

# Computer Generated Shapes in Mechanical Design with MATLAB\*

NICHOLAS J. SALAMON

Department of Engineering Science and Mechanics, The Pennsylvania State University, University Park,  
PA 16802, USA. E-mail: njsalamon@psu.edu

*Our undergraduate/graduate course Computer Methods in Engineering Design targets students who want to do mechanical design. Although we treat basic numerical methods that support design (solution of simultaneous equations, integration, interpolation), we emphasize curve fitting and constructions generated by splines, B-splines and Bézier curves, then transformations of point groups (objects), and computer graphics to manipulate objects, and culminate with solid modeling and simulation when students are in a position to appreciate such sophistication. In this paper we share our approach to teaching splines: some theory, some concepts important to learning it, key references, and a time frame for doing it; and touch upon transformations in computer graphics. We then show off results from student projects. MATLAB is the engine used to render the graphics.*

## EDUCATIONAL SUMMARY

1. The paper presents excerpts of theory, application, MATLAB code and examples from our course on computer methods for engineering design.
2. Students in aerospace, mechanical, industrial engineering and engineering science and mathematics departments have taken this course.
3. The level of this course is third year undergraduate to graduate in the United States.
4. The mode of presentation is typically lecture and discussion with computer demonstrations and practice in the classroom.
5. The course is an elective for all students.
6. The entire course requires 45 hours. The topics in this paper require 6 hours.
7. Student homework and projects require 3 hours per hour of lecture.
8. The novel aspects of this paper are: (1) the mix (and emphasis) of numerical methods topics is chosen with mechanical design applications in mind rather than the solution of differential equations, (2) problems/examples have a degree of realism and solutions depend upon creativity of the students and (3) custom MATLAB code developed from the theory is used not only to work with and graph data, but to draw pictures.
9. A single text which covers all the topics in this course is not available.

## INTRODUCTION

TRADITIONAL numerical methods courses in engineering do not cover computer-aided design;

they prepare students for the solution of differential and integral equations important for advanced analysis. Yet equally important are numerical methods that support synthesis, design and product development. The objective of our undergraduate/graduate course *Computer Methods in Engineering Design* is to learn the numerical methods underlying solid modeling and engineering design. It covers simultaneous equations, quadrature, rudiments of finite difference and finite element modeling, spline theory to generate curves, object transformations and computer graphics to manipulate objects, solid modeling to demonstrate and experience useful application of the methods and, after treatment of vector transformations and eigen problems, dynamic simulations. We are not aware of any textbook that covers this mix of topics, hence we provide a set of notes and MATLAB files to go with them [1]. While the obvious learning objective is to penetrate the black box of professional software, a more subtle one is to learn numerical methods at the same time; for example, while learning to manipulate objects, students also learn transformations and maps in general, hence later easily comprehend the eigenvalue problem which we also cover. The course targets that majority of students in aerospace, mechanical, industrial engineering and related disciplines with an applied mind-set who want to do mechanical design, enjoy working with computers and love to create graphics. Least one think that such a course will not appeal to the better students, we have been surprised to find that it attracts the crème de la crème—happily so because topics like spline theory are not soft mathematically.

We view solid modeling as the keystone to modern product development. Once a solid model is in hand, it enables: finite element stress

\* Accepted 9 May 2005.

analysis and engineering design, generation of technical drawings, rapid prototypes and manufacturing control-ware, project management, rendered illustrations, marketing, and so-forth. Computer methods that underlie solid modeling are many, but a principal one is spline theory. In *Computer Methods in Engineering Design*, we spend 15 class-hours learning: (1) the common splines constructed directly from polynomials (linear, quadratic and cubic), for a source, see Chapra and Canale [2]; (2) the B-splines (Basis splines), but we only use them for curve fitting because this writer has yet to figure an intuitively easy way to present NURBS (Natural Uniform Rational B-Splines) within 3 class-hours, see De Boor [3] and Cheney and Kincaid [4]; and (3) the Bernstein polynomials to generate one- and two-dimensional Bézier curves which, after wrestling with B-splines, students grasp quickly.

The objective of this paper is three-fold: (1) To suggest to you an approach to teaching numerical methods that is attractive to undergraduate engineering students; indeed, assessment reveals that students have to be reigned in from committing too much time to the projects; (2) To present succinctly an easy-to-learn development for Bézier curves and a summary of computer graphics transformations that pertain to included student work and (3) To show off some of this work. We do not discuss assessment, but suffice it to say, this elective course is fully enrolled and motivates students to learn. Although we find few students creative in an artistic sense, most are innovative and all take pride in their creations.

