TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

Validierung einer Gathering-Strategie zur Szenendiskretisierung für Partikeldaten im Kontext der Berechnung von ambienter Verdeckung

Maximilian Richter
(Geboren am 8. April 1993 in Schleiz, Mat.-Nr.: 3802290)

Betreuer: Dipl.-Medieninf. Joachim Staib

Dresden, 23. Januar 2018

Aufgabenstellung

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Validierung einer Gathering-Strategie zur Szenendiskretisierung für Partikeldaten im Kontext der Berechnung von ambienter Verdeckung

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 23. Januar 2018

Maximilian Richter

Kurzfassung

Abstract

Inhaltsverzeichnis

1 Introduction

1.1 Motivation

Particle-based simulations of high depth complexity require refined rendering techniques for visualization. Local lighting lacks visual cues for depth and occlusion and therefore introduce perception issues. Global illumination allows to visualize complex structures in particle data, that otherwise would remain unseen. For real-time rendering less expensive approximations like ambient occlusion are used. Ambient occlusion is especially useful for visualization, as it assumes no parameters for illumination and by that introduces no new information. Instead it mimics indirect diffuse light by calculating the level of occlusion for visible surfaces. The calculation of the ambient terms is still costly, but several methods exist to approximate it. One of them, Voxel Cone Tracing, uses an discretized voxel representation of the geometry. This requires a method to transfer particles into a voxel grid. Scattering iterates all particles and adds their partial contributions to the intersected voxels. Parallel scattering processes the particles simultaneously. Synchronization has to assure correct summation of the contributions whenever multiple spheres intersecting the same voxel cause race conditions. As synchronization increases, parallelism decreases. Therefore scattering performs the worse, the more dense the particle data is.

1.2 Task

To find a particle voxelization technique that better fits the parallel nature of graphics processing units a gathering approach is validated. Gathering determines for each voxel which particles it intersects with. Contrary to scattering each voxel is processed by an individual thread. This ensures that all write operations on one voxel happen sequentially and require no synchronization. For efficient particle search around voxels an acceleration data structure is required, as exhaustive search on all particles has $O(n^2)$ complexity. By sorting particles into spatial bins the search can be limited to near particles. The data structure has to be GPU based, as transferring particles to CPU and back would waste bandwith and synchronization would reduce performance. Starting from an unordered list of particles an algorithm is designed to first build a hashed uniform grid and then use it to calculate the voxel densities. To compare

the gathering approach against scattering both construction time and memory usage are measured for particle clouds of variable size and density. The created voxel data structure is used to calculate ambient occlusion using voxel cone tracing. To evaluate the quality of the proposed solution a ground truth renderer using ray-casting is implemented. The influence of both voxelization and voxel cone tracing on the determined error is examined.

1.3 Outline

The work is organized as follows. In chapter 2 related works are discussed. First concerning acceleration data structures on particles, followed by voxelization techniques and finally ambient occlusion for particle data. Chapter 3 presents the proposed gathering based voxelization technique. Chapter 4 details the techniques used to calculate ambient occlusion. In chapter 5 implementation details are given. Chapter 6 finally presents the results of the work.

2 Related Work

2.1 Ambient Occlusion

Zhukov [19] introduced ambient occlusion as an ambient light illumination model to reproduce effects of indirect light. The assumption is that the more open the scene around a point is, the more indirect light reaches this point. To determine the level of occlusion of a point, it's hemisphere has to be sampled in every direction. The more rays intersect close geometry, the more occluded the point is. Due to it's computational cost naive sampling is unsuitable for interactive visualization.

[21] pre-computes ambient occlusion using a limited amount of rays on the CPU or using shadow maps on the GPU. For rendering the ambient occlusion terms are then provided as color values to the vertices. As the pre-computation takes minutes, this implementation only works for static scenes.

To allow scenes with moving objects [22] converts all vertices to disc-shaped occluder elements. Then for each vertex the occlusion is calculated by adding the form factors of the other occluders. As the ambient occlusion term is calculated per vertex, detailed meshes are required. The pre-computation prevents this from being suitable for dynamic objects.

For scenes with dynamic objects it is necessary to recompute ambient occlusion per frame without pre-processing. [29] introduced screen space ambient occlusion. Instead of sampling polygonal meshes itself first the scene is rendered into a normal and a depth buffer. Each pixels sourrounding pixels are then sampled from this buffer to determine an approximated occlusion factor. ...

