TurboPK

TurboPk is a survivability/vulnerability/lethality (SVL) code highly optimized for modern desktop CPUs. Modern CPUs are designed for parallel processing. They are multicore, superscalar, hyperthreaded, and vectorized. These parallel processing features can result in very impressive performance ***if*** the code in question has been written to take advantage of them. TurboPk was written to do exactly that.

TurboPk is orders of magnitude faster than other point-burst codes. Figure 1 is a screen shot of TurboPk analyzing a set of 1000 burst points against a lightly armored vehicle. In this example the warhead ejects 2000 fragments. The burst points are all at the same height 5-meters above the ground plane. The weapon trajectory is a 45-degree diving angle. At each of the 1000 burst locations 20 Monte Carlo sample point-bursts were performed to compute average P_K. Figure 1 is a top view displaying average P_K versus burst location as a set of color-coded dots. Total run time was only 0.65 *seconds* on an Intel Core i7 desktop machine.

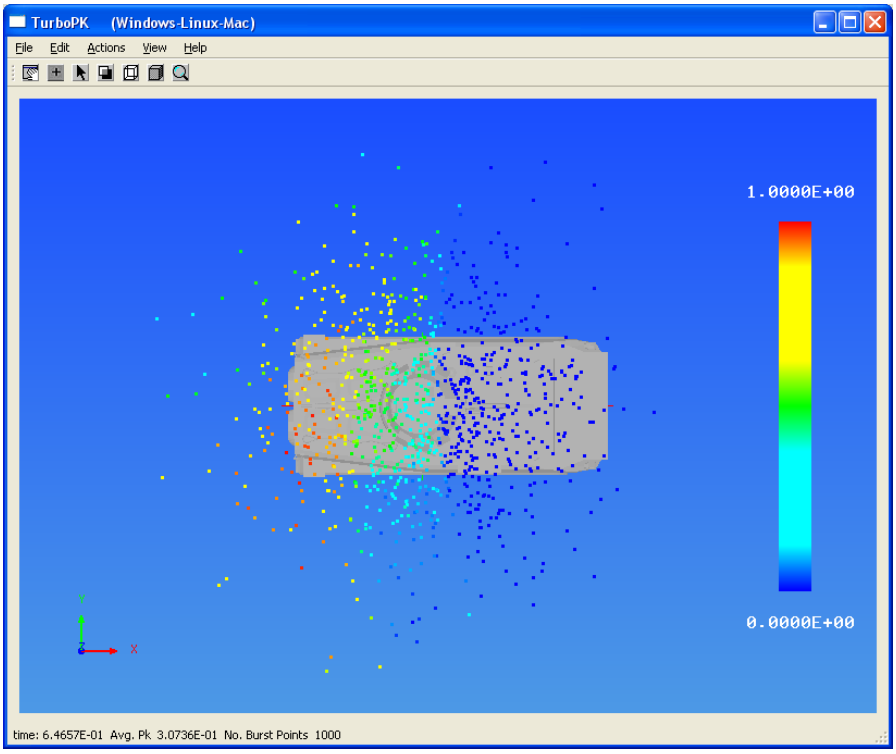

Figure 1. Example TurboPk run.

That kind of computing speed opens up a whole new range of possible SVL applications. A simple example is shown in Figure 2. The dialog box shown in

Figure 2 stipulates the parameters of the Rayleigh burst point distribution in TurboPK. Note the “spin buttons” next to each parameter field. Clicking on these spin buttons increments or decrements the parameter in question by a fixed amount. For example, the centroid coordinates are changed by one meter when clicking on their spin buttons. In Figure 2 the “Centroid X” decrement spin button was clicked twice, so the centroid x value changed from 0.0 (Figure 1) to –2.0 in Figure 2. Overall average P_K jumped from 0.28 to 0.73.

Reducing the samples-per-burst-point to 100 reduces the run time to 0.10 seconds while not changing the overall P_K much. That opens up the possibility of linking the centroid coordinates to the mouse location. As the user scrolls the mouse around P_K would be computed and displayed on the fly.

