A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces

Leonard McMillan

Abstract

Projected grid surfaces are a particularly interesting subset of bivariate parametric functions. Images generated using three-dimensional computer graphics methods and those resulting from ideal pin-hole cameras fall into this class. Often, a range image is provided in addition to the actual image, this may be in the form of a Z-buffer for synthetic images or stereo-disparity field for acquired images. In this research I report on an algorithm for computing the visible subset of a projected surface from any arbitrary view. This method has several important properties. It determines an occlusion-compatible paint order such that surfaces closest to the desired viewpoint are drawn last, thus overwriting any previously rendered surface. This painting order is determined independent of the range information. Also, facets of the image are drawn sequentially, taking advantage of local spatial coherency. This report also describes a method for the fast rendering of small triangles using a table driven approach.

Introduction

A classic problem in computer graphics is the display of bivariate functions and surfaces of the form $z = f(x, y)$. Several special-purpose visibility algorithms have been developed to render these specific types of surfaces, which are often called *gridded surfaces* or *height fields*. The purpose of this research is to develop a visibility algorithm for a related class of gridded surfaces, the *projected grid surface*.

One of the earliest solutions to the gridded surface problem was given by Kubert, Szabo, and Guilieri [Kubert68]. Later, Wright enhanced Kubert's algorithm resulting in the familiar floating horizon algorithm, which is still commonly used today [Wright73]. More recently, improved algorithms for rendering arbitrary views of bivariate surfaces have been suggested by [Anderson82] and [Wang90].

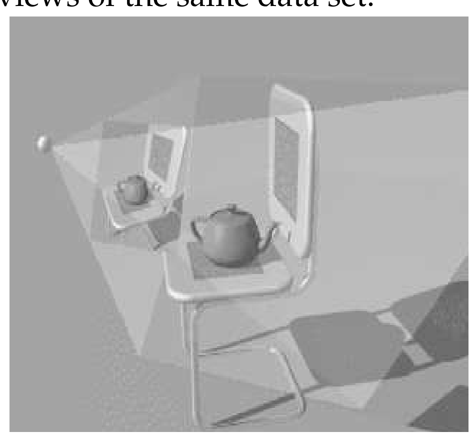
All of these algorithms for displaying gridded surfaces compute a hidden-line solution instead of a hidden-surface solution as is common in other computer graphics applications. This seems reasonable since a hidden-line representation can provide important insights into a surface's shape beyond what a smooth-shaded image can do. This is particularly true when the surface is, on average, smooth, as is the case for most mathematical surfaces.

There are, however, striking similarities between the popular floating horizon hidden-line algorithm and common hidden-surface algorithms, such as the Z-buffer algorithm. Both algorithms operate in image space and are tightly-coupled with the rasterization process. Both algorithms require a comparison operation for every pixel potentially displayed in order to determine the visibility at that sample location or column. The floating horizon algorithm, however, has the advantage that it is more space efficient since it only requires two row-sized buffers, while the Z-buffer algorithm requires a full screen-sized array of depth values.

In contrast, the hidden-line algorithms suggested by Anderson and Wang operate in object space. They resolve the visibility of line segments based on the unique properties of a gridded data structure. One advantage of their approach is that list-priority rendering techniques, which do not involve comparisons at each pixel displayed, can be used to generate the result. The major objective of this research is to adapt these object-space techniques to a different class of gridded surfaces.

The distinguishing characteristic between projected grid surfaces and bivariate surfaces is that the grid parameterization of a projected surface is imposed after projection. The actual three-space coordinates of these grid sample points are not constrained to lie on any sort of lattice or parametric grid, as bivariate functions are. Furthermore, the grid structure imposed upon a projected grid is only valid for a given center of projection; for any other point in space the surface is not guaranteed to lie on a grid. Thus, the grid coordinates only make sense when considered in conjunction with a given set of view parameters. In contrast, the parametric definition of a bivariate surface is independent of the viewpoint. This is equivalent to saying that the defining function, z = f(x,y), of the bivariate surface does not correspond to any perspective view of the function, whereas a projected grid surface does correspond to some view.

