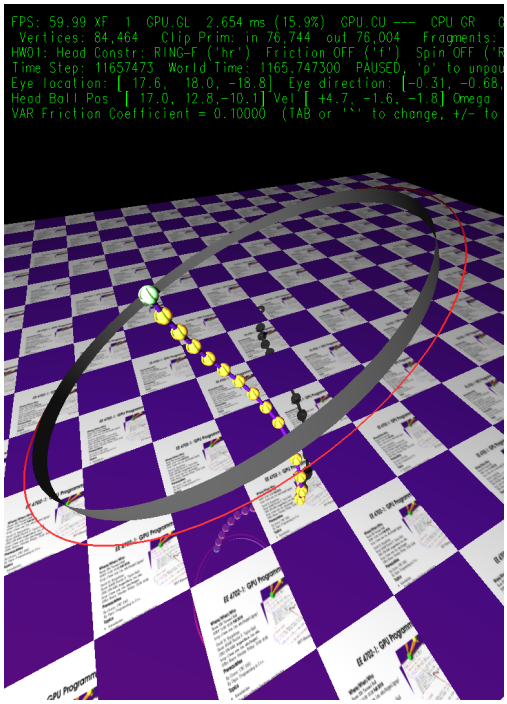
LSU EE 4702-1 **Homework 1** **Due: 12 September 2019**

Problem 0: Follow the instruction on the `https://www.ece.lsu.edu/koppel/gpup/proc.html` page for account setup and programming homework work flow. Compile and run the homework code unmodified. It should initially show a string of beads hanging from a ring. The ring is shown as a cylinder surrounded by a red circle. The red circle is rendered using code written in class (with a few changes), see the code in file `hw01-graphics.cc` near the comment `Live Classroom Demo`.



Assignment-Specific Model and User Interface

simulation starts the head (first, zero) ball is attached to the ring and can slide freely along the ring. Pressing `r` (lower-case) will toggle between the head ball sliding freely and the head ball rotating along the ring at a fixed rate. The green text line starting with HW01: shows the state of the ball's attachment to the ring and of the ring itself. On this line the text to the right of Head Contr (head ball constraint) is RING-F if the ball is sliding freely, RING-A (animated) if the ball is rotating with the ring, FREE if the ball is not attached to anything, and LOCKED if the ball is fixed in space (does not change its position). The last two states can be obtained by pressing `h` (head). (Press `h` and `r` to familiarize yourself with the behavior, and be sure to read the next Common User Interface section which describes other available UI features, such as moving the viewer and pausing simulation.)

When in mode RING-A the rotation rate is determined by variable VAR Rotation rate (variable `hw01.omega` in the code). See the Common User Interface section below to see how to change that and other variables.

For Problem 1 ring friction will be added to the simulation model. Pressing `f` toggles friction on and off (variable `opt_hw01_do_friction`, the state is shown to the right of Friction and by the color of the ring. When friction is turned on the ring color is yellowish, like sandpaper. This will have no effect until Problem 1 is properly solved. VAR Friction Coefficient is the coefficient of friction, variable `opt_hw01_fric_coefficient` in the code.

For Problem 2 the ring will be made to spin. When properly solved pressing `R` should turn this feature on and off, (variable `opt_hw01_spin`).

Common User Interface

release one end (to be precise, the ball at one end) and pressing `t` (tail) will grab or release the other end. (Actually, those keys toggle between the `OC_Locked` and `OC_Free` constraint of their respective balls.)

Press digits `1` through `4` to initialize different scenes, the program starts with scene 1. Scene 1 starts with the balls arranged almost vertically. Scene 2 starts with the balls arranged horizontally,

and they will start swinging. Each time scene 1 and 2 are initialized the ring is positioned to a new position. In scenes 3 and 4 the ring is always positioned the same way.

Press `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size. Initially the arrow keys, PageUp, and PageDown can be used to move around the scene. Press (lower-case) `b` and then use the arrow and page keys to move the tail ball around. Press `l` to move the light around and `e` to move the eye (which is what the arrow keys do when the program starts).

The `+` and `-` keys can be used to change the value of certain variables to change things like the light intensity, spring constant, and variables needed for this assignment. The variable currently affected by the `+` and `-` keys is shown in the bottom line of green text. Pressing `Tab` cycles through the different variables.

Look at the comments in the file `hw01.cc` for documentation on other keys.

Code Describing the Ring

The code in this assignment is based on `demo-2-springs.cc`, which shows balls connected by links. In the underlying physical model balls are modeled as point masses and links are modeled as ideal springs. The same is true for this assignment. In `demo-2-springs.cc` the head and tail balls could either be locked or free. When a ball is locked its position doesn't change. That's simulated by simply setting a locked ball's velocity to zero. Free balls are simulated based on the ideal-mass-and-springs model.

