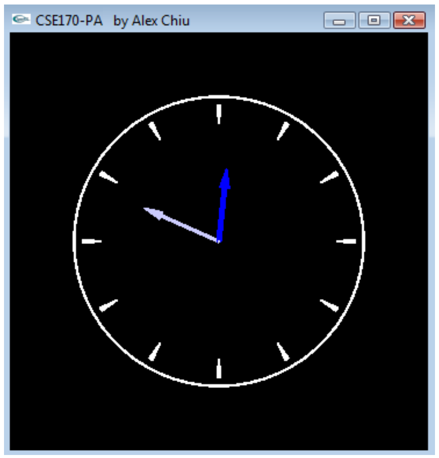
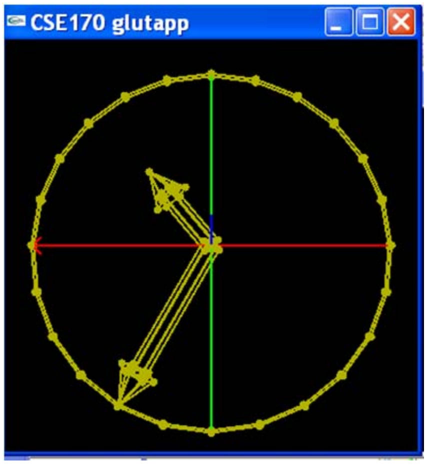
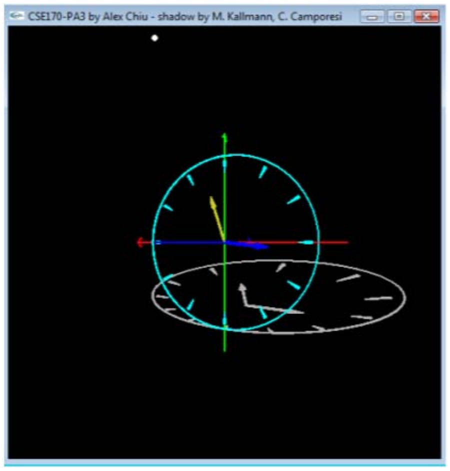
Programing Assignment #3

In this assignment you will implement and animate a stopwatch using transformations. Start this assignment from the glutapp3d support code. Below are some examples of how your stopwatch could look like:






CSE170-PA3 by Alex Chiu - shadow by M. Kallmann, C. Camporesi

Requirement 1 (30%) - Draw a stopwatch using your capsule primitive from PA2

a) Stopwatch (10%): the stopwatch itself can be a simple circle, or a thin cylinder centered at the origin and facing the Z direction.

   You may draw your stopwatch in any way. You can use several thin concatenated capsules to form the outer circular boundary of the stopwatch. You may add extra parameters to your capsule object in order to tell if the top and bottom spherical caps should be generated or not, in that way you can easily achieve cylinders and cones. A cone will be a tube having the radius at one extremity to be zero. You may also create additional classes SoTube and/or SoCone to implement your primitive objects if you would like to. You may choose how to customize your objects; however, you have to write your own code for each primitive generation that you create.

b) Hands (20%): the stopwatch will have two hands. Make sure that one hand is shorter than the other hand. You may use colors as you wish. The images above show two parameterized tubes being used for each hand: one for the main section, and another one to make an arrow at the end of the hand (doing this is not required).

Requirement 2 (30%) - Stopwatch animation

a) Hands animation (20%): one hand will rotate 6 degrees every second, such that a full turn will happen in 1 minute. The other hand will rotate 6 degrees every 1/60 of a second, such that a full turn will happen in 1 second. The speed of the animations must be controlled by a time function such that the animations remain correct regardless of the computer spped. GLUT has a time function that you can use and another one is provided below in case you'd like to test/compare both. You will need to call it from your "idle method" in order to check the elapsed time at every iteration, and then correctly update the hands of the stopwatch when needed. You should be able to obtain a precise stopwatch.

b) Stopwatch control (10%): Use the **space bar** to turn on and off the stopwatch, and use the **enter key** to restore the hands to the zero position. You may use another key to control the on/off display of the axis.

