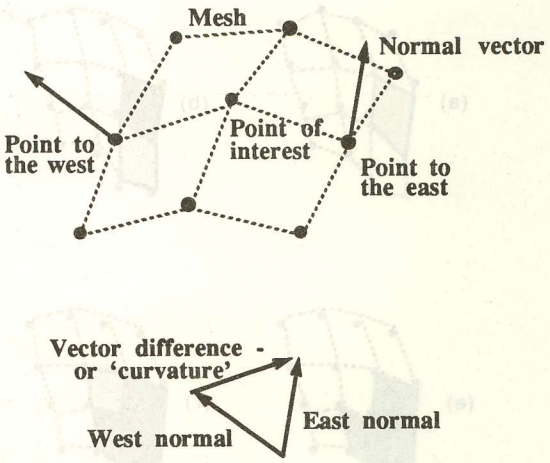


Fig. 8. Vector difference to represent curvature.

data point is assigned two "curvature" measures—one in the north–south direction and the other east–west. These measures are based on the length of the vector resulting from the subtraction of the normal vectors for the data points on either side of the data point being considered (see Fig. 8). Then, during the subdivision process, the location of maximum "curvature" is found in much the same way as the point of maximum error was located above. The latitude and longitude of this point then become the dividing lines for the new polygon. Because one of the points at the corner of a polygon will always have the highest curvature (since it was chosen as the place to subdivide the previous polygon), the corners should be excluded from the scan in order to prevent an infinite recursion.

3. POLYGON GROWTH

Another method to systematically generate surface-fitting polygons is to use a *polygon growth* technique (Fig. 2). First, a *seed* polygon is selected from the original set of data, as in Fig. 9(a). Then a neighboring polygon is selected and the two are *combined* into a larger trial polygon [Fig. 9(b) and (c)]. If the trial polygon passes the accuracy metric, additional neighbors are selected in an attempt at further growth, as in Fig. 9(d–g). When all attempts at further growth fail, the polygon is added to the list of polygons describing the reduced object, as shown in Fig. 9(h). A new *seed* polygon is selected and the process repeated. When there are no more polygons from which to choose a new seed, the process is terminated. Variations on this method include using a random or a sequential selection strategy for the next *seed* polygon.

This method of fitting polygons to the sampled data did not prove as useful as the adaptive subdivision methods, for several reasons. First, it is an essentially brute force method of surface fitting and very repetitive in nature. Many trial polygons are created, only to be "thrown out" because they exceed the error tolerance. This causes the procedure to execute quite slowly—orders of magnitude more slowly than the adaptive subdivision methods. Second, the data is not significantly reduced. Many of the generated polygons are

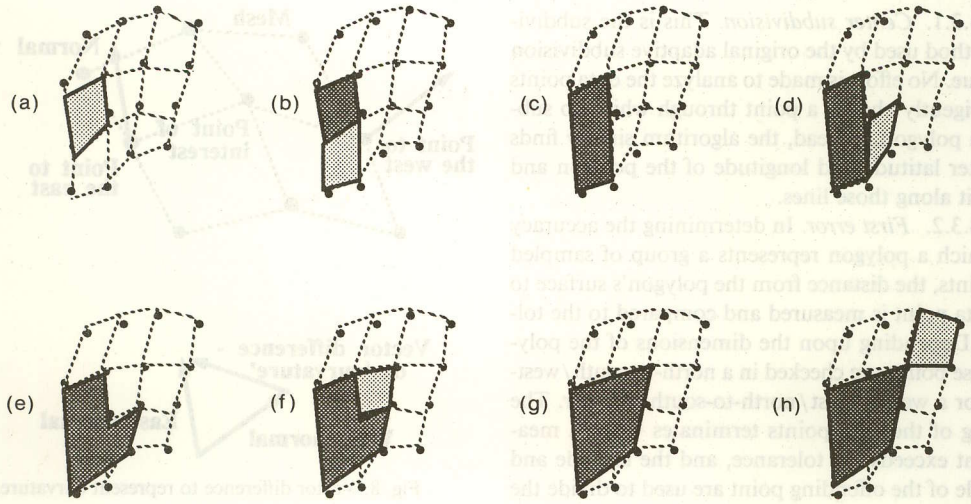


Fig. 9. Polygon growth.

of complicated shapes and use a large number of vertices. Thus many of the original sample data points are retained, increasing storage requirements and slowing drawing times.