The question arises: what is the importance of projected grid surfaces? Consider that virtually all naturally acquired imagtry, such as photographs and video images, are projected grids. Most three-dimensional computer graphics renderings correspond to projected grids. Therefore, an obvious application for the rendering of projected surfaces is the redisplay of existing images from a different point-of-view. If efficient methods for this redisplay could be found, it would allow the reuse of previously computed images in the computation of new views of the same data set.

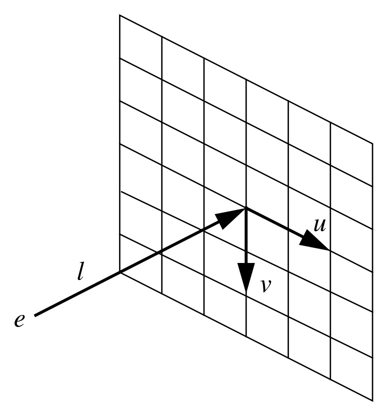


Definitions

A projected grid surface is a bivariate surface defined over some domain $x'_0, x'_1, \ldots, x'_n$ and $y'_0, y'_1, \ldots, y'^n$, where any point on the grid, $\boldsymbol{P}' = (x'_i, y'_j)$, is the projection of some real point, $\boldsymbol{P}$, in a higher dimensional space, such that $\boldsymbol{P} = (x, y, w)$, $x'_i = x/w$, and $y'_j = y/w$.

Projection grids differ from typical bivariate surfaces in that the grid structure is imposed after the projection rather than before. Points which lie on the projection grid, $\boldsymbol{P}'_{ij} = (x'_i, y'_j)$, are called *grid points*. The region of the projected grid surface which is bounded by the grid points (x'_i, y'_j), (x'_{i+1}, y'_j), (x'_{i+1}, y'_{j+1}), and (x'_i, y'_{j+1}) is the called the *facet*, $\boldsymbol{F}_{ij}$.

Any particular planar projection or *view*, V, can be characterized with the following four parameters: a center of projection, or eye position, $\vec{e}$; a vector from the eye to the origin of the viewplane, $\vec{l}$; and two basis vectors which sweep out the viewplane, $\vec{u}$ and $\vec{v}$, in the positive *i* and *j* directions respectively. This formulation is slightly more general than the typical characterization which likewise includes an eye position and a look-at vector, but in place of the two basis vectors it uses a field of view and an aspect ratio. The proposed form is capable of representing skewed projections (if l, u, and v are not orthogonal) and projections where the field-of-view is not symmetrical about the look-at direction. Both the original projection grid (denoted as V_0) and the resulting view (denoted as V_i) can be described with their own unique set of viewing parameters, $V_0 = \left(\vec{e}_0, \vec{l}_0, \vec{u}_0, \vec{v}_0\right)$, and $V_i = \left(\vec{e}_i, \vec{l}_i, \vec{u}_i, \vec{v}_i\right)$. The components of V are shown in the following diagram.



The locus of potential points, $P_x(i, j, t)$, that project onto a given grid point $\boldsymbol{P}'_{ij}$ of the projection grid described by view, V_x, are described by the following equation:

$$P_x(i, j, t) = \bar{e}_x + t\left(\vec{l}_x + i\vec{u}_x + j\bar{v}_x\right)$$

This implies that each grid point, (x'_i, y'_j), requires an associated value, t, as part of its definition in order to uniquely identify a given point. The t value is analogous to the z value stored in a Z-buffer, or the ray-length maintained by a raycasting algorithm. While t is an important element of the reprojection process, it is, surprisingly, not required to establish the relative visibility of facets.

The projection, $\Phi(V, P)$, of an arbitrary point, P, onto a planar view, V, is given by the projection formula:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix}_V = \begin{bmatrix} \vec{v} \times \vec{l} \\ \vec{l} \times \vec{u} \\ \vec{u} \times \vec{v} \end{bmatrix} (\bar{P} - \bar{e})$$

where $[x, y, w]_V$ is the homogeneous coordinate of the projected point $P' = \left(\frac{x}{w}, \frac{y}{w}\right)$, within the view V.

