1 (Practice 2) affine transformations

1.1 Warming up

1) Let the 3D point P [x, y, z] with three components expressed with Cartesian coordinates. How many components has a 3D point expressed with Homogeneous coordinates?

2) What kind of transformations expressed with matrices are impossible to realize in Cartesian coordinates?

3) Give the expression of a homogeneous matrix which scale uniformly about a factor 's' a 3D homogeneous point. Give the homogeneous matrix which translate about a vector $[t_x, t_y, t_z]$

4) In the course we showed how to find the rotation matrix of a 2D point (with the parametric equation of a circle). The matrix is $R=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$. Find the homogeneous rotation matrix for the rotation of a 3D point around the axis 'z' and for an angle of magnitude θ.

5) What is the value of the fourth homogeneous coordinate of a 3D vector? A 3D point?

6) Write and calculate the translation about $[t_x, t_y, t_z]$ of a vector $[v_x, v_z, v_y]$ with a homogeneous matrix.

7) Let T be the homogeneous matrix for the translation $\vec{t}=[t_x, t_y, t_z]$, and S the uniform scaling matrix of factor 's'. Compute the product of $M_1=T.S$ and then $M_2=S.T$.
   1. Let P be a 3D point. If we compute $M_1.P$ in which order the transformations are applied?
   2. From now on we use the values $s=2$ and $\vec{t}=[2,0,4]$.
   3. Transform the Point P(2 6 0) with the matrix M_1.
   4. Compute the inverse transformation of M_1 by inverting it. (Hint: compute the inverse of a matrix can be done by solving the system $M^{-1}.M=I$)
   5. Check the result by transforming back $P'=M_1.P$ with M_1^{-1}.

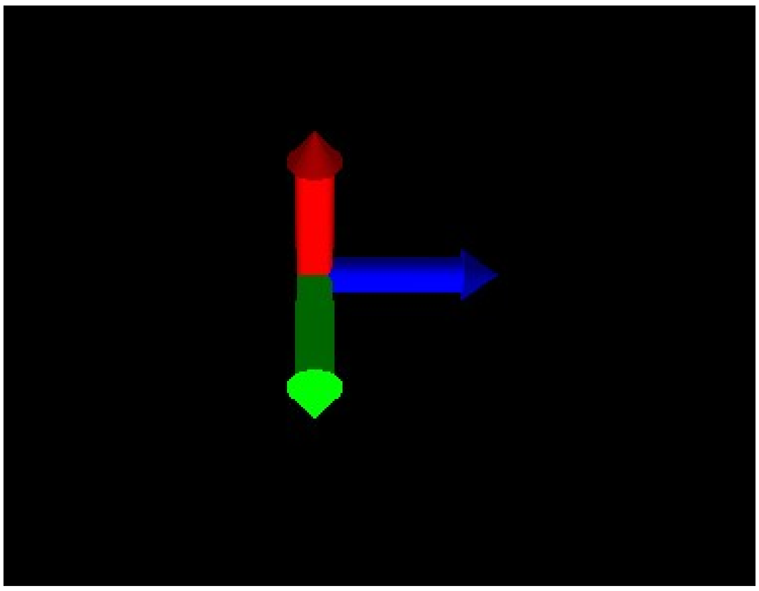
1.2 OpenGL training

Before doing anything download the materials associated to this practice compile and run the code. You'll work on this new code basis.

In 'main.c' double buffering, depth test, and light0 are already activated. Also the camera is already positioned, and the perspective characteristics setup.

1) Recall the meaning of the parameters of 'gluPersepective()' and 'gluLookAt()'
2) Try to sketch the position of the camera/cube/light in order to get more familiar with the code.

- We are going to draw a frame which should look like this:



Arrows will be made out of a cone and a cylindre. For the cone you can use 'glutSolidCone(diam, length, resx, resy)'. As for the cylinder GLUT doesn't provide a drawing function, we will use the GLU instead. I'll save you the trouble of figuring out how to use it:

```
void drawCylindre(float diam, float length)
{
    GLUquadricObj* obj = gluNewQuadric();
    gluCylinder(obj, diam, diam, length, 50, 50);
    gluDeleteQuadric(obj);
}
```

(Note that the function is already in your 'main.c')

- Write a function 'void drawFrame(float diam, float length)' which draws a frame as shown above. You'll have to use glPushMatrix(), glPopMatrix(), glTranslatef(), glRotatef(), glColor3f().
- Call drawFrame() where it is appropriate in order to draw it

- Now we will animate this frame. Use a global variable '_angle' and setup a call-back function when no events occurs which will increment angle between [0 360]. Use '_angle' to rotate the frame around some axis (for instance [1. 1. 1.]).

N.B: Set the callback with glutIdleFunc().

- We are going to handle the transformations by ourselves. Create two file 'mat4.h' and 'mat4.c' and implement the function signatures:

```
typedef GLdouble* Mat4;
Mat4 newMat4();
void deleteMat4(Mat4 m);

Mat4 mat4_identity();

Mat4 mat4_translation(double x, double y, double z);
```

```
Mat4 mat4_rotation(double angle, double x, double y, double z);
```

Because OpenGL handles by default column-major matrices you'll have to store in memory the elements as below:

```
m[0] = ; m[4] = ; m[ 8] = ; m[12] = ;
m[1] = ; m[5] = ; m[ 9] = ; m[13] = ;
m[2] = ; m[6] = ; m[10] = ; m[14] = ;
m[3] = ; m[7] = ; m[11] = ; m[15] = ;
```

You can find in the course the expression of the matrices in the chapter transformations.

- Once implemented you can use Mat4 to rotate the frame with a 'glMultMatrixd();' instead of the 'glRotatef()'.

- Stop animating

- Try to translate and rotate with two 'glMultMatrixd()'

- Implement `void mat4_mult(Mat4 m0, Maty4 m1, Mat4 res)`. Rotate and translate with the same matrix.

- Implement `void mat4_reflection(double nx, double, ny, double nz)`. Draw the frame twice with the second occurrence reflected along the plane of equation $f(P)=\vec{N}.\vec{P}-\vec{N}.\vec{O}=0$ With $\vec{N}=[1;1;-1]$ the normal to the plane and $\vec{O}=[0;2;0]$ a point lying on the plane.

Since the matrix of reflection is given for a plane passing through the origin you'll have to translate then reflect and translate back to make the plane pass through $\vec{O}$.

Otherwise said: you have to compute $M=T^{-1}.R.T$ multiply the current OpenGl matrix with M and draw the second occurrence of the frame.

- Re-enable the animation in order to rotate both frame.