Background

In computer graphics, it is convenient to render the scene with the camera at the origin and pointed in the **negative** z-direction. However, we want to be able to position the camera anywhere and render the scene. This means that we must transform from "world" coordinates to "camera" (or "eye") coordinates. To accomplish this, the user must provide 3 pieces of information (read in from an input file, refer to the example figure):

- the location (x,y,z) of the camera in "world" coordinates (**eye point, E**)
    - the location (x,y,z) that the camera is pointed at (**at point, A**)
    - the direction (x,y,z) that is "up" for the camera (**up vector, v_up**)

After obtaining the three coordinates for one of the above items, repeat the input in the form of an ordered triplet (x, y, z).

Using these supplied values, it is possible to construct a coordinate system where the camera is pointed along one of the coordinate axes. Then we can take all of our objects in world coordinates and transform them into camera coordinates. Instead of (x,y,z), the camera coordinates are usually labeled (u,v,n).

Computing n

The view normal (n) is easily obtained through point-point subtraction. Simply subtract the at point from the eye point (E - A) coordinate by coordinate. This is the direction along which the camera is pointed. Note that the camera is pointed in the **negative** n-direction.

Computing v

Obtaining v is more complex. First, v must be in the same plane as n and the up vector. This means that it is a **linear combination** of n and v_up which we can write as:

$v = \alpha * n + \beta * v\_up$

In addition, v must be orthogonal (at a right angle) to n (just as x, y, and z are all orthogonal to one another). We can use the **dot product** to enforce this requirement.

$a.b$ (a dot b) = $a_x*b_x + a_y*b_y + a_z*b_z$ (an obvious candidate for a macro)
where a_x is the first component of a, etc.
note: the dot product is just a number, a **scalar**

If two vectors are orthogonal, then their dot product is 0. Let's take the dot product of our formula for v with n:

0 = α*(n.n) + β*(v_up.n), which means that (solving for α and substituting)
(n.n)v = -β(v_up.n)n + β(n.n)v_up

To find (n.n)v, you will need to perform **vector-scalar multiplication** and **vector_vector addition**. First, you need to compute -β(v_up.n), which is just a scalar. Then, multiply this scalar by **each component** of n to obtain each component of -β(v_up.n)n. Now, add the components of -β(v_up.n)n and β(n.n)v_up together, component-by-component, to obtain (n.n)v.

I have multiplied v through by n.n (again, just a number) so that I can avoid doing division. When doing any integer arithmetic, we want to do the division last so that the rounding error involved in division does not accumulate with each subsequent arithmetic operation. This will not affect my answer as multiplying a vector by a scalar changes its length, but not its direction. In fact, since we are not concerned with the length of v, just its direction, we can set (n.n)/β = 1.

v = -(v_up.n)n + (n.n)v_up

Computing u

The last vector in our new coordinate system is u. It is obtained simply by noting that it must be orthogonal to both v and n. We can use the **cross product** to obtain u. The cross product of two vectors gives a third vector orthogonal to the first two. Note that you will actually obtain (n.n)u, but again, this is okay. Compute u using v x n.

a x b (a cross b) =
a_y*b_z - a_z*b_y (the first component of a x b)
a_z*b_x - a_x*b_z (the second component of a x b)
a_x*b_y - a_y*b_x (the third component of a x b)

