HW1: What you need to know as a noob as you start...

I have already completed ***HW1*** and I feel like more could have been provided for us to work on it. So I am starting this thread for those who have not yet started or are having problems with it.

I am in no way very good in C++, computer graphics, GL stuff or any other relevant material but I am hoping that we can gather some info to help others learn/do things faster. I am planning to update this message (if I can) with your inputs. And finally I would kindly ask the staff's understanding (and editing) if they think too much is revealed... So here we go...

What to do:

HW1 is about just filling in four functions in the given code to do two things: turn a teapot image horizontally and vertically. Well, actually this is what it looks like, we are infact moving our camera (see the "crystall ball interface" explanation in the first page (Introduction) of the homework.

Four functions for two things? Yes. Transform::left and Transform::up are called to do the deeds, but they should refer to and use Transform::rotate to do it. And the fourth function Transform::lookat is just for us to learn how to play with matrices and vectors of geometry to perform the lookat function. Infact, I think you may actually start working on it independently from the other three functions but it is better to leave it to the end.

All your code could take up to 10 to 20 lines of code really, so do not despair. The tough part is understanding, not coding.

If we get into more details, the left and up keys will be changing your camera locations by turning your camera around the "crystal ball"... This is unfortunately not a rotation around a fixed axis, well at least not after you start mixing horizontal and vertical movements. So you will need two vectors: eye and up... eye is your camera position, luckily always the same distance from the object (teapot) center, just revolving around a "crystal ball" for this exercise. And up is your second vector to figure out which way is, well, up! Infact, once you are revolving all around the "crystal ball", you would need to know the "up" direction in order to know which direction would be the "left" too. Hence, both "left" and "up" functions have the two vectors "eye" and "up" as arguments in addition to the "degrees" to turn. Speaking of degrees, remember that computer trigonometry operates on "radians" instead of "degrees", so a conversion will be necessary.

Once you know what direction (or around what arbitrary axis) you will turn thanks to your eye and up vectors, you can simply make use of the rotation matrix your code in Transform::rotate will provide you with. As you noticed, left and up are void functions, while rotate returns a mat3 and lookat returns a mat4.

This was all about the coding part to help you start. I am afraid I cannot get into more details from this point on as now you will have to figure how to apply the materials in the lecture for your implementation. And many other threads exist in the discussions on various topics about your steps after this point. I hope this much helps to begin your actual quest.

What we will get:

At the end, if all is done correctly, the teapot will rotate towards left when we press left and down when we press up. (I think there is a mismatch here so the cursor keys are moving neither the teapot nor your camera but doing a mix of both?) Make sure when you turn it all the way around (especially in the up direction), the teapot does not make sudden 180 degree turns, that means there is a problem with your implementation.

When you pres "g" nothing will happen if you correctly implemented your Transform::lookat function because both your code and the gluLookAt code will be showing the same view and you will not be able to see a difference.

Finally, you will simply press "i" and the code will perform its magic to produce preplanned movements (from the "input.txt" file passed through the command line) and taking pictures of their results. You will zip those image files as explained in the submission page and hopefully be done to play around with your code at your own pleasure. You do not need to bother with passing file name in the command line argument if you are using Visual Studio skeletons; otherwise, well, you do...

glm framework you'll wish you know:

First, you have the objects (data types?) to begin with... They are things like vec2, vec3, vec4, mat2, mat3, mat4 and more... But the kinds you will be using are even less in number than the ones I mentioned here. "vec"s are for vectors (one dimension) and "mat"s are for matrices (two dimensions). The number that follows indicate their dimensions so vec3 is a 3x1 vector and mat3 is a 3x3 matrix. You can declare and initialize and assign them as you would with other data types, such as:

```
    vec3 myVector = vec3(1.0, 2.0, cos(radians));
    mat3 myMatrix = mat3(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

But most often than not, you may do their value assignment by some functions and some vectors & matrices shown in the lectures. What functions? Well, here are some useful glm functions:

