Encoding 3D Surface Information in a Texture Vector

Michael Bailey
Dru Clark

University of California at San Diego and San Diego Supercomputer Center[1]

Abstract

A novel use of texture mapping is described. Surface scalar information, in this case the surface normal, is used as the 3D texture vector which is then transformed by the global rotation matrix to keep the normal oriented with the part, and by a special matrix that is used to isolate just the Z component of the transformed normal. The result is fast surface color fluctuations that are used within a human-in-the-loop interactive system for optimizing mechanical part orientation for fabrication.

The Problem

The UCSD/SDSC TeleManufacturing Facility (TMF) project has integrated a solid freeform fabrication (SFF) capability with the Internet to make the production of prototype parts easy and convenient ([BAILEY95]). As a result, manufacturing "amateurs," such as biologists, chemists, ecologists, geologists, and mathematicians are able to produce 3D solid models to better visualize their work.

The TMF is using a Laminated Object Manufacturing (LOM) machine from Helisys, Inc. [HELISYS97]. In the LOM process, the 3D object is made from layers of .0042" thick paper. In the LOM process, new paper is spooled into place and laminated to the layers beneath it with a hot roller. A laser cuts the part outlines at this level. The outlines are essentially the contour lines for this height on the 3D object. The process continues until the 3D part is completed. (See [BAILEY96] for more information on this and other SFF processes.)

After the laser cuts the part outlines, it needs to do something with the portion of this layer that is not part of the object being fabricated. This portion of the layer cannot be made to "fall away" on its own like some of the liquid-based SFF processes, so the LOM process crosshatches it as scrap. Layer-after-layer of this scrap is built up. These scrap "columns" must then be plucked from the resulting part after it is completed. A part in various stages of de-scrapping is shown in Figure 1. Some of these scrap columns fall away trivially, but some are so difficult to remove that one ends up accidentally ripping some paper that was meant to belong to the part's exterior surface. Clearly having the scrap fall away trivially is a major requirement for a quality part.

[1] Authors' address:
PO Box 85608
San Diego, CA 92186
mjb@sdsc.edu
dru@sdsc.edu

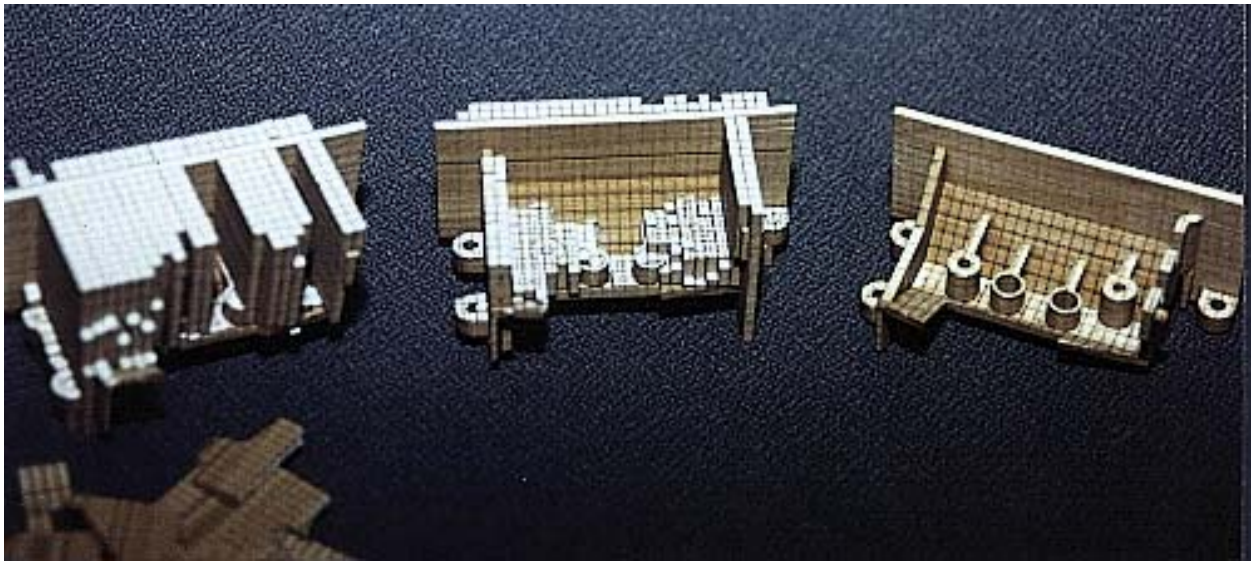

Figure 1: Various Stages of LOM De-scrapping

The ease with which a scrap column falls away is a function of how weak the connection is between the base of the column and the part exterior. This in turn is proportional to the density of contour lines at the base-part interface. The more contour lines, the more likely the scrap column will just fall off. From our experience, a contour line density of about 100 lines/linear inch will make this scrap column weak enough to fall off easily and preserve the quality of the part exterior.