It should be understood that prerequisite courses in linear algebra and computer algorithms and programming are necessary to students' success. Moreover, although MATLAB is the programming language used in the course, Mathematica would equally serve the purpose. However, a lower level language, C or Fortran, could work, but it would thwart efficient implementation and change the focus of the course because more detailed programming would be required.

## BÉZIER CURVES

We cover Bernstein polynomials and Bézier curves in 3 class-hours. We chose these topics over NURBS for three reasons: they are easy to learn, useful and produce very nice results. Our intention here is to present only key elements of development of the theory that we found appeals to students. Further details can be found in Buchanan and Turner [5]. It should be pointed out that the class has already covered both common and B-splines prior to this topic.

### Key elements to learning Bernstein polynomials

The Bernstein polynomials are defined in a local, normalized domain  $x$  by:

$$\begin{aligned} p(y; x) &= \sum_{i=0}^n \binom{n}{i} x^i (1-x)^{n-i} y(X_i) \\ &= \sum_{i=0}^n B_{n,i} y(X_i), x \in [0, 1] \end{aligned} \quad (1)$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

are the binomial coefficients,  $y$  is a given 'control' function defined at discrete values over the global domain  $X$  given by  $X_i = (X_n - X_0)i/n + X_0$  with the  $X_0, X_n$  beginning and end values of  $X$ , respectively, and

$$\begin{aligned} B_{n,i} &= B_{n,i}(x) = \binom{n}{i} x^i (1-x)^{n-i}, \\ i &= 0, 1, 2, \dots, n, x \in [0, 1] \end{aligned} \quad (2)$$

are the Bernstein basis functions.

The interesting feature of (1) is that data is generated in the normalized domain by  $y$  at equidistant points  $i/n$ , but no effort is made to pass the polynomial  $p$  through those points. This is a small leap for students who have just completed study of interpolating polynomials which of course do pass through the data.

The interesting feature of (2) is found by first noting that the binomial theorem (or distribution) [6] states that

$$\sum_{i=0}^n \binom{n}{i} a^i b^{n-i} = (a+b)^n.$$

Then indeed if we let  $a = x$  and  $b = 1 - x$  for any  $n$ , the individual terms of a binomial expansion (probabilities in the distribution) are the  $B_{n,i}$ . For example, if  $n = 2$ ,

$$\begin{aligned} [x + (1-x)]^2 &= x^2 + 2x(1-x) + (1-x)^2 \\ &\equiv B_{2,2} + B_{2,1} + B_{2,0} \end{aligned} \quad (3)$$

where terms of the center and rightmost expressions match, respectively. Moreover, the sum:

$$\sum_{i=0}^n B_{n,i} = [x + (1-x)]^n = 1^n = 1 \quad (4)$$

leads to a partition of unity which students can relate to also from study of interpolation.

Some students understand distributions (and Buchanan and Turner [5] inform us of Bernstein's connection with probability theory), but this does not matter here. What matters is that the general form of (3), specifically the first equation in (4), can be used to generate the individual terms of  $B_{n,i}$ ; better yet, we can write these down directly using Pascal's triangle (or Hui's or Khayam's, depending upon your allegiance) as shown in Fig. 1 where the

constant coefficients in any row follow by summing previous row coefficients in a particular fashion and the algebraic terms follow by multiplying previous rows by  $x$  from the right or  $(1-x)$  from the left; the details of both operations are easily discerned by inspection of Fig. 1. By using these formulas and the geometric patterns of Pascal's triangle, students gain a reasonable comfort level with Bernstein polynomials and are readily able to construct Bézier polynomials by hand before programming them to create Bézier curves.

#### An example using Pascal's triangle

In Fig. 1, we readily perceive from Pascal's triangle that terms on the left and right diagonals either vanish or are unity when  $x = 0$  or  $1$  and that all inboard terms provide zero contribution at  $x = 0$  or  $1$ . Clearly then Bernstein polynomials pass through the endpoints  $x = 0$  and  $1$ . Moreover, from the triangle, we can build directly Bézier curves. For an exercise, students are required to construct a Bézier curve, say from five control points:

$$Y = [y(0), y(1/4), y(2/4), y(3/4), y(1)]^T \quad (5)$$

where superscript  $T$  denotes the transpose of the vector and  $y$  may, but need not be, a particular function. To be consistent with  $Y$ , we choose from Fig. 1 the 4th degree set of Bernstein terms:

$$B_4 = [(1-x)^4, 4x(1-x)^3, 6x^2(1-x)^2, 4x^3(1-x), x^4] \quad (6)$$

to obtain a cubic polynomial

$$p(Y; x) = [B_4]\{Y\} \quad (7)$$

that forms a shape approximating, perhaps roughly, the control function or points. From this exercise, students should learn that the shape  $p$  is a poor match to the control shape  $y$ . For a better match, or more importantly, for a selection of shapes, we must turn to the computer. However, these 'hand' methods have served their purpose: students learn to construct Bézier curves simply from first principles and learn their properties as well.

