Project #: E-25-W40  Cost share #:  Rev #: 2
Center #: 10/24-6-R8024-0A0  Center shr #:  OCA file #:  Work type: RES
Contract#: AWD DTD 12/22/93  Mod #: DELIV SCHEDULE  Document: AGR  Contract entity: GTRC
Prime #:  Subprojects?: N  Main project #:  CFDA:  PE #:  
Project unit: MECH ENGR  Unit code: 02.010.126
Project director(s):  BRAS B A  MECH ENGR  (404)894-9667

Sponsor/division names: TEXAS INSTRUMENTS  /
Sponsor/division codes: 219  / 009

Award period: 931215 to 940614 (performance) 940630 (reports)

Sponsor amount  New this change  Total to date
Contract value  0.00  11,160.00
Funded  0.00  11,160.00

Cost sharing amount  0.00
Does subcontracting plan apply?: N
Title: DESIGN OF WIRE BONDING DIAGRAMS

PROJECT ADMINISTRATION DATA

OCA contact: E. Faith Gleason  894-4820
Sponsor technical contact  Sponsor issuing office
DR. DONALD ABBOTT  EDWARD A. IANNOTTI
(508)699-1580  (508)699-0000

TEXAS INSTRUMENTS INC.  TEXAS INSTRUMENTS INC.
SR. MEMBER TECHNICAL STAFF, MS 11-01  GEN MGR, WORLDWIDE LEADFRAMES, MS1117
34 FOREST STREET  34 FOREST STREET
ATTLEBORO, MA 02703  ATTELBORO, MA 02703

Security class (U,C,S,TS) : U  ONR resident rep. is ACO (Y/N): N
Defense priority rating :  Supplemental sheet
Equipment title vests with: Sponsor  GIT

Administrative comments -
* FINAL REPORT DUE DATE CHANGED TO 6/30/94.
REFERENCE TEXAS INSTRUMENTS P.O. #P 500182846 ON INVOICE AND CORRESPONDENCE.
GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

---

Closeout Notice Date 07/08/94

Project No. E-25-W40___________
Center No. 10/24-6-R8024-0A0_

Project Director BRAS B A______________
School/Lab MECH ENGR_______

Sponsor TEXAS INSTRUMENTS/______________

Contract/Grant No. AWD DTD 12/22/93___________
Contract Entity GTRC

Prime Contract No. _______________________

Title DESIGN OF WIRE BONDING DIAGRAMS

Effective Completion Date 940614 (Performance) 940630 (Reports)

---

<table>
<thead>
<tr>
<th>Closeout Actions Required</th>
<th>Y/N</th>
<th>Submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Invoice or Copy of Final Invoice</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Final Report of Inventions and/or Subcontracts</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Government Property Inventory &amp; Related Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Classified Material Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Release and Assignment</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Comments________________________________________________________________________________________________

---

Subproject Under Main Project No. ______________

Continues Project No. ______________

---

Distribution Required:

- Project Director: Y
- Administrative Network Representative: Y
- GTRI Accounting/Grants and Contracts: Y
- Procurement/Supply Services: Y
- Research Property Management: Y
- Research Security Services: N
- Reports Coordinator (OCA): Y
- GTRC: Y
- Project File: Y
- Other: N

---

NOTE: Final Patent Questionnaire sent to PDPI.
Final Report For The Development Of An Auto-Biasing Module

Prepared by
Zachary J. VerGow
Graduate Research Assistant

for
Dr. Donald Abbott
Texas Instruments

July 1, 1994
Introduction

There are many methods of metal forming. There are mechanical stamping, metal cutting, chemical etching, and many others. For Texas Instruments (T.I.) in their lead frame business, two methods are currently used: stamping and etching. However, mechanical stamping has a physical limitation in the size and accuracy of its series of punches and dies. As the lead frame gets more and more complicated and smaller, stamping will soon, and has already in some areas, reached its limit. This leads to chemical etching, a more complex and expensive process, but much more accurate and precise.

