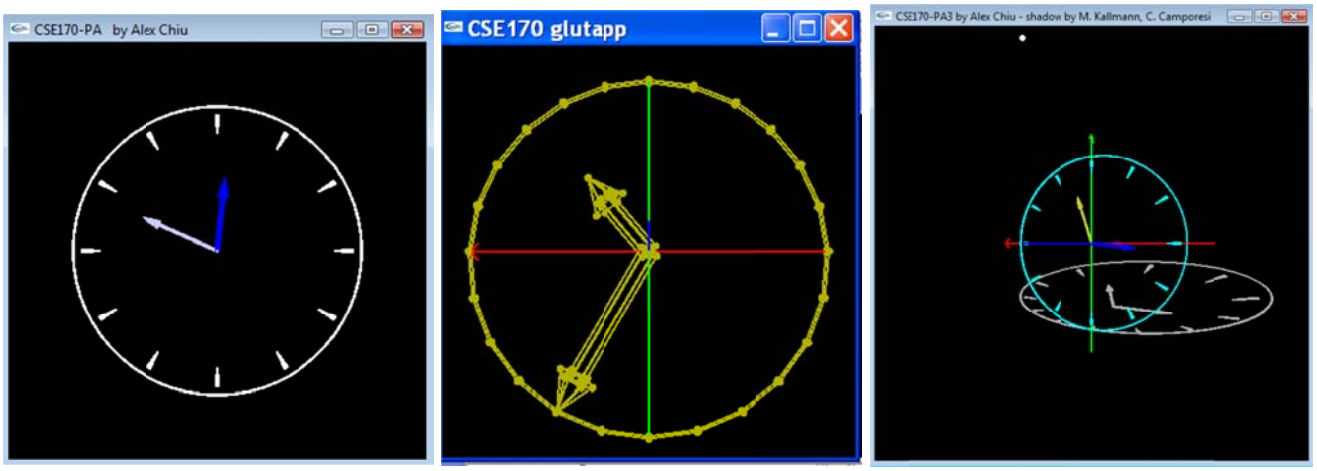
Programing Assignment #2

In this assignment you will implement and animate a stopwatch using transformations correctly placed to affect primitive objects in a scene graph.

You may start this assignment from the new example project **sigapp.7z**, which demonstrates how to perform some operations with SIG's scene graph, and gives one example of a time-controlled animation loop. If you prefer, you may also start from your previous sigmynode project, and just use sigapp to learn about the additional operations you will have to implement.

Below are some examples of how your stopwatch could look like. You will be able to take advantage of SIG's primitives in order to build interesting results with shaded primitives. (The examples below were actually made with "glut" and not SIG.)



Requirement 1 (30%) - Draw a stopwatch scene using primitive objects

a) The stopwatch itself can be a thin cylinder centered at the origin and facing the Z direction. You may as well use your torus scene node to make the outer contour. You may draw your stopwatch in any way, it just has to look correct.

b) Hands: 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 shape at the end of the hand.

   Study the provided information (lecture notes, sigapp.7z, etc.) to understand how to add primitives to a scene, and how to compose several primitives with transformations in order to build your stopwatch. Key to this assignment is the correct use and placement of SnTransform nodes in order to apply the correct rotations to the hands of the clock. As discussed in class the recommended approach is to keep pointers to the SnTransforms that you will need to update during your animation.

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 speed. SIG has a time function gs_time() that you can use. You will need to call it from your own animation loop in order to control your animated scene. An example of how to control an animation loop is given in the example project **sigapp.7z**. 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.

Requirement 3 (30%) - Shadow Animation and Control

a) Shadow Computation (15%): 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 original vertices to the shadow plane. You will do this at every iteration of your animation such that we can see the shadow moving together with the hands of the stopwatch.

   Start by studying the projection matrix discussed in class. 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. The right-most image at the beginning of this document shows a correct shadow result as an example. It also shows a point representing the “center of projection” position – this is very useful for debugging your code.

   There two main approaches to accomplish your shadow: 1) you may multiply yourself your projection transformation to every point of your objects, and place the result in a second clock object in the scene which will become the shadow, or 2) you may add your shadow transformation to the scene graph directly and let it affect a copy (or shared) version of the clock model coming next in the scene graph. In all cases you will need to well understand the different ways to use SIG’s scene graph.

b) Shadow animation (15%): by pressing **keys** ‘q’, ‘a’, ‘w’, ‘s’, ‘e’, ‘d’, update the position of the center of projection (the “position of the simulated flashlight generating the shadow”) 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.

Requirement 4 (10%): Overall quality.

Everything counts here and, once again, there is no need to do anything complex, just make sure your project looks well developed and you will get full points here!

Submission:

Please present and submit your PA as usual according to parules.txt. (Do not forget to upload your project before the deadline!)

Extra:

To give you some motivation, see below the most complex fully-animated mechanical clock built in our previous classes (by MS student Yusuke Niiro, Fall 2017):