Chrono Visualization

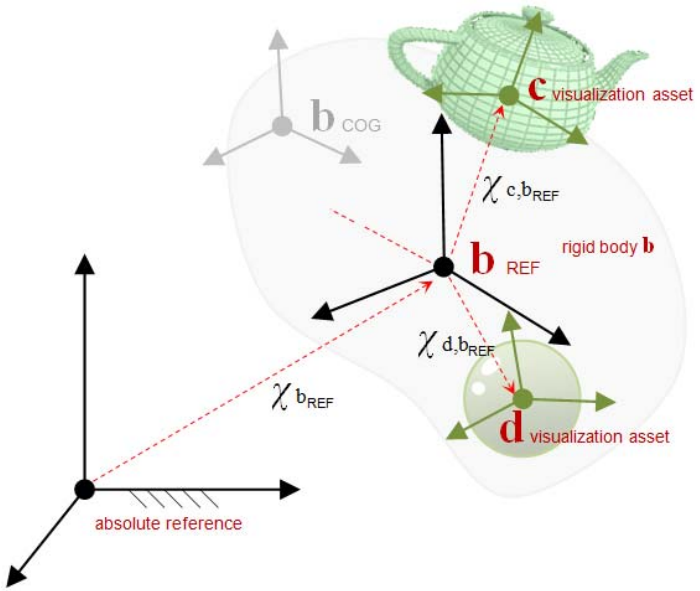
Run-time and off-line visualization support

Chrono visualization assets

Visualization assets

```
ChAsset
    ChVisualization
        ChSphereShape
        ChCylinderShape
        ChBoxShape
        …
```

- An arbitrary number of visualization assets can be attached to a body
- The position and rotation are defined respect to REF frame
- Visualization assets are used by postprocessing systems and by the realtime 3D interfaces



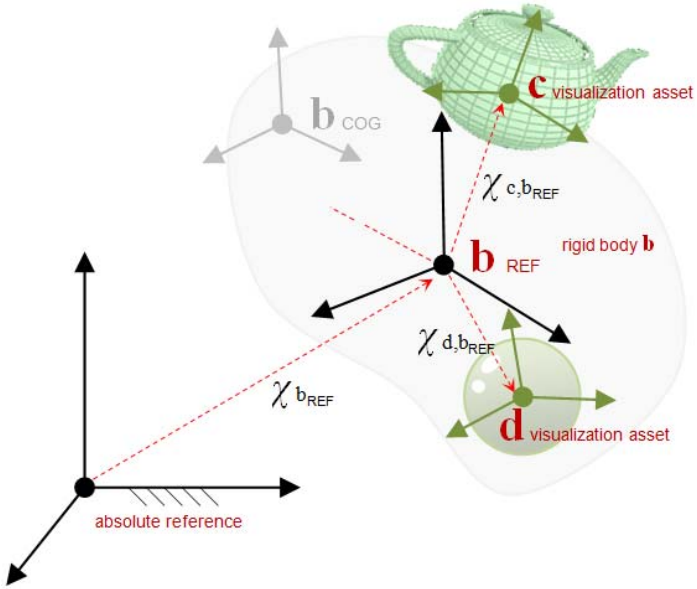
Visualization assets – construction (1/2)

- Example: add a box

```
auto box = std::make_shared<ChBoxShape>();
box->GetBoxGeometry().Pos = ChVector<>(0,-1,0);
box->GetBoxGeometry().Size = ChVector<>(10,0.5,10);
body->AddAsset(box);
```

- Example: add a texture

```
auto texture = std::make_shared<ChTexture>();
texture->SetTextureFilename(GetChronoDataFile("bluwhite.png"));
body->AddAsset(texture);
```



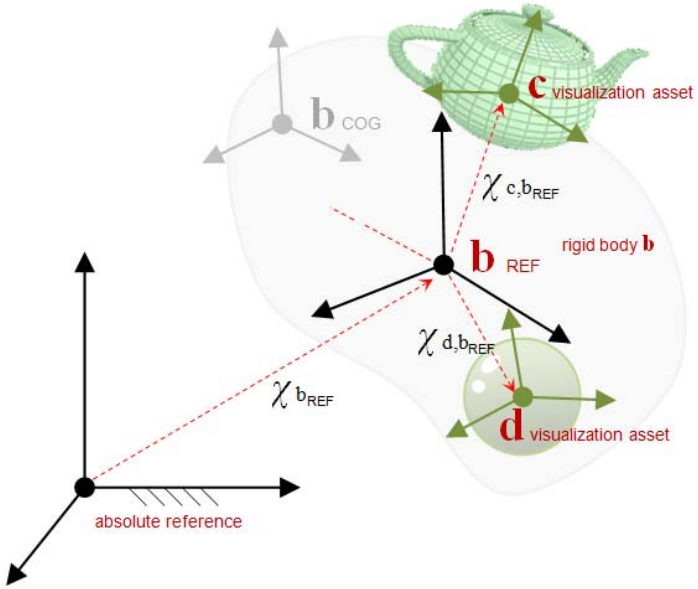
Visualization assets – construction (2/2)

- Example: add a mesh (reference to a Wavefront OBJ file)

```
auto meshfile = std::make_shared<ChObjShapeFile>();
meshfile->SetFilename("forklift_body.obj");
body->AddAsset(meshfile);
```

- Example:

```
auto mesh = std::make_shared<ChTriangleMeshShape>();
mesh->GetMesh()->LoadWavefrontMesh("forklift_body.obj");
body->AddAsset(mesh);
```