4. RESULTS

This project was undertaken to investigate possible methods for creating realistic ships, trucks, and other inhabitants of the real-time 3D visual simulators being developed at the Naval Postgraduate School. The majority of the research efforts were directed at developing the adaptive subdivision techniques, since this area seemed most promising for near term results.

The general test procedure consisted of viewing renderings of both the full data set and of a reduced data set. The data reduction program was repeated while the tolerance was adjusted based on trial-and-error and experience in order to achieve a good balance between the accuracy of the rendering and its drawing complexity. This process was then repeated for each of the variations of the adaptive subdivision method. The quality of the final renderings is necessarily a subjective judgment made by the user; however, every effort was made to be consistent across all of the reduction variations for each test subject.

Photos 1 through 3 show the results from applying the adaptive subdivision variations discussed in section 2 to three different digitized test objects. Photo 1 displays the results from a digitized tube of artist's paint. Photo 1(a) shows a rendering of the paint tube using all of the data produced by the digitizing process. Photo 1(b) shows the same tube of paint after applying a variation of the adaptive subdivision algorithm. In this case, the variation used subdivided each polygon into two subpolygons at the location of the maximum error, and extra vertices were retained in order to fill the edge gaps. Photo 2(a) shows a rendering of Spock's bust with the full set of data points. Photo 2(b) shows the rendering of the reduced data using the same technique (g2e) as used for the tube of paint in Photo 1. Photo 3 is from a linear scan of an ornamental carving; Photo 3(a) shows a rendering of the carving with the full set of sampled data points while Photo 3(b) shows the rendering of the reduced data. The carving method used subdivided each polygon into four parts at the latitude and longitude nearest the polygon's center. Photo 4 shows the wireframe of the reduced carving.

Each photo is accompanied by a table of various measures of the algorithm's performance. The entries labeled *polygons* and *vertices* list the number of polygons and the number of vertices retained from the original sampled data set. Since the polygons may not have four vertices per polygon, the number of polygons and the number of vertices only give a partial evaluation of the complexity of the final rendered object. In order to get a better estimation of complexity, the number of calls to the IRIS's graphics library are also tabulated (*library calls*).

The data structures used for storing the object consists of a table of vertices and a linked list of polygons. Each vertex structure in the table contains fields for the $x/y/z$-coordinates and the $i/j/k$-components of the unit normal vector at that vertex. The polygon structures are maintained in a linked list and contain a pointer to an array of pointers to entries in the vertex table. The *memory* entry in the tables gives the number of bytes needed to store such a data structure. And for the reduced data sets, the *tolerance* entry is the tolerance value used with that particular adaptive subdivision method.

The general results indicate that for data from objects with rough, complex surfaces, the best results are obtained when the strategy of subdivision at the point of greatest error is used, and extra vertices are retained in order to fill any edge gaps. This is not surprising, since such subdivision tends to reduce the measured error quickly. However, in some cases, this method results in a loss of detail in areas where the surface has sharp edges. In these cases, the center division variations can sometimes give better results, as in the case of the ornamental carving (Photo 3).

Better results were expected from the strategy of di-

viding at the point of maximum curvature. This technique seems to work well in areas of rapidly curving surfaces, but begins to suffer in areas that are relatively smooth and flat. In such areas, the difference between adjacent normal vectors tends toward very small values, possibly smaller than the digitizing camera's resolution, causing the choice of the subdivision point to become more random. A hybrid technique that uses curvature based subdivision for the first few levels of recursion and then switches to a greatest error technique when the surfaces become nearly flat may improve these results.

5. FUTURE DIRECTIONS

The polygon-fitting techniques discussed thus far create a polygonal surface that approximates sampled data by "adding" polygons until the approximation falls within a user specified tolerance. The user can control the accuracy of the final rendered object, but cannot directly control the final number of polygons (by running the procedure repeatedly and "tweaking" the tolerance, the user can zero in on the desired number of polygons). In many cases, it would be desirable to specify that the final approximation consist of a given number of polygons and have the data reduction procedure return the *best* approximation using that number of polygons.