#### Application: a bicycle seat

For an application of any significance, we turn to the computer. Students are required to write an algorithm using the recursion formula

$$B_{n,i}(x) = (1-x)B_{n-1,i}(x) + xB_{n-1,i-1}(x) \quad (8)$$

to compute  $B_{n,i}$  because it is computationally efficient. They then modify code provided to them to generate Bézier curves and turn in a report with a conclusion. Appendix I lists the MATLAB code to create shapes for a bicycle seat; it is similar to what students would produce. The code accesses function *bernstein\_basis*, based upon (8) and Buchanan and Turner's algorithm [5] to compute the basis functions and then *linear\_spline\_2* to interpolate values  $y(x)$  as needed along the control curve.

Execution of such code results in Fig. 2 where we see a series of shapes, the Bézier curves, generated using Bernstein polynomials of degrees 6, 9, 12 and

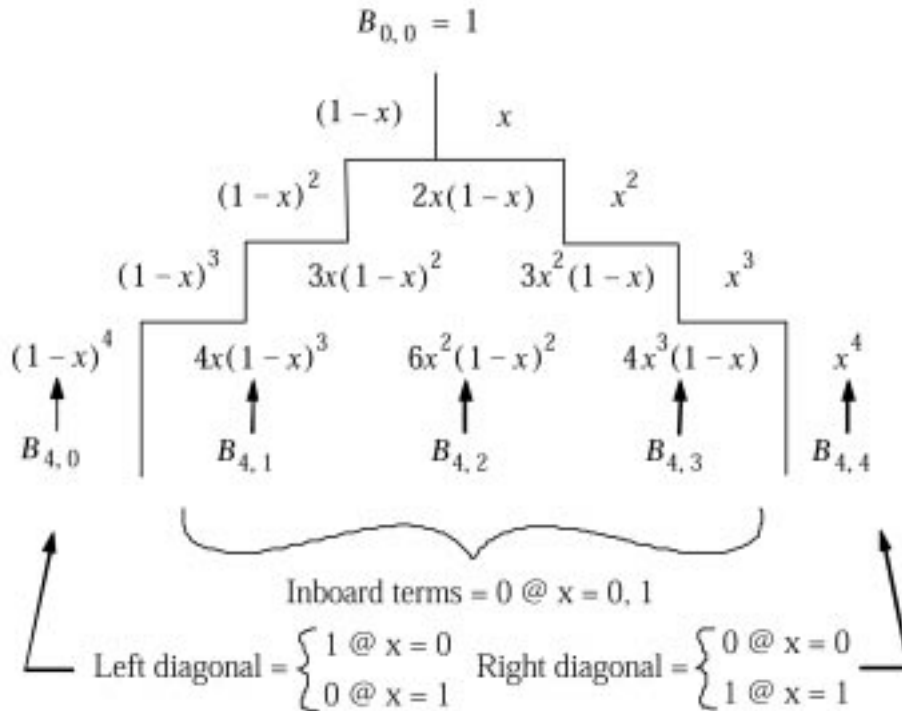


Fig. 1. Pascal's triangle where  $n, i$  denote the row, position of each term starting with 0,0 at the top and ending with 4,4 at the bottom right. Of course the triangle can continue.

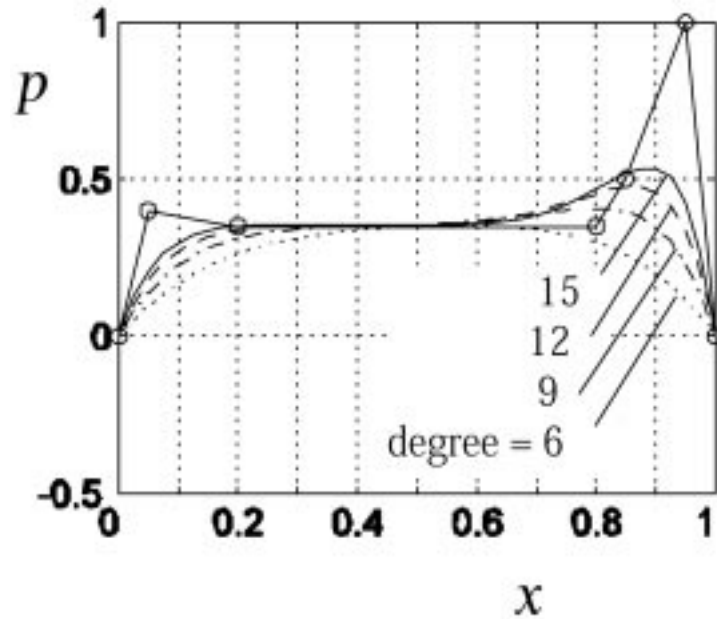


Fig. 2. A sequence of Bézier curves to design the shape of a bicycle seat based on an approximate control curve.

