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, 17. April 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 17. April 2018

Maximilian Richter

Kurzfassung

Abstract

Inhaltsverzeichnis

1 Introduction

Modern particle-based simulations contain several millions of particles. With increasing density and depth complexity they require refined rendering techniques for visualization. Local lighting lacks visual cues for depth and occlusion and therefore introduces perception issues. Global illumination allows to visualize complex structures in particle data, that otherwise would remain unseen. While ray tracing based techniques converge to exact results, their computation is costly. 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 more occluded a surface point is, the less light reaches it and the darker it has to appear. 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 intersect the same voxel causing race conditions. As synchronization increases, parallelism decreases. Therefore scattering performs the worse, the more dense the particle data is.

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.

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

This thesis' task is composed of two main components. To review the relevant literature

2.1 Spatial Particle Search

Gathering requires an efficient method to retrieve particles in a given region of space. Spatial particle search is related to physical simulation of particles, where properties are determined according to neighboring particles. To detect ...

[HKK07b] 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 [HKK07a] proposed an adaptive uniform grid for variable sized volumes. Here the computational region is sliced into planes of voxels that are stored as texels. To avoid the wasted memory associated with a high amount of empty cells, only regions that actually contain particles are represented as voxel planes.

Another method using textures to sort data into spatial bins is presented by [OBS09]. The bins are specified as 2D texture coordinates into a texture array. Each bin is composed of all texels that share it's 2D coordinate, one from each slice. A distinct texture counts the elements in each bin and determines which slice provides the next texel to store the input value in. To represent 3D data this implementation requires a mapping function from spatial coordinates to 2D texture coordinates. As the texture array's depth limits the number of items it does not suit this thesis' task.

The CUDA SDK [Gre10] evaluates the assignment of particles to uniform 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 every particle's cell is calculated as a hash. Using radix sort with the hash value as key 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 has shown to be less sensitive to dense and randomly distributed data.

[Hoe14] improves the sorting based algorithm presented in [Gre10]. Radix sort works best for data with sparse and very large keys. For efficient particle access it is not necessary to sort the particles finer than per bin. Therefore the radix sort used in [Gre10] is replaced with a prefix sum and subsequent counting sort. 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.

Spatial hashing is also used for collision detection of polygonal meshes in [THM+03]. The domain is implicitly subdivided into uniform grid cells by mapping them to a hash table, avoiding complex data structures. This way the number of elements is only limited by memory capacity and arbitrary non-uniform distributions of objects can be stored. This method has been implemented on CPU

[How16] implements several algorithms to determine neighbor lists for particle simulation. 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 effect others. As particle search for fixed sized voxels is conceptually equivalent to searching particles within a fixed cutoff radius, the uniform grid approach is most relevant for this work.

Similarly to this thesis [GLX+15] maps particle positions to mesh contributions. They compare a gathering approach to a scattering method. Unlike the method presented in [Gre10, Hoe14] their gathering algorithm uses a fixed amount of storage to assign particles to mesh points. Particles exceeding this storage are stored in a separate list and are scattered in an subsequent step. To perform better than scattering this approach requires the overflow list to be small. This sensitivity to the distribution of particles it is not desired for this thesis.

[ZD15] 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, without any representation of the distribution in between. The depth values are then connected to create a watertight mesh. While this approach allows efficiently extract the surface of volumetric data this thesis requires a full volumetric representation.

Another class of spatial search algorithms does not try to find neighbors for a given point but whether there is geometry at all. This is most relevant for tracing rays through scenes until they hit geometry. Hierarchical methods are adaptive to the distribution of geometry in the scene. They subdivide the scene

into regions of increasing resolution as the amount of geometry inside grows. These methods' effectiveness relies on their ability to skip large empty areas. While this allows for efficient traversal of rays with unknown length, for particle gathering only a neighbourhood of limited and constant size is considered.

[TODO: GPU photon mapping paper: "hash"grid = uniform grid -> http://www.inf.ed.ac.uk/publications/thesis/online/IM

2.2 Ambient Occlusion

[ZIK98] 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 a point's level of occlusion, 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.

[PG04] pre-computes ambient occlusion offline using ray tracing 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. The pre-computation takes minutes, therefore this implementation only works for static scenes.

[STCK13]

To allow scenes with moving objects [Bun05] converts all vertices to disc-shaped occluder elements. Then for each vertex the occlusion value is calculated by adding the form factors of the other occluders. Form factor represents how much occlusion a surface creates depending on disk area and distance to it. As the ambient occlusion term is calculated per vertex, detailed meshes are required to avoid artifacts. While this method does not require precomputation and therefore allows for movable, deformable meshes, it still imposes strict limitations of a few thousend elements on the vertex count to remain interactive.