We needed a graphical method to visually represent the de-scrapping difficulty so we could optimally orient a part. It could not require the application to do much computing between reorientations. With models typically having 100,000-300,000 polygons, this would have made the interactive response too slow to be useful.

Theory

Without loss of 3D generality, we will look at this problem in 2D by treating the vertical height as ΔZ and treating the horizontal distance as $\Delta H = \sqrt{\Delta X^2 + \Delta Y^2}$. The contour line density is then related to the slope of the part surface. This is well known to anyone who has used a topographic hiking map. Regions with very dense contour lines represent a steep slope.

The equation for the total number of contours produced as a function of total height and paper thickness is:

$$NCL = \Delta Z / t \qquad (1)$$

where:

NCL	is the number of contour lines
ΔZ	is the total height
t	is the paper layer thickness = 0.0042”

Dividing by the horizontal run distance gives the contour line density function:

$$CLD = NCL / \Delta H = (\Delta Z / \Delta H) / t \quad (2)$$

or

$$CLD = \tan\Theta / t \quad (3)$$

where:

CLD is the contour line density

Θ is the slope angle ΔZ/ΔH

The surface normal perpendicular to this slope is the vector (–ΔZ,ΔH). The Z component of this un-unitized surface normal is ΔH. Looking at equations (2) and (3), the Z component of the unitized surface normal is:

$$nz = \frac{\Delta H}{\sqrt{\Delta H^2 + \Delta Z^2}} \quad (4)$$

or

$$nz = \cos\Theta \quad (5)$$

The contour line density as a function of surface normal is then:

$$CLD = \frac{\sqrt{1 - nz^2}}{t * nz} \quad (6)$$

Implementation

This method was implemented in OpenGL. Most OpenGL programmers are familiar with the 2D texture coordinates (s,t) and the texture matrix. The typical way to use texture mapping is to specify an (s,t) pair at each polygon vertex and possibly use the texture matrix to rotate or scale (s,t) to become (s',t'), which then index into the texture image during display update.

A lesser-known feature of OpenGL texturing is that it can use 4D texture coordinates, each specified by an (s,t,r,q) quadruple. These are passed through the 4x4 homogeneous texture matrix as follows:

$$\begin{Bmatrix} s' \\ t' \\ r' \\ q' \end{Bmatrix} = \begin{bmatrix} Texture \\ Matrix \end{bmatrix} * \begin{Bmatrix} s \\ t \\ r \\ q \end{Bmatrix} \quad (7)$$

In the same way that homogeneous vertex coordinates are handled, the elements (s',t',r') are each divided by q' before use. (s't') are then used to index into the texture image. Currently, the resulting r' value is ignored, although [NEIDER93] says that there might be some use for it in the future.

In this method, the (s,t,r,q) quadruple was specified to be the unitized surface normal (nx,ny,nz,1.). The texture matrix was two concatenated matrices, one to rotate the normal the same amount as the part has been rotated, and one to turn the rotated Z component of the surface normal into the proper texture coordinates. We established a 256x1 pixel texture where the colors were used to represent the de-scrapping difficulty. Even though this is a texel coordinate range of 0-255, OpenGL defines this as 0. to 1. So, a matrix was needed to linearly map a rotated *nz* in the range -1. to 1. to a texture *s* coordinate in the range 0. to 1. so that it could index a color. The equation to do this is:

$$s' = .5*nz' + .5 \tag{8}$$

The complete concatenated texture matrix would then need to be:

$$\begin{Bmatrix} s' \\ t' \\ r' \\ q' \end{Bmatrix} = \begin{bmatrix} 0. & 0. & .5 & .5 \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix} * \begin{bmatrix} Rotation \\ Matrix \end{bmatrix} * \begin{Bmatrix} nx \\ ny \\ nz \\ 1 \end{Bmatrix} \tag{9}$$

The range of colors in the texture corresponded to a range of Z normal components of -1. to 1. To establish colors, a general definition of a "bad" Z component had to be established. From experience we knew that a CLD of 100 or more (which corresponds to nz ≤ 0.922) made de-scrapping easy. To these values, we assigned green. We then ramped the hue from green to red in the hue-saturation-value color space as CLD decreased from 100 to 0. This gave us good green-to-yellow-to-red behavior when the contour line density dropped below 100 per inch.

The unitized surface normals were used as the 3D texture coordinates inside the OpenGL triangle lists. The entire object was placed in a single OpenGL display list to make it fast to update the display whenever the part orientation changed.