Lead frames are the building blocks of semiconductors. A lead frame is technically the metallic frame onto which the silicon chip is mount and soldered. It is typically made from a base metal of copper that is shaped by either stamping or chemical etching. The base metal is then electroplated with a series of metals. The surface metal, such as silver or gold, must be a highly solderable, highly resistant to corrosion, and must bond well with gold. The basic lead frame shape consists of a central pad onto which the chip is mounted and many digits that run out from the pad that are the pins that stick out of an integrated circuit. The manufacturing of lead frames is very precise and delicate due to their fragility and cost.

The etching process currently being used at T.I. is photochemical. The base material, typically either copper or A-42, a nickel-steel alloy, is laminated with a light sensitive chemical resist. The laminated material is then passed through an exposure unit that shines an intense UV light on both the top and bottom of the material through glass photomasks which have the shape of the leadframe on it. That part of the resist that gets exposed to the light changes its properties and become resistant to acid etchant. The unexposed part is not resistant to the acid. The
material is then passed through a ferric chloride bath that etches the unexposed part of the material from both the top and bottom.

There are, however, inherent problems with chemical etching that must be compensated for. The largest problem is the actual etching process. When the ferric chloride etches the material, it does not etch in straight paths. Ideally the material would get etched in a direction exactly perpendicular to the surface. However, the ferric chloride undercuts a little to the side as it etches down through the material. When the top and bottom meet, a 'foot' is created. (See Fig. 1). Because of these features, the photomask must be made to compensate. The photomask compensation is the focus of this project.

![Figure 1 - Cross Section Of An Etched Cut](image)

To compensate for etching features, Texas Instruments has a well defined set of rules for biasing a lead frame drawing for creation of a photomask. Given a drawing of the desired lead frame, a drawing of the photomask is created based on this list of rules. The rules basically consist of methods of making certain parts wider, making holes smaller, and adding serifs onto corners to ensure sharp corner radii. However, T.I.'s current method of creating these biased
drawings is to have it done by an outside vendor, which is both costly and time consuming. Texas Instruments has expressed an interest in developing an in-house method for photomask biasing. This project was to develop the beginnings of an auto-biasing module for Pro/Engineer.

**Project Description**

From the original project proposal, the objective of this project was "to create a model on Pro/Engineer to create from the original prints the biased leadframe drawings that will be used for the fabrication of the manufacturing phototools". In order to achieve this objective, several steps had to be taken. These steps are discussed in the following paragraphs.

After conversations with Robert Alvarez, a T.I. employee actively involved in the development of Pro/Engineer (Pro/E) at Texas Instruments, and Paul Lomangino, the most experienced student on the Georgia Tech campus with Pro/Engineer, it was decided that Pro/Develop should be used to create the auto-biasing module. Pro/Develop is "a programmatic interface to Pro/ENGINEER which allows software developers to customize Pro/ENGINEER and/or new applications that can be directly integrated into the Pro/ENGINEER environment. Pro/DEVELOP consists of a library of 'C' functions which provide a supported interface to Pro/ENGINEER and direct access to the Pro/ENGINEER database" [Parametric Technology Corporation, 1993]

The next step was to secure a Pro/Engineer account for use on the Georgia Tech campus. The Computer Aided Engineering / Computer Aided Design (CAE/CAD) department at Georgia Tech maintains a working copy of Pro/Engineer for use by students. An account for Zachary
VerGow was attained from the CAE/CAD department. It was understood by the system administrator that Pro/Develop would be used to develop modules for use in Pro/Engineer. The problem with this setup was that the account was resident on a machine that was situated across the campus, and that in order to work on Pro/E, it was necessary to either go to an undergraduate computer laboratory or to export the screen to a local workstation. This caused difficulties in getting well established with the Pro/E system.

The next three months were spent learning how to use Pro/E. Having never had formal training in Pro/E, the learning time proved to be longer than expected. It was extremely important, as was emphasized by Alvarez [Alvarez, 1994], to learn Pro/E extremely well before even attempting to begin programming in Pro/DEVELOP (Pro/D). After spending a significant amount of time learning the basics of Pro/E, the next step was to begin to learn Pro/D. Learning Pro/D proved to more difficult than Pro/E.