For scenes with arbitrary geometry a method to decouple ambient occlusion calculation from polygonal detail is required. [Mit07] introduced screen space ambient occlusion, a method to approximate the occlusion terms as a postprocessing pass with screen space information only. Instead of sampling polygonal meshes itself, the scene is rendered into a normal and a depth buffer first. Then each pixel's surrounding pixels are sampled from the buffers to determine an approximated occlusion factor. This is done by sampling a sphere around each pixel's world position. Then each sample is reprojected into depth buffer to determine if it is occluded. For realtime performance undersampling is required, which introduces noise. To counteract, a blur step finalizes the ambient occlusion postprocessig. The method by [SA07] is conceptually similar but combines the screen space occlusion with occlusion from spherical proxy geometry for far-field occlusion. (!)

A different measure of occlusion is used in [BSD08]. From each sampling point rays in random directions are marched in screen space to find the maximum angle from which light falls in. These horizon angles are averaged resulting in an estimate of occlusion with the occlusion factor growing proportionally to the angle.

While these methods provide fast results, they also introduce errors by only considering screen space regions. Artifacts are typically introduced due to the combination of low resolution occlusion calculation and high depth complexity. The issue here is that only a single depth value per pixel is considered, ignoring any geometry behind. [BA12] reduces flickering using temporal filtering. Ambient occlusion depends only on geometry and is independent of the view. Therefore the last frame's occlusion values can be reprojected to the current view to reduce temporal variance. To avoid trailing effects the pixels are classified into a stable and unstable set and temporal filtering is only applied to the unstable, flickering pixels.

Another screen space artifact are halos around objects, when their background is distant. The depth discontinuity between the object's and the background's pixels the horizon sampling. While this applies for every geometry, it is most visible when moving objects pass static ones. [TP16] approaches this by seperating static and moving objects to distinct depth buffers. While this reduces the error, it does not solve the underlying problem. Due to screen space limitations increasing depth complexity causes increased artifacts. World space methods avoid this by directly sampling the geometry. [TP16] describes voxel cone tracing as one method to calculate world space ambient occlusion.

Voxel cone tracing has been introduced by [CNS+11]. The fundamental idea is to transfer the geometry into a multiresolution volumetric representation. The voxel hierarchy is then used to sample cones, approximating the amount of geometry in that region. For each cone a series of increasingly bigger voxels is sampled along the cones direction, each from a coarser resolution than before. The voxels' density values are blended along

[SGG15] applies voxel cone tracing to molecular dynamics data. A 3D texture is used as voxel data structure into which the particles are voxelized. To create the coarser resolutions the texture is mipmapped. The particles are then rendered using raycasting. For each pixel three cones are traced around the normal.

In [GKSE12] a grid stores the occupancy information derived from the neighboring atoms.

3 Ambient Occlusion

One method to enhance the quality of particle visualizations is to facilitate perception of depth and interleaving. For this purpose shadows and illumination are usefull cues, as natually occluded regions are likely to be shaded. Local illumination is not able to provide the desired global shading effect, as it's result depends exclusively on the light source and the surface normal, but not on the surrounding occlusive geometry.

Abbildung 3.1: image of local illumination

Global illumination

Using only direct light introduces perception issues, as every surface, that is not directly visible by a light source appears black. With interleaved data of high occlusion this leads to large regions being unable to perceive, as can be seen in 3.2. To prevent these regions from appearing entirely black, inter-reflection has to be accounted for. Most surfaces reflect some amount of the light reaching them diffusely through the scene. This attenuated light illuminates surfaces that are not directly visible to light sources and is therefore called indirect light. [REF REALTIME RENDERING]

As with local lighting, direct light depends on the intensity and position of light sources. This introduces additional information that is not related to the data set. As the rendering result does no longer depend exclusively on the data set but also on the light sources' properties the visualization is biased. For scientific visualization this is undesired. Therefore a method to implement indirect light independently of light position has to be found. One such method is ambient light. Here constant indirect radiance for the entire scene is assumed. As ambient light does not vary with direction every point's indirect illumination is

Abbildung 3.2: image of direct lighting

equal. The larger the solid angle of a light source, the softer are the shadows it casts. As ambient light illuminates evenly from all directions, shadows disappear due to maximum softness. Yet, shadows are essential for direction independent illumination like ambient lighting to display geometric details. Without shadows and directional variance of light objects appear flat, as seen in 3.3.

Abbildung 3.3: Ambient light only. Due to the lack of shadows objects appear flat.

To allow shadowing the amount of ambient light that illuminates a given point has to be determined depending on it's surrounding geometry. Ambient occlusion approximates this inversely as the lack of incident ambient light, which is assumed to be proportional to the degree of geometric occlusion.

The remainder of this chapter covers the computation of ambient occlusion. Therefor first the mathematical framework is presented in 3.1. As mentioned in 2.2 numerous methods exist compute ambient occlusion. For this thesis two object space methods are compared. In 3.2 ambient occlusion is calculated using ray casting. The quality of this method depends on the number of ray samples for each surface point. As the number of rays grows this method converges to the correct ambient occlusion value, but also achieves increasingly less interactive performance. Therefore this method is chosen as ground truth to evaluate the quality of the second method. In 3.3 a discretized volumetric representation of the scene is used to approximate ambient occlusion with voxel cone tracing.