In this assignment a ball can be under one of four possible constraints, Free, Locked, Ring-Animated, and Ring-Free. The constraint imposed is in member `ball->constraint`. The Free and Locked constraints are carried over from `demo-2-springs.cc`. The position and velocity of a ball under the Ring-Animated constraint is determined solely by the rotating ring and so the forces on such a ball have no effect. A ball under the Ring-Free constraint can move freely along the circumference of a ring, its motion is determined by forces tangent to the ring at its location.

Variables describing the ring and some other features in this assignment are in `struct HW01_Stuff` which is declared in the `World` class as `HW01_Stuff hw01`. In this discussion they will be referred to using the member name, `hw01`. The coordinate of the center of the ring is in `hw01.center`, the ring axis (normal) is in `hw01.axis`, and its radius is in `hw01.radius`. These three variables define the ring. Additional variables are provided for convenience and to describe the ball's position on the ring. The local x- and y-axis of the ring are in `hw01.x` and `hw01.y`.

The location of the ball on the ring (in radians) is specified by `hw01.theta`. The rotation rate (in radians per second) around the ring is specified by `hw01.omega`. The following code computes the position of the ball described by these variables:

```
pCoor pos =
  hw01.center +
  hw01.radius * ( cosf(hw01.theta) * hw01.x + sinf(hw01.theta) * hw01.y );
```

In `time_step_cpu` lambda function `pos_compute` computes this quantity.

Initialization of Ring Position

The variables describing the ring are initialized by code in `time_step_cpu` below the comment `Initialize Ring`. The initialization is performed whenever variable `hw01.rail_inited` is false. It is set to `false` when a scene is initialized, and when the head is released and then re-ringed. (Press `h` then `r`.)

The ring position is initialized in three possible ways. In all cases the ring is positioned so that the ring passes through the head ball. If the ball is not moving then the ring axis is set to either a random direction (scenes 1 and 2) or a predetermined direction (scenes 3 and 4). If the ball is moving, the local y axis, `hw01.y`, of the ring is set to the ball's direction of motion and the local x axis is chosen to be orthogonal to the global y axis. A passenger on the ball might feel that the

ball was smoothly captured by the ring. The rotation rate, `hw01.omega` is set to 1 if the ball is under constraint Ring Animated, otherwise it is set to zero.

Here is a condensed version of the code:

```
pNorm vdir(hball.velocity);
hw01.theta = 0;

if ( vdir.magnitude < 0.00001 ) {
    switch ( curr_setup ) {
    case 1: case 2:  // Random angle.
      hw01.axis = pNorm(.9*drand48(),1,.9*drand48()); break;
    case 3: // Almost straight up.
      hw01.axis = pNorm(0.2,1,-0.2); break;
    case 4: // Orthogonal to direction of chain.
      hw01.axis =
        cross( balls[chain_length-1].position - balls[0].position, pVect(0,1,0) );
      break;}

    hw01.x = hw01.axis.z != 0 && hw01.x != 0
      ? pNorm(hw01.axis.z,0,-hw01.axis.x) : pNorm(1,0,0);
    hw01.center = hball.position - hw01.x * hw01.radius;
    hw01.y = cross(hw01.axis,hw01.x);
    hw01.omega = hball.constraint == OC_Ring_Animated ? 1 : 0;
  } else {
    hw01.y = vdir;
    hw01.x = pNorm( cross( hw01.y, pVect(0,-1,0) ) );
    hw01.axis = cross(hw01.x,hw01.y);
    hw01.center = hball.position - hw01.radius * hw01.x;
    hw01.omega = vdir.magnitude / hw01.radius; // Match ball speed.
  }
```

The Time Step Routine and Ring Simulation

The time step routine has three main parts. In the first part the force on each ball is determined and written to `ball->force`. The second part, described above, initializes the ring if that needs to be done. The third part updates the velocity and position of each ball. The method used to update these is determined by `ball->constraint`. The code for the Free and Locked cases is similar to that in `demo-2-springs`, though the placement of the code is different.