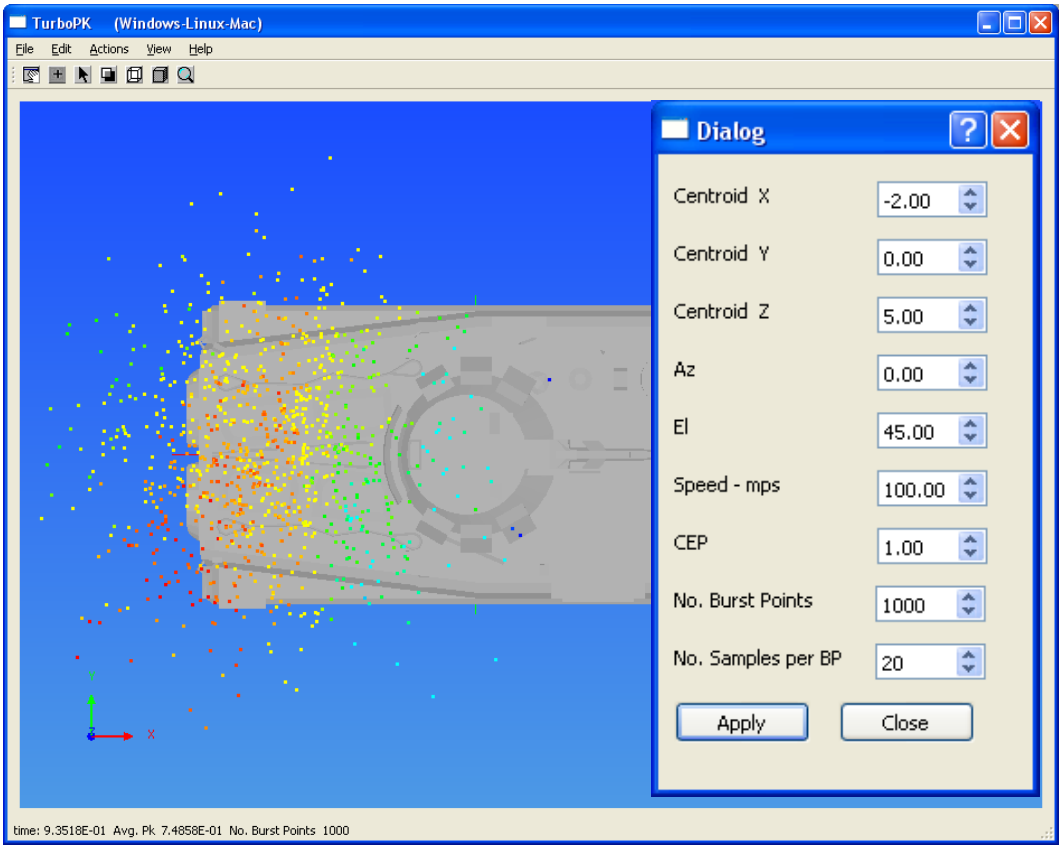

Figure 2. Moving the centroid x.

TurboPK also supports the analysis of projectile (bullet) impacts. Under this option the user specifies an approach angle pair (azimuth and elevation) and the code generates a grid of parallel shotlines that covers the target projection. A projectile impact event is simulated for each of the parallel shotlines and the computational results are displayed as a color map imposed on the target image. Several types of calculations are implemented. One calculation of interest to

armor designers is impact speed on the first vulnerable component seen along a shotline. Ideally that would be zero everywhere, but weight constraints almost always preclude that. Figure 3 is an example parallel shotline calculation for a 12.7 API projectile striking a lightly armored vehicle at 2700 fps. Figure 4 reduces the impact speed to 1900 fps. Run time in both cases is roughly 0.13 seconds for ½-inch shotline spacing.

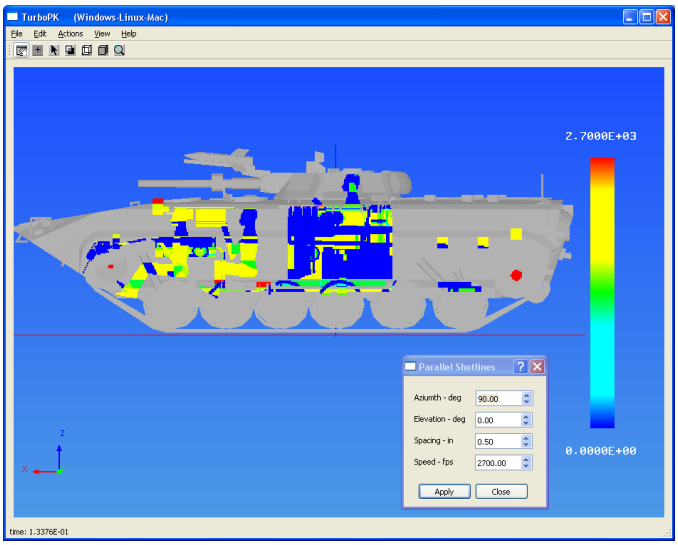

Figure 3. Parallel shotline example – 2700 fps.