15. We discover that higher degree curves approach the control curve. But importantly we learn that this is not the goal. Rather, it is to generate shapes from a simple sketch (making artistic craftsmanship unnecessary) and from these to select a shape that meets our design objectives. Indeed, usually our choice is not the highest degree curve. In part, it is this understanding that we look for in students' conclusions.

### COMPUTER GRAPHICS

The rudiments of two-dimensional computer graphics are taught in 2 class-hours. The goals are to enable students to manipulate objects on a computer display and to acquire a general sense of matrix transformations in preparation for further study, for examples, translation as the essence of extrusion in solid modeling and transformations in eigenproblems taught subsequently. We cover the basic transformations: translations, rotation about the origin of coordinates, scaling with respect to the origin and shear of an object, all of which fit the form:

$$O' = MO \Leftrightarrow \begin{bmatrix} x'_1 & x'_2 & \dots & x'_n \\ y'_1 & y'_2 & \dots & y'_n \\ 1 & 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (9)$$

where  $O, O'$  are sets of points that define the transformed and original objects, respectively, in homogeneous coordinates. We also cover rotation and scaling about a general point and reflections,

all of which can be written as products of the basic transformations. We point out that it is efficient computationally to concatenate a sequence of transformations, e.g.,  $M = M_1 M_2 \dots M_k$ , and only then apply the resultant transformation  $M$  to an object  $O$ . This development is standard, hence we go no further here; one choice among many texts is McMahon and Browne [7] whose text has an engineering flavor.

#### Application: a mosaic

An interesting application is the generation of mosaics. As an example, we have assigned a mosaic copied from the Al-Hambra in Granada, Spain, see Fig. 3. To begin, we provide the coordinates of Shape 1 shown bold in Fig. 3a and denoted as *obj0* in the code in Appendix II. Then using a code like *gt\_mosaic*, the students rotate Shape 1 about Point A in Fig. 3a to produce one complete tile block. (The function *gt\_rotation* is listed in Appendix IV.) Through a sequence of translations, a tile panel like that in Fig. 3b is produced; we don't include code for this, but it is straight forward. For an ultra-high grade, we challenge the students to create a mosaic of their own. In their conclusions, or by inspection of their creations, it is obvious that they are struck by the difficulty of doing this.

### DESIGN OF A VEHICLE: FRONT VIEW

Students spend 3 to 5 class-hours with professional computer-aided design software to gain first-hand experience with applications of much of the theory taught in the course. However, prior to working with such software, we assign a design project in which the above theory and other topics colligate to make it happen in MATLAB.

