Bullet Ray Vision

Lee A. Butler†
US Army Research Laboratory

Abe Stephens*
SCI Institute
University of Utah

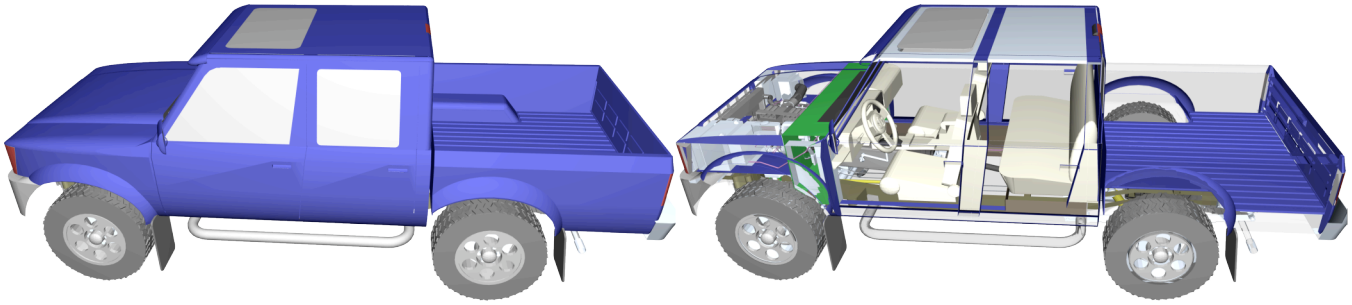

Figure 1. Bullet Vision

ABSTRACT

Prior to 2000 ray tracing was typically considered an expensive computation. As a result, many developers strove to minimize the cost of ray tracing in their applications. Today many older applications can be substantially improved by being re-designed to take advantage of modern, faster ray tracing algorithms. This work examines the integration of modern ray tracing techniques with a classic ballistic penetration algorithm and demonstrates the benefit gained. We also present a unique visualization of the ballistic penetration.

Keywords: Ballistics, projectile, penetration, THOR, ray tracing, visualization.

Index Terms: J.2.7 [Computer Applications]: Physical Sciences and Engineering — Engineering.

1 RELATED WORK

The interactive visualization performed by our prototype makes extensive use of transparent rendering, which has been used for ray trace based massive model visualization [8]. Although the images we produce are similar, the rendering problem is slightly different; in our system opacity is view dependent and computed using a ballistics penetration simulation instead of a constant, arbitrarily assigned opacity value.

2 BACKGROUND

Ballistic penetration simulations are conducted to predict the outcome of one or more *threats* against a *target*. The threat may be a single projectile, or one of many fragments from a source munition. The target is anything in the path of the threat. The term does not imply anything about intent to aim.

The two common objectives of ballistic simulation are to characterize how lethal a given threat is against a particular type of target (lethality) or the ability of a target to survive being subjected to a particular type of threat (survivability). The computation may be performed for a single threat path, such as when planning field experiments or supporting war-game simulations or it may be performed for a large number of threats to look for trends in ballistic performance and explore the set of all possible outcomes. This work focuses on the latter category of analysis.

† butler@arl.army.mil
* abe@sci.utah.edu

2.1 Genesis

The origins of ray tracing for ballistic penetration analysis can be traced back to a contract from the U.S. Army Ballistic Research Laboratory to the Mathematical Applications Group Inc. (MAGI) in 1956. The purpose of the contract was to adapt a technique used in optical lens design for ballistic penetration analysis [1]. As a result, there are strong similarities between the optical rendering process and the ballistic simulation process.

For lens design, light rays are traced from air to the lens surface. At the surface, a shading algorithm determines if the light path is bent due to the change in the index of refraction from the air to the lens glass. The shading algorithm may consider the quality of the glass to determine how much light of each wavelength is passed through the lens to the exit surface. At the exit surface, refraction may take place again and the light ray departs into the new medium.

In ballistic ray tracing, projectiles replace photons. The path of the projectile is referred to as the *shotline*. When the shotline reaches a target surface, a penetration algorithm determines what happens. A variety of factors such as the target and projectile materials, their relative velocity, area of interaction, mass, and incident angle may be used to determine if penetration will occur. If there is penetration, it may or may not be on the original path of the projectile. The amount of projectile and target material along the path as well as the factors mentioned above determine how much (if any) of the projectile arrives at the exit surface, and the new orientation, direction and velocity of the projectile.

