**Code name: *<DEMdemo\_ConePentration.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\_cone, mat\_type\_terrain, 0.8);

    DEMSim.SetMaterialPropertyPair("mu", mat\_type\_cone, mat\_type\_terrain, 0.7);

    // 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 cone\_speed = 0.03;

    float step\_size = 5e-6;

    double world\_size = 2;

    double soil\_bin\_diameter = 0.584;

    double cone\_surf\_area = 323e-6;

    double cone\_diameter = std::sqrt(cone\_surf\_area / math\_PI) \* 2;

    DEMSim.InstructBoxDomainDimension(world\_size, world\_size, world\_size);

    // No need to add simulation `world' boundaries, b/c we'll add a cylinderical container manually

    DEMSim.InstructBoxDomainBoundingBC("none", mat\_type\_terrain);

    // Now add a cylinderical boundary along with a bottom plane

**Description:**

**Cone\_speed:**

* The speed of the Cone that will be used in the simulation

**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)**

1. **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).

1. **m\_bounding\_box\_material**

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

**Block 4:**

double bottom = -0.5;

    auto walls = DEMSim.AddExternalObject();

    walls->AddCylinder(make\_float3(0), make\_float3(0, 0, 1), soil\_bin\_diameter / 2., mat\_type\_terrain, 0);

    walls->AddPlane(make\_float3(0, 0, bottom), make\_float3(0, 0, 1), mat\_type\_terrain);

**Description :**

**AddExternalObject():**

* Add an external object to the simulation system.

**AddCylinder():**

* Add a cylinder of infinite length, which is along a user-specific axis
* AddCylinder(const pos, const axis, const float rad, const material, const objNormal\_t normal = ENTITY\_NORMAL\_INWARD)

**AddPlane():**

* Add a plane with infinite size
* AddPlane(const float3 pos, const float3 normal, const std::shared\_ptr<DEMMaterial>& material)

**Block 5:**

    float terrain\_density = 2.6e3;

    double clump\_vol = 5.5886717;

    float mass = terrain\_density \* clump\_vol;

    float3 MOI = make\_float3(2.928, 2.6029, 3.9908) \* terrain\_density;

    // Then load it to system

    std::shared\_ptr<DEMClumpTemplate> my\_template =

        DEMSim.LoadClumpType(mass, MOI, GetDEMEDataFile("clumps/3\_clump.csv"), mat\_type\_terrain);

    my\_template->SetVolume(clump\_vol);

    // Decide the scalings of the templates we just created (so that they are... like particles, not rocks)

    double scale = 0.0044;

    my\_template->Scale(scale);

**Description:**

Define the terrain particle templates that will be used in the simulation.

**LoadClumpType():**

* An overload of LoadClumpType which loads sphere components from a file
* LoadClumpType(float mass, float3 moi, Filename , sp\_materials);

**SetVolume():**

* Set the volume of the templets

**Scale():**

* the help to scale the original templet as the size of the model.

**Block 6:**

 // Sampler to sample

    HCPSampler sampler(scale \* 3.);

    float fill\_height = 0.5;

    float3 fill\_center = make\_float3(0, 0, bottom + fill\_height / 2);

    const float fill\_radius = soil\_bin\_diameter / 2. - scale \* 3.;

    auto input\_xyz = sampler.SampleCylinderZ(fill\_center, fill\_radius, fill\_height / 2 - scale \* 2.);

    DEMSim.AddClumps(my\_template, input\_xyz);

    std::cout << "Total num of particles: " << input\_xyz.size() << std::endl;

**Description:**

This section create the Hexagonal Close Packed (HCP) Sampler

**SampleCylinderZ():**

* SampleCylinderZ(const float3& center, float radius, float halfHeight)
* Return points sampled from the specified Z-aligned cylindrical volume

**AddClumps():**

* Add the clumps that we created before
* AddClumps(input\_types, input\_xyz)