As described above Pro/D is a library of C routines for accessing Pro/E and its database. The Pro/E database contains all of the data necessary for describing a model in Pro/E. By accessing that database, a programmer can develop new functions that can help to tailor the system to a particular user's needs. In this project, the task was to develop a module that would bias a model of a lead frame for use in the manufacture of the phototool. This was done in Pro/D. The algorithm developed and the method of use is described in the next section.
The Auto-Biasing Module

In order to define a realistic task, it was decided that a module would be developed for biasing only the straights on the lead frame. In biasing the straights, a linear offset is applied to the edges of the leadframe. See Figure 2 for an example of what the module would create. It was also desired that future additions, such as serifs and holes, could be easily integrated into the existing module if time allowed.

![Figure 2 - Biasing The Straights](image)

In biasing the straights, a constant bias is applied to all edges of the lead frame. T.I. has established rules for how much the bias amount should be depending on the material type and the thickness of the lead frame. For proprietary reasons, no specific bias distances or rules will be listed in this report. It will suffice to say that because of the wide range of bias distances that exist, this module will specifically ask for the bias distance when initiated. In order to describe the process that is done to bias a model of a lead frame, the general algorithm will be provided. After the algorithm, the details of each step of the algorithm will be provided.
Auto-Bias Algorithm

1. choose the top face
2. determine the number of contours on the top face
3. request the bias distance
4. for each contour
5. determine the direction of the first edge
6. determine the ending point of the first edge
7. calculate angle between first edge and next edge
8. calculate and store new biased ending point (bep_old)
9. store first edge's biased ending point (fe_bep)
10. for each edge on the contour after the first edge
11. determine the direction
12. determine starting and ending points
13. calculate angle between current edge and next edge
14. calculate new biased ending point (bep_new)
15. draw line between old biased ending point (bep_old) and new biased ending point (bep_new)
16. bep_old = bep_new
17. continue
18. draw line between old biased ending point (bep_old) and first edge's biased ending point (fe_bep)
19. continue
20. regenerate the display

In step #1, the user will be asked to choose the top face of the lead frame. Because the model of the lead frame is 3 dimensional, the user is required to specify what the top face of the lead frame is. The top face cannot be assumed. Another reason for specifically asking the user to identify the top face is that the top face may consist of more than one surface, and consequently more than one contour. By specifying the top face, Pro/E will identify all of the surfaces and contours associated with the top face. See Figure 3.
Figure 3 - Surfaces, Contours, And Faces

Step #2 checks to see if there is more than one contour for the top face. A contour can be described as the loop of edges that contain a surface. If there is more than one face, a bias must be done for each contour. Step #3 then asks the user to input the bias distance for this particular lead frame. The rest of the steps actually perform the biasing.

Step #4 begins the loop through each contour in the top face. Step #5 then determines the direction of the first edge on the contour in the Pro/E database. It is important to know the direction of the edge in the Pro/E database so as to understand what the beginning and ending points of each edge are in the direction of the contour. Knowing what the direction of the edge is and the direction of the contour, the ending point of the first edge can be determined, step #6.

The next two steps involve computing where the biased edge should be located. Step #7 computes the angle between the first edge and the second edge. It is assumed that the two edges are either in an xy, xz, or yz plane in the coordinate system. By determining which direction is
common to both edges, the angle between the two edges can be calculated. With the angle between the two edges and the bias distance, the endpoint of the biased line can be calculated by Equation 1. Also, see Figure 4.

\[ y = x \tan \left( \frac{\frac{x}{2} - \alpha}{2} \right) \]  

[1]

With Equation 1 and the endpoint of the existing edge, the coordinates of the new biased endpoint can be calculated. It is important to note that the biased endpoint of the first edge will be the biased starting point for the second edge. This is done for two reasons:

- to ensure geometric continuity, and
- to reduce unnecessary computations.
The next step is to store the first edge's biased ending point for use by the next edge as its starting point. In the algorithm, given on page 6, it is stored in the variable, 'bep_old'. It is also necessary to store the first edge's biased ending point as 'fe_bep' for use later when its biased starting point is known.

The next step, #10, creates another loop that steps through the rest of the edges for each contour. The same calculations as those done on the first edge are done on every other edge on the contour. By using the biased ending point of the previous edge, bep_old, as the biased starting point for the current edge and calculating the new biased ending point, bep_new, of the current edge, a line can be drawn as the new biased edge. The last step in this loop is to store the new biased ending point of the current edge (bep_new) in bep_old for use by the next edge.

