CS 348C **Fall 1995-6**
Assignment 1 **Apostolos Lerios**

Due 10/19/95, 2:30p.

Notes

1. When you read this line, and provided you are taking the class for 3 units, please send mail to `tolis@cs.stanford.edu` and let me know that you have picked up the homework.
2. World Wide Web (WWW) documents referenced in this assignment can all be found under the course hierarchy

   `http://graphics.stanford.edu/courses/cs348c-95-fall/`
3. Please read the WWW page `requirements.html` for policies and regulations regarding your homework.
4. Feel free to use `Matlab` or `Mathematica` to avoid tedious tasks, such as solving huge linear systems or plotting in 3D.

Problem 1: Site Notation. *(15 points)*

Consider four points $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$, and $\mathbf{s}$ in 3D space. These points define an *arbitrary* tetrahedron T; for example, T may look like the tetrahedron in figure 1.

Let a be the length of side $\mathbf{pq}$, b be the perpendicular distance of $\mathbf{r}$ from the line $\mathbf{pq}$, and c be the perpendicular distance of $\mathbf{s}$ from the plane $\mathbf{pqr}$.

1. *(6 points)* Show that the volume V of T is $\frac{1}{6}abc$. Your proof may be based on some geometric construction, such as showing that T contains one-sixth of the volume of some associated parallelepiped. Alternatively, you may place T within an orthogonal coordinate system and use calculus.
2. *(3 points)* Show that

$$V = \frac{1}{6} \text{ abs } [((\mathbf{q}-\mathbf{p}) \times (\mathbf{r}-\mathbf{p})) \cdot (\mathbf{s}-\mathbf{p})].$$

   Here "$\times$" denotes the vector (cross) product operation, and "$\cdot$" stands for the dot product operation.
3. *(4 points)* Assume that the vertices of T have the following coordinates

$$\begin{aligned} \mathbf{p} &= (p_x, p_y, p_z) \\ \mathbf{q} &= (q_x, q_y, q_z) \\ \mathbf{r} &= (r_x, r_y, r_z) \\ \mathbf{s} &= (s_x, s_y, s_z). \end{aligned}$$

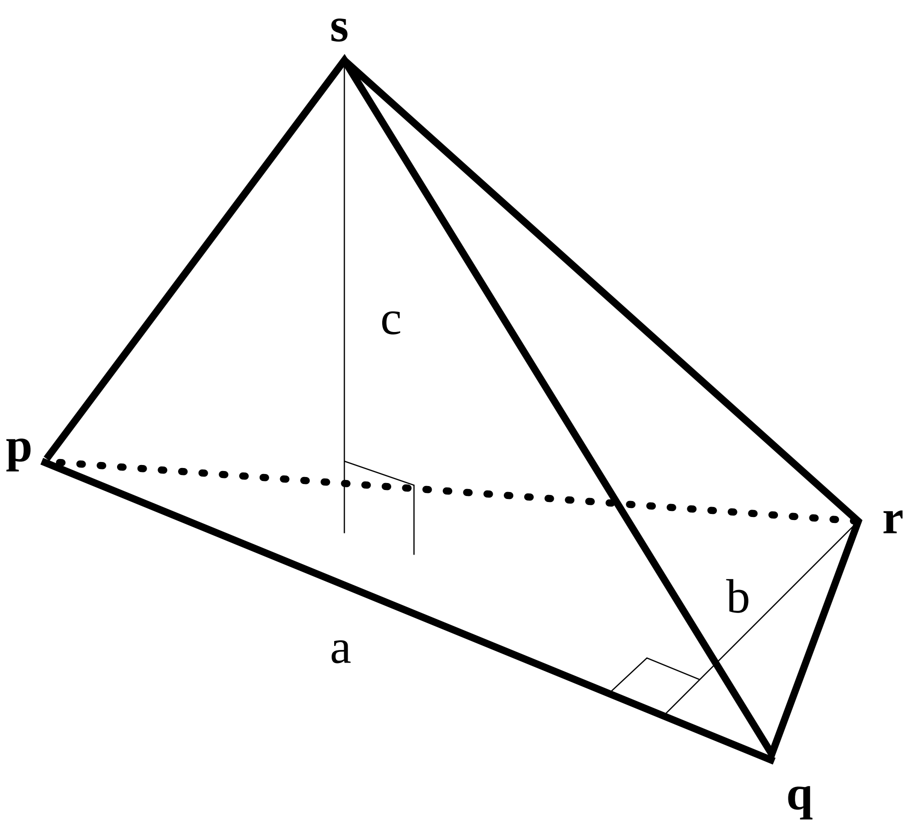

Figure 1: A sample tetrahedron for problem 1.