**Block 7:**

     // Load in the cone used for this penetration test

    auto cone\_tip = DEMSim.AddWavefrontMeshObject(GetDEMEDataFile("mesh/cone.obj"), mat\_type\_cone);

    auto cone\_body = DEMSim.AddWavefrontMeshObject(GetDEMEDataFile("mesh/cyl\_r1\_h2.obj"), mat\_type\_cone);

    std::cout << "Total num of triangles: " << cone\_tip->GetNumTriangles() + cone\_body->GetNumTriangles() << std::endl;

    // The initial cone mesh has base radius 1, and height 1. Let's stretch it a bit so it has a 60deg tip, instead of

    // 90deg.

    float tip\_height = std::sqrt(3.);

    cone\_tip->Scale(make\_float3(1, 1, tip\_height));

    // Then set mass properties

    float cone\_mass = 7.8e3 \* tip\_height / 3 \* math\_PI;

    cone\_tip->SetMass(cone\_mass);

    // You can checkout https://en.wikipedia.org/wiki/List\_of\_moments\_of\_inertia

    cone\_tip->SetMOI(make\_float3(cone\_mass \* (3. / 20. + 3. / 80. \* tip\_height \* tip\_height),

                                 cone\_mass \* (3. / 20. + 3. / 80. \* tip\_height \* tip\_height), 3 \* cone\_mass / 10));

    // This cone mesh has its tip at the origin. And, float4 quaternion pattern is (x, y, z, w).

    cone\_tip->InformCentroidPrincipal(make\_float3(0, 0, 3. / 4. \* tip\_height), make\_float4(0, 0, 0, 1));

    // Note the scale method will scale mass and MOI automatically. But this only goes for the case you scale xyz all

    // together; otherwise, the MOI scaling will not be accurate and you should manually reset them.

    cone\_tip->Scale(cone\_diameter / 2);

    cone\_tip->SetFamily(2);

// The define the body that is connected to the tip

    float body\_mass = 7.8e3 \* math\_PI;

    cone\_body->SetMass(body\_mass);

    cone\_body->SetMOI(make\_float3(body\_mass \* 7 / 12, body\_mass \* 7 / 12, body\_mass / 2));

    // This cyl mesh (h = 2m, r = 1m) has its center at the origin. So the following call actually has no effect...

    cone\_body->InformCentroidPrincipal(make\_float3(0, 0, 0), make\_float4(0, 0, 0, 1));

    cone\_body->Scale(make\_float3(cone\_diameter / 2, cone\_diameter / 2, 0.5));

    cone\_body->SetFamily(2);

    // Track the cone\_tip

    auto tip\_tracker = DEMSim.Track(cone\_tip);

    auto body\_tracker = DEMSim.Track(cone\_body);

**Description:**

Initialize the Cone that will be used in the simulation. The cone consist of two different parts which is the tip and the body. each part is assigned separately and then attached to form the cone.

**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 8:**

/ Now add a plane to compress the sample

    auto compressor = DEMSim.AddExternalObject();

    compressor->AddPlane(make\_float3(0, 0, 0), make\_float3(0, 0, -1), mat\_type\_terrain);

    compressor->SetFamily(10);

    DEMSim.SetFamilyFixed(10);

    auto compressor\_tracker = DEMSim.Track(compressor);

**Description:**

* In this section Plane to compress the sample is created and called compressor and track it

**Block 9:**

    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 10:**

DEMSim.Initialize();

    std::filesystem::path out\_dir = std::filesystem::current\_path();

    out\_dir += "/DemoOutput\_ConePenetration";

    std::filesystem::create\_directory(out\_dir);

    // Settle

    DEMSim.DoDynamicsThenSync(0.8);

    // Compress until dense enough

    unsigned int currframe = 0;

    unsigned int curr\_step = 0;

    unsigned int fps = 20;

    unsigned int out\_steps = (unsigned int)(1.0 / (fps \* step\_size));

    double compressor\_vel = 0.05;

    float terrain\_max\_z = max\_z\_finder->GetValue();

    double init\_max\_z = terrain\_max\_z;

    float bulk\_density = -10000.;