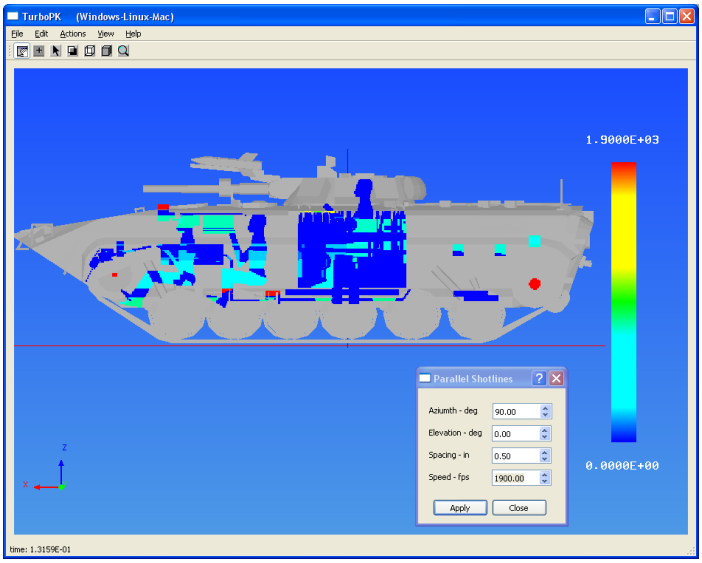

Figure 4. Parallel shotline example – 1900 fps.

With TurboPK it is possible for a user to spin the target around through a range of projectile approach angles and watch the results in near real time. One obvious

improvement would be to enable a user to move components around internally then rotate through a set of approch angles to see whether or not the new arrangement improves things. Another possibility is to compute how much additional RHA would need to be added to each shotline to bring the speed down to zero, then display the results as an “additional area density” map, which would aid armor designers in placing new panels.

TurboPK also has an option for analyzing parallel shotline grids for HEI projectiles. Figure 5 shows an example HEI calculation. The color indicates total target P_K, i.e., a fault tree is invoked after component P_K values have been computed. Shotline spacing is 1-foot, and 20 Monte Carlo repeats are done for each burst point. Run time was just under 1.0 second, so it is not unreasonable to Imagine doing a full 27-view analysis in one minute.

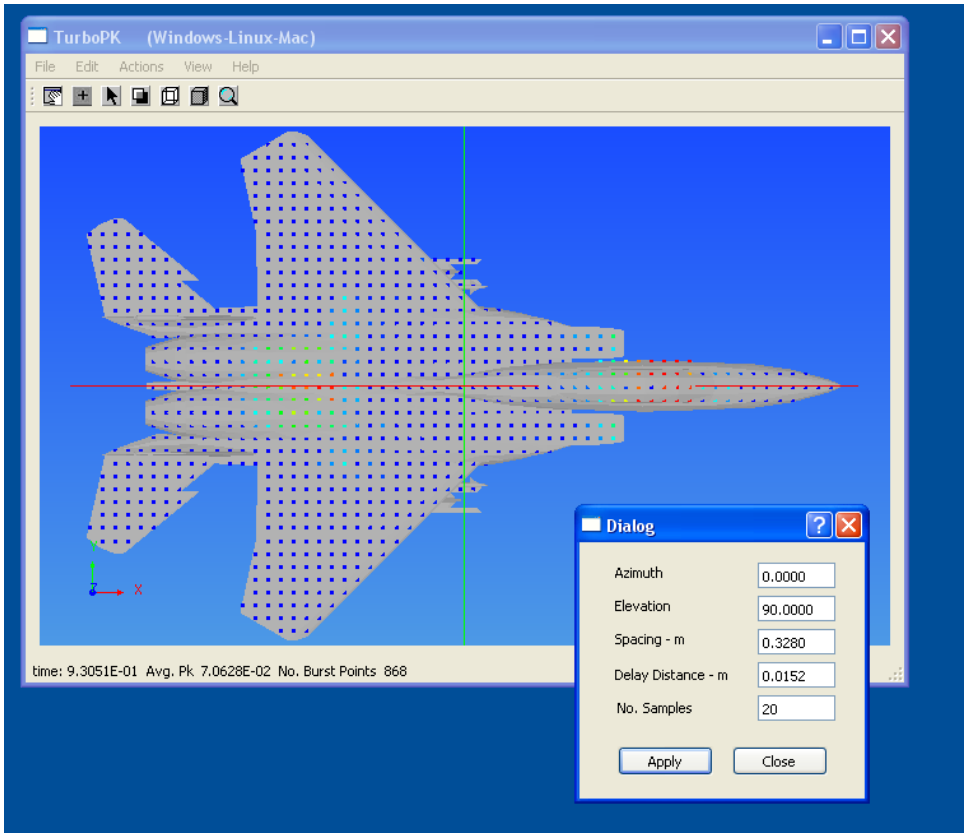

Figure 5. Example HEI analysis.

