Interactive Ray Tracing of Point-based Models

Ingo Wald and Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany

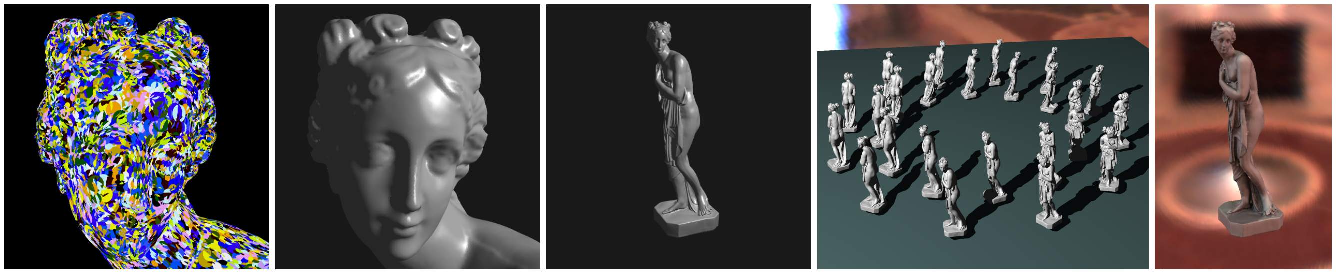

Figure 1: Interactive ray tracing of point-based scenes: a) ”Iphigenia” head, each point represented by a disc. b) The splats blendet to an implicit function and intersected using an acceleration structure ($6.8 fps@512\times512$ pixels). c) The full model ($15.9 fps@512\times512$ pixels). d) A complex scene of 24 Iphigenias (24 million points) with phong shader and shadows ($\sim 2 fps@640\times480$ pixels). e) Iphigenia, displayed interactively with a (precomputed) global illumination solution ($\sim 4 fps@400\times600$ pixels). All frame rates measured on a single PC.

1 Introduction

In recent years, point-based methods have gained significant interest, as their simplicity and independence of connectivity make them a simple and powerful tool in both modelling and rendering. Still, their use for high-quality and photorealistic rendering is still in its infancy, in particular for interactive applications. This paper sketches new developments in interactively ray tracing point based models, including both complex models and photorealistic shading.

2 System Overview

Our method consists of several, inter-playing ingredients:

Part I: Implicit Surface Model. Given a model with only (singular) points and normals, computing an intersection with a ray requires to either traverse “thick” rays or to construct a non-singular surface out of those points. We use the implicit surface model proposed by [Adamson and Alexa 2003]: Each point p_i is equipped with a spherical support $w_i(x)$ (with radius r_i). An implicit function $f(x)=0$ is then defined as $f(x)=(x-\bar{P}(x))\bar{N}(x)$, where $\bar{P}(x)=\frac{\sum_i w_i(x)p_i}{\sum_i w_i(x)}$ and $\bar{N}(x)=\frac{\sum_i w_i(x)n_i}{\sum_i w_i(x)}$ are the weighted averages of the positions resp. normals of all points (p_i,n_i,r_i) overlapping x.

Part II: Choosing optimal “Splat” Radii. Instead of having the user decide on the “optimal” splat radius, we used the method by [Wu and Kobbelt 2004]. This method computes a radius for each point such that a) the model’s surface is completely covered, and b) the radii are minimized (see Figure 1a). This not only avoids holes in the model, but additionally minimizes the overlap of different splat supports, which is crucial for fast surface intersection.

Part III: Fast Surface Intersection. Given an implicit surface model, we compute the ray-surface intersection using a two-step method: First, we subdivide space using a kd-tree, and determine whether a voxel may contain the surface at all. For those voxel-s, we store a list of all splats whose support overlaps the given voxel.

Once a ray reaches a voxel, it has to be intersected (only) with the splats contained in that voxel. Instead of an iterative procedure (like Newton-iteration), regularly sampling the ray interval inside the voxel has shown to work best: Taking N samples $t_0,t_1,\ldots$ along the ray, there is a surface intersection if there is an i with $sign(f(t_i))\neq sign(f(t_{i+1}))$, and the hitpoint is linearly interpolated between t_i and t_{i+1} depending on $f(t_i)$ and $f(t_{i+1})$, respectively.

In practice, we use a fixed number N of sample points, which allows for computing $f(t)$ for all t_i in parallel using a very fast data-parallel SIMD implementation. As the kd-tree already encloses the surface quite tightly, even a small $N=4$ in practice reaches good results.

Part IV: Efficient Ray Traversal For building the kd-tree, we use a kd-tree construction method that is specifically targeted towards point-based methods, and which aims at enclosing the surface as tightly as possible. This results – almost independent of model size – in less than two ray-surface intersection per ray.

Traversal steps are *significantly* cheaper than even the highly optimized voxel intersection method described above, and total rendering cost is usually still dominated by the surface intersection cost. As the latter is almost constant, the total rendering cost depends only very weakly on model complexity, yielding comparable performance for both a 32,000-point version and the 1-million-point version of the Iphigenia model.

3 Results and Conclusion

Figures 1b+c show two views of the Iphigenia model (1 million points) with pure ray casting, running at 4.5 and 8.7 frames per second (at 512×512 pixels), respectively. As a stress test, Figure 1d shows a scene with 24 such statues in various poses, totalling 24 million points. Even with additional shadows, we achieve interactive performance of 2.1 frames per second at 640×480 pixels. All experiments are performed on a *single* dual-2.4GHz Opteron PC.

Finally, Figure 1e shows the Iphigenia model with a specially designed method that precomputes global illumination in a directionally dependent way. Using this method we can render this model with direct and indirect illumination, shadows, and highlights at ~ 4 frames per second at 400×600 pixels.

In summary, we have sketched a complete interactive point-based ray tracing framework that – on a single PC – allows for interactive ray tracing performance even for highly complex point-based models and sophisticated ray traced effects.

References

ADAMSON, A., AND ALEXA, M. 2003. Ray tracing point set surfaces. In *SMI ’03: Proceedings of the Shape Modeling International 2003*, IEEE Computer Society, Washington, DC, USA, 272.

WU, J., AND KOBBELT, L. 2004. Optimized sub-sampling of point sets for surface splatting. In *Proceedings of Eurographics 2004*, 643–652.