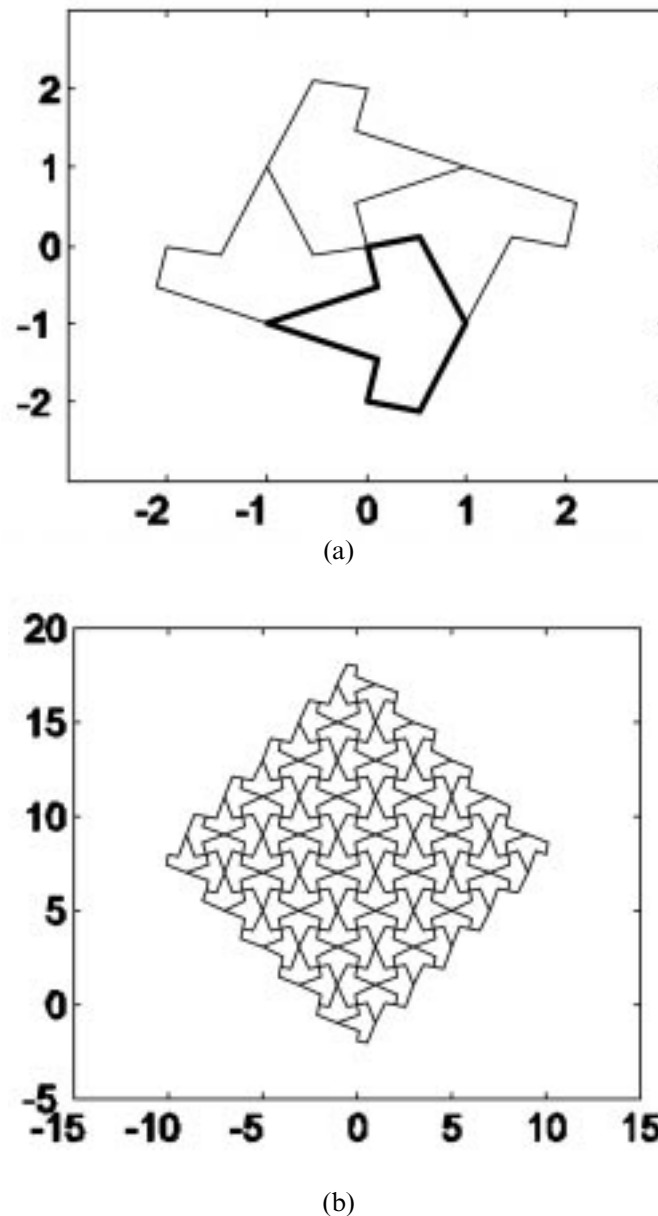


Fig. 3. Mosaic tiles copied from the Al-Hambra, Granada, Spain: (a) A single tile block formed by rotating copies of Shape 1 (bold lines) counter-clockwise around Point A three times; (b) a tile panel formed by translating blocks into tangent positions.

The best of these projects have been designs of bicycles and automobiles, all in two-dimensions of course (which makes selection of attractive topics difficult). Here we present the design of the front view of a vehicle; most students chose an automobile.

The result shown in Fig. 4 received the highest grade in the class. All projects were very good, but this one illustrates painstaking attention to detail (note the tread in the tires and the intricate grill). All curved lines are Bézier curves, including the steering wheel and bottom (contact surface) of the tires. All objects (lines, shapes, components) are transformed (rotated, scaled and/or translated) into position. Except for the steering wheel, only half the object is generated and then a copy is reflected about the centerline to produce the whole front view.

Of course we provide some help: (1) in class we develop together a starter code (Appendix III with function `bezier_curve` and `gt_translation` in Appendix IV) and (2) we provide a sketch done by hand on paper to illustrate what is expected. From that point on, other than some consulting, the students work on their own.

Related to the automobile are other, more technical aspects of the course. For example, suspension of the chassis is related to vibrations and eigenproblems. One semester things moved along so well we were able to program mass vibrations into a movie in MATLAB. Thus we try to tie as many topics as we can together in order to show their inter-relation and demonstrate the importance of computer methods in engineering.