[30], [31], [32]

While these methods provide fast results, they also introduce errors by only considering screen space regions. Object space methods avoid this by directly sampling the geometry. -> raycasting ground truth

[25], [26], [27]

[9], [10] + [23]

2.2 Particle Search Data Structures

Gathering requires an efficient method to retrieve particles in a given region of space. Neighboring particle search is related to physical simulation of particles, where movement is determined by the properties of surrounding particles.

[11] was first to present entirely GPU based particle simulations. As no general purpose compute API was available it was implemented using OpenGL shaders. To create a three-dimensional grid, particle indices were stored in the RGBA channels of a two-dimensional texture. Because only four particles can be stored per grid cell, a small cell size has to be chosen to avoid artifacts. This leads to significant amount of memory occupied by empty cells, as SPH simulations fill the simulation volume in a non uniform way. To avoid reduced performance due to high memory transfer [6] proposed an adaptive uniform grid for variable sized volumes. Here the computational region is sliced into planes of voxels that are stored as texels of a texture. To avoid the wasted memory associated with a high amount empty cells the size of each plane is determined in advance, considering only regions that actually contain particles.

Another method using textures to sort data into spatial bins is presented by [14]. The bins are specified as 2D texture coordinates into a texture array. Each bin is composed of the texels that share it's 2D coordinate. A distinct texture counts the elements in each bin and determines which slice provides the next texel to store the input value in. As this implementation only allows 2D data and the texture array's depth limits the number of items it does not suit this thesis' task.

The CUDA SDK [5] evaluates the assignment of particles to grid cells with atomic operations against sorting. Atomic operations are used to update the number of particles per cell and in a second array the indices of these particles. With sorting for every particles it's cell is calculated as a hash. Using radix sort the particles are then sorted based on their hashes. Finally for each cell it's start in the sorted list is calculated. Due to memory coherence and reduced warp divergence they found that sorting achieves highest performance. Additionally it was less sensitive to dense and randomly distributed data.

[13] improves to the sorting based algorithm presented in [5]. Their main contribution is to replace radix sort with counting sort and prefix sum. Once prefix sum calculated bins final address the sorting is achieved by copying each particle to the address of its bin. Additionally they replaced the computation of each bin's particle count with an atomic addition during particle assignment to cells. These optimizations allowed to significantly reduce kernel calls per frame and increase performance.

Spatial hashing is also used for collision detection in [7]. The domain is implicitly subdivided into grid cells by mapping them to a hash table. ...

[12] implements several algorithms to determine neighbor lists. They state that stenciled cell lists and

linear bounding volume hierarchies work better for data sets with variable cutoff radius, while uniform grids are better suited for fixed cutoff radius. In molecular dynamics the cutoff radius is the maximum distance in which particles can effect each other. Note that particle search for fixed sized voxels is conceptually equivalent to searching particles within a fixed cutoff radius.

Similary to this thesis [4] maps particle positions to grid contributions. ...

[8] proposes a memory efficient method to voxelize particles to surfaces. They start with a view adaptive voxel grid and voxelize the particles using splatting. To compress the grid only surface entries and exits are stored. The depth values are then connected to create a watertight mesh. While this approach works for surfaces this thesis requires a full volumetric representation.

3 Ambient Occlusion

3.1 Ray Casting

idea + hemisphere sampling ray sphere intersection math

3.2 Voxel Cone Tracing

cone tracing idea + cone approximation with voxels construction of hierarchy

4 Voxel Gathering

Review of possible data structures considering requirements of gathering approach.

4.1 Grid Construction

Presentation of a fast particle access data structure including underlying structure (grid..) and acceleration methods (sorting..).

4.2 Particle Voxelization

(basically how to transfer particles to voxel contributions)

5 Algorithm

5.1 Voxel Gathering

5.1.1 Grid Construction

5.1.2 Particle Voxelization

5.2 Ambient Occlusion

5.2.1 Voxel Cone Tracing

-> hemisphere of n cones per frame -> variable cone size + variable voxel grid size for comparison

5.2.2 Ray Casting

-> GPU Pathtracer on spheres -> progressive pipeline -> also on voxels for comparison

6 Results

6.1 Evaluation

1. Qualitativ

VCT on standard resolution grid - ray tracing on particle data -> Fehlerquellen:

VCT on high resolution grid -> voxelization error

Ray casting/Fine cones on voxel grid -> cone tracing error

2. Quantitativ

Scattering vs Gathering Performance + Speicherverbrauch

6.2 Discussion

7 Conclusion

Literaturverzeichnis