Algorithm

The desired rendering algorithm will need to enumerate the facets *F1, F2, F3, …, FN*, where $N = (n-1)(m-1)$, in such a way that if $\boldsymbol{F}_j$ occludes $\boldsymbol{F}_i$ then $j > i$. I will call this order *back-to-front occlusion compatible*. This approach is similar to Anderson's algorithm for bivariate functions with the exception that his visibility algorithm enumerates the facets in a front-to-back occlusion order. Anderson's choice of a front-to-back order requires that some representation of the grid perimeter be maintained to aid in deciding what parts or edges of subsequent facets need to be rendered. This representation requires auxiliary storage for maintaining a list of edges defining the perimeter. This list must then be queried before each new facet's edge is displayed, and the display must be updated if any part of the facet is visible. In contrast, a back-to-front ordering requires no additional storage because the proper occlusion is handled by the drawing order in a way similar to the classic painter's algorithm [Rogers85].

The algorithm for redisplaying a projected surface from an alternate viewpoint, V_x, is described as follows:

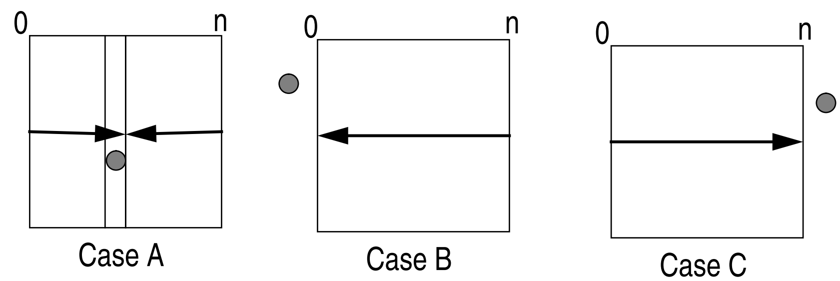
1) Compute the projection $e' = \Phi(V_0, e_x)$.
2) Generate a proper enumeration order based on the value of e'.
3) Reproject, $P' = \Phi(V_x, P_0(i, j, t))$, facets from the original projected grid to the new view position, V_x, according to the enumeration order of step 2.
4) Resample the final projected grids to generate the final image.

In order to generate a proper enumeration order based on the value of e', several cases must be considered. In the following case analyses, $e' = (x, y, w)$ represents the projected eye position. The enumeration along the i and j directions are considered independently. When the w element of e' is positive, the facets can be enumerated as follows:

Case A) If $x' = x/w$ falls within the range 0 to $n-1$, then the list of facets must be partitioned into two groups; the first enumerated from $F_{0,j}$ to $F_{x',j}$ and the second ordered from $F_{n-1,j}$ to $F_{x'-1,j}$.
Case B) If $x' = x/w$ is less than zero, then a single ordering from $F_{n-1,j}$ to $F_{0,j}$ is required.
Case C) If $x' = x/w$ is greater than or equal to n, then a single ordering from $F_{0,j}$

to $F_{n-1,j}$ is needed.

The three cases are shown below with the eye's projected position shown as a darkened circle.



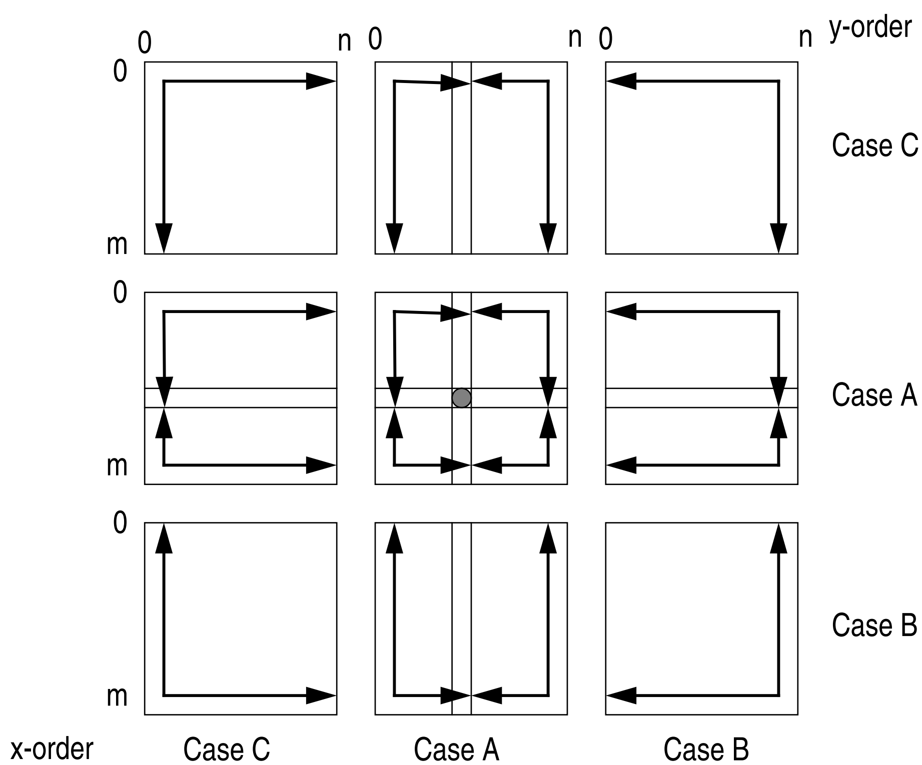
Next, the facets are ordered in y according to the following three cases.