```
    glm::cross(firstVector, secondVector) //cross prod.
    glm::dot(firstVector, secondVector)   //dot prod.
    glm::normalize(singleVector)          //normalize
    // argument vectors are of type vec3 in our case
```

Oh, there is also the matter of accessing singular values within the vectors or matrices. You can do that as you would with arrays such as eye[0], eye[1] or eye[2]. But you can also use eye.x, eye.y, and eye.z to the same effect.

Finally, there is this "row-major" vs. "column-major" discussion going around but as a noob, I feel that it is already taken care in the code so you can just stick with the format shown in the lectures. We will likely delve into that issue when we are required to type ALL of the code. No problem, I can wait... :)

My recommendations for starting:

These are not necessary, but will probably come in handy for noobs like me at the beginning...

You may consider putting the following line in your Transform::left and Transform::up functions to see what is happening to your eye (and/or up) vectors for debugging purposes:

```
    printf("Coordinates: %.2f, %.2f, %.2f; distance: %.2f\n",
eye.x, eye.y, eye.z, sqrt(pow(eye.x, 2) + pow(eye.y, 2) +
pow(eye.z, 2)));
```

If the final number (the distance) seems to be changing in this output ---> something's wrong! :)

Again, if you are a noob like me, start by using simple implementations of the eye and up vectors in the corresponding functions (without coding or using the rotate function) to see what is happening. You can use the 2D rotation formulas for this and this is, of course, incorrect; but it will give you a feeling of what the code does. It is also a pleasure having that teapot dance around early on. If you take this path, you may also encounter some other problems like the teapot moving towards you or away from you but do not mind it at this point, hopefully when you incorporate 3D axis rotations through the rotation function's returned matrix later, things will improve.

I started by having the left function always turn the view around the vertical (y) axis (again, incorrect) to feel some accomplishment. Then I implemented the rotate function (which returns a mat3, if you remember, obviously to be used somewhere) and use it within the left function. When you complete these steps, it is time for the up function, which is a little bit trickier. Of course, if you prefer, you can also make another incorrect implementation for the up function (as I did) just to see that teapot doing more complicated stuff. With the rotate function in use, you will strive to make your left and up functions do the correct things and you may even find out other problems in your left and rotate functions after you have a go at the up function. But believe me, when the teapot starts turning on your screen, you feel more easy doing new things. (That does not mean it gets easier, it's just how you feel...) :D

One important remark on your "up" direction (vector): if you are wondering why on earth the direction of up would ever need to change, the simple answer is: you are no longer standing on the earth. :) Think of when you move your camera (eye) right at the top of the teapot, looking down at it; at that point, the "up" direction for the teapot (or the "real world") is quite different from the "up" direction for your camera. ;) Also when you go up a 180 degrees, you (camera) will be seeing the teapot upside down, right? That's another example of your "up" being different from the "real world up".

I cannot say much about the lookat function except that once you implement it, you finally get out of the teapot (when you press "g"). Three things to look out for: 1) The original code just returns an empty mat4 object. Remember to return the mat4 object you constructed by updating the return call at the last line as indicated in the comment in the code. 2) If you see nothing but the blue emptiness, maybe you are looking at the wrong direction? Try checking your +s and -s in calculations. 3) If the teapot seems (or moves) wobbly, you may either be making a mistake in your calculation or some mix up with your vectors.

Oh by the way, the moment pressing "g" does not seem to work is the moment your lookat function started working... :)

Finally maybe a checklist may be helpful at the very end before you submit:

1. Does your teapot turn left when you press left, and turn downwards when you press up (you know its actually the camera moving in the other direction but it is easier to state it this way)? If so, your directions are correct for the grader specs.
2. Do you see no difference (except some message lines popping up in your terminal window) when you press "g" on your keyboard? If so, you successfully implemented your lookat function.
3. Does the teapot stay at a fixed distance no matter how you turn it?
4. When you mix left and up movements randomly and do full 360 degree turns, does the teapot turn smoothly, without making sudden 180 degree jumps (no sudden upside down turns or switching the handle and spout)? If it does, you better check what you do with your eye and up vectors in the code.
5. This, I am just guessing, but if pressing "i" does not start an automatic routine to take images of some pre-determined rotations, you will probably have to check if the provided "input.txt" file is supplied as a command line argument to your code. VS users will probably never have this problem as it is set in the skeleton code of the project but may still go through the menu system to see it there; other systems will have type them in their command line running the code.

I hope this helps. Other recommendations are welcome (as long as they do not interfere with the honor code). And apologies for the more enlightened coders if this material seems too long and too simple and not particularly useful, it's just the stuff I wish I knew (as a noob) before starting this homework...