Abbildung 3.4: Ambient occlusion. Note that ambient light is attenuated the more, the more the pixel is occluded.

3.1 Ambient Occlusion Term

To compute ambient occlusion a for a surface point **p** it's hemisphere Ω of possible incident light directions is integrated with a visibility function v. The hemisphere Ω is spanned orthogonal to **p**'s normal. The visibility function v determines whether a ray from **p** in direction ω intersects the local occluding geometry. For intersections v returns zero, for non blocked rays it returns one. v is cosine weighted with the angle θ between **p**'s normal and ω, resulting in the following formula:

$$a(\mathbf{p}) = \frac{1}{\pi} \int_{\Omega} v(\mathbf{p}, \omega) \cos(\theta) d\omega \tag{3.1}$$

[REF 9.14 Realtime Rendering] With an uniform distribution of rays across the hemisphere the cosine weighting is necessary to account for the higher influence of light incoming close to the point's normal. Alternatively importance sampling can be used, which weights each ray equally but cosine distributes them on the hemisphere.

Abbildung 3.5: Influence of rays depending on their angle of incidence.

With equation 3.1 fully enclosed geometry like objects inside a closed room will have an a value of zero,

rendering the calculation useless. To bypass that, the hemisphere radius can be limited to a distance that prevents that every ray hits the enclosing geometry. The smaller the hemisphere's radius, the less global are the resulting shadows, which hides precious information on coarse occlusion.

[TODO: PHYSICAL MODEL INTERREFLECTION -> REALTIME RENDERING] Another solution is to replace the binary visibility function with a continuous distance based attenuation ρ. [ZIK98] denoted this method as obscurance:

$$a(\mathbf{p}) = \frac{1}{\pi} \int_{\Omega} \rho(d(\mathbf{p}, \omega)) \cos(\theta) d\omega \tag{3.2}$$

Here $d(\mathbf{p}, \omega)$ calculates the distance between **p** and the closest intersection with the geometry in direction ω. The higher the distance, the less light is occluded by the intersection. Therefore ρ has to be a function that attenuates occlusion with distance. [ZIK98] describes this function as non-linear mapping from distance to incident light, with incident light converging to one as the distance increases.

Abbildung 3.6: ρ attenuation function of distance.

Note that while recent literature beschreibt ao als fehlen von licht wohingegen zhucov menge an licht bestimmt.

3.2 Ray Casting

Ray casting denotes the method of determining the first object intersecting a ray. Therefore the scene's objects are compared with a ray using ray-surface intersection tests. To calculate ambient occlusion with ray casting a hemisphere on the surface point's normal is sampled with rays. For each ray the first particle intersecting it is determined and it's intersection distance is then used to calculate ambient occlusion term. By averaging the ambient occlusion values of all rays the point's final ambient occlusion value is obtained.

Each ray from **p** computes the visibility function for a single direction.

Hemisphere Sampling

As the ambient occlusion term is cosine weighted, a uniform distribution of the rays on the hemisphere is required. One method to represent points on a sphere are spherical coordinates θ and ϕ, where θ is the polar angle and ϕ is the azimuthal angle. Selecting θ from a uniform random distribution results in an increased sample density around the polar direction, which is the surface normal. This is due to [...] and results in overestimation of the solid angle around the normal.

Abbildung 3.7: Spherical Coordinates.

Several methods exist to uniformly sample points on a sphere. Idea: reduce number of sampling points where their density is too high (polar axis). This can be achieved with an arccos distribution of points... [TODO].

Abbildung 3.8: Non-uniform distribution using spherical coordinates vs. uniform distribution using arccos distribution

Ray-Sphere Intersection

With an uniform distribution of the rays each of them has to be intersected with the scene geometry. For particle data the geometry consists exclusively of spheres. Therefore a method to determine if and

where a ray intersects a sphere has to be applied. -> https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection

Abbildung 3.9: Intersection of a ray with a sphere.

3.3 Voxel Cone Tracing

Voxel Cone tracing uses a discretized voxel representation of the scene geometry to efficiently approximate illumination. Instead of casting rays into the scene, entire cones are traced. This discretization allows to sample a point's hemisphere with a finite set of cones, allowing for real-time performance. Approximating entire solid angles of rays with single cones is a reasonable approximation as the rays are both spatially and directionally coherent.

A cone is approximated as a series of voxels of increasing size. Therefor a hierarchy of voxels is required, each level representing the finer levels at a coarser resolution. To resemble a cone, starting from the finest resolution a series of increasingly larger voxels is traced. The resolution a voxel is sampled from is determined by the width of the cone at the sampling position. Intuitively, the further the cone is traced, the coarser the voxels get.

