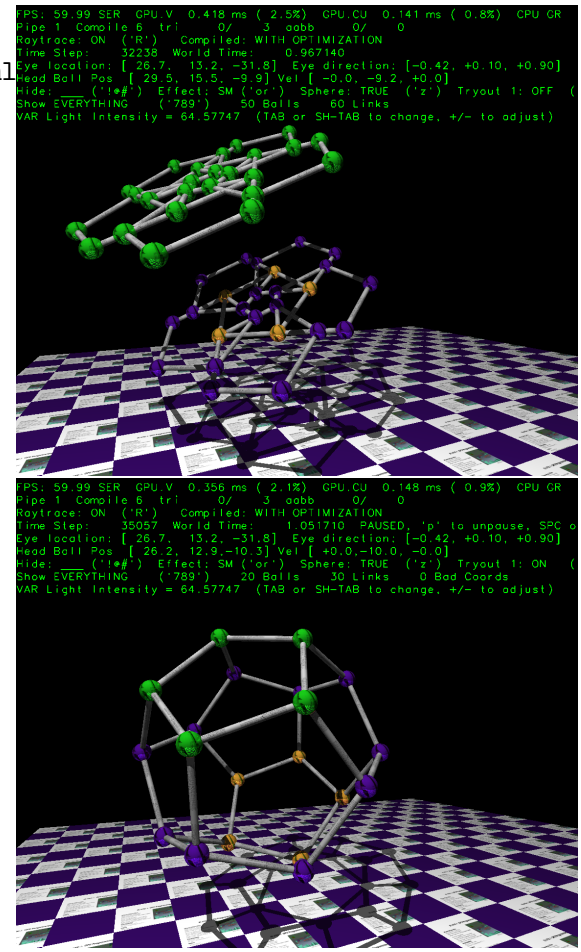
LSU EE 4702-1 **Homework 2** **Due: 7 October 2025**

All of the code for this assignment is in the course repo. HTMLized versions of the assignment file are at `https://www.ece.lsu.edu/koppel/gpup/2025/hw02.cc.html`.



Problem 0: Follow the instructions 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 jumble of balls connected by links, see the screenshot to the upper right. The jumble is constructed by the code in routine `balls_setup_7` in the unmodified code. The routine is supposed to use transformation matrices to compute the coordinates of a regular dodecahedron, but those transformation matrices are just placeholders. In this assignment those placeholders are to be replaced by correct transformations. The screenshot to the lower right is from a correctly solved assignment.

User Interface

The code can display several scenes, numbered 0 through 9. The currently displayed scene is identified in the penultimate line of green text. For this assignment use scenes 7-9. Press `0` to select scene 0, `1` for scene 1, etc. Scene 3 shows a crude corona virus particle, Scene 4 shows a wheel, scene 5 a top (as in a spinning child's toy), scene 6 is a very crude parachute, and scene 7-9 are for this assignment. The code for scenes $0 \leq x \leq 7$ is in routine `World::ball_setup_`x. The code for scenes 8 and 9 are also in `World::ball_setup_7`.

Press `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size. Press `F12` to generate a screenshot. The screenshot will be written to file `hw02.png` or `hw02-debug.png`. Press `F10` to start recording a video, and press `F10` to stop it. The video will be in file `hw02-1.webm` or `hw02-debug-1.webm`.

Initially the arrow keys, `PageUp`, and `PageDown`, can be used to move around the scene. Using the `Shift` modifier when pressing one of these keys increases the amount of motion, using the `Ctrl` modifier reduces the amount of motion. Use `Home` and `End` to rotate the eye up and down, use `Insert` and `Delete` to rotate the eye to the sides.

After pressing `l` the motion keys will move the light instead of the eye, after pressing `b` the motion keys will move the head ball around, and after pressing `e` the motion keys operate on the eye.

The simulation can be paused and resumed by pressing `p` or the space bar. Pressing the space bar while paused will advance the simulation by 1/30 s. Gravity can be toggled on and off by pressing `g`.

The + and - keys can be used to change the value of certain variables. These variables specify things such as the gravitational acceleration, dynamic friction, and variables that may be needed for this assignment.

The variable currently affected by the + and - keys is shown in the bottom line of green text. Pressing Tab and Shift-Tab cycles through the different variables. To locate variables which can be set this way, and to see how they were set, search for `variable_control.insert` in the assignment file.