Many aspects of optical rendering have their analogues in ballistic ray tracing. Participating media and global illumination both involve scatter as a result of photons interacting with the environment. In ballistic computation fragmentation, and behind-armor debris are similar effects. Reflection becomes ricochet. The ballistic analysis ray generation process is identical to orthographic projection rendering. For a particular ballistic simulation, the analyst/user typically selects a small number of threat directions (called *views*) to compute. To keep execution time of the simulation reasonable, the number of views is typically between 3 and 42.

2.2 Evolution of Applications

The design of many existing ballistic simulation applications separates the geometric interrogation from the ballistic interaction calculations. This separation was driven by two design choices. First, a single threat would be represented by a single shotline.

Second, shotlines and the resultant target intersections would be pre-computed, saved, and thus available for re-use. The ballistic penetration computation is run as a separate step after the shotline process. This design allows simulation using different penetration algorithm parameters (or even different penetration algorithms) while reusing the same shotlines. A byproduct of this design is that shotlines are traced all the way through the target geometry because there is no way of knowing what point along the path the threat is stopped.

A benefit of these design principles is that the development of penetration equations and algorithms is completely separated from the geometric intersection calculations. This modularity means that the ballistician needs to know very little about the nature of the shotline process. For simple penetration algorithms, this is convenient. However, more modern algorithms may generate fragments and other effects that require tracing additional rays. As a result, these algorithms are actually hampered in their design and performance by the architecture of the application.

The following conditions were once true, but are no longer valid:

- Ray tracing is expensive and slow.
- It is cheaper to save the shotlines and reuse them than to re-compute them.
- Penetration algorithms need only a single shotline through the geometry for their computation.

2.3 Change in Simulation Objectives

Over time the goal of ballistic simulations has shifted. From 1960–1980 the emphasis was on looking for trends in results: what are all the possible outcomes for projectile A vs. target B? More recently the emphasis has shifted towards probability analysis: of the possible outcomes of this interaction, what are the probabilities of each and which will be chosen this time? This drive towards sampling the statistical outcome space has driven up the number of shotlines needed to compute a given simulation run.

Like other areas of computing, the ballistic analysis sector has seen a shift over the last 20 years from ballistic analysts as software and algorithm implementers to one where they are software package users. The community has taken advantage of increased speed in computers. As a result, very few analysts reuse shotlines, even when it would be possible. It is more convenient to rerun the entire simulation from start to finish than to identify the shotline file and reuse it.

The combination of acceleration in ray-tracing algorithms with the changing nature of analysis design makes reusing the ray path less attractive than it was a quarter century ago.

2.4 THOR Penetration Equations

Ballistic penetration equations are developed from data obtained from physical test measurements and experiments. The THOR equations [2] are an example of a ballistic penetration algorithm. They roughly correspond to the Phong [3] illumination model. From [2] the THOR equations are:

(eq 1) $V_{50(ft/sec)} = 10^{c} \bullet (h_{(in)} \bullet A_{f(in^2)})^{\alpha} \bullet W_f^{\beta}{}_{(grains)} \bullet \sec \theta^{\gamma}$

(eq 2) $V_{r(ft/sec)} = V_{(ft/sec)} - 10^{c} \bullet (h \bullet A_f)^{\alpha} \bullet W_f^{\beta} \bullet \sec \theta^{\gamma} \bullet V^{\lambda}_{(ft/sec)}$

(eq 3) $W_{r(grains)} = W_f - 10^{c} \bullet (h \bullet Af)^{\alpha} \bullet W_f^{\beta} \bullet \sec \theta^{\gamma} \bullet V^{\lambda}$

Equation 1 calculates V_{50}: the velocity where there is a probability of 0.5 that the projectile penetrates the material of the target. Equation 2 calculates the residual velocity of the projectile after passing through the target. Equation 3 calculates the residual weight of the projectile. The terminology used for the projectile is usually fragment; hence, the subscript f in the equations.

The terms c, α, β, γ and λ are coefficients for the individual materials and are empirically derived. From [2], we have the values in Table 1 as coefficients for 2024-T3 aluminum:

Eqn	Coefficients				
	c	α	β	γ	λ
V_{50}	6.185	0.903	-0.941	1.098	
V_r	7.047	1.029	-1.072	1.251	-0.139
W_r	-6.663	0.227	0.694	-0.361	1.901

Table 1. Material Coefficients for 2024-T3 Aluminum

The other terms in the equations are velocity V, thickness h, impact area A_f, weight W_f and angle of obliquity θ.