The quality of this method depends on the resolution of the scene's voxel representation. To allow high resolutions a memory efficient method to store the voxel information is required. [CNS+11] uses a sparse voxel octree. [TODO: kurz was ist das]. Sparse data structures are especially useful for scenes with few polygonal meshes of high resolution, as they allow to Assuming the ... is tightly packed around the

+ trilinear interpolation to allow spatially smooth transition of the voxel valus

[sagen wie samples entlang cone verteilt werden] As the size of the voxels will rarely match the width of the cone at each sampling position [CNS+11] proposed quadrilinear interpolation of the voxels. Additionally to the trilinear interpolation within each voxel hierarchy level a linear interpolation between two subsequent levels is applied.

Abbildung 3.10: Multiresolution voxel hierarchy with a cone approximated by voxels.

Abbildung 3.11: Quadrilinear interpolation.

[CNS+11] used the voxels to represent discretized surface information of a polygonal scene. For this thesis a discretized representation of particle data has to be stored. Therefor instead of surface information a density value is stored per voxel, representing "how many particles"[TODO better formulation] occupy the voxel's volume. As this model does not use geometric surfaces, but volumetric density information, both the visibility function v and distance based attenuation ρ cannot be used. Instead a density based measure δ based on the volume rendering integral is used.

$$a(\mathbf{p}) = \frac{1}{\pi} \int_{\Omega} \delta(\mathbf{p}, \omega) \cos(\theta) d\omega \tag{3.3}$$

The volume rendering integral...

The fundamental observation that allows voxel cone tracing for ambient occlusion is, that the integration of the hemisphere Ω can be subdivided into a sum of n integrals on partial hemispheres Ω_i:

$$a(\mathbf{p}) = \frac{1}{n} \sum_{i=1}^{n} \int_{\Omega_i} \delta(\mathbf{p}, \omega_i) \cos(\theta_i) d\omega_i \tag{3.4}$$

Each Ω_i is integrated separately and added to the final occlusion value. If the subdivision of Ω is regular,

each of the partial integrals resembles a cone. To approximate ambient occlusion n cones are traced starting from **p**.

Abbildung 3.12: A hemisphere resembled by cones.

The number of cones determines both performance and quality of the calculation. As the number of cones on a hemisphere increases, their opening angle decreases. A low opening angle results in finer voxels resembling the cone, which increases quality. Apart from that a higher number of cones naturally increases the computational load. Therefore, for real-time rendering a trade-off between quality and performance has to be found.

Abbildung 3.13: Showing multiple sample counts of the Raycasting

4 Particle Voxelization

This chapter covers the process of transferring particle data into a voxel hierarchy. The voxel hierarchy is organized as complete octree. By definition this data structure is composed of levels of increasing resolution, with each level being an uniform grid of eightfold resolution of the previous level. As stated in 3.3 each voxel represents the density of particles within its respective region. Using this model a voxel hierarchy can be simply created with mip-mapping of the voxel density values, propagating them from the finest to the coarsest level. Therefore the voxelization into the finest level is of main interest. The chosen voxelization model represents the particles within a particular region using density values. Each voxel represents the accumulated particle volume within its volume.

Abbildung 4.1: Particle to voxel model (shows correlation of particle density and voxel value)

Let p be a particle and v_{xyz} the voxel at grid position (x, y, z) with its density value d_{xyz}. Whenever a particle intersects a voxel it contributes a part of its volume to the voxel's density value. The contribution is a function c that ranges from 0 for emptiness to 1 for complete coverage. Its value depends on the way the particle intersects the voxel, which can be categorized into three cases.

1. The voxel lies entirely within the particle. As the particle occupies the voxel completely, it contributes maximum density, resulting in: $c(p, v_{xyz}) = 1$
2. The particle lies entirely within the voxel. Here the particle's contribution equals its volume normalized to the voxels volume: $c(p, v_{xyz}) = \frac{V(p)}{V(v_{xyz})}$
3. Particle and voxel intersect each other partially. For this case the intersection volume of particle and

voxel has to be determined. [TODO REF Fast computation of accurate sphere-cube intersection volume] evaluates methods to compute this intersection for their performance and precision. The voxelization's purpose is to accelerate ambient occlusion computation and therefore a method that trades accuracy for performance was chosen. The voxel is sampled with points on a regular grid and the number of those within the p is accumulated as n_p. The ratio of n_p to the total number of samples s yields the final contribution value: $c(p, v_{xyz}) = \frac{n_p}{s}$

Abbildung 4.2: Intersection cases

Once all contributions are added to the voxel density values the hierarchy has to be created. To create a coarser octree level from a finer level groups of eight neighboring voxels are averaged into a single, coarser voxel. This process is repeated until the root of the voxel octree is reached. Due to the averaging the coarser levels contain increasingly rough estimates of the particle density inside them.