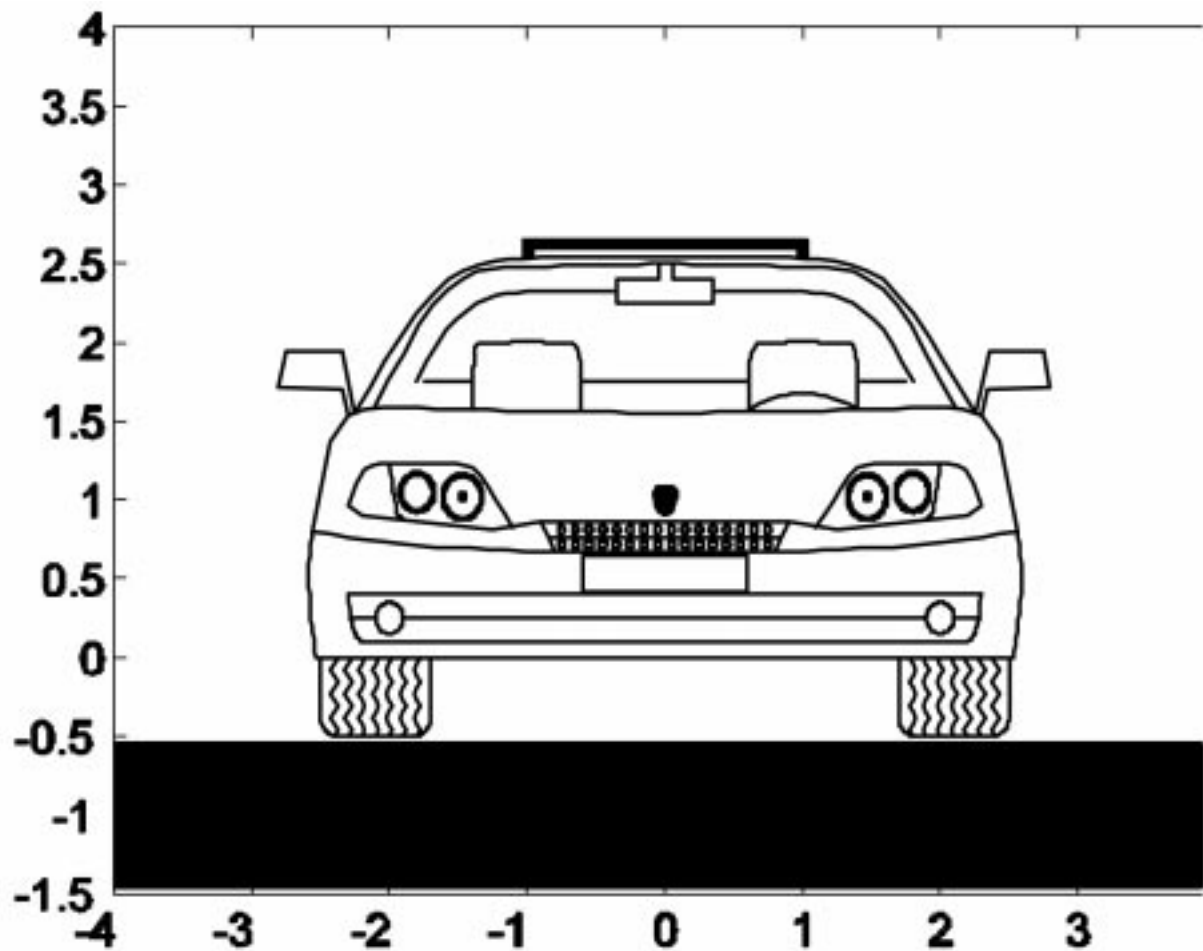


Fig. 4. Front of a commercial sports car, complements of Sunny Siu (Spring, 2004).

### CONCLUSION

This paper is based upon the proposition that the mathematical fundamentals underlying computer methods in design are important to engineers. Students learn the rudiments of generating curved shapes and the usefulness of mathematics to numerically design objects. In learning these methods, students also learn fundamental numerical methods, however the course is not traditional; for example, we only treat the finite difference modeling and solution of differential equations in one dimension as an application for solution of simultaneous equations.

One innovative feature of this course is that students draw their own creations mathematically.

Although learning the math is difficult, we find that design applications appeal to students and motivate them to learn. Indeed they take pride in their 'products', but sometimes get so involved they spend too much time doing intricate details. Despite their very good work and attention to detail, engineering students lack artistic creativity; their designs for the most part mimic those of others. Perhaps this is but an early step in learning.

*Acknowledgments*—Support from The Leonhard Center for the Enhancement of Engineering Education at Penn State through NSF Grant EEC 99-73002 is gratefully acknowledged. I thank Gary Schubert, artist, for supplying the Al-Hambra mosaic and Wensheng Yu for help with MATLAB codes. And I apologize to all students whose excellent work could not be included here.

### REFERENCES

1. N. J. Salamon (2002). <http://www.esm.psu.edu/courses/emch407/njs/default.html>
2. Steven C. Chapra and Raymond P. Canale, *Numerical Methods for Engineers*, McGraw-Hill, New York (1988).
3. Carl De Boor, *A Practical Guide to Splines*, rev. ed., Vol. 27 of Applied Mathematical Sciences Series, Springer-Verlag, New York (2001).
4. W. Cheney and D. Kincaid *Numerical Mathematics and Computing*, 3rd ed., Brooks/Cole Publishing Co. (1994).

5. James L. Buchanan and Peter R. Turner *Numerical Methods and Analysis*, McGraw-Hill, New York (1992).
6. CRC Standard Mathematical Tables, 15th edition, The Chemical Rubber Company, Cleveland (1967).
7. Chris McMahon and Jimmie Browne, *CADCAM: Principles, Practice, and Manufacturing Management*, Addison-Wesley, Reading (1998).