Requirement 3 (40%) - Shadow Animation and Control

a) Shadow Computation (20%): you will now use a projective transformation to compute a shadow for your stopwatch. Consider that your stopwatch is transparent. Now imagine that it is standing on a table and you will be moving a flashlight in front of it such that the shadow of the contour of the stopwatch and its hands appear on the surface of the table. In your project the surface of the table is the plane parallel to the XZ plane and passing by the bottom-most vertex of the stopwatch.

   You will simulate a shadow by drawing a second time the entire stopwatch after transforming its coordinates by a transformation matrix that projects the vertices to the shadow plane. You will do this at every iteration such that we can see the shadow moving together with the hands of the stopwatch. Use **key '/'** to turn on and off the display of the shadow. Read the sections of the book and of L04-mathnotes.pdf that cover projection transformations. There are different ways to achieve a correct projection. You will typically need to combine a projection transformation with translations in order to correctly specify the local frame expected by the projection operation.

   For example, the right-most image at the beginning of this document shows a shadow implemented using the projection matrix given in the L04-mathnotes.pdf document. It also shows a point representing the "light source" position – this is very useful for debugging your code.

   Be aware that your chances of just applying the transformation and having a correct shadow right away are very small. You need to understand the parameters, the considered frame of reference, etc. Strange results will appear if a parameter is not correctly encoded. These are difficult issues to debug and the best way to proceed is to test your projection step by step, for example by first working with only a few points.

   The recommended implementation is to call your stopwatch drawing function two times, one for the stopwatch itself, and another one for its shadow.

   For example, like in the following way:

```
// Encode your stopwatch object in its own class:
...
SoStopwatch _sw; // reminder: "So" classes represent "Scene Objects"
...

// At the right time, draw the object normally:
_sw.draw ( UsualSceneTransformation, UsualSceneProjection );
...
```

```
// Then compute the shadow projection matrix. For ex. you may pass
// a GsMat ShadowProjMatrix by reference to a function to compute it:
Compute_shadow_transformation ( parameters, ShadowProjMatrix );
...
// Now draw the object again, but projected as a shadow:
_sw.draw ( ShadowProjMatrix*UsualSceneTransformation, UsualSceneProjection );
```

b) Shadow animation (10%): by pressing **keys** ‘q’, ‘a’, ‘w’, ‘s’, ‘e’, ‘d’, update the position of the center of projection (the “position of the flashlight”) along the X, Y and Z coordinates, and have the shadow be re-computed with respect to it. Have these keys generate correct projections while the hands move. These keys will also be useful to test and debug your projection matrix.

c) Overall result (10%): make sure that your shadow looks good and the obtained results are correct in all cases.

Notes:

- As we have not seen shaded objects yet your stopwatch will be only in wireframe. It is ok to include shading if you already know how to do it, however you must use your own implementation to draw everything that appears on the screen.
- You may use a gray color for the shadow lines, but any other color is fine.
- The TA will ask you to explain how your projection matrix works, so be sure you know what you are doing!
- <u>For some students this may be the most complex PA to get a 100%, so start early!</u>
- Below is a simple time function that you can use:

```
#ifdef WIN32
#include <sys/types.h>
#include <sys/timeb.h>
#else
#include <sys/time.h>
#endif

double get_time ()
{
  #ifdef WIN32
   // if better precision is needed in Windows, use QueryPerformanceCounter
   _timeb tp;
   _ftime_s(&tp);
   return 0.001*(double)tp.millitm + (double)tp.time;
  #else
   timeval tp;
   if ( gettimeofday(&tp,0)==-1 ) return 0;
   return 0.000001*(double)tp.tv_usec + (double)tp.tv_sec;
  #endif
}
```

- The GLUT function that you can also use is `glutGet(GLUT_ELAPSED_TIME)`, it returns the number of milliseconds since `glutInit` was called (or first call to `glutGet(GLUT_ELAPSED_TIME)`).