Code name: *<DEMdemo_BallDrop.cpp>*

Block 1:

```
DEMSolver DEMSim;
    DEMSim.SetVerbosity(STEP_METRIC);
    DEMSim.SetOutputFormat(OUTPUT_FORMAT::CSV);
    DEMSim.SetOutputContent(OUTPUT_CONTENT::ABSV);
    DEMSim.SetMeshOutputFormat(MESH_FORMAT::VTK);
    DEMSim.EnsureKernelErrMsgLineNum();
    DEMSim.SetContactOutputContent(OWNER | FORCE | POINT);
```

Description:

DEMSim.SetVerbosity(STEP_METRIC);

- *Set the verbosity level of the solver.*
- *param verbose "QUIET", "ERROR", "WARNING", "INFO", "STEP_ANOMALY", "STEP_METRIC", "DEBUG" or "STEP_DEBUG". Recommend "INFO".*

void SetOutputFormat(OUTPUT_FORMAT::CSV);

- *Sphere and clump output file format.*
- *Format Choice among "CSV", "BINARY" or "CHPF"..*

void SetOutputContent(const std::vector<std::string>& content);

- *Specify the information that needs to go into the clump or sphere output files.*
- *Content A list of "XYZ", "QUAT", "ABSV", "VEL", "ANG_VEL", "ABS_ACC", "ACC", "ANG_ACC", "FAMILY", "MAT"*

void SetContactOutputContent(const std::vector<std::string>& content);

- *Specify the information that needs to go into the contact pair output files.*
- *Content A list of "CNT_TYPE", "FORCE", "POINT", "COMPONENT", "NORMAL", "TORQUE", "CNT_WILDCARD", "OWNER",*

void SetMeshOutputFormat(const std::string& format);

- *Specify the output file format of meshes. Format A choice between "VTK", "OBJ".*

Block 2:

```
auto mat_type_ball = DEMSim.LoadMaterial({{"E", 1e10}, {"nu", 0.3}, {"CoR", 0.6}, {"mu", 0.3}, {"Crr", 0.01}});
  auto mat_type_terrain = DEMSim.LoadMaterial({{"E", 5e9}, {"nu", 0.3}, {"CoR", 0.8}, {"mu", 0.3}, {"Crr", 0.01}});
  // If you don't have this line, then CoR between mixer material and granular material will be 0.7 (average of the
  // two).
  DEMSim.SetMaterialPropertyPair("CoR", mat_type_ball, mat_type_terrain, 0.6);
  // Should do the same for mu and Crr, but since they are the same across 2 materials, it won't have an effect...
```

Description:

This section is about the materials that will be used and their properties.

.LoadMaterial ( )

Initializing materials properties like

- ***E:** Young's Modulus, which is a measure of the material's stiffness.*
- ***nu:** Poisson's Ratio, which is a measure of the material's ability to be compressed or elongated.*
- ***CoR:** Coefficient of Restitution, which defines how 'bouncy' a material is. A value of 1 would mean a perfectly elastic collision, while a value of 0 would mean a perfectly inelastic collision (no bounce).*
- ***mu:** Friction coefficient, which measures the resistance of a material to sliding over another.*
- ***Crr:** Rolling resistance coefficient, which measures the resistance of a material to roll over another.*

.SetMaterialPropertyPair()

- *This function seems to be used to explicitly set a property between two interacting materials*

Block 3:

```
    float step_size = 1e-5;
    double world_size = 10;
    DEMSim.InstructBoxDomainDimension({0, world_size}, {0, world_size}, {0, world_size});
    DEMSim.InstructBoxDomainBoundingBC("top_open", mat_type_terrain);
```

Description:

Step_size:

- Step size in a simulation like DEM often refers to the time increment for each simulation step

world_size:

- Represents the size of the simulation domain or the "world" in which the particles will interact.

.InstructBoxDomainDimension(X_Domain, Y_Domain, Z_Domain)

- Set up a box-shaped domain for the simulation

InstructBoxDomainBoundingBC(m_user_add_bounding_box, m_bounding_box_material)

A. **m_user_add_bounding_box**
- Instruct if and how we should add boundaries to the simulation world upon initialization.
- `none', (NO bondaries)
- `all' (add 6 boundary planes)
- `top_open' (add 5 boundary planes and leave the z-direction top open).

B. **m_bounding_box_material**
- Also specifies the material that should be assigned to those bounding boundaries.

Block 4

```
 auto projectile = DEMSim.AddWavefrontMeshObject((GET_DATA_PATH() /
"mesh/sphere.obj").string(), mat_type_ball);
    std::cout << "Total num of triangles: " << projectile->GetNumTriangles() << std::endl;

    projectile->SetInitPos(make_float3(world_size / 2, world_size / 2, world_size / 3 * 2));
    float ball_mass = 7.8e3 * 4 / 3 * 3.1416;
    projectile->SetMass(ball_mass);
    projectile->SetMOI(make_float3(ball_mass * 2 / 5, ball_mass * 2 / 5, ball_mass * 2 / 5));
    projectile->SetFamily(2);
    DEMSim.SetFamilyFixed(2);
auto proj_tracker = DEMSim.Track(projectile);
```

Description:

Initialize the ball that will be used in the simulation.

