CS 420/686 Game Engineering
Practice Problems #1

Assume the following for all problems:

- Use row vectors (except question 10!)
- Use a right-handed coordiate system.
- All models point down the z axis in their object space (with "up" poining along y axis in object space)

1. Spaceship 1 has the rotation matrix (in world space):

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

   And the position (in world space) $[p_x, p_y, p_z]$

   You want to place Spaceship 2 directly in front of Spaceship 1 (in Spaceship 1's reference frame), 20 units away, pointing towards Spaceship 1. Spaceship 1 and spaceship 2 should have the same up vector.

   Give the rotation matrix (in world space) and the position (in world space) of Spaceship 2

2. A character in your game has the rotation matrix (in world space)

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

   And the position (in world space) $[p_x, p_y, p_z]$

   You wish to place a camera 10 units directly to the right of the character, pointing at the character (that is, looking at the character's right shoulder), with the same up vector as the character.

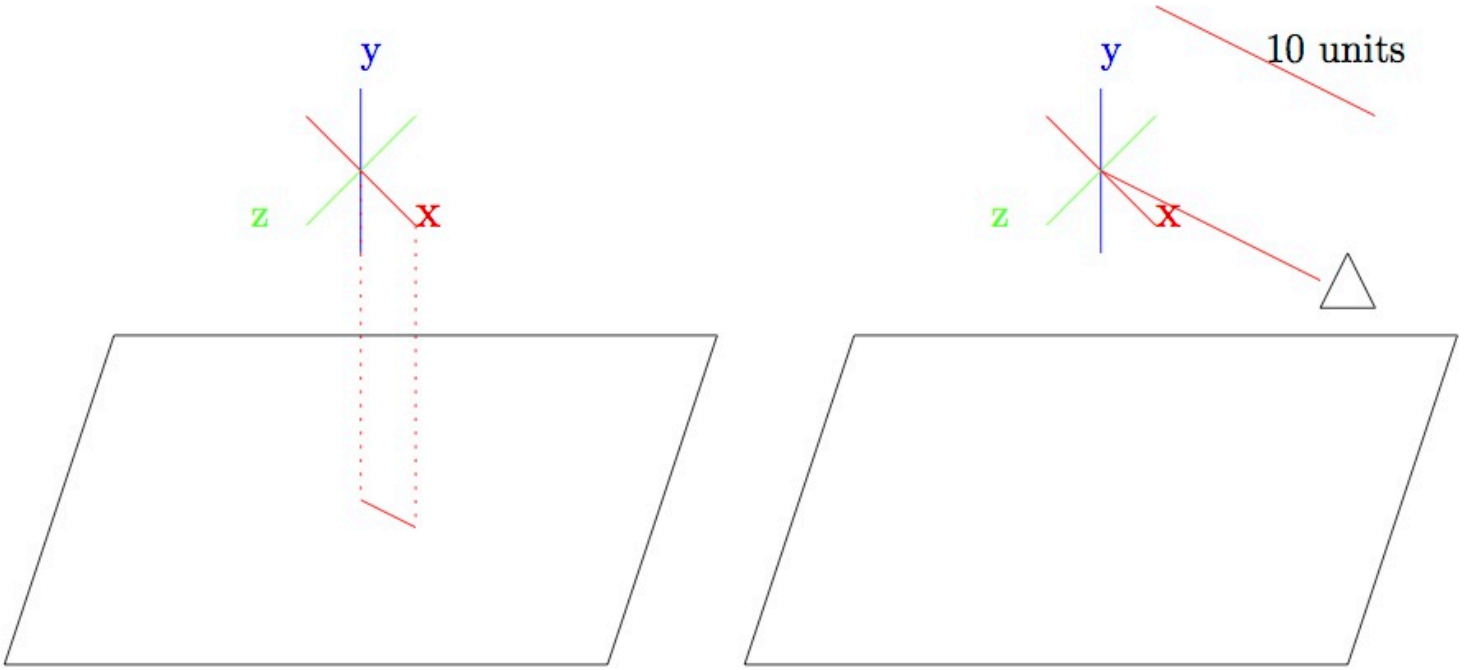
   Give the rotation matrix (in world space) and the position (in world space) of the camera.

3. A character in your game has the rotation matrix (in world space)

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

   And the position (in world space) $[p_x, p_y, p_z]$

   The ground is perpendicular to the world up vector. You wish to place a camera 10 units away from the character, along the line you get when you project the x axis to the ground plane. The camera should have the same up vector as as the world, and be pointed at the character. (See diagram) You can assume that $[m_{11}, m_{12}, m_{13}]$!= [0, 1, 0] and $[m_{11}, m_{12}, m_{13}]$!= [0, -1, 0] (so that x has a non-zero projection to the ground plane).