The code for Ring-Animated computes the position and velocity based on the fixed motion. Here is a condensed version:

```
case OC_Ring_Animated:
    // Ball is attached to ring and rotates with it at fixed rate omega.

    // Compute amount by which ring will rotate during this time step.
    const double delta_theta = delta_t * hw01.omega;

    // Update velocity vector.
    ball->velocity = hw01.omega * hw01.radius *
      ( -sinf(hw01.theta) * hw01.x + cosf(hw01.theta) * hw01.y );
```

```
    // Update the position angle and the position Cartesian coordinate.
    hw01.theta += delta_theta;
    ball->position = pos_compute(hw01.theta);
  break;
```

The code for the Ring-Free constraint uses `ball->force`, but not all of it. Because the ball is constrained to move along the ring the only force that matters is force in the direction along the ring (the tangent). The code starts by computing the tangent, `dtan`, and using that to find the amount of force in that direction, `force_tan`:

```
pVect center_to_ball = pVect(hw01.center,ball->position) / hw01.radius;
pVect dtan = cross(hw01.axis,center_to_ball);
const float force_tan = dot( ball->force, dtan );
```

Force `force_tan` is then used to compute the change in velocity, `delta_v` and from that a change in rotation rate, `delta_omega`. Then `hw01.omega` is updated and used to update the velocity, and `hw01.theta` is updated and used to update the position:

```
const float delta_v = force_tan / ball->mass * delta_t;
const float delta_omega = delta_v / hw01.radius;
hw01.omega += delta_omega;
hw01.theta += hw01.omega * delta_t;
ball->velocity = hw01.omega * hw01.radius * dtan;
ball->position = pos_compute(hw01.theta);
```

The alert student might suspect that `ball->velocity` isn't needed for the head ball attached to the ring. True, it's not needed to determine the position, but it is used when determining whether the spring is gaining or loosing energy, and it is needed when the ball's constraint is changed of Free.

Problem 1: In the unmodified assignment code the model does not account for friction on the ring. In this problem account for such friction when the ball is sliding along the ring and variable `opt_hw01_do_friction` is `true`. The code to be modified is in routine `time_step_cpu` near the comment `Put Problem 1`.

As you may remember from physics, the amount of dynamic frictional force on an object sliding over a surface is determined by the force of the object on the surface and a coefficient of friction. Speed of motion is irrelevant.

Recall that the forces on the ball are accumulated in `ball->force`. As described in Problem 0, the force along the tangent accelerates the ball, and so this component of force does not contribute to friction.

Let f_t denote the magnitude of the ball's force along the tangent (`force_tan` in the code) and let f_r denote the magnitude of the ball's force on the ring (something that must be solved for this problem). Let k denote the friction coefficient, (`opt_hw01_fric_coefficient`). Then the magnitude of frictional force will be kf_r and the direction is in the opposite direction of the ball's motion.

As with ordinary force, the frictional force will result in a change in rotation rate, $\Delta\omega = \frac{kf_r\Delta t}{mr}$, where m is the ball mass and r is the ring radius, with an important caveat: the interval Δt can not be so long as to result in a change in direction. Instead, ω should be just set to zero.

So, to solve the problem first compute f_r and with that $\Delta\omega = \frac{kf_r\Delta t}{mr}$. Computing f_r is straightforward. Of course, I don't need to remind anyone that $f_r \neq f - f_t$, where f is the total force on the ball.

Next, figure out whether to add or subtract $\Delta\omega$ from `hw01.omega`. If `hw01.omega` changes sign, set it to zero.

If this is solved correctly the chain should come to a stop before reaching the lowest point on the ring, unless the ring is steeply titled or the chain is bouncing around.

Problem 2: When variable `opt_hw01_spin` is true and the ring is visible, spin the ring around an axis parallel to the xz plane. Press `R` to toggle the state of `opt_hw01_spin`. The ring should spin at rate `hw01.opt_spin_omega`.

Put most of the solution for this at the end of `time_step_cpu`, search for `Put Problem 2` in the code. An `if` statement with an appropriate condition has been prepared.

To solve this problem, first compute a rotation axis and matrix. This only needs to be done once each time a new ring is chosen. Put the rotation matrix and any other variables that are initialized once used each time step in `HW01_Stuff`. The code below shows how to construct a transformation matrix that rotates around axis `spin_axis` by `theta` radians, and to use it to rotate a vector. The rotated vector is re-normalized to correct small errors that change vector length.

```
pMatrix xform_spin; // Put this in HW01_Stuff.
hw01.xform_spin.set_rotation( spin_axis, theta );
pNorm before(2,1,2);  // Some unit vector.
pVect after = pNorm( hw01.xform_spin * before );
```

Then use the rotation matrix each time step to rotate the appropriate variables describing the ring. The default spin rate, in `hw01.opt_spin_omega`, is 1 radian per second, and so the ring should make a complete rotation in $2\pi \approx 6.283$ seconds.

Take care not to re-compute the rotation matrix unnecessarily.