Case A) If $y' = y/w$ falls within the range 0 to $m\text{-}1$, then partition the facets into two groups; the first ordered from $F_{i,0}$ to $F_{i,y'}$ and the second ordered from $F_{i,m-1}$ to $F_{i,y'-1}$.

Case B) If $y' = y/w$ is less than zero, then a single ordering from $F_{i,m-1}$ to $F_{i,0}$ is used.

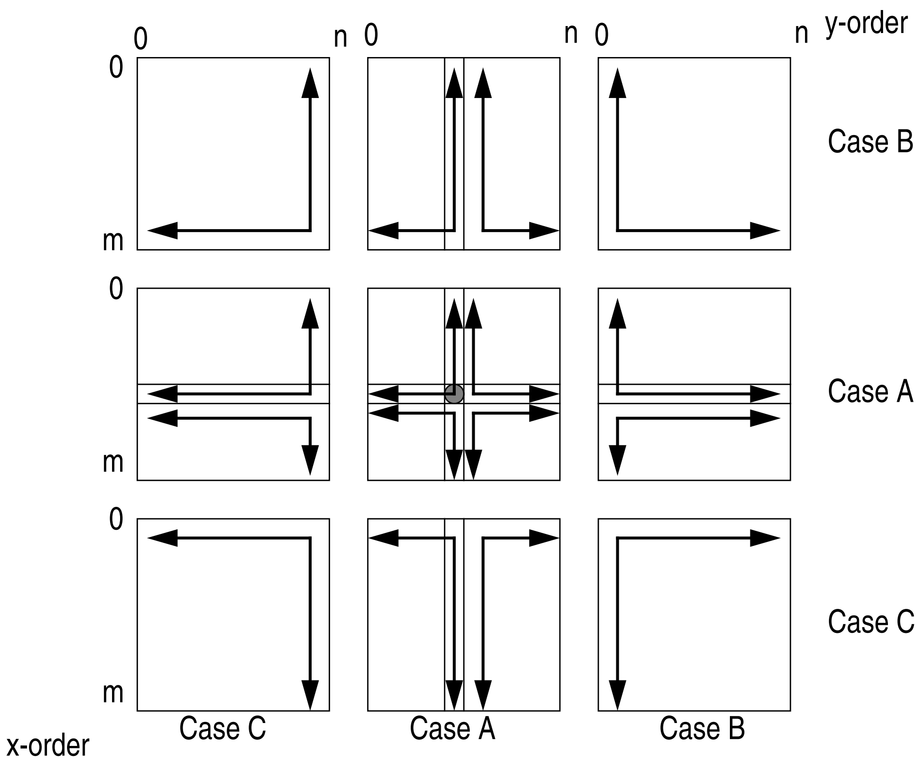
Case C) If $y' = y/w$ is greater than or equal to m, the order is from $F_{i,0}$ to $F_{i,m-1}$.

The following diagram shows the nine possible enumeration orders after both the x and y orders have been determined. All of the projected grid enumerations are shown rela-



tive to the single eye position shown in the center of the figure.

A similar breakdown results when the sign of w is negative. The following diagram shows the nine possible enumerations based on the projected coordinate of e'.