Results

Figure 2 shows a lens base for a lighting system that was submitted for fabrication.[2] As a prototype part, it was desirable to screw the lens caps onto the threads to be sure that they would fit without any interference. Thus, the threads needed to be the highest quality portion of the part, possibly at the expense of the surface quality of the base.

But, as Figure 2 shows, the original orientation of the part resulted in the difficult de-scrapping region being right in the middle of the threads. This means that the most delicate and crucial portion of the prototype would be the place with the most torn-up surface finish. The part was dynamically reoriented using the method described until the colors showed that the threads would be easy to de-scrap. This is shown in Figure 3.

But, of course, reorientation cannot make *all* of the horizontal difficult areas go away, it can just move them to somewhere else. Rotating the view shows where they have gone. In Figure 4, the difficult area has moved to the other side, but it is along the rim and at the base of the threads. This is acceptable as it will not hurt the threads. If re-orienting the part simply moved the difficult area to another crucial place in the model, then we would have continued to iterate until the most acceptable solution was found. The finished part is shown in Figure 5.

[2] This part was designed by UCSD Masters student Mike Arnstein using the Pro/Engineer solid modeler.

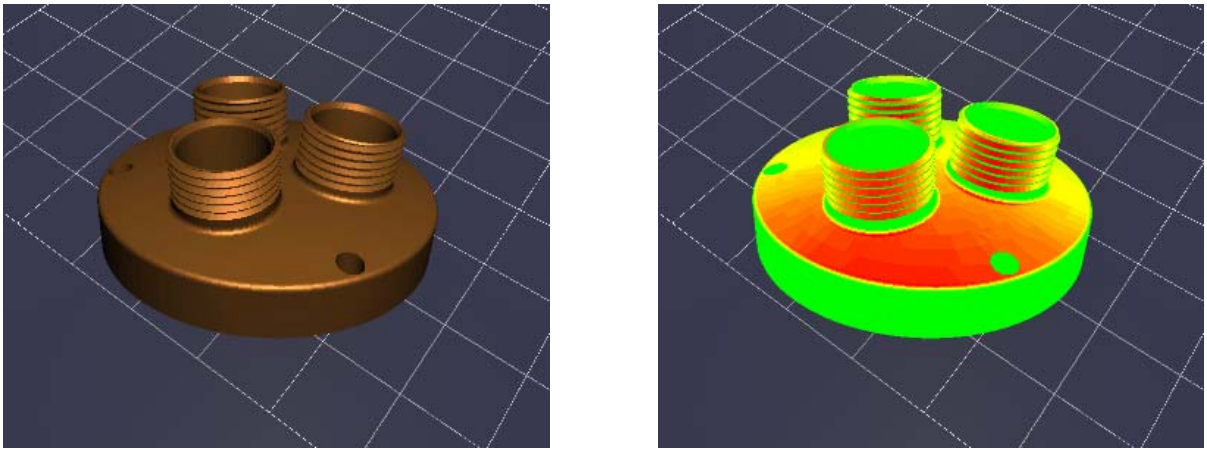

Figure 2: The Difficult De-scrapping Area is in a Bad Place

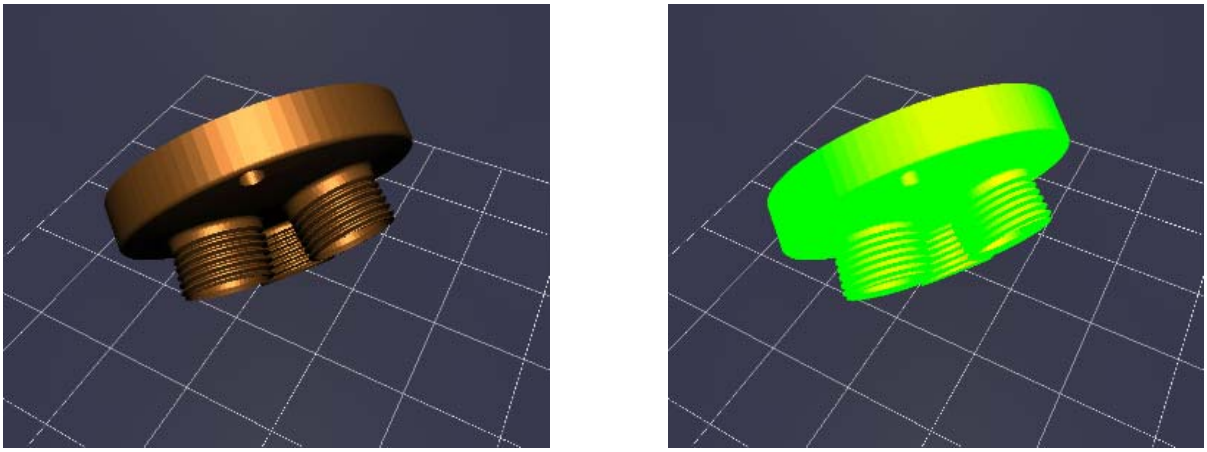

Figure 3: A Better Orientation for a Quality Finish on the Threads

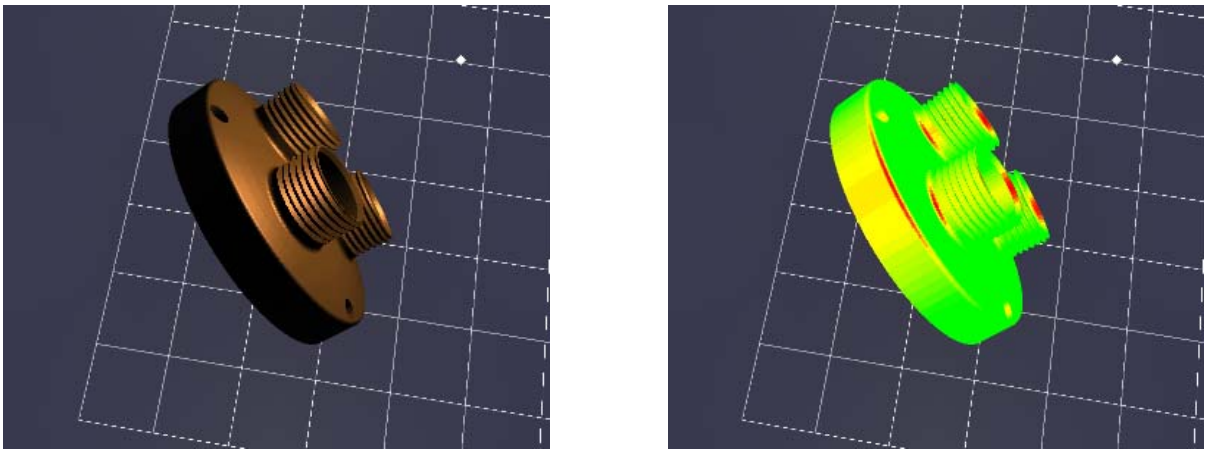

Figure 4: The Difficult De-scrapping Area has Moved to a Better Place

Figure 5: Finished Part with a Fitting on One of the Threads

Summary and Future Work

This method is not just another use of color to represent scalar values. This method turns the scalar value into a texture coordinate that uses a dynamic OpenGL texture matrix to change the color distribution with part orientation. The significance of this approach is:

- The entire part can be placed in a static display list because it does not have to be altered between reorientations.