An example from [2] considers a 200 grain fragment impacting a 0.08-inch-thick 2024-T3 aluminum plate at an angle of 20° and a velocity of 5000 ft/sec, with an impact area of 0.25_{in^2}. In this instance equations 1 through 3 become:

$$V_{50} = 10^{6.185} \cdot (0.08 \cdot 0.25)^{0.903} \cdot 200^{-0.941} \cdot \sec(20\text{deg})^{1.098}$$
$$= 327.5 \text{ ft/sec}$$

$$V_r = 5000 - 10^{7.047} \cdot (0.08 \cdot 0.25)^{1.029} \cdot 200^{-1.072} \cdot \sec(20\text{deg})^{1.251} \cdot 5000^{-0.139}$$
$$= 5000 - 225$$
$$= 4775 \text{ ft/sec}$$

$$W_r = 200 - 10^{-6.663} \cdot (0.08 \cdot 0.25)^{0.227} \cdot 200^{0.694} \cdot \sec(20\text{deg})^{-0.361} \cdot 5000^{1.901}$$
$$= 162.8 \text{ grains}$$

After impact and penetration, the fragment is travelling at 4775 ft/sec, and is 162.8 grains. If the V_{50} velocity had been greater than the fragment velocity, the simulation would not perform penetration computation.

3 BENCHMARKS

For this work a reference implementation of THOR was created in the BRL-CAD CSG-based ray-tracing engine. This engine was selected because it is used by many ballistic analysis applications. THOR penetration was run on each of four views: top, right, front, and an off-axis view along an azimuth of 35° and an elevation 25° from the target. The target chosen was a simple pickup truck (see Figure 2). Each view consisted of a 512x512 grid of rays with a single shot per grid location. All tests were run on the same hardware: a MacBook Pro running a 2.33GHz Intel Core Duo CPU. Each test was run three times.

The first test run represents the traditional execution flow, where rays are passed all the way through the geometry before penetration equations are performed. The second test was run with the penetration equation computation interspersed with the geometry intersection.

The average run times in seconds for each execution are shown in Table 2. The first row is for the classic, full ray traversal before penetration computation. The second performs penetration as the ray traverses the geometry.

Overall, computing penetration during ray traversal resulted in a 27% speed improvement. For front views the difference is a 50% reduction in runtime.

Figure 2. Target geometry

This is because the projectile is stopped significantly before passing through the rest of the target. In other views, the projectile passes through a significant amount of the target, and less speedup is achieved. There is no significant difference in the timing for the right view.

Ray depth	Top	Right	Front	A35
Full	9.22	10.52	6.27	12.36
Partial	7.30	10.54	3.15	7.60

Table 2. Computation Time (seconds)

4 Bullet Vision

When such computations are performed the obvious desire is to visualize the results of the ballistic penetration. The projectile is deemed to stop if its velocity is below the V_{50} threshold or if the weight falls to zero. To facilitate visualization, the penetration equation routine was modified to perform a Lambertian shading of the object in which the projectile stopped. This shading allows the user to see into the target object as if looking with bullets instead of photons. Figure 2 shows the geometry used for the ballistic simulation. Figure 3 shows the results of rendering the objects in which the projectile stopped.

For these renderings, all objects in the model were treated as being made of 2024-T3 aluminum. For an actual ballistic simulation the coefficients for each of the materials present should be used for their respective parts.

One of the interesting qualities of the image is that it visually reveals that penetration is sensitive to the impact obliquity. For example, the projectile has no trouble penetrating the front of the left front fender flare over the tire. On the rear of the fender, where the incident angle becomes very small, the projectile does not penetrate. This leaves the rear of the fender in the ballistic rendering visible. Figure 4 shows the same view but with the initial velocity set higher.

5 Interactive Rendering

To further improve performance, the Manta ray tracer [7] was substituted for the BRL-CAD ray tracer. This allowed the ballistic penetration simulation to be visualized interactively. For additional context, transparency was used to shade portions of the geometry that had been penetrated. The simulation with online visualization runs at an average 3.9 fps over the camera path pictured in the video on an Intel Core 2 2.66 Ghz processor with four cores. Solving the THOR penetration computation with 14 exponential calculations after each intersection was particularly expensive. The current version of the code uses ray packets but does not use SSE instructions during the THOR calculations. SSE code paths are used throughout other components of the renderer.