The enumeration order would typically be implemented as nested for loops. Thus, the projection of the desired eye position, e', would first be used to determine whether the surface is to be subdivided into one, two, or four sheets. Next, the starting value, terminal value and increment direction for each of the sheets is assigned. The facets are then painted by looping through the three-space coordinates of the grid points within a given sheet. These grid points are then reprojected to the desired view, V_x. The resulting image will be the correct hidden surface solution for the original projected grid surface when viewed from the new viewpoint.

Implementation Results

A demonstration program, called *vwarp*, that uses the list-priority algorithm for redisplaying projected surfaces has been implemented on a standard UNIX workstation running the X-window system. The program accepts the names of two inputs files as command line arguments. The first is a file of t values representing the radial distance from the center of projection to the object visible at each pixel in the image. The second file contains a true-color band-interleaved image with three bytes representing the color at each pixel (red, green, and blue channels). Once the program has started, the original projected grid surface is displayed on the screen. The user can then supply interactive input to the program using the mouse.

The mouse interface operates as follows:

- Any cursor motion while the first button is depressed will cause the eye position to move in the direction of the cursor. Thus, if the cursor moves to the left, the scene will be reprojected as if the scene were being viewed from a position that is to the left of the original projection. As a result, the apparent motion of objects in the scene will be in a direction opposite of the mouse's motion. The first button allows left/right and up/down changes in the eye's position.
- Any up or down cursor motion when the second mouse button is depressed will cause the eye's position to move into (up) or out of (down) the scene. The motion is along the look-at direction. Any left or right motion of the mouse with the second button is depressed is ignored.
- The third button pops up an options menu. The desired option is selected by releasing the third button while a particular option is highlighted. Options are highlighted when the cursor lays over the text describing the option. If the user does not wish to select an option from the menu, then the third button should be released while the cursor is outside of the window.

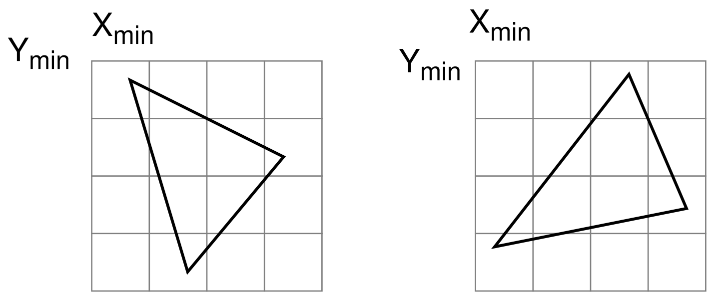
The names of the various menu options and their actions follows:

1) **Print Eye** outputs the current coordinate of the center of projection.
2) **Double Range** increases the rate of the eye's motion relative to the mouse's.
3) **Half Range** decreases the rate of the eye's motion relative to the mouse's.
4) **Dblbuff On/Off** alternates between single and double buffering.
5) **Points** displays only the projected grid points.
6) **All Tiles** displays all facets of the projected grid surface.
7) **Small Tiles** displays only those facets that fit within a 4×4 bounding box.
8) **Reset** moves the eyepoint to the center-of-projection of the original projected grid.

Vwarp supports either an 8-bit pseudocolor or a 24-bit Truecolor or Directcolor visual. In pseudocolor mode it uses 216 colormap entries.

The "Small Tiles" option reflects an interesting optimization that is used in the *vwarp* program. Early in the development it was noticed that the majority of facets, which are rendered as two triangles between each set of four neighboring grid points, project to very small polygons, most composed of no more than four pixels. For such polygons the setup for the rasterization process far exceeds the time required to write out the pixels. As a result, performance of the "Points" mode was an order of magnitude faster than the faceted "All Tiles" mode. This was particularly disconcerting since, on average, little more than twice the number of pixels were actually written in the faceted case. As a result I attempted to come up with a rasterization technique which would require considerably less setup.

The resulting technique was a rasterizer for small triangles based purely on table look-ups. The new "tiny triangle" routine takes three vertices as inputs and determines if the required triangle will fit within a 4×4 bounding box. If it does not. then the routine returns with an error code. If the triangle fits within the bounding box, it is then classified as one of the two types shown in the following figure.