As the contributions are added independently with overlapping particles this approach can result in density values larger than 1. While note being physically plausible this also results in errors during the mipmapping process. Assume a situation where eight particles entirely occupy a single voxel, leading to a density value of 8. If the seven neighboring voxels are empty this would result in the upper level voxel having a mipmapped density value of 1, despite only one of his children actually contains geometry. This has to be prevented by clamping the density value of each voxel to 1 before mipmapping.

[TODO DISADVANTAGE: DOUBLE REPRESENTATION OF GEOMETRY]

The remainder of this chapter covers algorithms for the voxelization of particles. First in 4.1 the scattering method is described with both its advantages and limitations, including details on the underlying hardware restrictions of modern graphics processors. In 4.2 gathering is presented as a method to bypass these restrictions. As it requires an acceleration data structure for spatial access on the particles in 4.3 an algorithm to create an uniform spatial partitioning is described. The proposed data structure is finally utilized to implement efficient algorithms for gathering and scattering, explained in 4.3.5 and 4.3.6

respectively.

4.1 Scattering Method

Generally scattering describes the process of transferring irregular data to regular data. In case of particle voxelization the particles itself are irregular with respect to their position and the voxels are regular in form of an uniform grid. As we transfer from particles to voxel cells, scattering is a glyph centric method. To transfer a particle to voxel contributions scattering first determines the voxels it intersects. With the uniform structure of the voxel grid this can be done efficiently by quantizing the vertices of a bounding box around the particle to grid coordinates. For each intersected voxel the particle's contribution is then calculated according to the intersection cases. Therefore for n being the number of particles and c being the number of voxels each particle contributes to, scattering has a complexity of $\mathcal{O}(n \cdot c)$.

Abbildung 4.3: Scattering a sphere to intersected voxels

Scattering is conceptually simple and requires no acceleration data structure. Also it is easily parallelized among threads. On parallel architectures the particles are processed simultaneously, with a distinct thread spawned for each particle. Each thread first determines the voxels its particle intersects and then iterates them contributing to their density value. However this method for parallel execution is flawed. Whenever simultaneously processed particles intersect the same voxel synchronization has to assure correct accumulation of the contributions. For dense particle data this can significantly reduce parallelism. Also no assumption is made on the particle ordering. Spatially unsorted data results in scattered memory accesses with significantly reduced performance, due to low cache utilization. In the following both limitations are discussed in detail.

Synchronization

PROBLEM

Abbildung 4.4: Scattering two particles with shared voxels

BACKGROUND

uninterruptable read-modify-write memory operation on specific address that serializes contentious updates from multiple threads [REF http://on-demand.gputechconf.com/gtc/2013/presentations/S3101-Atomic-Memory-Operations.pdf]

executed by memory controller -> formerly only bitwise operations like addition and logical operations were supported. -> recent architectures also implemented float atomics

Whenever high number of atomic operations per particle could be performance limitation

https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

Scattered Memory Access

As no spatial sorting is assumed for the particles they can be completely unsorted. In this context unsorted means that particles stored sequentially in memory lie in entirely different regions in space and therefore contribute to distant voxels. The more scattered sequential particles are spatially, the more scattered their writes to voxels are in memory. Due to the caching system of modern graphics processors this dramatically decreases efficiency of write operations.

[TODO: Background on GPU Coalescing]

4.2 Gathering Method

Gathering is a method that does not require atomic operations for correct voxelization. Instead of scattering particles to voxels, the voxels gather the contributions from the particles. For each voxel first the particles it intersects with are determined, before their contribution is calculated and added to the voxels

density. This process

The gathering approach is diametral to scattering.

- cell centric - regular -> irregular

This, on the one hand, requires a fast lookup structure for the spheres, because intersecting spheres have to be determined for every cell. On the other hand, this process is easily parallelizable across the cells, where each threads works on only one cell.

Of importance here is: While with scattering based on the data of one particle multiple writes to different locations occur. With gathering reads from multiple locations in memory lead to a single write operation.

ADVANTAGES

LIMITATIONS

Gathering lange becannt aus GPU COMPUTING. -> TRADEOFF: weniger atomic mehr arbeit that means that the particles regularly are processed multiple times. One time for each contribution it makes.

Without an acceleration structure each voxel would have to iterate all particles. With n being the number of particles and v being the number of voxels, this results in $\mathcal{O}(n \cdot v)$ complexity.

Abbildung 4.5: Gathering a voxel from intersecting spheres

REQUIREMENTS FOR DATA STRUCTURE: SPATIAL ACCESS + Sorting

Suitable Data Structures

-requirements (spatial access + efficient buildup) -> fixed cutoff radius case

Review of possible data structures considering requirements of gathering approach.

With fixed-radius nearest neighbor search the target is to find all neighboring particles within a given radius around a position.

+ NNS is base for all kinds of lagrangian problems where neighbors are searched that influence a particular region in space

Abbildung 4.6: Showing the conceptual similarity of fixed radius NNS and fixed size particle search around voxel

-> darauf kommen das uniforme unterteilung bester ansatz fuer diesen anwendungsfall ist.