This second approach to the problem can be viewed as a search. The nodes of the search correspond to sets of **P** polygons to be used in the rendering, and the search space consists of all possible combinations of those **P** polygons out of the **N** original polygons. The goal of the search then is to find the set of polygons that *best* approximates the original data. Because the problem is combinatorial in nature (*i.e.,* the search space grows as the factorial of **N**), brute force search techniques would not be feasible for most real world problems, therefore, techniques borrowed from the artificial intelligence field such as simulated annealing and the use of heuristics are to be investigated [12, 13].

5.1. *Search*

In trying to find the *best* approximation to the original data, one is trying to minimize an error function (*i.e.,* maximize the accuracy). One approach to minimizing a function is to choose a possible solution and evaluate the function. By carefully changing the function's arguments in small increments, its value can be made to decrease. When no further changes to the arguments results in a decrease in the function's value, a minima has been found.

A problem that arises with this incremental improvement strategy is that this procedure can become trapped at a local minimum and, therefore, never find the global minimum (Fig. 10). This is because since only changes that move closer to the goal (minimum error) are allowed, there is no way of climbing over a small "hump" in the function's value. One way to minimize this effect is to repeat the incremental improvement process many times, each from different starting points and simply retaining the best solution.

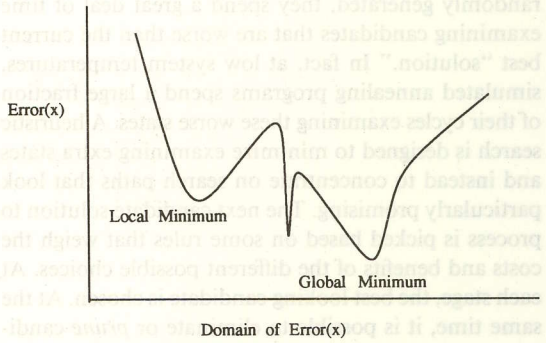


Fig. 10. Local minima in an error function.

However, this can fail if the minimum is located in a narrow "valley" in the error curve.

5.2. *Simulated Annealing*

Simulated annealing is similar to the incremental improvement technique except that "uphill" excursions are allowed in a controlled way. An initial guess for the best solution is formed and evaluated. The best solution is then randomly permuted in an effort to improve it. If the permutation results in a lower error, then it becomes the new best solution and the permutation/evaluation cycle is repeated. If the new guess has a higher error, a probability function is used to determine whether to replace the current best solution.

The probability is determined by a system parameter called the system *temperature*. During the initial stage of the algorithm, the temperature is high and the probability that a more costly solution is allowed to replace the current solution is also high. As the process continues, the temperature is gradually reduced and the system "cools." This results in a lower probability of keeping a solution that is "uphill" from the current one. Eventually, the system *freezes* and no more changes are possible; the current guess is then the solution.

At high temperatures, the solution is allowed to make many long random uphill jumps, effectively conducting a broad survey of the search space. As the system cools, the uphill jumps get progressively smaller and less frequent until the system freezes. Since all better solutions are accepted and only some of the poorer guesses are accepted, the overall trend is a movement toward smaller errors. The occasional "uphill" excursion helps keep the solution from getting stuck at a local minimum.

Simulated annealing does not guarantee the most optimal solution, but it has been used effectively to obtain near-optimal solutions for very large and complex problems [12]. By choosing an initial set of polygons with which to render a digitized object and permuting this set in a consistent way, this technique may be useful in data reduction efforts.

5.3. *Heuristic Searches*

Stochastic search techniques such as simulated annealing examine a large number of states in the search space, and because each candidate successor state is

randomly generated, they spend a great deal of time examining candidates that are worse than the current best "solution." In fact, at low system temperatures, simulated annealing programs spend a large fraction of their cycles examining these worse states. A heuristic search is designed to minimize examining extra states and instead to concentrate on search paths that look particularly promising. The next candidate solution to process is picked based on some rules that weigh the costs and benefits of the different possible choices. At each stage, the best looking candidate is chosen. At the same time, it is possible to eliminate or *prune* candidates that look bad, or look worse than the current solution by some threshold value.

If we consider the solution to be a set of **P** polygons, then our method for generating candidate solutions might be to exchange a polygon in our set **P** with one not in the set. A possible heuristic to use in choosing which polygons to exchange would be to swap out the polygon with the largest curvature (*i.e.,* is most nearly flat) with one that has a small curvature. Similarly, we may instead choose to rank the edges or vertices to discover which vertices, edges, and polygons carry more meaning, etc. The idea is to use one's knowledge about the system and the problem in order to more quickly guide the program to the solution.