Show that

$$V = \frac{1}{6} \operatorname{abs} \begin{vmatrix} 1 & p_x & p_y & p_z \\ 1 & q_x & q_y & q_z \\ 1 & r_x & r_y & r_z \\ 1 & s_x & s_y & s_z \end{vmatrix}.$$

4. *(2 points)* What does the sign of the expression

$$\begin{vmatrix} 1 & p_x & p_y & p_z \\ 1 & q_x & q_y & q_z \\ 1 & r_x & r_y & r_z \\ 1 & s_x & s_y & s_z \end{vmatrix}$$

tell us about the relative positions of T's vertices? What is the sign of this expression for the *particular* tetrahedron shown in figure 1?

Problem 2: Barycentric Coordinates. *(15 points)*

In this problem, you are given the Cartesian coordinates of $n+1$ points on the plane. You have to find the barycentric coordinates of one of these points, $\mathbf{p}$, with respect to the other ones, $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Stated differently, you have to express $\mathbf{p}$ as a barycentric combination of $\mathbf{p}_1, \ldots, \mathbf{p}_n$. To refresh your memory, this means that you have to look for real numbers $a_1, \ldots, a_n$, called the barycentric coordinates, such that

$$a_1 + \cdots + a_n = 1$$

and

$$\mathbf{p} = a_1\mathbf{p}_1 + \cdots + a_n\mathbf{p}_n.$$

There are three possible outcomes to your search, and thus three valid answers to each of the parts that follow:

- You realize that there can't exist numbers $a_1, \ldots, a_n$ with the above properties. Just saying so (without elaborate justification) is a fine answer to the problem.
- You realize that there are infinite ways in which $a_1, \ldots, a_n$ can be selected. Saying so *and* listing two possible barycentric coordinates, along with your derivation, is an adequate answer.
- You realize that there is one and only one way to select $a_1, \ldots, a_n$. In this case, state the answer and show your work.

Here are the six parts of this problem; for some of them, it's easier to use algebra, for some it's easier to draw a diagram, and for some it's easier to relate them to previous ones:

1. *(3 points)* $\mathbf{p} = (.5, .25)$, $\mathbf{p}_1 = (0,0)$, $\mathbf{p}_2 = (0,1)$, and $\mathbf{p}_3 = (1,0)$.
2. *(3 points)* $\mathbf{p} = (-6,-6)$, $\mathbf{p}_1 = (1,2)$, $\mathbf{p}_2 = (5,1)$, and $\mathbf{p}_3 = (6,4)$.
3. *(3 points)* $\mathbf{p} = (5,11)$, $\mathbf{p}_1 = (2,5)$ and $\mathbf{p}_2 = (6,13)$.
4. *(3 points)* $\mathbf{p} = (1,3)$, $\mathbf{p}_1 = (5,11)$, $\mathbf{p}_2 = (2,5)$, and $\mathbf{p}_3 = (6,13)$.
5. *(2 points)* $\mathbf{p} = (0,0)$, $\mathbf{p}_1 = (2,5)$ and $\mathbf{p}_2 = (6,13)$.
6. *(1 point)* $\mathbf{p} = (0,0)$, $\mathbf{p}_1 = (1,3)$, $\mathbf{p}_2 = (2,5)$, and $\mathbf{p}_3 = (6,13)$.

Problem 3: Homogenous Coordinates. *(15 points)*

In this problem, we use the notation (p_x, p_y) to represent the Cartesian coordinates of a point on the regular plane (also called the *Euclidean* plane). Also, (p_x, p_y, p_z) represents the Cartesian coordinates of a point in the regular 3D space. And, lastly, the notation $(p_w; p_x, p_y)$ represents the homogenous coordinates of a point on the *projective* plane.

To refresh your memory, $(\lambda p_w; \lambda p_x, \lambda p_y)$ represents the same projective point as the triple $(p_w; p_x, p_y)$, for any non-zero λ. So, when we say that $(p_w; p_x, p_y)$ are "the homogenous coordinates" of a projective point, what we actually mean is that $(p_w; p_x, p_y)$ is "one of the infinite triples of homogenous coordinates" for the point. Also, recall that $(0; 0, 0)$ is a bogus triple; that is, it does not represent any point on the projective plane. Finally, points on the projective plane come in two varieties:

Finite points are those projective points with $p_w \neq 0$. A projective point $(p_w; p_x, p_y)$ of this kind corresponds to the Euclidean point $(p_x/p_w, p_y/p_w)$.

Points-at-infinity are those projective points with $p_w = 0$. These points have no corresponding Euclidean points.

