Volumetric Ray Tracing

1062[1]

[1]OTH Regensburg, Germany

Abstract

1. Introduction

In Computer Graphics, objects usually are represented as a set of geometric primitives (e.g. triangles), displaying the surface of the object. However, this approach is not always suitable. For example, if the original data representation of the object is volumetric data (which might be produced by medical 3D scans [**?**]), the traditional rendering technique would necessitate the creation of an intermediate surface representation that might introduce unwanted artifacts [**?**]. Another issue arises if the object has no well-defined surfaces to which geometric primitives could be fitted, such as a cloud or fog [**?**]. In such cases, volumetric ray tracing might be used, a technique in which rays are cast through a volume which contains information about its optical properties(e.g color and opacity), sampled at various points within the volume, accumulated and projected on a 2D image (see figure 1) [**?**]. A description of this algorithm is presented in this paper, alongside several strategies for improving computational speed, such as (probabilistic) early termination of a ray, and sampling at lower resolutions within the volume.

2. Derivation Of The Rendering Equation

Volumetric ray tracing follows the same basic principle as classical ray tracing in which an image is rendered by spanning a pixel plane in front of the camera and casting one or multiple rays through each pixel. That means for a point $\vec{x}$ on the plane we cast a ray in direction $-\omega$ and calculate the amount of light $\vec{x}$ receives from direction ω, called $L(\vec{x},\omega)$ [**?**]. To do this, we calculate $\vec{y}$, the closest intersection point of the ray with a piece of geometry. At $\vec{y}$, we calculate the amount of light transported from $\vec{y}$ in direction ω (either through reflection or emission), called $L_e(\vec{y},\omega)$ (The exact method for calculating L_e is a topic of classical raytracing [**?**] and will not be further evaluated in this paper. In the following, we assume L_e to be a known quantity). Thus, we get the equation

$$L(\vec{x},\omega) = L_e(\vec{y},\omega) \tag{1}$$

However, this equation assumes that the light has no interactions between $\vec{x}$ and $\vec{y}$, which is only true if the ray travels through a vacuum. If the light travels through a medium that interacts with it (called a participating medium), the interactions change the light along the ray [**?**](see figure 1). In most cases, these interactions are small enough to be ignored, but in certain scenarios (e.g if there is fog, clouds or smoke present) they might have a noticeable effect. The types of interactions we need to consider are absorption, emission, in-scattering and out-scattering [**?**].

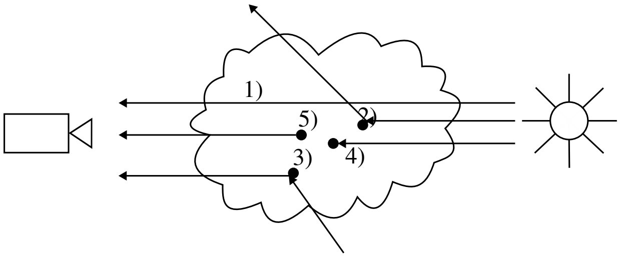


Figure 1: Projected cylinder.

Absorption and out-scattering attenuate the light intensity along the ray, in-scattering and emission add to it.

2.1. Absorbtion And Out-Scattering

At first, let us consider only out-scattering and absorption, which occur due to tiny particles floating around in the volume (such as water droplets in clouds and fog, or dust particles in the air). Of course, these particles are far too numerous to be directly simulated, but their distribution in 3D space can be stochastically modeled (similar to how detailed surface structures can be modeled by micorfacets in classical ray tracing). QUESTION! We follow Max [**?**] in our development of the stochastical model. To do so, consider a close-up look at a ray section traveling through the volume. This section can be assumed to be a cylinder with a base area E and a height Δh, through which the ray travels from top to bottom. Within this cylinder, there exists a certain number of out-scattering and ab-