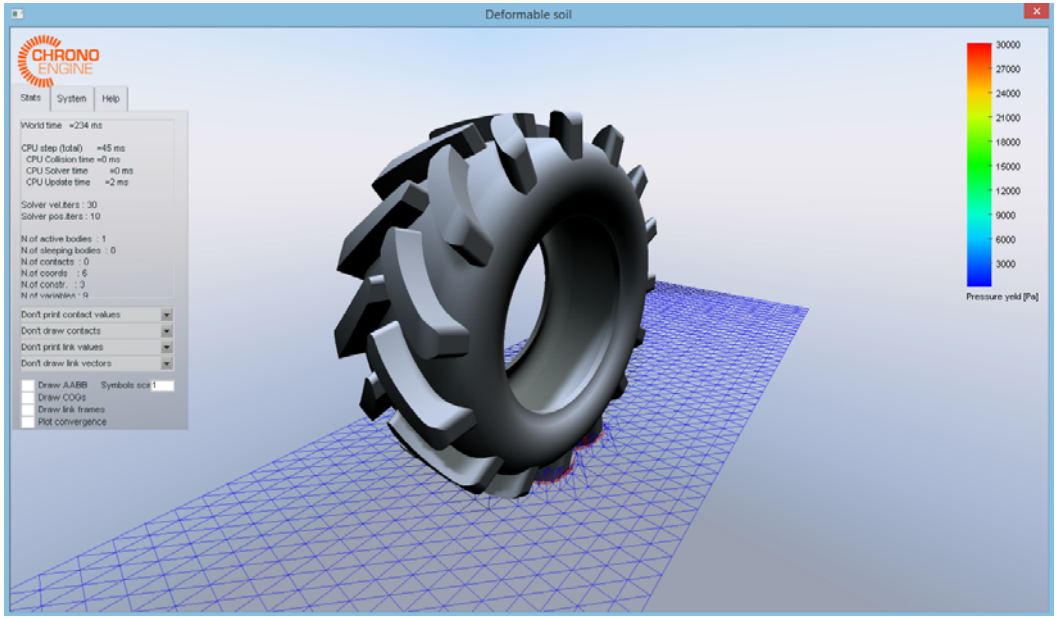


Run-time visualization with Irrlicht

Chrono::Irrlicht module

Irrlicht visualization

- Enable the IRRLICHT module in CMake when configuring Chrono.
- This module uses the Irrlicht3D library for showing objects in real-time interactive views.
- Most demos in Chrono use this visualization system.



Irrlicht visualization

- An Irrlicht application object must be created
- Lights and camera must be added

```
// Create a Chrono::Engine physical system
ChSystem my_system;

...

// Create the Irrlicht visualization
ChIrrApp application(&my_system,                                  // physical system
                     L"Deformable soil and deformable tire",      // title
                     core::dimension2d<u32>(1280, 720),           // window size
                     false,                                       // full screen?
                     true,                                        // support stencil shadows?
                     true);                                       // antialiansing?

// Easy shortcuts to add camera, lights, logo and sky in Irrlicht scene:
application.AddTypicalLogo();   // optional (Chrono logo in top left)
application.AddTypicalSky();    // optional (surrounding horizon gradient as sky)
application.AddTypicalLights(); // creates two default lights
application.AddTypicalCamera(core::vector3df(1.0f, 1.4f, -1.2f),  // camera position
                             core::vector3df(0, 0, 0));           // camera aim point
```

Visualization of Chrono objects

To be visualized by Irrlicht, Chrono objects must contain some visualization assets (sphere, boxes, meshes), **plus a ChIrrNodeAssset asset**; it is like 'flagging it' for Irrlicht. If you forget this, object will be simulated but will remain INVISIBLE.
This can be done quickly with shortcuts:

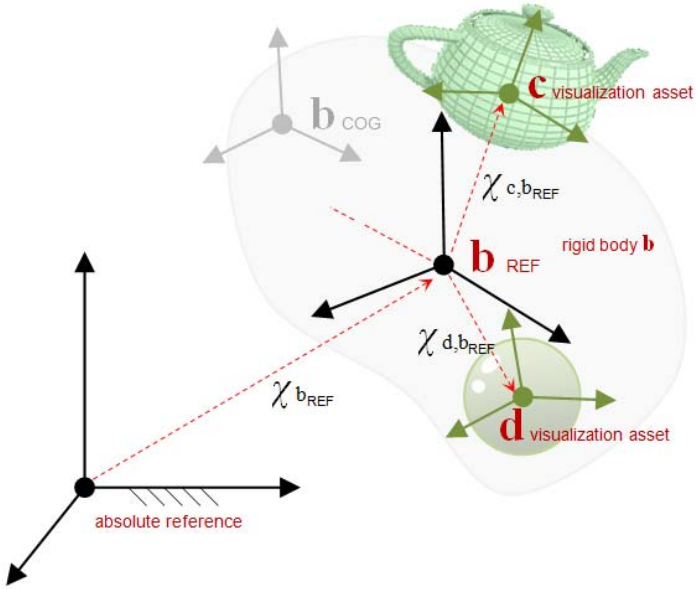
- After all asset creation in all bodies, do:

```
application.AssetBindAll();
application.AssetUpdateAll();
```

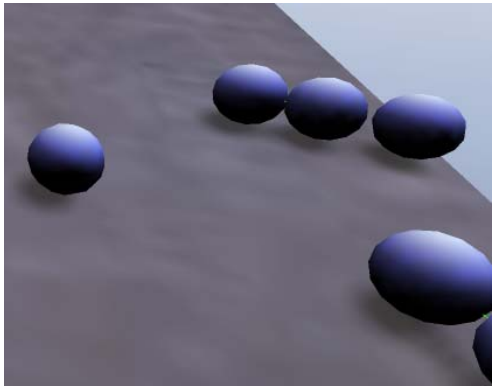
- Alternatively, you can attach the ChIrrNodeAsset to single bodies via:

```
application.AssetBind(body);
application.AssetUpdate(body);
```

(This is useful especially when you are continuously creating objects, ex. in a waterfall of particles, because you can call AssetBind**All**() only once).