When last edge of the contour has been biased, the last step, #17, is to draw the biased line for the first edge. Because the starting point of the biased line is equal to the biased ending point of the last edge on this contour and because the biased ending point of the first edge was stored in fe_bep, the line can be drawn to complete the bias for that contour. The whole sequence from steps #5 to #17 are repeated for every contour on the top face.

The very last step, #20, is regenerate the new model to verify its completeness. At this point, there are a series of lines that represent the biased drawing of the lead frame. The auto-bias module is done at this point. The user may then perform whatever other operations are necessary (such as other biasing operations), dimension the drawing as is appropriate, and send out the print for phototool manufacture.
Future Work

The code for this auto-biasing module has been written (see Appendix 1). Unfortunately, it was never tested. In order to install this module for use with the seat of Pro/E at Georgia Tech, the entire campus would be affected. Every user on campus would start Pro/E and see a menu item for an auto-biasing command. Hence, it is not possible to install and test this code on this seat of Pro/E for both proprietary and systems reasons. Therefore, the first thing to be done in the future is to install this module and fully test the code. The algorithm is sound, and the code is free of syntax errors. However, it is not known if this code is fully functional.

In addition, there are many other aspects of biasing that were not addressed here. However, because of the manner in which this module was written, it will not be a difficult task to add in other features. For example, in order to include serifs in this module, instead of directly connecting the two biased lines from the edges at a point, insert two more lines to make the serif triangle. Since, the angle of the corner is already known, the dimensions of the serif can be calculated. This biasing module was intentionally designed to allow for ease of future upgrades.

Personal Observations

One of the goals of this project was to teach me how to use Pro/E. That goal was achieved thoroughly. I feel as if I know Pro/E, at least certain parts of it, better than AutoCad, which is a major step. I am beginning to understand why so many companies are pushing to get Pro/E as the major design environment. It is powerful, flexible, and, eventually, even intuitive. I see the potential benefits of Pro/E much better now.
Also, I feel as if I deeply understand the inner workings of Pro/DEVELOP. It was extremely difficult to get up to speed with Pro/D. But once you understand the intricacies involved, it is an extremely powerful tool. By programming with Pro/D, I have also gained a much better understanding of the C language which will be of great assistance in my work for Rick Fay. I would like to thank Texas Instruments for giving me this opportunity.

**Conclusions**

The goal of this project was to develop a module for Pro/E that would provide the beginnings of an automated biasing function. This was achieved. An effective, flexible, and efficient algorithm was developed for Pro/DEVELOP. This algorithm is capable of biasing the straight edges of lead frame via several geometric and trigonometric relations. It is extremely unfortunate that the code could not be tested. Some directions for future work are also identified.