4.3 Uniform Spatial Partitioning

The fundamental idea of spatial partitioning is to insert particles into space subdividing bins so that whenever particles in a specific region are requested, only those inside the bins of this particular region have to be searched. With uniform partitioning space is divided into equal sized bins that resemble a uniform grid. Uniformity allows for efficient insertion into and lookup from bins by simply quantizing the requested position to the spatial partitioning's resolution.

To allow efficient particle processing the particles have to be sorted by their bins. This allows for two major optimizations. Firstly with spatial partitioning the particles are processed in chuncks of a bins size. When these particles are sequential in memory the read operations to fetch them are coalesced. This is essential for the performance of the gathering approach. Secondly as stated in 4.1 spatially unsorted particles result in scattered writes for voxel contributions. Sorting the particles by bin essentially sorts them spatially, so that those which intersect the same voxels are sequential in memory. These particles' contribution writes are coalesced, allowing efficient scattering.

TODO: OVERALL CONCEPT The algorithm presented by [Hoe14] constructs a uniform spatial partitioning by first applying a hash function to every particle position. The process of spatial partitioning presented by ...

In the following first an evaluation of suitable sorting algorithms is given.

TODO: REMAINDER 4.3.5 4.3.6

Abbildung 4.7: Image shows particles assigned to hash grid cells and their position in memory before and after

4.3.1 Sorting Algorithms

AKA RADIX SORT VS COUNTING SORT

[Gre10] Green employs radix sort to sort the particles by their bin. Radix sort is a key based sorting algorithm where elements are sorted by their key. [Hoe14] Hoetzlein

Radix sort ...

Counting sort sorts list elements by their key.

4.3.2 Spatial Hashing

»A hash function is any function that can be used to map data of arbitrary size to data of fixed size.« -> YES WE DO HASHING

As the particles have to be sorted spatially, a method has to be found that assigns keys based on spatial similarit so that neighboring particles share the same key.

Assumtion: Particle xyz range 0..1 -> simple rounding formula

4.3.3 Address Calculation

Prefix sum

4.3.4 Particle Sorting

Out of place sorting -> seperate array

In place sorting -> moving auxillary storage

Abbildung 4.8: Image shows list of values transferred to a prefix sum

4.3.5 Gathering Algorithm

Mace differentiation between hug and voxel octree clear! Image showing the process

Voxel -> Hash Cells -> Particles

Let k be the average number of particles inside a bin and b be the number of bins that have to be searched for each voxel. For n particles the complexity of this method is $\mathcal{O}(n \cdot k \cdot b)$. As $k \cdot b \ll v$ complexity can be significantly reduced compared to brute force 4.2.

```
parallel foreach voxel:
    foreach intersected bin:
        foreach particle within bin:
            if particle intersects voxel: add partial particle mass to voxel
```

4.3.6 Coalesced Scattering Algorithm

Sorted Particles -> Voxels

5 Implementation

5.1 Software Architecture

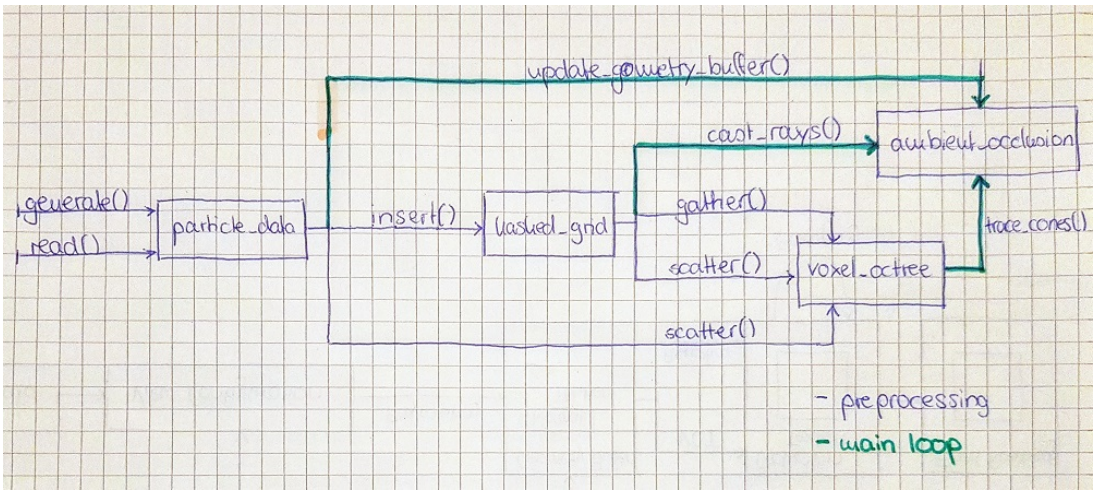

Abbildung 5.1: Overview of the software architecture.