## APPENDIX I: BÉZIER CURVES

```
function bezier_bikeSeat
% Application: Bezier curve using Bernstein basis polynomials to
% to create the shape of a bicycle seat
% Wensheng Yu and NJ Salamon, Feb 2001
% Revised: 20 June 2004, NJS
xp=[0.0 0.05 0.20 0.80 0.85 0.95 1.0]; % Set control point data
yp=[0.0 0.40 0.35 0.35 0.50 1.00 0.0];
np=length(xp);
figure;
n2=41; % Set the number of plot points between data
x=linspace(0.0,1.0,n2);
p=x; % This just sets up the 'size' of poly p

nlow = 6; ninc = 3; % <----- Set range of Bernstein degrees
nhigh = nlow + ninc*3;
for n = nlow:ninc:nhigh; % n = degree of Bernstein functions
    np1=n+1;
    x1=linspace(0.0,1.0,np1); % Compute control x-points and. . .
    y1=linear_spline_2(xp,yp,x1); % . . . interpolate for y-points
    for i=1:n2;
        xx=x(i);
        b1=bernstein_basis(n,xx); % We supply this function
        p(i)=y1*b1'; % Compute Bezier curve for each degree n
    end;
    if n==nlow;
        plot(x,p,'k:'); % Plot the poly's
        hold;
    elseif n==nlow + ninc*1;
        plot(x,p,'k-.');
    elseif n==nlow + ninc*2;
        plot(x,p,'k--');
    elseif n==nlow + ninc*3;
        plot(x,p,'k-');
    end;
end;

for i=1:np;
    plot(xp(i),yp(i),'ko'); % Plot the control points
    if i<np;
        xpl=[xp(i) xp(i+1)];
        ypl=[yp(i) yp(i+1)];
        plot(xpl,ypl,'k'); % Plot control curve segments
    end;
end;

axis([0 1 -0.5 1]);
grid;

function f = linear_spline_2(x,y,xx)
% LINEAR_SPLINE_2 linear spline returns y(xx) by using a linear spline
% between points x where x, xx and y are vectors.
n=length(x);
nxx=length(xx);

%for any xx(j) out of range, stop the program
if (xx(1)<x(1)) | (xx(nxx)>x(n));
    error('out of x range!');
end;
```

```

%loop for all the xx(j) points
for j = 1:nxx;

    %for each xx(j), try to find which data interval it belongs
    x1=xx(j);
    for i = 1:n-1;
        if (x1>=x(i)) & (x1<=x(i+1));
            m=i;
            break;
        end;
    end;

    %calculate y(xx(j))
    f(j) = y(m)*(x1-x(m+1))/(x(m)-x(m+1))...
        +y(m+1)*(x1-x(m))/(x(m+1)-x(m));
end;

function b = bernstein_basis(n,xx)
%BERNSTEIN_BASIS returns values of Bernstein basis polynomials....
% ....Bn,i(xx), n >= 1 at xx
% b is a vector of length n+1, b(i+1)=Bn,i(xx), i=0,1,...,n;
% Wensheng Yu and NJ Salamon, Feb 2001; Revised Mar 2002: NJS

if ((xx<0.0) | (xx>1.0));
    error('xx out of range in function 'bernstein_basis'!');
end
xx_1=1.0-xx;
b=1:n+1; % sets b(1)=1 and index to [1,n+1], not [0,n];
    % other values of b overwritten below.
for i=1:n; %starting from B_1,1, building Bn,i from Bn-1,i;
    b(i+1)=xx*b(i);
    for j=i-1:-1:1;
        b(j+1)=xx_1*b(j+1)+xx*b(j);
    end;
    b(1)=xx_1*b(1);
end;

```

## APPENDIX II: GRANADA MOSAIC

```

function gt_mosaic
% Generate a single Al-Hambra tile block
% Wensheng Yu and NJ Salamon, Feb 2001
% Revised: 20 June 2004, NJS

n=1;
obj0=[-1 .1125 0 .5392 1 .5392 0 .1125 -1; ...
    -1 -.5392 0 .1125 -1 -2.1125 -2 -1.4608 -1];
objrow3=ones(1,length(obj0)); obj0=[obj0;objrow3];
figure; plot(obj0(1,:),obj0(2,:)); hold;

% rotation to create one tile block
xr=0;yr=0;
obj1=gt_rotation( 90,1)*obj0; plot(obj1(1,:),obj1(2,:), 'b');
obj2=gt_rotation( 180,1)*obj0; plot(obj2(1,:),obj2(2,:), 'b');
obj3=gt_rotation( 270,1)*obj0; plot(obj3(1,:),obj3(2,:), 'b');

```

## APPENDIX III: AUTO FRONT

```

function autoFront
% Plot the front view of an auto using translation, rotation,
% scaling, reflection, and concatenations and import shapes
% from spline and/or Bezier functions.
% Scheme: Do one half of symmetric features, then reflect
% these w.r.t. the y-axis. Add unsymmetric features.
% Origin of coordinates is centered at bottom of bumper.
% Author: NJ Salamon, 10 Apr 04

%input paramters;