- This method takes advantage of the ubiquitous texture-mapping acceleration that is appearing in all levels of graphics systems.

We believe that there are even more less-than-obvious uses for the 3D OpenGL texture coordinates beyond this one. One that is of particular interest to us is to encode a "peak-ness" length and direction into a 3D texture vector to help orient the part so that long thin protrusions do not get built vertically (where they act like a weak "stack of checkers") and instead get built horizontally (where they act like a very strong set of laminated cantilever beams).

Web Page

For more information see: `http://www.sdsc.edu/tmf`

Acknowledgments

This work was supported by the National Science Foundation under grant MIP-9420099.

Authors

Mike Bailey is the director of the TeleManufacturing Facility project. Mike received his Ph.D. from Purdue University in Mechanical Engineering with an emphasis on computer aided engineering and computer graphics. Since that time, Mike has taught and conducted research at Sandia National Labs, Purdue University, Megatek Corporation, and the University of California at San Diego / San Diego Supercomputer Center. His research interests include Internet-based scientific visualization, solid freeform fabrication, and computer integrated mechanical engineering.

Dru Clark is a masters student in Applied Mechanics and Engineering Sciences at the University of California at San Diego and a student researcher on the TMF project at SDSC. His research interests are prototyping techniques for engineering and scientific visualization, modeling, and animation.

References

[BAILEY95]
Michael Bailey, "Tele-Manufacturing: Rapid Prototyping on the Internet with Automatic Consistency-Checking," *IEEE Computer Graphics and Applications*, November 1995.

[BAILEY96]
Michael Bailey, "The Use of Solid Rapid Prototyping in Computer Graphics and Scientific Visualization, " SIGGRAPH Course Notes for *The Use of Touch as an I/O Device for Graphics and Visualization*, August 1996.

[HELISYS97]
Laminated Object Manufacturing product literature, Helisys, Inc., Torrence, CA, 1997.

[NEIDER93]
Jackie Neider, Tom Davis, and Mason Woo, *OpenGL [1.0] Programming Guide*, Addison-Wesley, 1993.