Soft shadow casting



- If you want optional soft-shadow-casting:

  A) use AddLightWithShadow():

```
...
application.AddTypicalLogo();   // optional (Chrono logo in top left)
application.AddTypicalSky();    // optional (surrounding horizon gradient as sky)
application.AddTypicalLights(); // creates two default lights
application.AddTypicalCamera(core::vector3df(1.0f, 1.4f, -1.2f), core::vector3df(0, 0, 0));
application.AddLightWithShadow(core::vector3df(1.5f, 5.5f, -2.5f), core::vector3df(0, 0, 0), 3, 2.2, 7.2, 40, 512,
                               video::SColorf(0.8f, 0.8f, 1.0f));
```

  B) after creating all objects, enable soft-shadow-casting with:

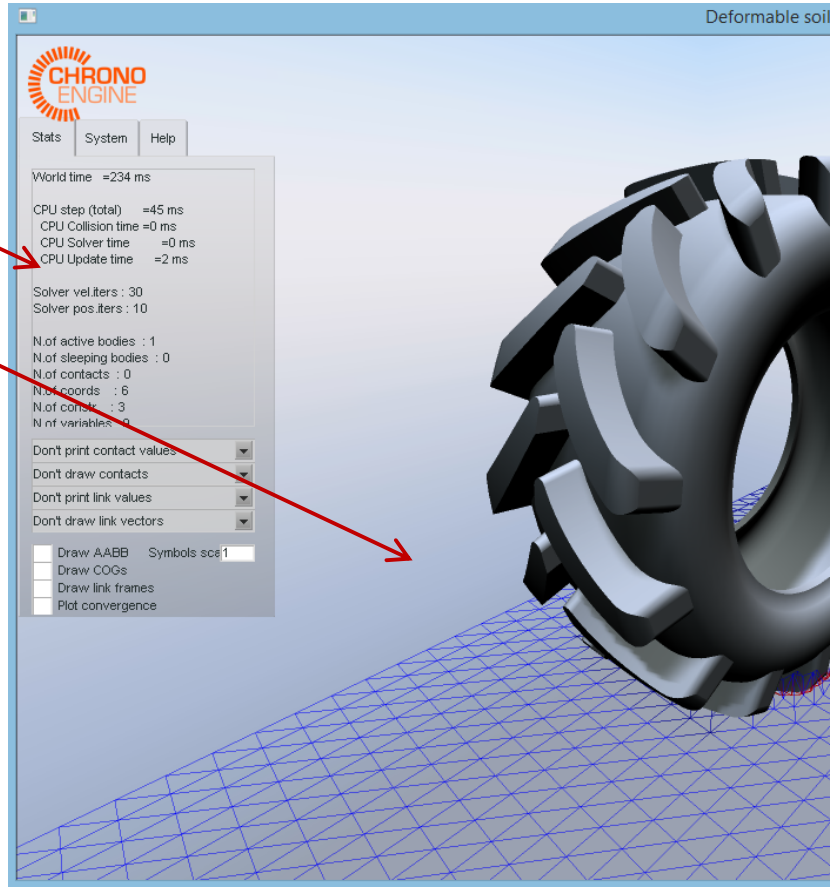
```
application.AssetBindAll();
application.AssetUpdateAll();
application.AddShadowAll();
```

- Note: soft-shadow-casting may slow down 3D refresh

Irrlicht view GUI

The base ChIrrApp provides some basic GUI:

- Press «i» to open this diagnostic window
- Use mouse LMB and RMB to rotate-pan the view
- Use mouse wheel to zoom
- Press «space» to start-stop the simulation
- Press «p» to advance one step a time
- Press «print screen» to activate video capture
- Etc. (see Help tab in panel)