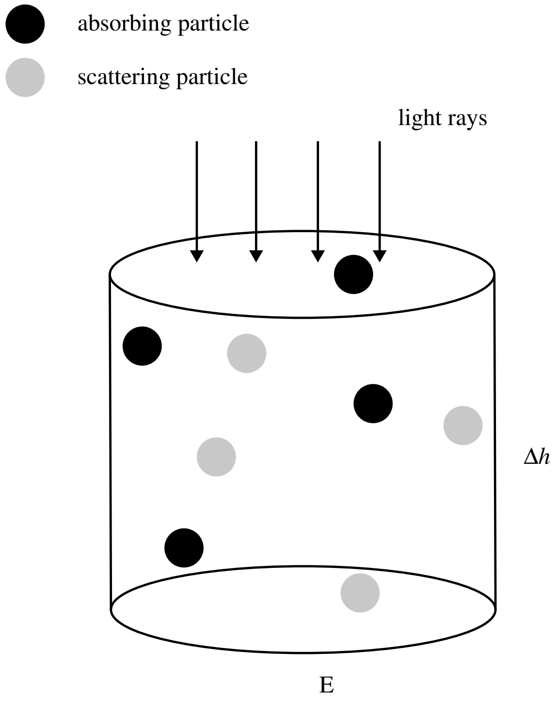


Figure 2: Projected cylinder.

sorbing particles, defined as $n_s = \rho_s E \Delta h$ and $n_a = \rho_a E \Delta h$ respectively, where ρ_s and ρ_a are the densities of the particles. From the top-down view, the particles will occupy an area of $n_s A = \frac{\rho_s A E \Delta h}{E}$ and $n_a A = \frac{\rho_a A E \Delta h}{E}$, respectively, if we assume that the particles do not overlap each other, which is reasonable if the densities and the height do not become too large.

This can be simplified to $\rho_s A \Delta h$ and $\rho_s A \Delta h$, which gives us the fractions of light which are stopped in the cylinder, either through absorbtion or out-scattering. By letting Δh approach 0, we see that for each infinitessimaly small cylinder slice with height dh, the change in intensity is proportionally to $-(\rho_s A + \rho_a A)dh$. Formulated as a differential equation, this results in

$$dL = -(\rho_s(h)A + \rho_a(h)A)L(h)dh \tag{2}$$

At this point, we define the scattering coefficient $\mu_s = \rho_s(h)A$, the absorption coefficient $\mu_a = \rho_a(h)A$ and the extinction coeffcient $\mu_t = \mu_s + \mu_a$, which give a measure of how much light is lost due to scattering, to absorption, and in total. Thus, equation [REFERNCE] can be simplified to

$$dL = -\mu_t(h)L(h)dh \tag{3}$$

which solution is

$$L(h) = L_0 e^{-\int_0^h \mu_t(s)ds} \tag{4}$$

Rearranging this results in the equation

$$\frac{L(h)}{L_0} = e^{-\int_0^h \mu_t(s)ds} \tag{5}$$

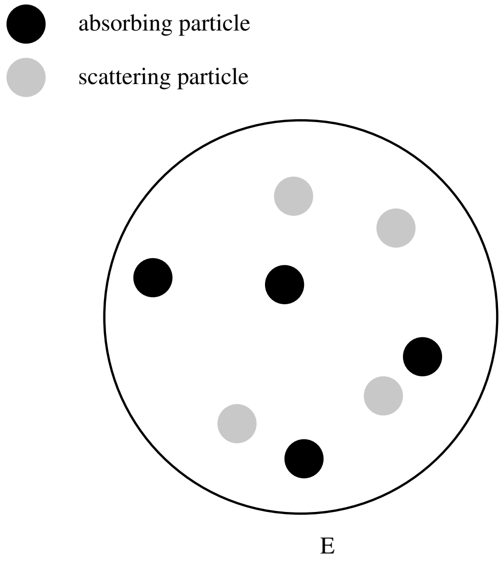


Figure 3: Projected cylinder.

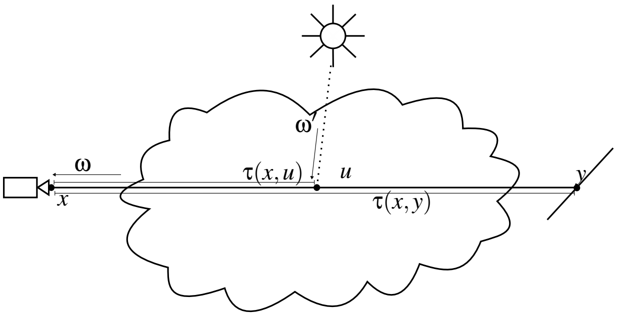


Figure 4: Projected cylinder.

describing the ratio of the light that travels a distance h unimpeded. In the following, we call this quantity the transmittance, and refer to it as $\tau(\vec{x},\vec{x'})$ (the ratio of light that arrives at $\vec{x}$ from $\vec{x'}$). Thus, going back to equation 1, we can now specify how much L_e gets attenuated, and get the equation

$$L(\vec{x},\omega) = \tau(\vec{x},\vec{y})L_e(\vec{y},\omega) \tag{6}$$

This equation is still not accurate, since we also need to consider in-scattering and emission along the ray.