In the first four parts of this problem, you are given equations of Euclidean curves; that is, equations which Euclidean points satisfy if and only if they lie on the curve. In each part, you are given two equations, each defining one curve. Your task is twofold:

- For each curve separately, you must find its equation on the projective plane. This means, find an equation which is satisfied by these projective points (and no other) whose corresponding Euclidean points, if any, are on the given curve. This process is called *homogenization* of an equation, and it yields a projective curve given a Euclidean one.
- Find those projective points, if any, which lie on both projective curves. To be more precise, list all the triples $(p_w; p_x, p_y)$ which satisfy the equations of both projective curves.

In the last part of this problem, you are expected to plot a projective curve. The standard way of plotting curves (drawing two perpendicular lines for the two axes, etc.) works only for Euclidean curves; for projective curves, this visualization method cannot depict the points-at-infinity as these projective points have no corresponding Euclidean ones. Stolfi (see the course reader) describes several methods of visualizing the projective plane. The one you are expected to use is the *spherical model*, which works as follows:

- You first draw a 3D coordinate system.
- Then, you draw a sphere of radius 1, centered at the origin.
- Finally, you map the projective plane onto this sphere. That is, you pair every point on the projective plane with *two* points on the surface of the sphere. In particular, you draw the projective point $(p_w; p_x, p_y)$ at two places:

  $$(p_w/l, p_x/l, p_y/l) \quad \text{and} \quad (-p_w/l, -p_x/l, -p_y/l)$$

  where $l = \sqrt{p_w^2 + p_x^2 + p_y^2}$.

Here are the five parts of this problem; x and y are the Cartesian coordinates of the Euclidean point (x, y).

1. *(3 points)* $x + y = 2$ and $x - y = 4$.
2. *(3 points)* $3x + 2y = 4$ and $-3x - 2y = 1$.
3. *(3 points)* $xy = 1$ and $x = 1$.
4. *(3 points)* $xy = 1$ and $x = 0$.
5. *(3 points)* Plot and label the curves of part 4. Also, mark and label all their intersection points, if any, on your drawing.

Problem 4: Maps. *(15 points)*

Recall that maps are operations which associate every single point $\mathbf{p}$ to some other point $\mathbf{q}$, called the image of $\mathbf{p}$. Here, we will concentrate on maps which associate points on the projective plane to other points on the projective plane. These maps come in many varieties, some of which can be concisely expressed in the form

$$\mathbf{q} = \mathbf{A}\mathbf{p}$$

where $\mathbf{A}$ is a 3-by-3 matrix and both $\mathbf{q}$ and $\mathbf{p}$ are written as column vectors of the form

$$\begin{pmatrix} w \\ x \\ y \end{pmatrix}.$$

Here are three important varieties of maps which can be expressed in this form:

Linear: a map is linear if $\mathbf{A}$ has the form

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & a & b \\ 0 & c & d \end{pmatrix}.$$

Affine: a map is affine if $\mathbf{A}$ has the form

$$\begin{pmatrix} 1 & 0 & 0 \\ e & a & b \\ f & c & d \end{pmatrix}.$$

Projective: a map is projective if $\mathbf{A}$ has the form

$$\begin{pmatrix} i & g & h \\ e & a & b \\ f & c & d \end{pmatrix}.$$

In each of the three parts of this problem, you are given four points $\mathbf{p}_1, \ldots, \mathbf{p}_4$, on the projective plane. You are also given their desired images $\mathbf{q}_1, \ldots, \mathbf{q}_4$. Your task is twofold:

- Find a map that is linear, affine, or projective and which indeed manages to associate $\mathbf{p}_1$ with $\mathbf{q}_1$, $\mathbf{p}_2$ with $\mathbf{q}_2$ etc. — the other points on the projective plane may be mapped to anything you wish.
- In finding such a map, you should prefer a linear map (if one exists) to an affine one, and an affine one (if one exists) to a projective one.

In other words, for each of the three parts, you are required to find the *simplest* $\mathbf{A}$ which performs the desired map.

Here are the three parts of this problem. For the sake of brevity, we use Cartesian coordinates (x, y) for the projective point $(1; x, y)$. Also, for all three parts, assume that

$$\mathbf{p}_1 = (0,0), \mathbf{p}_2 = (1,2), \mathbf{p}_3 = (-1,2), \mathbf{p}_4 = (-3,-6).$$

1. *(5 points)* $\mathbf{q}_1 = (0,0)$, $\mathbf{q}_2 = (4,11)$, $\mathbf{q}_3 = (0,5)$, and $\mathbf{q}_4 = (-12,-33)$.
2. *(5 points)* $\mathbf{q}_1 = (5,4)$, $\mathbf{q}_2 = (7,8)$, $\mathbf{q}_3 = (3,8)$, and $\mathbf{q}_4 = (-1,-8)$.
3. *(5 points)* $\mathbf{q}_1 = (0,0)$, $\mathbf{q}_2 = (.25,.5)$, $\mathbf{q}_3 = (-.5,1)$, and $\mathbf{q}_4 = (.375,.75)$.