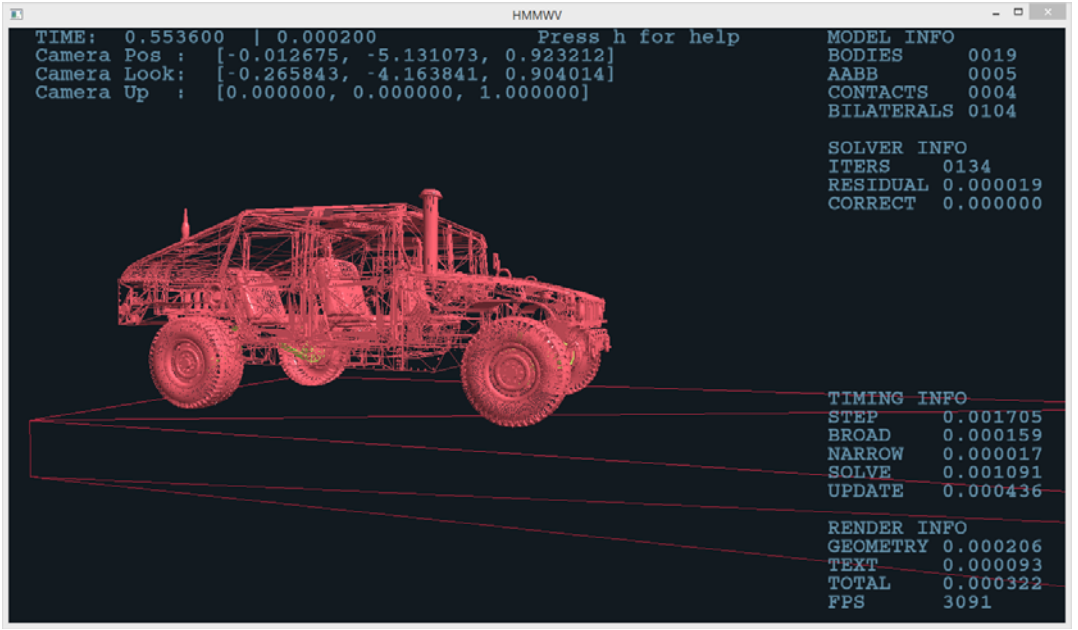


Run-time visualization with OpenGL

Chrono::OpenGL module

OpenGL visualization

- Enable the OpenGL module in CMake when configuring Chrono.
- Dependencies: GLFW (window inputs and events), GLEW, GLM (math).
- This module uses the OpenGL library for rendering objects in real-time interactive views.
- Limited support of assets (e.g. no FEA meshes)
- Taylored for efficient rendering of large scenes simulated with Chrono::Parallel