**Description:**

* **Initialize():** Initialize the simulation system
* **Create\_directory():** Create the path from the output
* Initialize the length of the simulation and the number of frames per second.
* Some of variables created to evaluate the simulation and change it

**Block 11:**

   while (bulk\_density < 1500.) {

        float matter\_mass = total\_mass\_finder->GetValue();

        float total\_volume = math\_PI \* (soil\_bin\_diameter \* soil\_bin\_diameter / 4) \* (terrain\_max\_z - bottom);

        bulk\_density = matter\_mass / total\_volume;

        if (curr\_step % out\_steps == 0) {

            char filename[200], meshname[200];

            sprintf(filename, "%s/DEMdemo\_output\_%04d.csv", out\_dir.c\_str(), currframe);

            sprintf(meshname, "%s/DEMdemo\_mesh\_%04d.vtk", out\_dir.c\_str(), currframe);

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

            // DEMSim.WriteMeshFile(std::string(meshname));

            std::cout << "Compression bulk density: " << bulk\_density << std::endl;

            currframe++;

        }

        terrain\_max\_z -= compressor\_vel \* step\_size;

        compressor\_tracker->SetPos(make\_float3(0, 0, terrain\_max\_z));

        DEMSim.DoDynamics(step\_size);

        curr\_step++;

    }

**Description:**

This code snippet is creating a simulation loop that appears to be modeling some compression process until a specified bulk density is achieved. It's saving simulation data at specified intervals and continuously updating the simulation dynamics and the position of a compressor. **If statement**

* works as the curr\_step is larger than out\_step **(% mean remain from the division** )

**compressor\_tracker->SetPos(make\_float3(0, 0, terrain\_max\_z));**

* Updating Terrain and Compressor Position:
* The terrain\_max\_z variable is decremented by compressor\_vel \* step\_size, and the position of the compressor tracker is updated accordingly.

**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.DoDynamics(frame\_time);**

* This to advance the simulation by one step.

**Block 12:**

    while (terrain\_max\_z < init\_max\_z) {

        if (curr\_step % out\_steps == 0) {

            char filename[200], meshname[200];

            sprintf(filename, "%s/DEMdemo\_output\_%04d.csv", out\_dir.c\_str(), currframe);

            sprintf(meshname, "%s/DEMdemo\_mesh\_%04d.vtk", out\_dir.c\_str(), currframe);

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

            // DEMSim.WriteMeshFile(std::string(meshname));

            float matter\_mass = total\_mass\_finder->GetValue();

            float total\_volume =

                math\_PI \* (soil\_bin\_diameter \* soil\_bin\_diameter / 4) \* (max\_z\_finder->GetValue() - bottom);

            bulk\_density = matter\_mass / total\_volume;

            std::cout << "Compression bulk density: " << bulk\_density << std::endl;

            currframe++;

        }

        terrain\_max\_z += compressor\_vel \* step\_size;

        compressor\_tracker->SetPos(make\_float3(0, 0, terrain\_max\_z));

        DEMSim.DoDynamics(step\_size);

        curr\_step++;

    }

**Description:**

This section include a while loop for the simulation. This section works to remove the compressor to its initial state.

**Block 13:**

  // Remove compressor

    DEMSim.DoDynamicsThenSync(0.);

    DEMSim.DisableContactBetweenFamilies(0, 10);

    DEMSim.DoDynamicsThenSync(0.2);

    terrain\_max\_z = max\_z\_finder->GetValue();

**Description:**

The code snippet appears to be part of a larger simulation workflow, where a compressor is being removed from the simulation, and certain simulation parameters are being updated accordingly.

**DEMSim.DisableContactBetweenFamilies(0, 10):** method is called to disable contact interactions between particle families 0 and 10, which might correspond to the compressor and the soil particles, respectively.