Type A: X_{min} and Y_{min} are determined by a single vertex.

Type B: X_{min} and Y_{min} are determined by two vertices.

The coordinates of each vertex are then quantized to the nearest half pixel and then normalized by subtracting the value of X_{min} and Y_{min}. The resulting coordinate values are combined to form an index into one of two tables based on the triangle's type. Each of these tables has 2^{14} entries when the coordinates are represented to half a pixel accuracy, and 2^{20} entries for a quarter-pixel accuracy. Each entry is composed of 32 bytes, two for each of the 16 pixels in the 4×4 grid.The values stored, S_1 and S_2, are scaled versions of the baricentric coordinates for two of the vertices at the given pixel. The scaling function is $S_i = 255^*w_i+1$. The value of the third coordinate can be found by subtracting the two known coordinates from $S_3 = 257 - S_1 - S_2$. If any of S_1, S_2, or S_3 are negative, then S_1 and S_2 are set to zero.These weighing values can be used to interpolate arbitrary parameters defined at the triangle's vertices (colors in this case). This scaling is also accomplished with a table lookup using a single table with 2^{16} entries, corresponding to the 8 bits of S_i and the 8 bits used to represent the color value.

The tiny triangle renderer works as follows:

1) compute the bounding box for the three input vertices V_1, V_2, and V_3
2) if the bounding box is larger than 4×4, return with a "too big" error
3) determine the triangle type A or B
4) subtract x_{min} and y_{min} from V_1, V_2, and V_3 and munge results into a table index
5) for each of the 16 pixels read the table entry
6) if S_1 is non-zero then
7) compute $S_3 = 257 - S_1 - S_2$
8) red = mult[S_1][V_1.r] + mult[S_2][V_2.r] + mult[S_3][V_3.r]
9) green = mult[S_1][V_1.g] + mult[S_2][V_2.g] + mult[S_3][V_3.g]
10) blue = mult[S_1][V_1.b] + mult[S_2][V_2.b] + mult[S_3][V_3.b]

11) write pixel with (red, green, blue)

The tiny polygon routine is about 5-6 times faster than a standard polygon renderer. The resulting performance is about one third of the speed of displaying the grid points only, while twice as many pixels are written.

Conclusions

I have developed an algorithm for the redisplay of projected grid surfaces from arbitrary view points. The algorithm suggested has many advantages over alternative approaches. It generates a back-to-front occlusion-compatible ordering of the grid facets which can be rendered using a simple painter's algorithm. This ordering is determined by examining a single point, the position of the new center of projection. The ordering is independent of the depth values at the grid points on the projected grid surface.

There are many potential applications for this technique. It can be used as an interactive rendering technique for environments where only small changes in the viewpoint are expected. This approach allows the results of non-real-time rendering techniques, such as ray tracing, to be reprojected from new viewpoints interactively. It might also be used for post-rendering correction in order to compensate for the latency of classical graphical rendering systems. In this application, the rendering of a scene begins with a viewing position and orientation defined at t_N. However, by the time the rendering process has completed, at t_{N+1}, the actual head position may have changed. The algorithm given could be used to reproject the computed image to correspond to the current head position.

Other potential applications include progressive and adaptive refinement where a previously computed image might be first reprojected in order to provide an initial intermediate-quality result, capable of directing the efforts of a higher quality renderer.

Acknowledgments

This research was done a course project requirement for COMP 236 in the spring semester of 1993. The course was taught by J. Turner Whitted.

References

[Anderson82] Anderson, D., "*Hidden Line Elimination in Projected Grid Surfaces*," **ACM Transactions on Graphics**, October 1982.

[Kubert68] Kubert. B., J. Szabo, and S. Giulieri, "*The Perspective Representation of Functions of Two Variables*," **Journal of the Association for Computing Machinery**, April 1968.

[Rogers85] Rogers, D.F., **Procedural Elements for Computer Graphics**, McGraw-Hill, New York, NY, 1985.

[Wang90] Wang S.C. and J. Staudhammer, "*Visibility Determination on Projected Grid Surfaces*," **IEEE Computer Graphics and Applications**, July 1990.

[Wright73] Wright, T., "*A Two-Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables*," **IEEE Transactions on Computers**, January 1973.