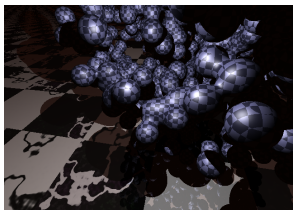
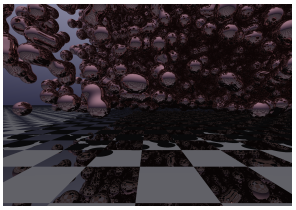
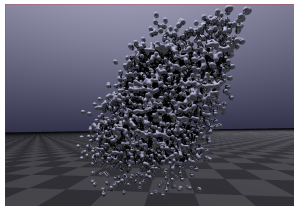
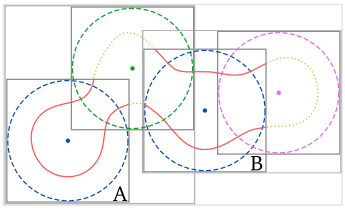
BVH for Efficient Raytracing of Dynamic Metaballs on GPU

Olivier Gourmel, Anthony Pajot, Loïc Barthe, Mathias Paulin*
IRIT, University of Toulouse, France

Pierre Poulin†
Université de Montréal

(a) Raytracing of metaballs, with shadows and 3 levels of mirror reflections: (left) 1000 metaballs at 13 fps at a 640 × 480 resolution without multi-sampling; (center) 200K metaballs at 0.3 fps at 1600 × 1200 with 4 rays per pixel; (right) 3000 metaballs at 1.1 fps at 1600 × 1200 with 4 rays per pixel.



(b) BVH with partial MCS in red and full MCS outlined in red and dotted orange.

1 Introduction

Metaballs [Bloomenthal 1997] are effective to represent fluids and similar complex and deformable geometries, but their implicit nature makes difficult their visualization in real time. A common strategy is to tessellate the resulting isosurface and to render it on GPU, but it scales poorly as the number of metaballs increases. Kanamori et al. [2008] efficiently raycast thousands of metaballs without intermediate representations. Their method assumes that rays are shot from a single viewpoint, thus preventing secondary effects (no shadows, reflections, etc.), and is limited to polynomial density functions. We propose to exploit the culling capacity of dynamic bounding volume hierarchies (BVH) [Wald 2007], the secant method for ray-surface intersection, and CPU-GPU parallelism to alleviate the restrictions of their method. This results in a general raytracing method, allowing arbitrary ray intersection (visibility, shadow, reflection, refraction, etc.) with metaballs of any finite-support at interactive performances.

2 Raytracing Metaballs on GPU

Definitions: A set of metaballs creating a connected surface is defined as a *metaball connected set* (MCS). A dynamic scene can consist of a varying number of MCSs, with completely changing configurations at each frame.

BVHs for Metaballs: While a BVH can be efficiently built [Wald 2007], it does not scale well with large MCSs, as the BVH would contain only a few leaves, each encompassing a large MCS composed of many metaballs. Moreover at any given point, the majority of the metaballs do not contribute to their MCS, because of their finite support. Each metaball has a bounding box defined by its support, and will be included in one leaf node in the BVH. During construction of the BVH, all metaballs potentially contributing to the MCS of one metaball belonging to a BVH node, are efficiently duplicated in this node. Once a leaf node is reached, other conservative tests (sphere-sphere and maximum combined spheres) are performed to remove some non-contributing metaballs. As the number of duplicated metaballs cannot be predicted, dynamic allocation is needed, hence the BVH is constructed on the CPU. In Figure 1(b), the leftmost leaf node of the BVH contains two metaballs (the green one being duplicated), the uppermost leaf node has three metaballs, etc. For leaf nodes A and B, the red curves indicate the shape of the partial MCS, in their respective leaf nodes, that will be tested for intersection.

Ray-Metaball Intersection: Our method runs in two steps. First we traverse the BVH and search for a point P_i along the ray and inside the first leaf node intersected by the ray. The projections on the ray of the center of each (non-duplicated) metaball in this leaf node are good candidates for intersection. We then check the value of the density function at these projected points and keep the nearest among those inside the leaf partial MCS. To compute the intersection point, we need a point P_o outside the MCS, for instance the origin of the ray. The intersection point is found by using the secant method with the interval $[P_o, P_i]$ as an initial guess. The secant method has some advantages over other iterative methods: it is very robust and quickly converges (in about 10 iterations, implying 10 evaluations of the density function). Kanamori et al. [2008] use Bezier clipping, which in some cases needs slightly fewer evaluations, but the number of evaluations increases with the degree of the polynomial density function.

CUDA Implementation: We store the metaballs and BVH nodes in textures to benefit from texture cache during random memory access due to BVH traversal. Animation of the metaballs and rendering are done on GPU, while BVH construction is done on CPU. This implies memory transfers, but also an inherent CPU-GPU parallelism that is exploited by using streams and asynchronous execution and transfers. The GTX-280 GPU allows transfer and computation simultaneously on GPU and CPU. By using buffers and forward computation of up to two frames, the animation is computed and downloaded to the CPU while the BVH begins to build, and rendering is performed while the BVH is completed and copied on the GPU.

3 Results

We introduced techniques to robustly and efficiently raytrace large numbers of metaballs at interactive framerates, while still capturing secondary effects such as shadows and mirror reflections (Fig. 1(a)). Due to the finite support of a metaball, metaball density increases will directly affect performances. Moreover, BVH construction can become a limiting factor when the number of metaballs grows over 200K in our tests. Future work will focus on those issues.

References

BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann, August.

KANAMORI, Y., SZEGO, Z., AND NISHITA, T. 2008. GPU-based fast ray casting for a large number of metaballs. In *Eurographics*.

WALD, I. 2007. On fast construction of SAH based bounding volume hierarchies. In *Symposium on Interactive Ray Tracing*.

*email: {gourmel, pajot, lbarthe, paulin}@irit.fr
†email: poulin@iro.umontreal.ca