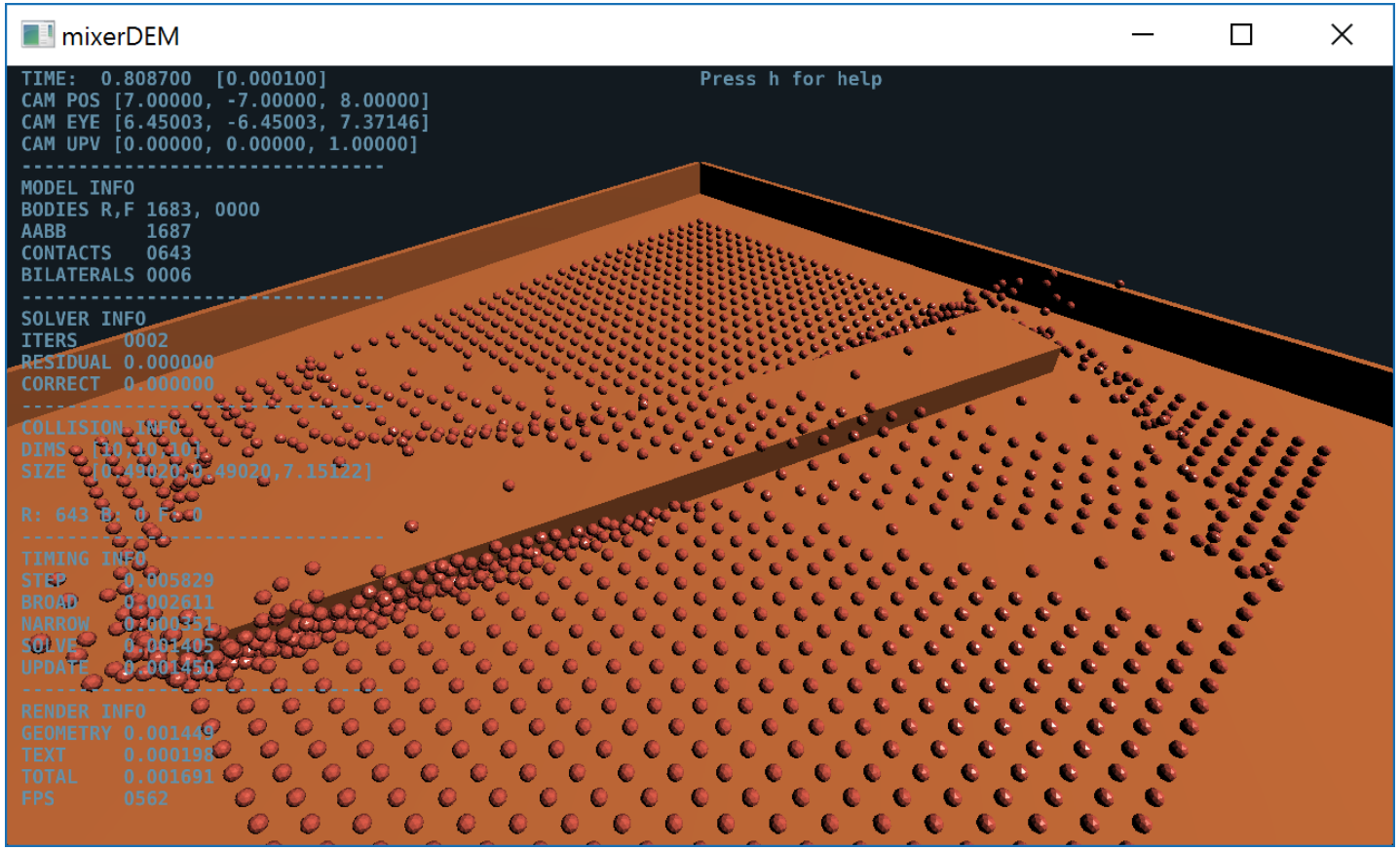


Using Chrono::OpenGL with Chrono

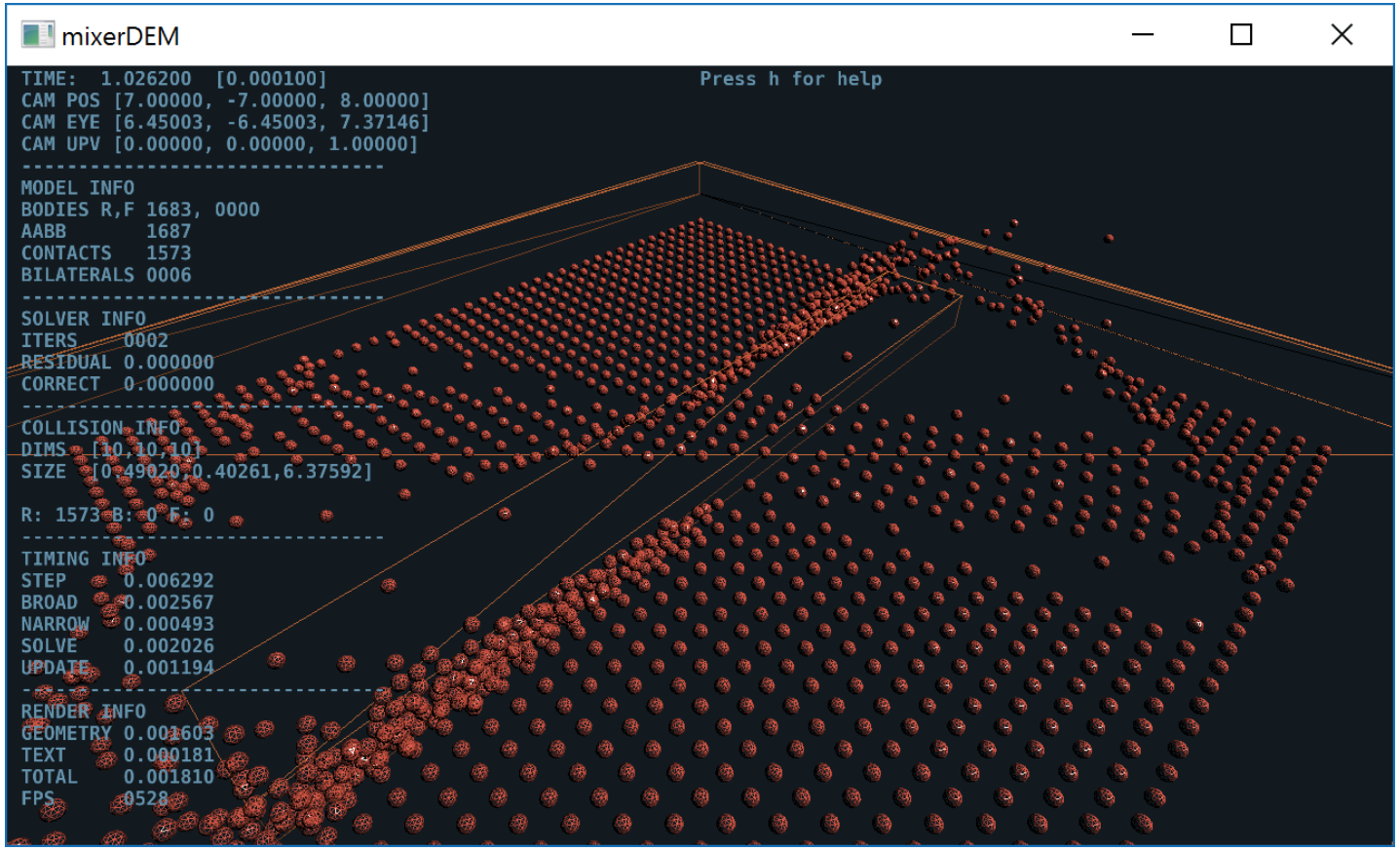
```
#include "chrono_opengl/ChOpenGLWindow.h"

opengl::ChOpenGLWindow& gl_window = opengl::ChOpenGLWindow::getInstance();
gl_window.Initialize(1280, 720, "mixerDVI", &msystem);
gl_window.SetCamera(ChVector<>(0, -10, 0), ChVector<>(0, 0, 0), ChVector<>(0, 0, 1));

while (true) {
  if (gl_window.Active()) {
    gl_window.DoStepDynamics(time_step);
    gl_window.Render();
  } else {
    break;
  }
}
```

```
#include "chrono_opengl/ChOpenGLWindow.h"

opengl::ChOpenGLWindow& gl_window = opengl::ChOpenGLWindow::getInstance();
gl_window.Initialize(1280, 720, "mixerDVI", &msystem);
gl_window.SetCamera(ChVector<>(0, -10, 0), ChVector<>(0, 0, 0), ChVector<>(0, 0, 1));

gl_window.StartDrawLoop(time_step);
```