Input Data

Design Considerations

Graphics API WHY OPENGL -> `https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html` chapter 39.2.6 -> all disadvanteges of old opengl implementations are removed with compute shaders

5.2 Uniform Spatial Partitioning

Seperated into three phases:

- assign particle top bin - calculate address of each bin -

5.2.1 Spatial Hashing

5.2.2 Address Calculation

Parallel Prefix sum. Prefix sum papers.

5.2.3 Particle Sorting

5.3 Particle Voxelization

The voxelization process has been implemented

5.3.1 Scattering

Compute Shader Method

UNSORTED INPUT FROM PARTICLE-DATA or SORTED INPUT FROM HASHED-UNIFORM-GRID

By offsetting the particle position by the particle radius in each dimension a bounding box can be constructed. By quantizing the corners to grid coord the voxels are determined.

Rasterization Method

Use rasterizer to emit multiple two dimensional slices of voxels for each particle.

5.3.2 Gathering

5.4 Particle Rendering

Deferred Shading

Conservative Rasterization

http://jcgt.org/published/0002/02/05/paper.pdf

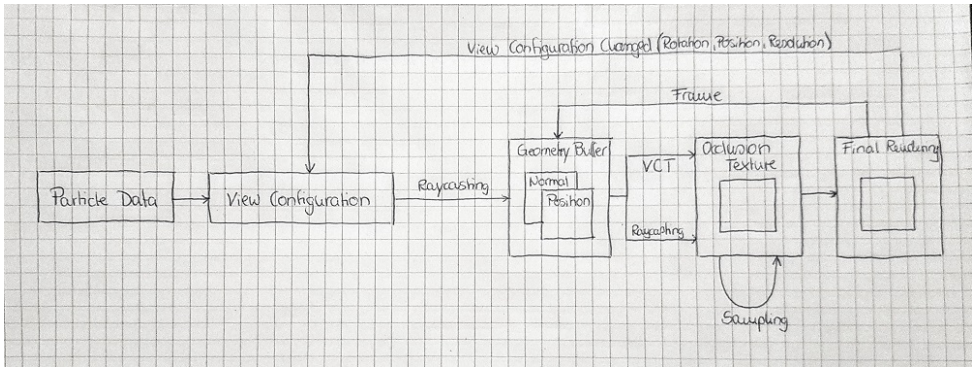

Abbildung 5.2: Overview of the rendering pipeline.

Abbildung 5.3: Image of both Position Buffer (z-value) and Normal Buffer

Perspective Projection

Raycasting

5.5 Ambient Occlusion

5.5.1 Ray Casting

-> GPU Raycast on spheres

- dda algorithm

-> progressive pipeline

5.5.2 Voxel Cone Tracing

-> Voxel Cone Tracing

-> hemisphere of n cones per frame

Abbildung 5.4: Images comparing thin line DDA with thick line DDA and the resulting error due to missed particles

6 Evaluation

6.1 Qualitative Results

Evaluates the voxelization quality with respect to the ground truth

6.1.1 Influence of Particle Data Structure

-Fehler in abhaengigeit der Datensaetze

-2 Partikel die sich schrittweise voneinander wegbewegen

-dense vs sparse Data

-flat data (voxel pattern)

6.1.2 Influence of Voxel Octree Resolution

Vergleich: VCT mit ray casting auf partik data -> Fehlerquellen: VCT on high resolution grid -> voxelization error Ray casting/Fine cones on voxel grid -> cone tracing error

MUSS ICH QUALITAET VERSCHIEDENER RAYCASTING ZEITEN UNTERSUCHEN?

WIE TRENNE ICH VOXELISIERUNGSFEHLER UND CT FEHLER?

6.1.3 Influence of Cone Configuration

Influence of number and opening angle of cones

More but sharper cones

Left side image that shows distribution of cones - Right side the rendered result

Length of the cones

6.2 Quantitative Results

6.2.1 Runtime Performance

Scattering vs Gathering Performance

Einzelne Schritte des Gatherings

Teuerste Schritte in shadern

6.2.2 Memory Consumption

...

7 Discussion

8 Conclusion

[GLX+15] [Gre10] [HKK07a] [THM+03] [ZD15] [GKSE12] [SGG15] [HKK07b] [How16] [Hoe14] [OBS09] [ZIK98] [PG04] [Bun05] [CNS+11] [Mit07] [BSD08] [BA12] [TP16] [SA07]

Literaturverzeichnis

[BA12] BAVOIL, Louis ; ANDERSSON, Johan: Stable SSAO In Battlefield 3 With Selective Temporal Filtering. In: *Game Developer Conference Course, San Francisco, CA*, 2012

[BSD08] BAVOIL, Louis ; SAINZ, Miguel ; DIMITROV, Rouslan: Image-space horizon-based ambient occlusion. In: *ACM SIGGRAPH 2008 talks*, ACM, 2008, S. 22