Normalization

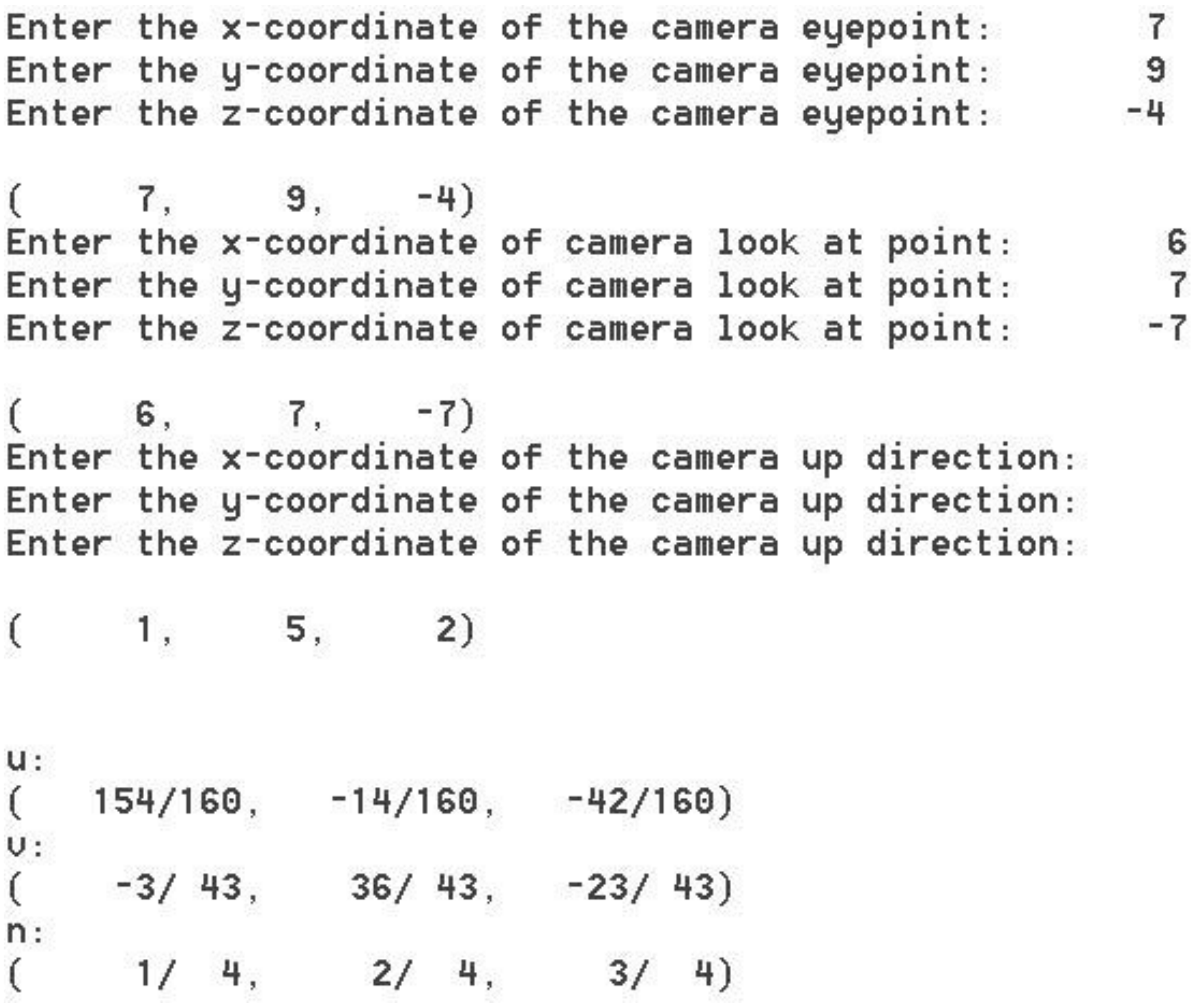
For the transformation to camera-coordinates to work correctly, the lengths of each of our 3 vectors must be 1. The dot product can be used to obtain the length of a vector:

a_len = √a.a

To **normalize** a vector, simply divide each component of the vector by the length of the vector. Thus, the division is done as the very last arithmetic operation. In fact (refer to the example figure), we can avoid division error by reporting our results as a rational number. In this way, the only error present is the error associated with finding the square root (done using sqrt.h and sqrt.obj).

Use the provided batch file (camera_build.bat) to assemble and link your program. Your program must work with this build file. Put header files (io.h, debug.h, sqrt.h) in the Assembly directory. This will make it much easier for me to grade your programs.

Your program must work with an input file (camera_input.txt) and have output formatted as in the below figures:



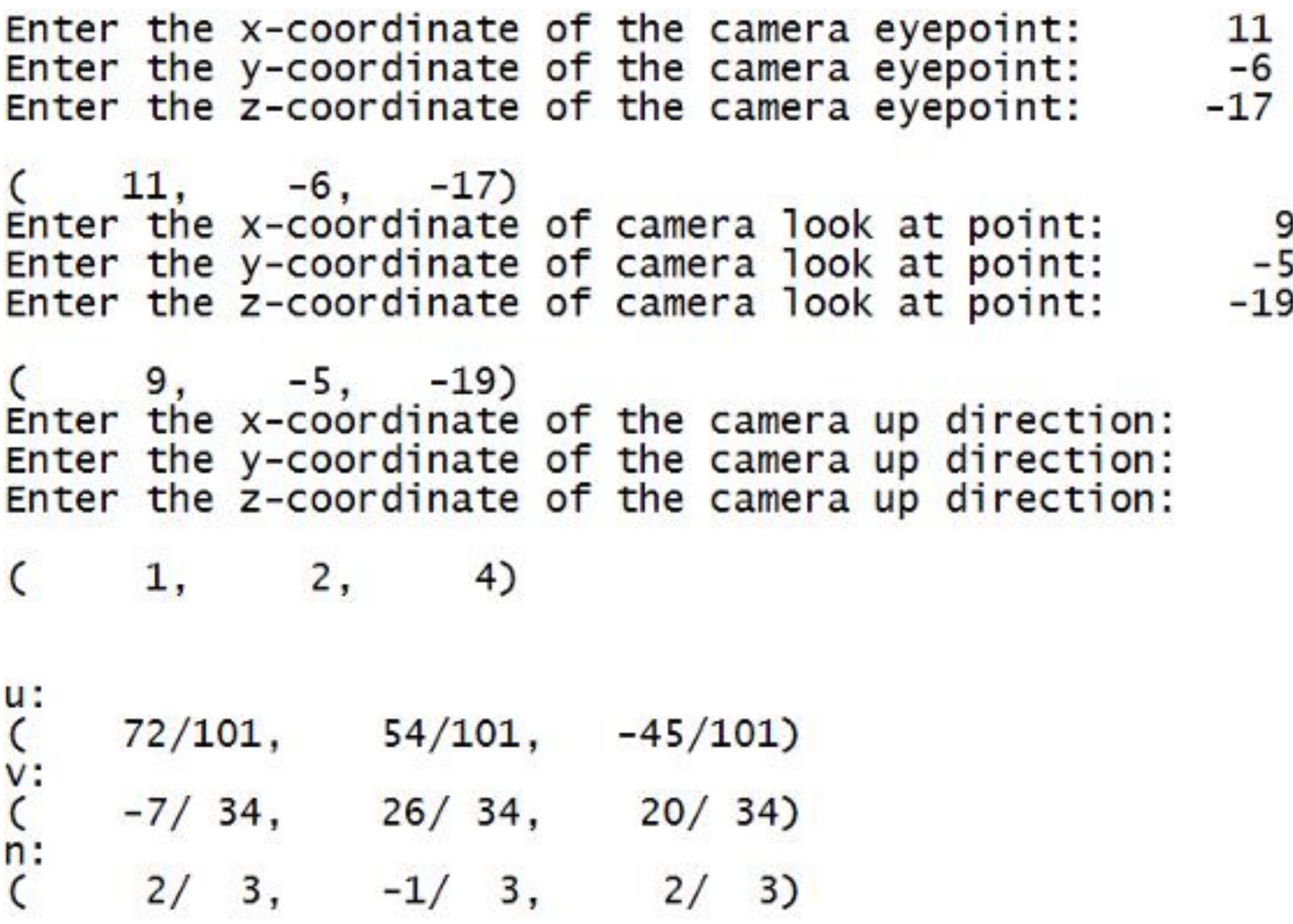
```
Enter the x-coordinate of the camera eyepoint:      7
Enter the y-coordinate of the camera eyepoint:      9
Enter the z-coordinate of the camera eyepoint:     -4

(     7,      9,     -4)
Enter the x-coordinate of camera look at point:      6
Enter the y-coordinate of camera look at point:      7
Enter the z-coordinate of camera look at point:     -7

(     6,      7,     -7)
Enter the x-coordinate of the camera up direction:
Enter the y-coordinate of the camera up direction:
Enter the z-coordinate of the camera up direction:

(     1,      5,      2)


u:
(   154/160,   -14/160,   -42/160)
v:
(    -3/ 43,    36/ 43,   -23/ 43)
n:
(     1/  4,     2/  4,     3/  4)
```



```
Enter the x-coordinate of the camera eyepoint:      11
Enter the y-coordinate of the camera eyepoint:      -6
Enter the z-coordinate of the camera eyepoint:     -17

(    11,     -6,    -17)
Enter the x-coordinate of camera look at point:      9
Enter the y-coordinate of camera look at point:     -5
Enter the z-coordinate of camera look at point:    -19

(     9,     -5,    -19)
Enter the x-coordinate of the camera up direction:
Enter the y-coordinate of the camera up direction:
Enter the z-coordinate of the camera up direction:

(     1,      2,      4)


u:
(    72/101,     54/101,    -45/101)
v:
(    -7/ 34,     26/ 34,     20/ 34)
n:
(     2/  3,     -1/  3,      2/  3)
```

Program Specification:

- Compiling, Linking, Running
  - Basic I/O (using io.h)
    - Obtaining user input
    - Using an input file
    - Displaying computation results
  - Data Types
    - WORD
    - BYTE
  - Basic Instructions and Computations
    - Move
    - Add
    - Subtract
    - Multiply
  - Write **Numerous** Macros
    - Computation Macros

- Display Macros

Submission:
Submit Electronically:

- camera.asm