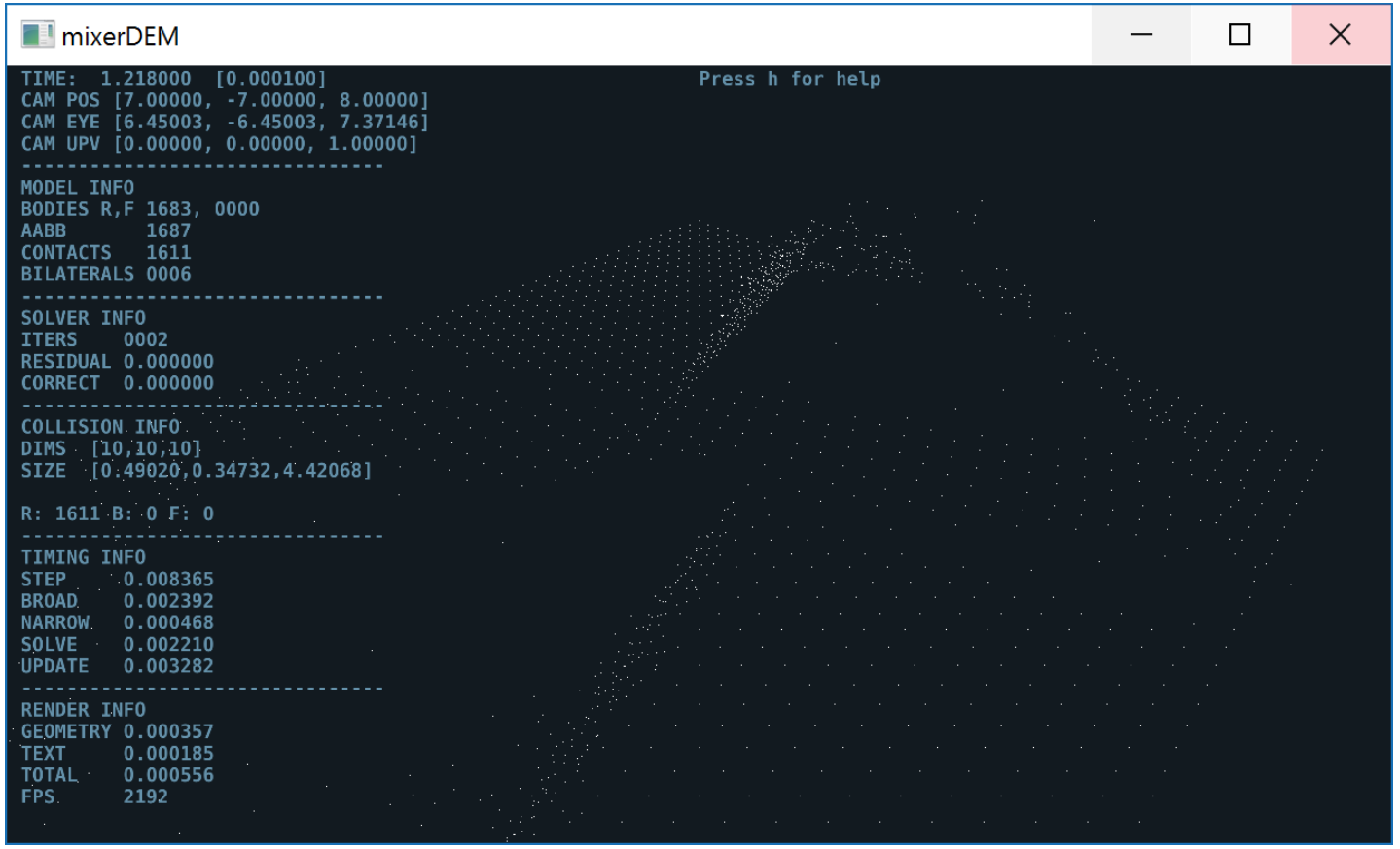
Solid rendering



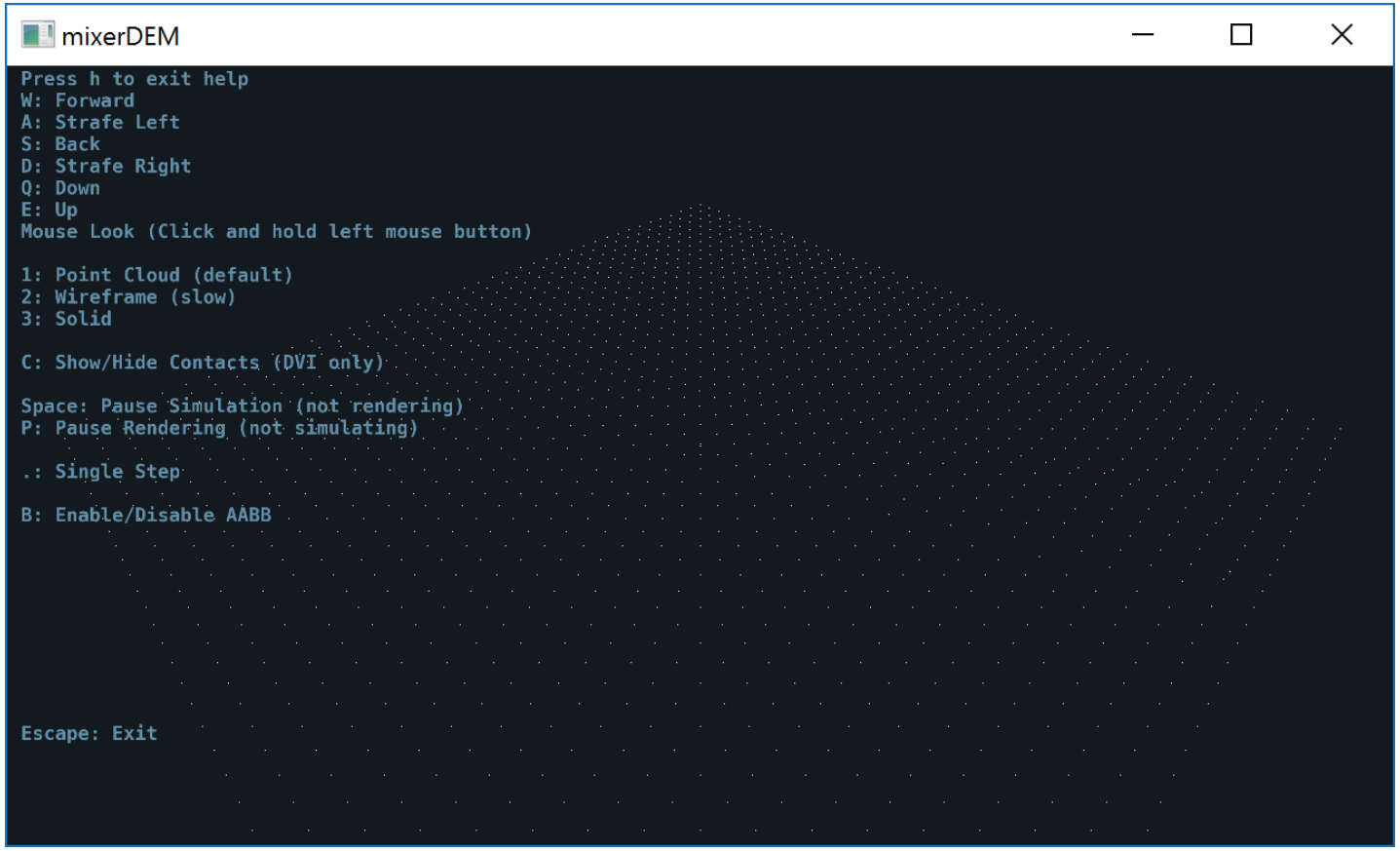
Wireframe rendering



Point-cloud rendering



OpenGL view GUI

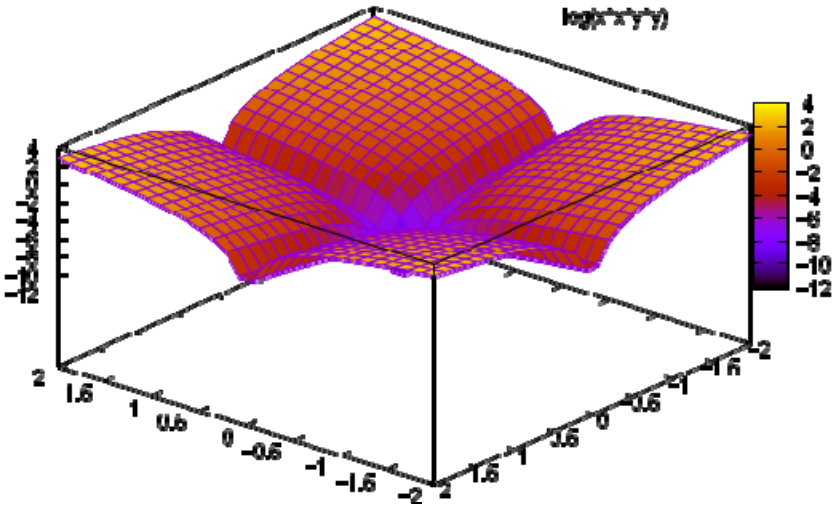


Post processing with Gnuplot

Chrono::Postprocess module and Gnuplot support

GNUplot interface

- Enable POSTPROCESSING module in CMake when you configure Chrono, and compile it.
- GNUplot must be installed on your computer
- The ChGnuPlot class can be used to generate GNUplot scripts from Chrono
- The ChGnuPlot class can be used to directly call GNUplot from Chrono