Problem 5: Parametric Polynomial Curves. *(40 points)*

This is a programming problem, in part: you have to write the drawing portion of `polly`, a program that allows the user to visualize and manipulate parametric polynomial curves in space. Also, you have to use `polly` to answer some simple questions.

Programming part. *(35 points)*

Here is what you have to do:

1. Log on to one of the Silicon Graphics Indigo workstations in the Sweet Hall graphics lab (room 026)[1]. Their names are `raptor1` through `raptor14`, all in the `stanford.edu` domain. Remote logins are not possible on the first five machines. To use these machines, you must go to Sweet Hall in person; to open the combination lock of the lab, press 5 and 1 together, release both, press 2 and 3 together, release both, and turn the knob.
2. Get the source code of `polly` from the WWW. It is available as a `tar` (archive) file named

   `assignments/first/polly.tar`

[1] Any other Silicon Graphics machine should work just as well, provided the development environment of Open Inventor is installed.

Save this file in some empty directory.

3. Unpack the archive by typing `tar xf polly.tar` in the command line. The following files should appear in your directory:

   - `Makefile`
   - `draw.c`
   - `polly.c++`
   - `polly.h`
   - `xsupport.c++`
   - `xsupport.h`

   You do not need your copy of `polly.tar` any longer.

4. Type `make`. Compilation and linking should succeed. You do not need the resulting executable named `polly`.

5. Peruse `polly.h`. It contains the headers and documentation of the functions you have to write.

6. Write the drawing code. To this end, modify the skeleton code of `draw.c`, mainly by writing the body of its three functions.

7. Make sure your code works correctly.

8. Comment your code adequately and carefully.

9. Mail your edited `draw.c` to `tolis@cs.stanford.edu`, entering `polly` in the subject line. If you discover any bugs after mailing in your code, just mail the updated copy; only your latest mail prior to the deadline will be used for grading purposes.

To help you in the development of your work, we provide a documented working executable of `polly`; see the WWW page

`software/polly/index.html`

A copy of the executable is also available on the Sweet Hall machines under

`/usr/class/cs348c/software/polly`

This copy is also available via anonymous ftp to `ftp.stanford.edu` under

`/pub/class/cs348c/software/polly`

Hints:

- Do not worry about issues relating to numerical instability; Lagrange polynomials are perfectly acceptable.

- Efficiency matters to the extent that the user interacts comfortably with the program. Do not spend hours devising your own, super-efficient fixed-point representation. Also, make sure that you activate the optimization flags in the `Makefile` when you are assessing efficiency.
- Recall that a point on the Bézier curve $\mathbf{p}(t)$ of degree $d = n - 1$ is given by
$$\mathbf{p}(t) = \mathbf{b}_1 B_1(t) + \cdots + \mathbf{b}_n B_n(t).$$
where $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are the Bézier points of the curve, and
$$B_{i+1}(t) = \frac{d!}{i!(d-i)!} t^i (1-t)^{d-i}$$
are the Bernstein polynomials. Also, an interpolating curve of degree $n-1$, passing through $\mathbf{p}_1, \ldots, \mathbf{p}_n$ at times $t_1, \ldots, t_n$ is given by
$$\mathbf{p}(t) = \mathbf{p}_1 L_1(t) + \cdots + \mathbf{p}_n L_n(t)$$
where
$$L_i(t) = \frac{\prod_{j=1, j\neq i}^{j=n} (t - t_j)}{\prod_{j=1, j\neq i}^{j=n} (t_i - t_j)}$$
are the Lagrange polynomials.

Written questions. ***(5 points)***

Using either your version of `polly`, or the provided executable, answer the following questions:

1. *(3 points)* Execute `polly` using the command-line

   ```
   polly -count 8
   ```

   Now move the fifth control point upwards. How many wiggles ("ups" and "downs") does the resulting *interpolating* curve have? What about a *Bézier* curve?

2. *(2 points)* Execute `polly` twice using the command-lines

   ```
   polly
   ```

   and

   ```
   polly -points 0 0 0 6 0 0 3 0 0 9 0 0
   ```

   In both cases, the resulting curve is a line (of which you see a small segment). Animate the movement of a particle along each of these lines, and describe what you see in *at most* a sentence or two.

The first question demonstrates the advantages of Bézier over interpolating points for curve specification: the former provide easy, local control of the curve's shape. The second question demonstrates the fact that two curves may have the same shape but very different parameterizations.