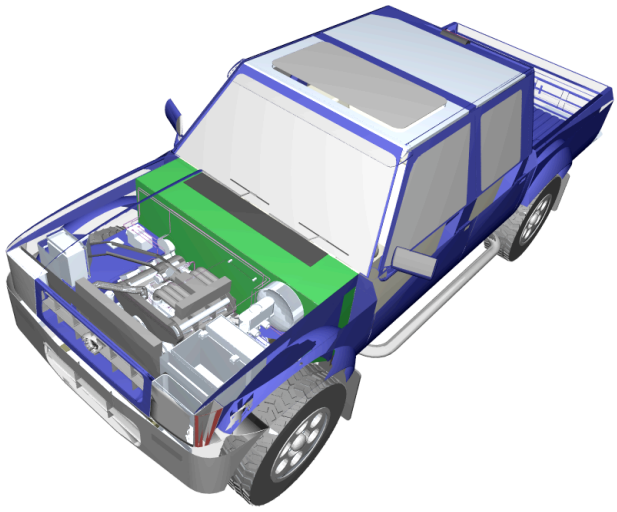

Figure 3. Bullet vision rendering

The interactive visualization used a fast-building BVH acceleration structure [9] because it was readily available within Manta. BVH traversal time was the dominant component of the computation. Restarting BVH traversal is necessary between intersections since leaf nodes might overlap even if an ordered traversal is used.

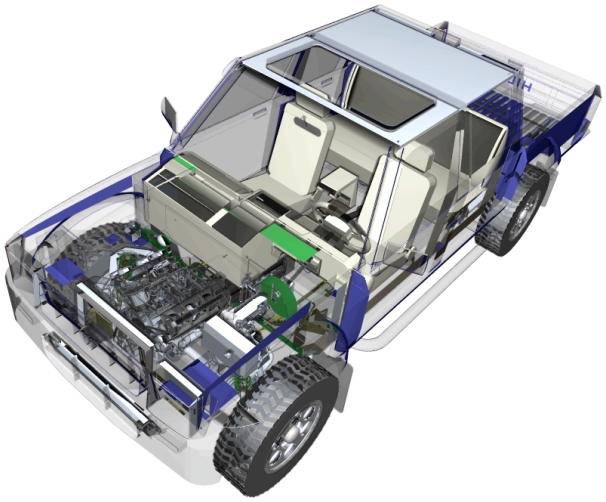

Figure 4. Bullet vision with higher initial velocity

6 Conclusion

The architecture of many legacy ray-tracing applications inhibits the speed of execution. Design choices that were valid 10 years ago or more no longer apply. In particular, excessive partitioning of the computation can have a severe impact on performance. For efficiency, most optical-rendering engines intertwine the geometric intersection computation with rendering. Scientific applications such as ballistic analysis stand to benefit from the same structure. Modern packet tracing engines can provide sub-

stantial benefit to non-rendering applications, and can produce online visualization as a by-product of the scientific computation.

6.1 Future Work

The rendering of the last object hit is useful but could be improved by providing the viewer cues as to how far into the object the projectile stopped.

The kd-tree acceleration structure used by other transparent rendering systems [8] may offer advantages over BVH during traversal because leaves are necessarily traversed in order. This would permit the transparent renderer to collect many intersection points along a ray without restarting the traversal from the top of the tree. A more detailed analysis is a topic of future research.

In our system the THOR penetration calculation with 14 exponential operations was the second most significant portion of the runtime. There is opportunity for eliminating or pre-computing some terms. In addition, since the THOR equations are a mathematical fit to experimental data, a significant benefit may be achieved by fitting a less-costly function to the data.

REFERENCES

[1] A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles, MAGI-6701, AD847576, August 1967.

[2] Robert E. Ball. The Fundamentals of Aircraft Combat Survivability Analysis and Design.—2nd ed.; AIAA Education Series. American Institute of Aeronautics and Astronautics, Inc., Reston, VA, 2003, 320-326.

[3] Foley, van Dam, Feiner, and Hughes. Computer Graphics – Principles and Practice.—2nd ed.; C. Addison Wesley, 1997.

[4] Ingo Wald. Realtime Ray Tracing and Interactive Global Illumination. Computer Graphics Group, Saarland University, 2004.

[5] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005), 24(3):1176-1185, 2005.

[6] Alexander Reshetov Omnidirectional Ray Tracing Traversal Algorithm for kd-trees. In Proceedings of the IEEE Symposium on Interactive Ray Tracing, 2006, 57–60.

[7] James Bigler, Abe Stephens, and Steven G. Parker. Design for Parallel Interactive Ray Tracing Systems. In Proceedings of the IEEE Symposium on Interactive Ray Tracing, 2006, 187-196

[8] Abe Stephens, Solomon Boulos, James Bigler, Ingo Wald, and Steven G. Parker. An Application of Scalable Massive Model Interaction using Shared Memory Systems. In Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization, 2006 pp 19-26

[9] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Heirarchies. In ACM Transactions on Graphics vol 26 number 1