Real Time Ray Tracing for Physically Simulated Scenes

Colin Fowler, Steven Collins and Michael Manzke,
GV2 Group, School of Computer Science & Statistics,
Trinity College Dublin

Introduction

The physics engine has become an ubiquitous component of graphics simulations in recent years. As computing object collisions is expensive, physics engines employ a range of techniques to alleviate these costs. Similarly, ray-tracing is computationally expensive and has been subject to much research aiming to increase its speed to make it viable in real-time simulation. Currently, there is no interaction between the ray-tracer and the physics engine. We take an aggregate approach, treating the system as a whole and investigate areas where the physics engine can provide information to accelerate the ray-tracer.

The physics engine has knowledge of the state of all objects in the simulation in terms of location, velocity and acceleration. We propose that such information will accelerate the ray-tracer which operates on the same objects.

Sleeping

Sleeping is a physics engine acceleration technique that clusters spatially coherent physics objects into groups known as "islands". When the linear and angular velocities of all objects in an island have fallen below a threshold, it is marked as sleeping. It is no longer necessary to check these objects against any other sleeping object for collision or perform other calculations on them such as applying gravitational forces. Sleeping dynamic objects therefore act like static objects in this state. If an active object enters the sleeping island's AABB, the islands is reactivated for more low-level, higher precision collision detection. Relieving the engine of such calculations can greatly decrease the computational overhead of a physics simulation.

Integration

In a simulation with multiple copies of the same rigid body, instancing may be used in order to reduce memory overhead. Although this greatly improves ray-tracer speed, there is still overhead in building the top-level BVH and transforming the rays. In typical physics simulation involving gravity and friction, moving objects naturally tend to come to rest. Without constant energy injection, all initially moving objects will eventually come to a stop. As previously noted, such rest states offer an opportunity for acceleration in the physics engine through sleeping. A similar technique, which we call "integration", can be applied in the same scenarios to accelerate the renderer. As static objects are fixed, their local coordinate system can be set to match the world coordinate system. This means that no ray transformations are necessary when intersecting them. Dynamic objects that have come to rest may be thought of as a special case. Instead of transforming each ray that potentially intersects these objects, we transform the now resting object's primitives into the world coordinate system and then add this geometry to the static geometry. This has two effects:

- Reducing the number of primitives in the top-level BVH, therefore lowering top-level acceleration data structure (ADS) build time and the time spent in the top-level ADS.
- Reducing the number of ray transformations required.

When a previously integrated object is moved, the current static ADS becomes invalid. We can switch back to the original static ADS and re-add the previously integrated objects back into the dynamic hierarchy to regain a valid state. Objects still in a sleeping state may then be integrated again. If the original data structures are still stored in memory, the performance cost of switching or "reactivation" is very small. Integration, however, has a cost as it requires an ADS build. In order for integration to be beneficial, the cycles spent performing the integration build must at least be recouped by the ray-tracer before a reactivation occurs. In a dynamic system with many objects moving, deactivation and reactivation events may follow closely.

Deciding to Integrate

An island should be integrated when $C_i < t_n$, where C_i is the ADS build cost for integration and t_n is the gain achieved by integration. C_i is not precisely known but can be approximated easily based on previous ADS builder performance with known primitive input counts. t_n may be costly to calculate or completely unpredictable due to user input. It can therefore be approximated based on several previous integration/reactivation events. In order to increase the accuracy of t_n, we track events where integration is precluded from happening by our current heuristic values and the costs for integration had it occurred. This allows t_n to converge more rapidly.

Figure 1 : Fairy Forest. Two high polygon dynamic characters (a fairy and a dragon fly) move through a mostly static scene.

Reducing C_i presents more opportunities for integration and therefore better renderer performance. We take advantage of our pre-built static geometry to speed up the insertion of deactivated objects into the ADS. A from-scratch rebuild of a tree-based ADS using an *O(nlog n)* builder has a running time of *O((k+n)log(k+n))* where *n* is the number of triangles in the original static scene and *k* is the number of triangles in the complete set of deactivated objects.

By using the static geometry's ADS as a starting point, we reduce this operation to *O(k log(k+n))*. We perform *k* insertions into the tree, each insert taking *O(log(k + n))* time.

When compared to full rebuilds in practice, insertion yields similar ray-tracing speeds although building several orders of magnitude faster.

Geometry Caching

If an island that was previously integrated reactivates, we cannot use the current static scenery ADS any longer as it contains the now-moving objects. We may switch back to the original pristine scenery. However, it possible that the world-state is now similar to a previous configuration where some or all of the still sleeping islands were sleeping. To exploit this, we introduce geometry caching.

The cache contains snapshots of the static scene geometry taken before each new island integration. Each snapshot is also marked with the islands that have been integrated into the snapshot. Initially the cache contains only the pristine static geometry. This is marked as containing no integrated islands and is the "worst case" fallback position.

When an island has gone to sleep, we first save our current static geometry as an object in the cache, marked with the list of islands integrated. We then integrate the new sleeping island into the static geometry. If a previously sleeping island wakes up, we scan the cache and remove any cache objects marked as containing that island as these are now invalidated.

When a static ADS is added to the cache, we also store in the cache entry a list of the islands integrated into that ADS. The ADS builder requests from the cache an ADS with a list of the islands it needs to integrate. If a cache entry is found that contains all of the requested islands already integrated, this means that the builder has no further work to do. We refer to this as a full cache hit. If the cache contains an entry with only a subset of the requested islands integrated, we refer to this as a partial cache hit. As this partial hit contains already integrated islands, the builder need only insert the non integrated islands.

Partial cache hits greatly increase the performance of the cache with 52% of cache requests providing geometry that contains between 90% and 99% of the islands requested already integrated.

Results

Our results (see Figure 2) show that deactivation and integration can yield substantial frame-rate increases in the ray-tracer as a result of lowering ray-transformation overheads and the cost of building the top-level data structure over the dynamic objects.

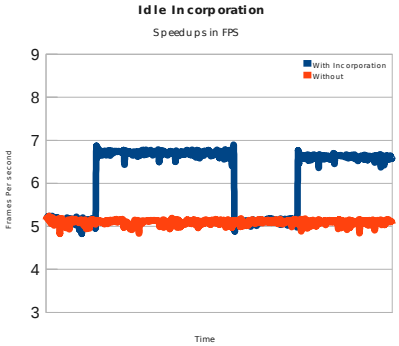

Figure 2 : An experiment with the deactivation and reactivation of 8000 objects in the Sponza Atrium scene. By integrating idle objects frame-rate improves substantially.