Assignment-Specific User Interface

For this assignment use scenes 7-9. Press 7 to show scene 7, etc. Scene 7 shows the entire dodecahedron (or what's supposed to be a dodecahedron). Scene 8 shows only the initial pentagon from which the remainder of the dodecahedron is to be constructed. Scene 9 shows half of the dodecahedron. The penultimate line of green text shows which of the assignment scenes is active.

The penultimate line also shows the number of Balls, Links, and Bad Coords. For scenes 7-9, the number of balls is the number of vertices. In a correct solution Scene 7 should show 20 balls (vertices). Links shows the number of edges, which should be 30 in a correct solution. Bad Coords show the number of coordinates in array `coors` which are not at the expected distance from the dodecahedron center when the dodecahedron is constructed (but not later). Because `coors` contains duplicated coordinates, the number near Bad Coords can be larger than Balls.

The code for scenes 7, 8, and 9 are all in `World::ball_setup_7`. Each time scenes 7, 8, and 9, are started the position, size, and orientation of the dodecahedron will be slightly different.

Code Generation and Debug Support

The compiler generates an optimized version of the code, `hw02`, and a debug-able version of the code, `hw02-debug`. The `hw02-debug` version is compiled with optimization turned off, which makes it easier to debug. When needed, you are strongly encouraged to run `hw02-debug` under the GNU debugger, `gdb`. See the material under "Running and Debugging the Assignment" on the course procedures page. **You must learn how to debug.** If not, you will be at a severe disadvantage *for the rest of your life, if that's how long you stubbornly resist learning!*

To help you debug your code and experiment in one way or another, the user interface lets you change *tryout* variables. There are two Boolean tryout variables, `opt_tryout1` and `opt_tryout2`, and one floating-point tryout variable `opt_tryoutf`. You can use these variables in your code (for example, `if ( opt_tryout1 ) { x += opt_tryoutf; }`) to help debug, to help familiarize yourself with how the code works, or to experiment with new ideas. Keys `y` and `Y` to toggle the values of the Boolean variables; their values are shown in the green text at the label `Tryout 1:` and `Tryout 2:`. The user interface can also be used to modify host floating-point variable `opt_tryoutf` using the `Tab`, `+`, and `-` keys, see the previous section.

Regular Dodecahedron

A *regular dodecahedron* is constructed from 12 regular pentagons of edge length a by joining the pentagons at their edges so that each vertex is on the surface of a sphere, called the *circumscribed sphere*, of radius $r_c = a\phi\sqrt{3}/2$. There are a total of 20 vertices and 30 edges. The angle between two adjacent pentagons, called the *dihedral angle*, is $2\arctan(\phi)$ radians, where ϕ is the Golden Ratio, $\phi = \frac{1+\sqrt{5}}{2}$. The largest sphere that can fit inside a dodecahedron is called the *inscribed sphere*, its radius is $r_i = a\sqrt{\frac{\sqrt{5}\phi^5}{20}}$.

The code in `ball_setup_7` computes a random location, size, and orientation for a dodecahedron and computes useful information about it. Variable `center_pentagon_0` is the coordinates of the center of the first pentagon. Unit vectors `ax`, `ay`, and `az` are used to compute the pentagon coordinates. Comments in the code describe the other variables.

For this assignment a dodecahedron will be constructed by placing coordinates in vector `coors`. The first five elements of `coors` are the coordinates of a pentagon. Balls corresponding to these coordinates are gold. In the unsolved assignment these coordinate values are correct, but they are computed parametrically (shown below), in Problem 1 they are to be computed using transformations.

```
const float d_theta = 2 * numbers::pi / n_sides;
for ( int i=0; i<5; i++ )
  {
    const float theta = d_theta * i;
    pVect dir = cosf(theta) * ax + sinf(theta) * ay;
    pCoor pos = center_pentagon_0 + rc_pentagon * dir;
    coors << pos;
  }
```

Notice that coordinates are added to the `coors` vector using the `<<` operator. This is equivalent to `coors.push_back(pos)`.