AddWavefrontMeshobject(File name, materials type):

- Load a mesh-represented object that you want to use. The file should be with extension .obj . we can create the 3d model using blender or fusion and then save it as obj.
- **SetInitPos(x,y,z):** initialize the position of the object
- **SetMass():** intilize the mass of the object
- **SetMOI():** initialize the Moment of inertia of the object
- **SetFamily():**Mark all entities in this family to be fixed
- **Track():** Track the movement of the object

Block 5:

```
float terrain_rad = 0.25;
    auto template_terrain = DEMSim.LoadSphereType(terrain_rad * terrain_rad *
terrain_rad * 2.6e3 * 4 / 3 * 3.14,terrain_rad, mat_type_terrain);
```

Description:

LoadSphereType(float mass,float radius, material);

- This function works to load the type of clump with mass , radius and materials properties

Block 6:

```
    float sample_halfheight = world_size / 8;

    float3 sample_center = make_float3(world_size / 2, world_size / 2, sample_halfheight + 0.05);

    float sample_halfwidth = world_size / 2 * 0.95;

    auto input_xyz = DEMBoxHCPSampler(sample_center, make_float3(sample_halfwidth, sample_halfwidth, sample_halfheight), 2.01 * terrain_rad);

    DEMSim.AddClumps(template_terrain, input_xyz);

    std::cout << "Total num of particles: " << input_xyz.size() << std::endl;
```

Description:

This code initializes the clump block

DEMBoxHCPSampler(float3 BoxCenter, float3 HalfDims, float GridSize):

- A wrapper for a HCP sampler of a box domain.

AddClumps(input_types, input_xyz);

- Load clumps (of the same template) into the simulation.
- input_types Vector of the types of the clumps (vector of shared pointers).
- input_xyz Vector of the initial locations of the clumps.

Block 7:

```
    DEMSim.SetInitTimeStep(step_size);
    DEMSim.SetGravitationalAcceleration(make_float3(0, 0, -9.81));
    DEMSim.SetMaxVelocity(15.);
    DEMSim.SetExpandSafetyAdder(5.);
```

Description:

These commands are configuring the simulation environment to have specific characteristics like a defined gravitational field, a maximum allowable velocity for particles, and other parameters that will affect how the simulation runs.

Block 7:

```
DEMSim.Initialize();

    path out_dir = current_path();
    out_dir += "/DemoOutput_BallDrop";
    create_directory(out_dir);

    float sim_time = 6.0;
    float settle_time = 2.0;
    unsigned int fps = 20;
    float frame_time = 1.0 / fps;

    std::cout << "Output at " << fps << " FPS" << std::endl;
    unsigned int currframe = 0;
```

Description:

- **Initialize():** Intialize the simulation system
- **Create_directory():** Create the path from the output
- Initalize the length of the simulation and the number of frames per second

Block 8:

```
    // We can let it settle first
    for (float t = 0; t < settle_time; t += frame_time) {
        std::cout << "Frame: " << currframe << std::endl;
        char filename[200], meshfilename[200];
        sprintf(filename, "%s/DEMdemo_output_%04d.csv", out_dir.c_str(), currframe);
        sprintf(meshfilename, "%s/DEMdemo_mesh_%04d.vtk", out_dir.c_str(), currframe);
        DEMSim.WriteSphereFile(std::string(filename));
        DEMSim.WriteMeshFile(std::string(meshfilename));
        currframe++;
        DEMSim.DoDynamicsThenSync(frame_time);
        DEMSim.ShowThreadCollaborationStats();
    }
```

Description:

This section include for loop for the simulation. This section works for the settling the box clumps that we had created. This section will return the output of the simulation as shown

sprintf(filename, "%s/DEMdemo_output_%04d.csv", out_dir.c_str(), currframe);

- This is for the file name

DEMSim.WriteSphereFile(std::string(filename));

- This for the data inside the file

DEMSim.DoDynamicsThenSync(frame_time);

- This to advance the simulation by one step.

Block 9:

```
    // Then drop the ball. I also wanted to test if changing step size method works fine here...
    step_size *= 0.5;
    DEMSim.UpdateStepSize(step_size);
    DEMSim.ChangeFamily(2, 1);

    std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now();
    for (float t = 0; t < sim_time; t += frame_time) {
        std::cout << "Frame: " << currframe << std::endl;
        char filename[200], meshfilename[200], cnt_filename[200];
        sprintf(filename, "%s/DEMdemo_output_%04d.csv", out_dir.c_str(), currframe);
        sprintf(meshfilename, "%s/DEMdemo_mesh_%04d.vtk", out_dir.c_str(), currframe);
        // sprintf(cnt_filename, "%s/Contact_pairs_%04d.csv", out_dir.c_str(), currframe);
        DEMSim.WriteSphereFile(std::string(filename));
        DEMSim.WriteMeshFile(std::string(meshfilename));
        // DEMSim.WriteContactFile(std::string(cnt_filename));
        currframe++;

        DEMSim.DoDynamicsThenSync(frame_time);
        DEMSim.ShowThreadCollaborationStats();
     }
```

Description:

This section include for loop for the simulation. This section works for falling the ball and interact with the clumps we had created. This section will return the output of the simulation.

std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now();

- In this line, A high-resolution timer is started to measure the time taken for the simulation loop that follows.

Block 10:

```
    std::chrono::high_resolution_clock::time_point end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> time_sec = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
    std::cout << time_sec.count() << " seconds (wall time) to finish the simulation" << std::endl;

    DEMSim.ShowTimingStats();
    DEMSim.ShowAnomalies();
    std::cout << "DEMdemo_BallDrop exiting..." << std::endl;
```

Description:

std::chrono::high_resolution_clock::time_point end =: the timer is stopped

The following codes after this code calculate the time of the simulation by subtracting the end and start time

ShowTimingStats():

- return wall time and percentages of wall time spend on various solver tasks

End of the code