```



```

% Set the stage
road = [-3.5, 3.5]; [-.5, -.5];
plotobj(road);
axis([-4, 4, -1, 4]);

% Roof line
xp = [0.0 1.5 2.3 2.4]; % Set this
yp = [3.5 3.5 3.5 1.8]; % Set this
degree = 8;
roof_line = bezier_curve(xp, yp, degree);
plotobj(roof_line);

% Fender line
xp = [.50 .75 1.2 1.8]; % fender line
yp = [1.5 1.8 1.2 1.25];
degree = 3;
fender_line = bezier_curve(xp, yp, degree);
fender_line = xy2homo(fender_line);
mrot = gt_rotation(90, 1, 1.8, 1.25);
fender_line = mrot*fender_line;
mtrans = gt_translation(2.4--1.8, 1.8--1.25);
fender_line = mtrans*fender_line;
plotobj(fender_line);

% Bumper
% Simple bumper to test adding another object onto the same plot
xp = [0.0 2.3 2.3 0.0];
yp = [0.0 0.0 0.5 0.5];
bumper = [xp;yp]; % Needs a rounded end
plotobj(bumper);

% ----- FUNCTIONS -----
function datahomo = xy2homo(xydata)
n = length(xydata(1,:));
datahomo = [xydata; ones(1,n)];

function plotobj(obj)
plot(obj(1,:),obj(2,:));
grid;
hold on;

```

#### APPENDIX IV: OTHER REFERENCED FUNCTIONS

```

function m=gt_rotation(theta,deg,xr,yr)
% GT_ROTATION(THETA,DEG,XR,YR) rotates point A to point B thru
% angle theta around center of rotation, point (xr,yr)
% The transformation matrix R(theta,xr,yr) takes the form
%
% cos(theta) -sin(theta)  xr*(1-cos(theta))+yr*sin(theta)
% sin(theta)  cos(theta)  yr*(1-cos(theta))-xr*sin(theta)
% 0          0          1
%
% THETA is in radians by default; if DEG is present, a non-zero
% value means THETA is in degrees; if xr,yr are not present,
% the rotation is around the coordinate origin;
% Author: Wen-Shen Yu and NJ Salamon, 2001

if nargin==1;
    deg=0; xr=0; yr=0;
elseif nargin==2;
    xr=0; yr=0;
elseif nargin==4;
    error('number of input arguments wrong for gt_rotation!');
end;

if deg~=0;
    theta=theta*pi/180;
end;

```

```

ct=cos(theta);st=sin(theta);
m=[ct -st xr*(1-ct)+yr*st; st ct yr*(1-ct)-xr*st; 0 0 1];

function m=gt_translation(tx,ty)
% GT_TRANSLATION(TX,TY) translates point A to point B by a
% displacement of (tx,ty)
% The transformation matrix T(tx,ty) takes the form
%
%   1  0  tx
%   0  1  ty
%   0  0  1
%
% Note: gt_ stands for Geometrical Transformation
% Author: Wen-Shen Yu and NJ Salamon, 2001

m=[1 0 tx; 0 1 ty; 0 0 1];

function curve=bezier_curve(xp,yp,degree)
% Generates plot points for drawing shapes
% Author: njs, 1 Apr 04

% Set the control points and the degree range for Bezier curves
% xp = [0.0 1.5 2.3 2.4]; % Set this
% yp = [3.5 3.5 3.5 1.8]; % Set this
% xp = [0.0 1.5 2.0 2.1 2.2 2.35 2.4]; % Set this
% yp = [3.5 3.5 3.8 5.0 4.7 4.5 1.8]; % Set this
% degree = 3; % Set this
npts = 21; % Set this
np = length(xp);
xn = linspace(0.0,1.0,npts); % xn = normal coord's.
aa = xp(1); bb = xp(np);
xc = (bb - aa).*xn + aa; % xc = actual coord's
y = xn;
n = degree;
np1 = n+1;
x1 = linspace(xp(1), xp(np), np1);
y1 = linear_spline_2(xp,yp,x1); % Insure correct no. of control pts
for i = 1:npts;
xx = xn(i);
b1 = bernstein_basis(n,xx);
y(i) = y1*b1';
end
curve = [xc; y];

```

**Nicholas J. Salamon**, Ph.D. (Northwestern University, USA) has been a professor at Penn State since 1985. Prior to that he was associate professor at West Virginia University and assistant professor at the U. of Wisconsin—Milwaukee. He has taught mechanics of materials at both the introductory and advanced level off and on since 1975 and does research in stress analysis of materials and structures with the emphasis on computer analysis. His hobby is hiding away in the forests of Nittany Lion country.