After the first pentagon, the code adds five more pentagons, one for each edge of the original pentagon. The coordinates of the new pentagon are found by applying transformation `m` to the coordinates of the first pentagon:

```
for ( int i=0; i<5; i++ )
  {
    pCoor p0 [[maybe_unused]] = coors[i];
    pCoor p1 [[maybe_unused]] = coors[(i+1)%5];

    pMatrix_Translate tr( pVect(p0,p1) ); // Wrong.
    pMatrix m = tr;

    for ( int j=0; j<5; j++ ) coors << m * coors[j];
  }
```

Balls for these new coordinates are shown in purple (unless they are coordinates of the first pentagon). Each new pentagon shares edge `p0`,`p1` with the first pentagon but is rotated at angle θ around the axis formed by `p0` and `p1`. The transformation `m` above is supposed to do that, but it doesn't, it just translates the coordinates. This is to be fixed in Problem 2.

At this point in the code `coors` describes six pentagons, the first pentagon and five more. To complete the dodecahedron six more pentagons need to be added. That can be done by rotating the existing points in `coors` around the center of the dodecahedron, in variable `center`. That is done by the code:

```
pMatrix_Translate trd( 5 * az );
pMatrix ms = trd;
for ( int i=0; i<n_coord_demi; i++ ) coors << ms * coors[i];
```

Of course, `ms` is not the correct transformation. In Problem 3 a correct transformation is to be computed.

The coordinates in `coors` are then used to initialize positions of balls and links. This code does not need to be modified for this assignment. The first step in doing so is to identify duplicated coordinates. In a correctly constructed dodecahedron each coordinate will appear three times. The code considers two coordinates to be identical if their distance is less than 10^{-4} units. A ball is constructed for each unique coordinate. A link is constructed for each unique edge. Ball ids are

used to construct a key for a link, and a stdlib map is used to check whether a link has already been constructed.

Resources

A good reference for C++ is `https://en.cppreference.com/w/`. A past homework assignment that may be helpful is 2018 Homework 1. In particular look at the alternative solution to this assignment.

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of C++ syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out programming and graphics resources. It is okay to make use of AI LLM tools such as ChatGPT and Copilot to generate sample code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Student Expectations

To solve this assignment students are expected to avail themselves of references provided in class and on the Web site, such as for C++ programming and to seek out any additional help and resources that might be needed. (Of course this doesn't mean asking someone else to solve it for you.) Students are expected to experiment to learn how graphics work, and how to code C++ sequences. Experimentation might be done on past homework assignments. Students are also expected to learn what error messages mean by consulting documentation and by asking others (including Dr. Koppelman), and also to develop debugging skills. It is the students' responsibility to resolve frustrations and roadblocks quickly. (Just ask for help!)

This assignment cannot be solved by blindly pasting together code fragments found in class notes or past assignments. Solving the assignment is a multi-step learning process that takes effort, but one that also provides the satisfaction of progress and of developing skills and understanding.

Problem 1: The existing code in `World::ball_setup_7` computes the vertices of the first pentagon parameterically. Modify it so the coordinates are computed using a transformation matrix. Starter code is provided. Please don't waste computation by computing the transformation matrix in the loop.

Note that the balls of the first pentagon are shown in gold. It may help to use Scene 8 when working on this problem.

Problem 2: The code in `World::ball_setup_7` near `HW 2 Problem 2` uses transformation matrix `m` to move the coordinates of the first pentagon (`coors[0]` through `coors[4]`, accessed in the `j` loop) to a new position for each edge of the first pentagon, and it inserts these coordinates in `coors`. The edge of the first pentagon is in `p0` and `p1`. A placeholder is shown for `m`. Correctly compute `m` so that it rotates the first pentagon coordinates by angle `theta_dihedral` (a variable in the code) around the axis formed by `p0` and `p1`.

If this problem is solved correctly half a dodecahedron will be constructed. Call that a *demi-dodecahedron*. The balls of the demi-dodecahedron are shown in gold and purple.

Problem 3: For this problem compute the remaining coordinates of the dodecahedron by applying a transformation to the demi-dodecahedron coordinates. Starter code is shown near `HW 2 Problem 3`. Compute transformation matrix `ms` so that it places the vertices in the correct location. The transformation should rotate the coordinates around the center of the dodecahedron (the center of the circumscribed sphere). More than one rotation will be necessary. The vertices added here are shown in green.