- Also used to generate .EPS vector plots

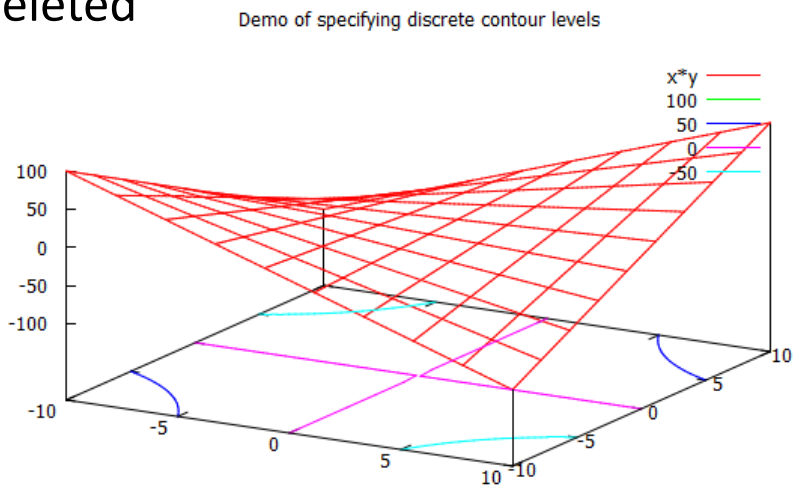


GNUplot interface: example 1

- Example: generate a .gpl script:

```
ChGnuPlot mplot("__tmp_gnuplot_1.gpl");
mplot << "set contour";
mplot << "set title 'Demo of specifying discrete contour levels'";
mplot << "splot x*y";
```

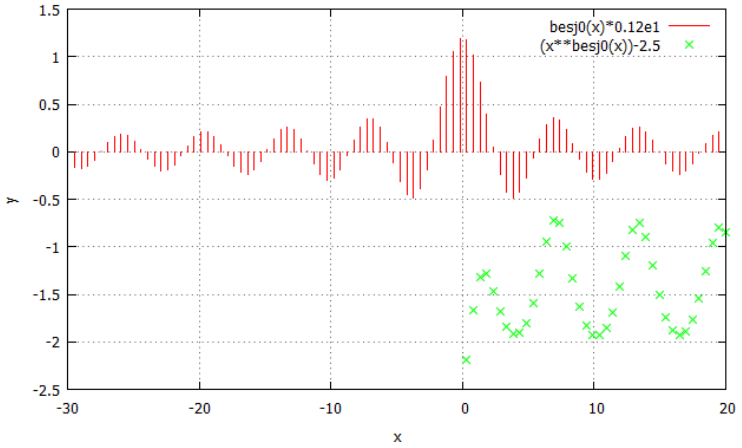
- When the mplot object goes out of scope and is deleted
  - the .gpl script is saved on disk
  - GNUplot (if available on PATH) is launched with that .gpl, and the window with the plot opens