**Max\_z\_finder-> GetValue():** reflect changes in the terrain elevation after the compressor removal.

**Block 14:**

float sim\_end = 7.0;

    fps = 2500;

    float frame\_time = 1.0 / fps;

    std::cout << "Output at " << fps << " FPS" << std::endl;

    // Put the cone in place

    double starting\_height = terrain\_max\_z + 0.03;

    // Its initial position should be right above the cone tip...

    body\_tracker->SetPos(make\_float3(0, 0, 0.5 + (cone\_diameter / 2 / 4 \* tip\_height) + starting\_height));

    // Note that position of objects is always the location of their centroid

    tip\_tracker->SetPos(make\_float3(0, 0, starting\_height));

    // The tip location, used to measure penetration length

    double tip\_z = -cone\_diameter / 2 \* 3 / 4 \* tip\_height + starting\_height;

    // Enable cone

    DEMSim.ChangeFamily(2, 1);

    float matter\_mass = total\_mass\_finder->GetValue();

    float total\_volume = math\_PI \* (soil\_bin\_diameter \* soil\_bin\_diameter / 4) \* (terrain\_max\_z - bottom);

    bulk\_density = matter\_mass / total\_volume;

    std::cout << "Bulk density: " << bulk\_density << std::endl;

    double tip\_z\_when\_first\_hit;

    bool hit\_terrain = false;

    unsigned int frame\_count = 0;

**Description:**

This code initialize the properties for the second simulation which consist the movement of the cone.

* First the time and step size initialize.
* Then the location of the cone is sit by using SetPos**(x,y,z):** initialize the location of the cone (body +tip)
* **Change Family (from , to) :** is a method to switch the family number for enabling

**Block 15:**

    std::chrono::high\_resolution\_clock::time\_point start = std::chrono::high\_resolution\_clock::now();

    for (float t = 0; t < sim\_end; t += frame\_time) {

        // float terrain\_max\_z = max\_z\_finder->GetValue();

        float3 forces = tip\_tracker->ContactAcc();

        // Note cone\_mass is not the true mass, b/c we scaled the the cone tip! So we use true mass by us

        forces \*= cone\_tip->mass;

        float pressure = std::abs(forces.z) / cone\_surf\_area;

        if (pressure > 1e-4 && !hit\_terrain) {

            hit\_terrain = true;

            tip\_z\_when\_first\_hit = tip\_z;

        }

        float penetration = (hit\_terrain) ? tip\_z\_when\_first\_hit - tip\_z : 0;

        std::cout << "Time: " << t << std::endl;

        std::cout << "Z coord of tip: " << tip\_z << std::endl;

        std::cout << "Penetration: " << penetration << std::endl;

        std::cout << "Force on cone: " << forces.x << ", " << forces.y << ", " << forces.z << std::endl;

        std::cout << "Pressure: " << pressure << std::endl;

        if (frame\_count % 500 == 0) {

            char filename[200], meshname[200];

            std::cout << "Outputting frame: " << currframe << std::endl;

            sprintf(filename, "%s/DEMdemo\_output\_%04d.csv", out\_dir.c\_str(), currframe);

            sprintf(meshname, "%s/DEMdemo\_mesh\_%04d.vtk", out\_dir.c\_str(), currframe++);

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

            DEMSim.WriteMeshFile(std::string(meshname));

            DEMSim.ShowThreadCollaborationStats();

        }

        DEMSim.DoDynamicsThenSync(frame\_time);

        tip\_z -= cone\_speed \* frame\_time;

        frame\_count++;

    }

**Description:**

This code snippet demonstrates a structured approach to advancing a DEM simulation, monitoring key parameters, and saving simulation data at specified intervals.

**std::chrono::high\_resolution\_clock::now()**

* the high accuracy clock it on to measure the duration of the simulation loop

**ContactAcc();**

* ContactAcc(size\_t offset)

**First IF statement :**

* Check if the tip of the cone teach the sample by measuring the pressure
* This will help to measure the penetration if return true

**Second IF statement :**

* This make sure the output of the simulation return every 500 frame which can help to reduce the number of files and accelerate the simulation.

**Block 16:**

    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;

**Description:**

**std::chrono::high\_resolution\_clock::time\_point end** =:

* The timer is stopped which will be usedto measure the simulation length
* The codes after this code calculate the time of the simulation by subtracting the start from end time.

**End of the code**