6. CONCLUSIONS

Current work on this project has focused on the systematic generation approaches to the simplification of rendered objects. Both adaptive subdivision and polygonal growth methods have been implemented, along with several of the variations. The initial results are very encouraging; sample data of a human bust consisting of 112,640 points was successfully reduced to 12,821 polygons using the extra vertex-greatest error variation of the adaptive subdivision method (g2e). This drastic reduction in object complexity resulted in an order of magnitude increase in drawing speeds. Further investigation into improving the accuracy metrics, and into rules concerning how to subdivide a polygon should result in greater reductions.

Other issues include:

- loss of features that are finer than the digitizing camera's resolution.
- methods of combining several sets of data from the camera for objects too large to digitize in one pass.
- being able to have different accuracy metrics for different parts of the same object (*e.g.,* when working with a digitized bust, may want to require higher accuracy in the area of the face than in the back of the head).
- automatically adjusting the tolerance based on surface features.
- using surface color information that will be provided by the next generation of the Cyberware Laboratories rapid digitizer.
- defining boundaries for areas of an object's surface, so that simplification occurs within these boundaries. This would allow assignment of colors to parts of the object without fear of a polygon crossing from one color to another.
- eliminating the artifacts produced by the subdivision methods.
- improving execution times of the polygonal growth technique.
- investigate various data structures for storing and manipulating the objects.

These issues represent areas of current investigation; for the future, research into the use of search techniques will be expanded. Small scale programs written to learn and experiment with these techniques need to be scaled up and adapted to the data reduction problem. Effective heuristic rules need to be devised in order to allow such techniques to be used efficiently, and ways of measuring how accurate a rendering will look to a user need to be investigated.

Our goal is to be able to use these processes to create models for the simulators developed here at NPS. By using the digitizing process on scale models, the design, creation, and coding of vehicles and other objects for these simulators can be made easier, faster, and more realistic. Initial results are good, indicating that this is a promising area of investigation.

REFERENCES

1. M. Zyda, R. McGhee, R. Ross, D. Smith, and D. Streyle, Flight simulators for under $100,000. *IEEE Computer Graphics and Applications* **22**(1), 19–27 (January 1988).
2. D. Addleman and L. Addleman, Rapid 3D digitizing. *Computer Graphics World* **8**(11), 41–44 (November 1985).
3. K. Akeley and T. Jermoluk, High performance polygon rendering. *Computer Graphics* **22**(4), 239–246 (August 1988).
4. G. K. Weeks and C. E. Phillips, *The Command and Control Workstation of the Future—Subsurface and Periscope Views.* Masters Thesis, Naval Postgraduate School, Monterey, CA (June 1989).
5. M. J. Zyda, M. A. Fichten, and D. H. Jennings. *Graphics Workstations and 3D Visual Simulation: Some Performance Expectations and Measurements.* Naval Postgraduate School Technical Report NPS52-88-022 (July 1989).
6. R. P. Strong and M. C. Winn, *The Moving Platform Simulator II: A Networked, Real-time Visual Simulator with Distributed Processing and Line-of-Sight Display.* Masters Thesis, Naval Postgraduate School, Monterey, CA (June 1989).
7. W. O. Breden and J. J. Zanoli, *Visualization of High Resolution Digital Terrain.* Master Thesis, Naval Postgraduate School, Monterey, CA (June 1989).
8. F. J. M. Schmitt, B. A. Barsky, and Du Wen-Hui, An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics* **20**(4), 179–188 (August 1986).
9. M. F. Barnsley, et al.: *The Science of Fractal Images.* New York: Springer-Verlag (1988).
10. G. S. P. Miller, The definition and rendering of terrain maps. *Computer Graphics* **20**(4), 39–48 (August 1986).
11. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vectchi, Optimization by simulated annealing. *Science* **220**, 671–680 (1983). Reprinted in *Neurocomputing,* 554–568, MIT Press (1988).
12. Silicon Graphics, Inc., *GT Graphics Library User's Guide* (1988).
13. R. E. Korf, Search: A survey of recent results. In *Exploring Artificial Intelligence,* H. E. Shrobe (Ed.), San Mateo, CA: Morgan Kaufmann Publishers Inc. 197–239 (1988).
14. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design—A Practical Guide,* Academic Press Inc., New York (1988).