4. An object initially has the transformation vector

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

after applying a transform M, we are left with:

$-m_{11}$	$-m_{12}$	$-m_{13}$
$-m_{31}$	$-m_{32}$	$-m_{33}$
m_{21}	m_{22}	m_{23}

Does the transformation M contain a reflection?

5. Recall that quaternions are used to represent displacements (though they can also be used to represent absolute orientation, as a displacement from a reference orientation). For each of the following pairs of quaternions, describe how the displacement for quaternion 1 differes from the displacement for quaternion 2

   a. [w, (x, y, z)] and [-w, (x, y, z)]
   b. [w, (x, y, z)] and [w, (-x, -y, -z)]
   c. [w, (x, y, z)] and [-w, (-x, -y, -z)]
   d. [w, (x, y, z)] and [w, (-x, y, z)]
   e. [w, (x, y, z)] and [w, (x, -y, z)]
   f. [w, (x, y, z)] and [w, (x, y, -z)]
   g. [w, (x, y, z)] and [w, (-x, -y, z)]

6. Describe the change in orientation described by the following quaternions:

   a. [1, (0, 0, 0)]
   b. [0, (0, 1, 0)]
   c. [1/sqrt(2), (1/2, 1/2, 0)]

7. You have a spaceship whose position and orientation ore defined by the Ogre variables:

```
Ogre::Vector3 shipPos;
```

```
Ogre::Quaternion shipOrientation;
```

The spaceship has an orbiting satelite whose position and orientation (relative to the spaceship) are:

```
Ogre::Vector3 satelitePosLocal;
Ogre::Quaternion sateliteOrientationLocal;
```

a. Give code to calculate the position and orientation of the satelite in world space:

```
Ogre::Vector3 satelitePosGlobal = ...
Ogre::Quaternion sateliteOrientationGlobal = ...
```

b. You have a point in satelite space:

```
Ogre::Vector3 pointInSateliteSpace;
```

Give code to transform that point into world space:

```
Ogre::Vector3 pointInWorldSpace = ...
```

c. You have a point in world space:

```
Ogre::Vector3 pointInWorldSpace;
```

Give code to transform that point into satelite space:

```
Ogre::Vector3 pointInSateliteSpace = ...
```

8. Given the following bit of code:

```
class foo
{
    foo(int a, char *b, char *c) : x(a) {
       strcpy(y,b);
       z = new char[strlen(c) + 1];
       strcpy(z,c);
    }
    int x;
    char y[10];
    char *z;
};
class bar
{
   bar(int g, char * h, char *i) : d(g, h, i)
   {
      e = new foo(g, h, i);
   }
   foo d;
   foo *e;
};


int main()
{
   bar v(3, "hello", "there");
   bar *vPtr = new bar(5, "test", "string");
  // Point A
}
```

Show the contents of the stack and heap at Point A

9. Give the output of the following bit of code (WARNING: This is tricky. Trickier than you might think ...):

```
#include

class A
{
public:
  void P1() {printf("P1 in A\n");}
  virtual void P2() {printf("P2 in A\n");}
  virtual void P3() {printf("P3 in A\n");}

};
```

```
class B : public A
{
public:
  void P1() {printf("P1 in B\n");}
  void P2() {printf("P2 in B\n");}
};


class C : public B
{
public:
  void P1() {printf("P1 in C\n");}
  void P2() {printf("P2 in C\n");}
  void P3() {printf("P3 in C\n");}
};


int main()
{
  A aVar;
  B bVar;
  C cVar;
  A *aPtr = new C();
  B *bPtr = new C();

  aVar.P1();
  aVar.P2();
  aVar.P3();
  printf("--------\n");
  bVar.P1();
  bVar.P2();
  bVar.P3();
  cVar.P1();
  cVar.P2();
  cVar.P3();
  printf("--------\n");
  aPtr->P1();
  aPtr->P2();
  aPtr->P3();
  bPtr->P1();
  bPtr->P2();
  bPtr->P3();
  printf("--------\n");
  aVar = cVar;
  aVar.P1();
  aVar.P2();
  aVar.P3();
  printf("--------\n");
  bVar = cVar;
  aPtr = &(bVar);
  aPtr->P1();
  aPtr->P2();
  aPtr->P3();
}
```

10. You have calcuated a series of transformations for a transforming a vector v (assuming v is a row vector) to get the result:

    ```
    r = vABC
    ```

    However, you now realize that your system uses column vectors instead of row vectors! Give a formula for the equivalent column vector r' (in terms of the column vector v', and the transformation matrices A, B, and C),