[Bun05] BUNNELL, Michael: Dynamic ambient occlusion and indirect lighting. In: *GPU Gems 2*. Addison-Wesley Professional, 2005, Kapitel 14, S. 223–233

[CNS+11] CRASSIN, Cyril ; NEYRET, Fabrice ; SAINZ, Miguel ; GREEN, Simon ; EISEMANN, Elmar: Interactive indirect illumination using voxel cone tracing. In: *Computer Graphics Forum* Bd. 30 Wiley Online Library, 2011, S. 1921–1930

[GKSE12] GROTTEL, Sebastian ; KRONE, Michael ; SCHARNOWSKI, Katrin ; ERTL, Thomas: Object-Space Ambient Occlusion for Molecular Dynamics. In: *Proceedings of IEEE Pacific Visualization Symposium*, IEEE Computer Society, 2012, S. 209 – 216

[GLX+15] GUO, Xiangyu ; LIU, Xing ; XU, Peng ; DU, Zhihui ; CHOW, Edmond: Efficient Particle-mesh Spreading on GPUs. In: *Procedia Computer Science 51 ICCS*, Elsevier, 2015, S. 120–129

[Gre10] GREEN, Simon: *Particle Simulation using CUDA*, NVIDIA Corporation, 2010. `http://hoomd-blue.readthedocs.io/en/stable/nlist.html`

[HKK07a] HARADA, Takahiro ; KOSHIZUKA, Seiichi ; KAWAGUCHI, Yoichiro: Sliced data structure for particle-based simulations on GPUs. In: *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, 2007, S. 55–62

[HKK07b] HARADA, Takahiro ; KOSHIZUKA, Seiichi ; KAWAGUCHI, Yoichiro: Smoothed Particle Hydrodynamics on GPUs. In: *Proceedings of Computer Graphics International*, IEEE Computer Society, 2007, S. 63–70

[Hoe14] HOETZLEIN, Rama: *Fast Fixed Radius Nearest Neighbors: Interacti-*

ve Million-Particle Fluids, GPU Technology Conference, 2014. http://on-demand.gputechconf.com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu.pdf

[How16] HOWARD, Michael: *HOOMD-blue Neighbor lists*, Univerity of Michigan, 2016. http://hoomd-blue.readthedocs.io/en/stable/nlist.html

[Mit07] MITTRING, Martin: Finding next gen: Cryengine 2. In: *ACM SIGGRAPH 2007 courses*, ACM, 2007, S. 97–121

[OBS09] OAT, Christopher ; BARCZAK, Joshua ; SHOPF, Jeremy: *Efficient Spatial Binning on the GPU*. AMD Technical Report, 2009

[PG04] PHARR, Matt ; GREEN, Simon: Ambient Occlusion. In: *GPU Gems*. Addison-Wesley Professional, 2004, Kapitel 17, S. 279–292

[SA07] SHANMUGAM, Perumaal ; ARIKAN, Okan: Hardware Accelerated Ambient Occlusion Techniques on GPUs. In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. New York, NY, USA : ACM, 2007 (I3D '07), S. 73–80

[SGG15] STAIB, Joachim ; GROTTEL, Sebastian ; GUMHOLD, Stefan: Visualization of Particle-based Data with Transparency and Ambient Occlusion. In: *Proceedings of the 2015 Eurographics Conference on Visualization*, Eurographics Association, 2015, S. 151–160

[STCK13] SLOAN, Peter-Pike ; TRANCHIDA, Jason ; CHEN, Hao ; KAVAN, Ladislav: Ambient Obscurance Baking on the GPU. In: *SIGGRAPH Asia 2013 Technical Briefs*. New York, NY, USA : ACM, 2013 (SA '13), S. 32:1–32:4

[THM+03] TESCHNER, Matthias ; HEIDELBERGER, Bruno ; MÜLLER, Matthias ; POMERANTES, Danat ; GROSS, Markus: Optimized Spatial Hashing for Collision Detection of Deformable Objects. In: *Proceedings of Vision, Modeling, Visualization*, Aka GmbH, 2003, S. 47–54

[TP16] TATARINOV, Andrei ; PANTELEEV, Alexey: *Advanced Ambient Occlusion Methods for Modern Games*. http://developer.download.nvidia.com/gameworks/events/GDC2016/atatarinov_alpanteleev_advanced_ao.pdf. Version: 2016. – Game Developers Conference 2016

[ZD15] ZIRR, Tobias ; DACHSBACHER, Carsten: Memory-efficient On-the-fly Voxelization of Particle Data. In: *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization*, Eurographics Association, 2015, S. 11–18

[ZIK98] ZHUKOV, Sergey ; IONES, Andrei ; KRONIN, Grigorij: An Ambient Light Illumination

Model. In: *Proceedings of the Eurographics Workshop: Rendering Techniques '98*, Springer, 1998, S. 45–56