In its current form TurboPK relies exclusively on triangle geometry. That might change in the future if CPU or GPU vendors add hardware for ray tracing NURBS or parametric surfaces, but for the time being TurboPK will stick to triangles. We do not see that as a limitation as all popular CAD programs support

triangle geometry export as STL (Stereolithography) files. To turn STL geometry models into TurboPK SVL models we provide a companion program called **STL Model Builder**. Figure 6 shows this program in action after the user has loaded a set of 110 STL files for a drone aircraft model. The actual CAD model was done in SolidWorks and the 110 STL files were exported from SolidWorks. The dialog box allows the user to specify / edit SVL properties like Covart material ID, component Pk/h function ID, and so forth. So you can turn STL geometry files into a TurboPK SVL-capable model. TurboPK Pk/h functions and fault trees adhere to Covart-4 standard formats.

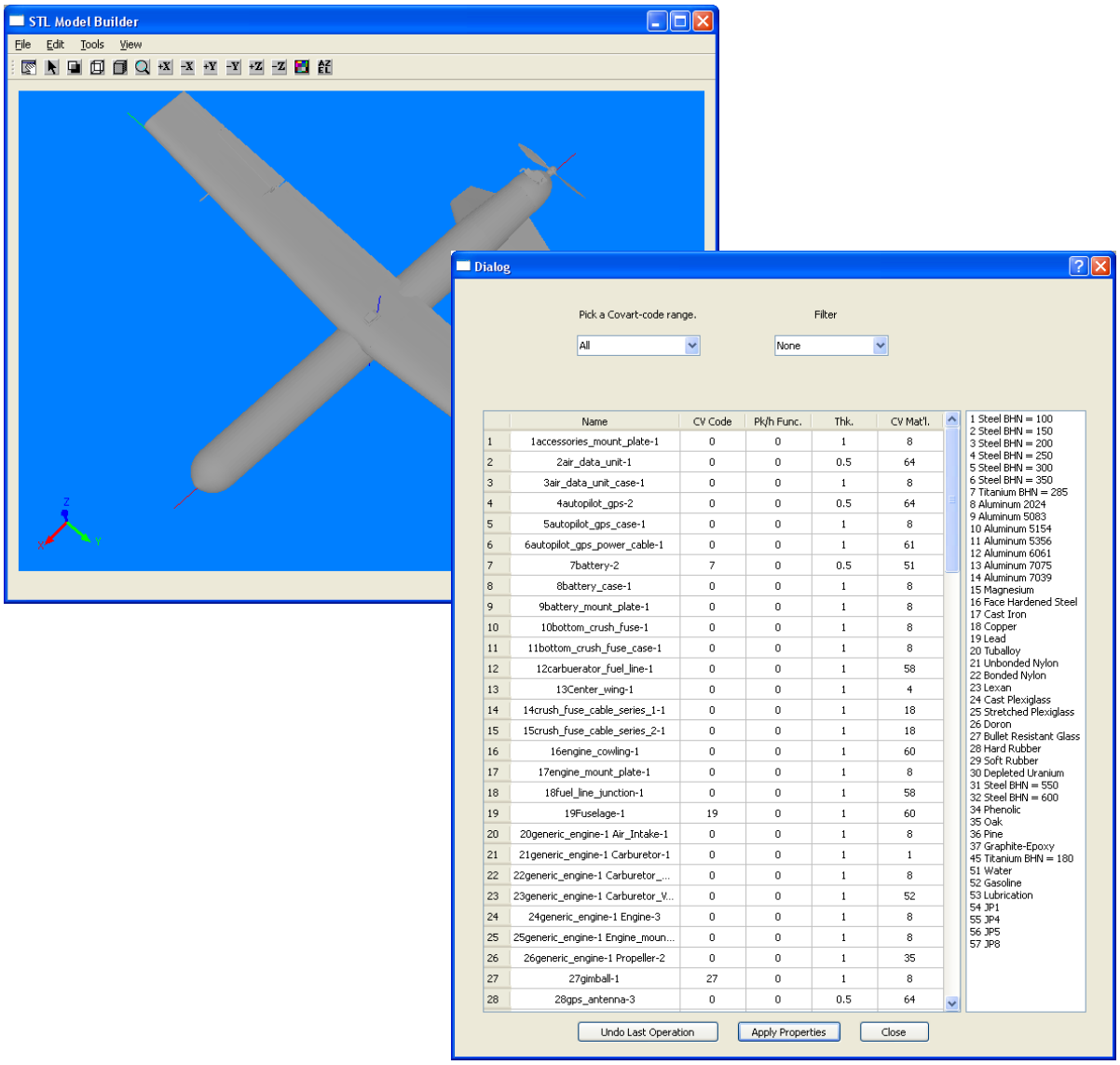

Figure 6. STL Model Builder.