Additionally, I have personally gained a lot of knowledge about Pro/ENGINEER. I understand how to use it as a design tool, how to use it as a drafting tool, and how to use it as a programming tool. I would like to thank Paul Lomangino for all the direct support with Pro/D. I would also especially like to thank Don Abbott for all the help he has been in many ways.
References


```c
#include "prodevelop.h"
#include "user_wchar_t.h"
#include "prorefer.h"
#include "select3d.h"
#include "ptc_geom.h"
#include "proentype.h"
#include "umath_ext.h"
#include <stdio.h>
#include <math.h>

/* ***************************************************************
 * TI AutoBiasing module
 * by Zachary J VerGow
 * - requires pro_develop example programs
 */

#define MAX_DATA 100

void ti_bias ()
{
    char *p_face, *p_edge, *p_edge_next, *p_contour
    char *first_edge;
    double dbl_buff, y_distance, angle, bep[3];
    double bep_old[3], bep_new[3];
    Select3d *select;

    /****** ask the user to select the lead frame face *************/
    promsg_print(MSGFIL, "BIAS Select a face");
    if ( pro_select("FACE", 1, &select, 0, 0) <= 0) break;
    p_face = select[0].selected_ptr;
    promsg_print(MSGFIL, "BIAS Enter the bias distance");
    promsg_getdouble(&dbl_buff, NULL);

    for ( p_contour = prodb_first_face_contour(p_face);
        p_contour ! = 0;
        p_contour = prodb_next_face_contour(p_contour) )
    {
        /* Get the first edge on the contour */
        if ( (first_edge = prodb_first_cntr_edge(p_contour)) ! = 0 ) {
            /* Hold onto first edge ptr */
            p_edge = first_edge;
            /* Find the next edge */
            p_edge_next = prodb_next_edge_on_face(p_edge, p_face);
            /* Find the angle between these edges */
            angle = angle_between_edges(p_edge, p_edge_next, p_face);
            /* Calculate the displacement distance */
            y_distance = linear_displacement(dbl_buff, angle);
            /* Calculate the new biased end point */
            if ( biased_end_point(dbl_buff, y_distance, p_edge, p_contour, bep) <=0)
                break;
            /* Store biased end point */
            for (i=0;i<3;i++) bep_old[i] = bep[i];
            /* Store edge pointer */
            p_edge = p_edge_next;
        }
    }
}```
/* Traverse all other edges on the contour */
do
   /* Go to the next edge */
p_edge_next = prod_b_next_edge_on_face(p_edge, p_face);
   /* Perform same operations as first edge */
   angle = angle_between_edges(p_edge, p_edge_next, p_face);
y_distance = linear_displacement(dbl_buff, angle);
   if (biased_end_point(dbl_buff, y_distance, p_edge, p_contour, bep_new) <= 0) break;
   /* Draw the new biased line */
   user_draw_line(bep_old, bep_new);
   /* Store all the points and pointers for next loop */
   for (i = 0; i < 3; i++) bep_old[i] = bep_new[i];
   p_edge = p_edge_next;
   /* Until back at first edge */
   while (p_edge != first_edge && p_edge != 0);  
   /* Draw the last line */
   user_draw_line(bep_old, fe_bep);

}


double angle(p_edge, p_edge_next, p_face)
char *p_edge, *p_edge_next, *p_face;
{
   double begin[3], end[3], begin_next[3], end_next[3];
   double alpha, beta;
   begin_end_points(p_face, p_edge, begin, end);
   begin_end_points(p_face, p_edge_next, begin_next, end_next);
   if ((end[0] != begin_next[0]) || (end[1] != begin_next[1]) || (end[2] != begin_next[2])) break;

   if (begin[0] == end_next[0]) {
      if ((begin[1] - end[1]) == 0.0)
         alpha = 3.1415927 / 4.0;
      else
   }
   else
      if ((end_next[1] - begin_next[1]) == 0.0)
         beta = 3.1415927 / 4.0;
      else
\[
\beta = \arctan(\frac{\text{fabs}(\text{end}_\text{next}[2] - \text{begin}_\text{next}[2])}{\text{fabs}(\text{end}_\text{next}[1] - \text{begin}_\text{next}[1])})
\]

\[
\beta += \text{quadrant_check}(\text{end}_\text{next}[2] - \text{begin}_\text{next}[2], \text{end}_\text{next}[1] - \text{begin}_\text{next}[1])
\]

\[
\text{if} (\text{begin}[1] == \text{end}_\text{next}[1]){
\text{if} ((\text{begin}[2] - \text{end}[2]) == 0.0)
\text{alpha} = 3.1415927 / 4.0;
\text{else}