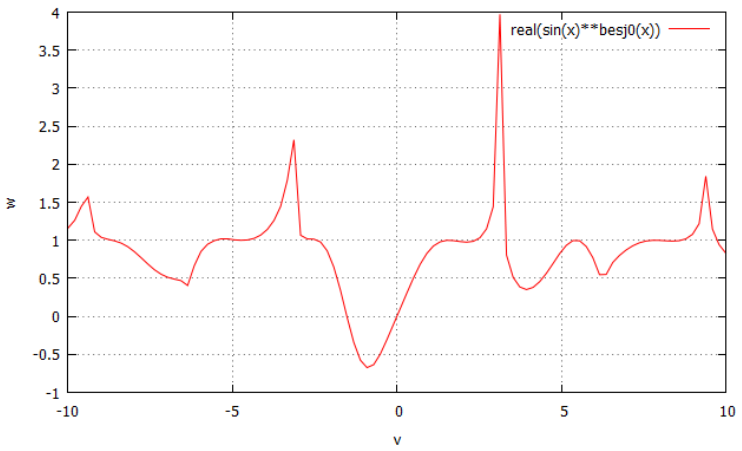


GNUplot interface: example 2

- Make 2 plots
- Save them in EPS

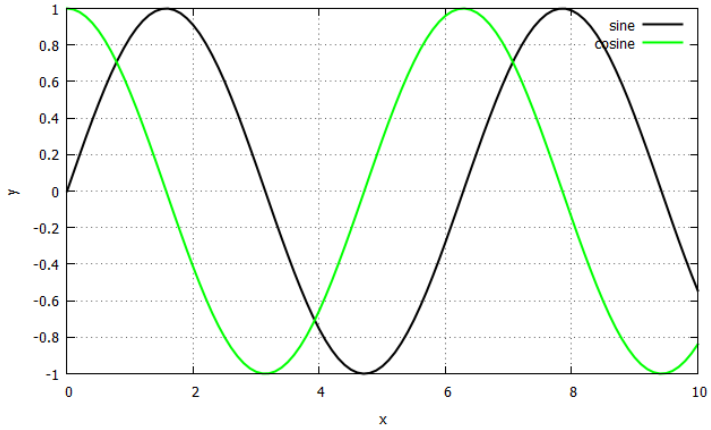
```
ChGnuPlot mplot("__tmp_gnuplot_2.gpl");
mplot.SetGrid();

mplot.OutputWindow(0);
mplot.SetLabelX("x");
mplot.SetLabelY("y");
mplot << "plot [-30:20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2.5 with points";

mplot.OutputWindow(1);
mplot.SetLabelX("v");
mplot.SetLabelY("w");
mplot << "plot [-10:10] real(sin(x)**besj0(x))";

mplot.OutputEPS("test_eps.eps");
mplot.Replot();  // repeat last plot
```





GNUplot interface: example 3

- Plot from .dat files




```
// Step 1.
// create a .dat file with three columns of demo data:
ChStreamOutAsciiFile mdatafile("test_gnuplot_data.dat");
for (double x = 0; x < 10; x += 0.1)
    mdatafile << x << ", " << sin(x) << ", " << cos(x) << "\n";

// Step 2.
// Create the plot.
// NOTE. In this case you pass the .dat filename, the columns IDs, title and custom settings
// NOTE. You can have multiple Plot() calls for a single Output,
// they will be overlapped as when you use commas in gnuplot:  "plot ... , ... , ..."
ChGnuPlot mplot("__tmp_gnuplot_3.gpl");
mplot.SetGrid();
mplot.SetLabelX("x");
mplot.SetLabelY("y");
mplot.Plot("test_gnuplot_data.dat", 1, 2, "sine", " with lines lt -1 lw 2");
mplot.Plot("test_gnuplot_data.dat", 1, 3, "cosine", " with lines lt 2 lw 2");
```

GNUplot interface: example 4

- Plot from embedded data (vectors, functions) without .dat files:

```
 // create demo data in a pair of x,y vectors
ChVectorDynamic<> mx(100);
ChVectorDynamic<> my(100);
for (int i = 0; i < 100; ++i) {
    double x = ((double)i / 100.0) * 12;
    double y = sin(x) * exp(-x * 0.2);
    mx(i) = x; my(i) = y;
}
// ..or create demo data in a ChFunction_Recorder
ChFunction_Recorder mfun;
for (int i = 0; i < 100; ++i) {
    double x = ((double)i / 100.0) * 12;
    double y = cos(x) * exp(-x * 0.4);
    mfun.AddPoint(x, y);
}
// ..or create demo data in two columns of a ChMatrix
ChMatrixDynamic<> matr(100, 10);
for (int i = 0; i < 100; ++i) {
    double x = ((double)i / 100.0) * 12;
    double y = cos(x) * exp(-x * 0.4);
    matr(i, 2) = x;
    matr(i, 6) = y * 0.4;
}

// Create the plot using the Plot() shortcuts.

ChGnuPlot mplot("__tmp_gnuplot_4.gpl");
mplot.SetGrid();
mplot.Plot(mx, my, "from x,y ChVectorDynamic", " every 5 pt 1 ps 0.5");
mplot.Plot(mfun, "from ChFunction_Recorder", " with lines lt -1 lc rgb'#00AAEE' ");
mplot.Plot(matr, 2, 6, "from ChMatrix", " with lines lt 5");
```