2.2. Emission And In-Scattering

For this purpose, let us consider any arbitrary point $\vec{u}$ on the ray from $\vec{x}$ to $\vec{y}$. At $\vec{u}$, the particles of the medium emit some light towards ω [?], which we call $\varepsilon(\vec{u},\omega)$. In the cylindrical model, the probability of finding an absorbing particle at position $\vec{u}$ is $\mu_a(\vec{u})$. We assume that the particles responsible for absorbtion are also responsible for emission [?], meaning the amount of light emitted at

$\vec{u}$ towards ω is $\mu_a(\vec{u})\varepsilon(\vec{u},\omega)$. In-scattered light can arrive at $\vec{u}$ from every direction, which means we need to integrate over the sphere Ω surrounding $\vec{u}$ [**?**]:

$$\int_{\Omega} L(\vec{u},\omega‘)d\omega‘ \tag{7}$$

However, not all light arriving at $\vec{u}$ is scattered towards ω. The so called phase function $f_p(\vec{u},\omega,\omega‘)$ gives us the probality that light hitting a particle at $\vec{u}$ from ω‘ is reflected towards ω [**?**]. In this sense, it is analogous to the BRDF in classical ray tracing. Furthermore, the amount of light scattered towards ω also depends on the probability that in-scattering particles are present at $\vec{u}$ [**?**]. We assume that in-scattering particles are the same as out-scattering particles and get the equation

$$L_e^s(\vec{u},\omega) = \mu_s(\vec{u})\int_{\Omega} f_p(\vec{u},\omega,\omega‘)L(\vec{u},\omega‘)d\omega‘ + \mu_a(\vec{u})\varepsilon(\vec{u},\omega) \tag{8}$$

describing the total amount of light being added to the ray at $\vec{u}$. We call this quantity L_e^s and use the superscript s to differentiate it from L_e(which describes light reflection at solid surfaces). Not all of $L_e^s(\vec{u},\omega)$ arrives at $\vec{x}$, since the newly added light also needs to travel through the volume where it is attenuated by the transmittance $\tau(\vec{x},\vec{u})$. To get the total amount of light added to the ray, we sum up the light contribution of all points along the ray by integrating over it [**?**]:

$$\int_{\vec{x}}^{\vec{y}} \tau(\vec{x},\vec{u})L_e^s(\vec{u},\omega)d\vec{u} \tag{9}$$

2.3. The Rendering Equation

Adding this to the light arriving from $\vec{y}$ gives the total amount of light arriving at $\vec{x}$ from ω.

$$L(\vec{x},\omega) = \int_{\vec{x}}^{\vec{y}} \tau(\vec{x},\vec{u})L_e^s(\vec{u},\omega)d\vec{u} + \tau(\vec{x},\vec{y})L_e(\vec{y},\omega) \tag{10}$$

This integral is difficult to solve, especially since the expression $L(\vec{x},\omega)$ appears on both sides of the equation. In the following, we will describe several methods for approximating a solution.

3. Ray Marching Algorithm

In this chapter we will discuss an algorithmic solution [**?**, **?**] to a simplified version of the rendering equation we derived in the previous section.

3.1. Simplified Rendering Equation

In the following, we will ignore all scattering effects in the medium (meaning μ_s is 0) and assume that the medium only absorbs and emitts light [**?**], which can be understood as a scenario where all external light has been evenly distributed within the volume. This is analogous to only rendering ambient lighting in classical ray tracing, which assumes that the light is evenly distributed in the scene. Furthermore, we will also ignore all other geometry in the scene and only focus on the volume. The term $\tau(\vec{x},\vec{y})L_e(\vec{y},\omega)$, which describes the light contribution from the nearest intersection with a piece of geometry, is omitted. Thus, we arrive at the equation

$$L(\vec{x},\omega) = \int_{\vec{x}}^{\vec{y}} \tau(\vec{x},\vec{u})\mu_a(\vec{u})\varepsilon(\vec{u},\omega)d\vec{u} \tag{11}$$

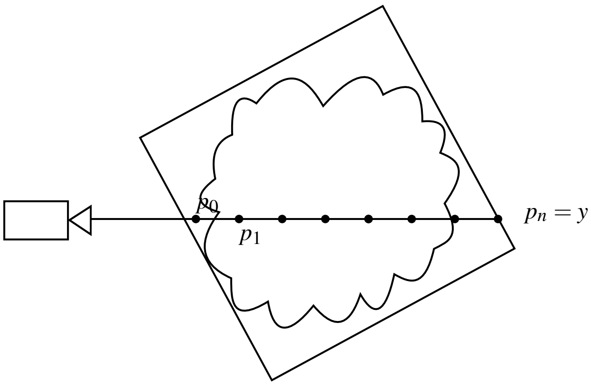


Figure 5: Projected cylinder.