\text{alpha} = \arctan(\text{fabs}((\text{begin}[0] - \text{end}[0])/(\text{begin}[2] - \text{end}[2])))
\text{alpha} += \text{quadrant_check}(\text{begin}[0] - \text{end}[0], \text{begin}[2] - \text{end}[2]);
\}
\text{if} ((\text{end}_\text{next}[2] - \text{begin}_\text{next}[2]) == 0.0)
\text{beta} = 3.1415927 / 4.0;
\text{else}
\text{beta} = \arctan(\text{fabs}((\text{end}_\text{next}[0] - \text{begin}_\text{next}[0])/(\text{end}_\text{next}[2] - \text{begin}_\text{next}[2])))
\text{beta} += \text{quadrant_check}(\text{end}_\text{next}[0] - \text{begin}_\text{next}[0], \text{end}_\text{next}[2] - \text{begin}_\text{next}[2]);
}\]

\[
\text{return} (\text{fabs}(\text{alpha} - \text{beta}));
\]

double quadrant_check(double rise, double run)
{
\text{double tempo;}
\text{if } ((\text{rise} >= 0.0) \&\& (\text{run} >= 0.0)) \text{tempo} = 0.0;
}
double
if ((rise >= 0.0) && (run < 0.0)) tempo = 3.1415927 / 2.0;
if ((rise < 0.0) && (run <= 0.0)) tempo = -3.1415927 / 2.0;
if ((rise < 0.0) && (run > 0.0)) tempo = 0.0;
return(tempo);
}
double
y_distance(bias, angle)
double bias, angle;
{
    double tempo;
    tempo = bias * tan((3.1415927 / 2.0 - angle) / 2.0);
    return(tempo);
}
int
begin_end_points(p_face, p_edge, begin, end)
char* pface, *p_edge;
double begin[3], end[3];
{
    if (prodb_edge_direction(p_edge, p_face) > 0)
    {
        pro_eval_xyz_edge(p_edge, 0.0, begin, 0, 0, 0);
        pro_eval_xyz_edge(p_edge, 1.0, end, 0, 0, 0);
    }
    if (prodb_edge_direction(p_edge, p_face) < 0)
    {
        pro_eval_xyz_edge(p_edge, 0.0, end, 0, 0, 0);
        pro_eval_xyz_edge(p_edge, 1.0, begin, 0, 0, 0);
    }
    if (prodb_edge_direction(p_edge, p_face) == 0) break;
}
int
bias_end_point(bias, y, p_edge, p_face, bep)
double bias, y, bep[3];
char* p_face, *p_edge;
{
    double begin[3], end[3], scale, alpha, dummy[3];
    int accept_flag;

    /* begin by finding beginning and ending points */
    begin_end_points(p_face, p_edge, begin, end);

    /* figure out what plane you're in */
    if (begin[0] == end[0]) /* in yz plane */
    {
        /* find angle of line */
        if ((end[1] - begin[1]) == 0.0)
        {
            alpha = 3.1415927 / 4.0;
        }
        else
        {
        }
        /* figure out which quadrant you're in */
    }

user_get_scale(SEL_3D_EDG, p_edge, &scale, dummy, dummy);
if (prodb_edge_direction(p_edge, p_face) > 0)
{
    t = (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}
if (prodb_edge_direction(p_edge, p_face) < 0)
{
    t = 1 - (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}

alpha += 3.1415927 / 4.0;
bep[1] += bias * cos(alpha);
bep[2] += bias * sin(alpha);
accept_flag = 1;
}

if (begin[1] == end[1]) /* in zx plane */
{
    /* find angle of line */
    if ((end[2] - begin[2]) == 0.0)
    {
        alpha = 3.1415927 / 4.0;
    }
    else
    {
        alpha = atan(fabs((begin[0] - end[0])/(begin[2] - end[2])));
    }
    /* figure out which quadrant you're in */
    alpha += quadrant_check(end[0] - begin[0], end[2] - begin[2]);
}

user_get_scale(SEL_3D_EDG, p_edge, &scale, dummy, dummy);
if (prodb_edge_direction(p_edge, p_face) > 0)
{
    t = (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}
if (prodb_edge_direction(p_edge, p_face) < 0)
{
    t = 1 - (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}

alpha += 3.1415927 / 4.0;
bep[2] += bias * cos(alpha);
bep[0] += bias * sin(alpha);
accept_flag = 1;
}

if (begin[2] == end[2]) /* in xy plane */
{
    /* find angle of line */
    if ((end[0] - begin[0]) == 0.0)
    {
        alpha = 3.1415927 / 4.0;
    }
    else
    {
alpha = atan(fabs((begin[1] - end[1])/(begin[0] - end[0])));
/
figure out which quadrant you're in */
alpha += quadrant_check(end[1] - begin[1], end[0] - begin[0]);

user_get_scale(SEL_3D_EDG, p_edge, &scale, dummy, dummy);
if (prodb_edge_direction(p_edge, p_face) > 0) {
    t = (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}
if (prodb_edge_direction(p_edge, p_face) < 0) {
    t = 1 - (scale + y) / scale;
    pro_eval_xyz_edge(p_edge, t, bep, 0, 0, 0);
}

alpha += 3.1415927 / 4.0;
bep[0] += bias * cos(alpha);
bep[1] += bias * sin(alpha);
accept_flag = 1;
}

if (accept_flag == 0) return(0);
else return(1);