For this chapter, we assume that the medium is contained within a cuboid boundary [**?**, **?**]. Since there is no geometry, we can't cast the ray until it intersects with the geometry. Instead, the endpoint $\vec{y}$ of the ray will be the point where the ray leaves the cuboid boundary(see figure **??**).

3.2. Discrete Approximation To The Simplified Equation

$L(\vec{x},\omega)$ is approximated by casting a ray in direction $-\omega$ and sampling it at evenly spaced points along the ray (see figure **??**) and summing up their contributions [**?**]. The distance s between the sampling points should be smaller than the volumes nyquist limit[CITE], otherwise important details are missed[CITE]. We start sampling at the first point within the volume, which has a distance of s_0 to $\vec{x}$. The i-th sample point then is described by

$$p_i = \vec{x} + s_0(-\omega) + si(-\omega) \tag{12}$$

The light contribution of a point p_i is considered to be the light contribution of the ray segment between p_{i-1} and p_i.

$$\int_{p_{i-1}}^{p_i} \tau(p_0,p_{i-1})\tau(p_{i-1},p_{i'})\mu_a(p_{i'})\varepsilon(p_{i'},\omega)dp_{i'} \tag{13}$$

As a simplification, we assume all variables to be piecewise constant [**?**] on the ray segments. This yields

$$\prod_{1\leq j\leq i} (\tau(p_{j-1},p_j))\cdot\mu_a(p_i)\varepsilon(p_i,\omega)s \tag{14}$$

for one line segment between p_{i-1} and p_i. The quantity $\mu_a(p_i)\varepsilon(p_i,\omega)s$ describes the emitted light from p_i and will from now on be refered to as the color $c(i)$. The product is the total attenuation (or transperancy) from $\vec{x}$ to p_i, $\tau(p_{j-1},p_j)$ is the attenuation from p_{j-1} to p_j and can be calculated by

$$\tau(p_{j-1},p_j) = e^{-s\mu_t(j)} \tag{15}$$

However, in computer graphics it is more common to use the opacity instead of the transparency. Therefore, we will refer to

$\tau(p_{j-1}, p_j)$ as $1-\alpha(j)$ from now on, where $\alpha(j)$ is the opacity. Inserting this in formula for $\tau(\vec{x}, p_i)$ yields $\prod_{1\leq j\leq i}(1-\alpha(j))$.

Again, we can refer to this quantity in terms of opacity rather than transperancy be using the equality

$$1-\beta(i) = \prod_{1\leq j\leq i}(1-\alpha(j)) \tag{16}$$

where $\beta(i)$ is the accumulated opacity between $\vec{x}$ and p_i. Using these results, we can approximate the integral in equation 11 as the sum

$$L(\vec{x}, \omega) = \sum_{1\leq i\leq n}(1-\beta(i))c(i) \tag{17}$$

which can be calculated in a single for loop [?]. Since $\beta(i)$ equals $\beta(i-1)(1-\alpha(i))$, it is not necessary to completely recalculate β for every step.

3.3. Underlying Volume Data

Due to our assumption that the variables μ_s, μ_a, μ_t and ε are piecewise constant, the opacity and color can easily be computed, if the volume provides such information. Often however, the volume is already defined in terms of color and opacity, in which case α and c can be sampled directly from it. In some cases, the volume might contain other information such as density (e.g. for medical 3D scans), in which case a preprocessing step [?] must produce color and opacity. If the volume is defined as a ternary function, sampling a point x works simply by calculating the value of the function at x. If the volume is defined as a 3D array of voxels, the value must be found through interpolation (usually through trilinear interpolation [?], but other methods such as monte carlo interpolation [?] are possible as well). When interpolating color, c must be weighted with its associated opacity before interpolation [?].

3.4. Direct Illumination

Until now, we have ignored scattering in the ray marching algorithm. In the following, we describe an approach to simulate single scattering (that is all light rays that are scattered only once) developed by Kajiya and Von Herzen [?] based on the work of Blinn [?]. This works as a two-step process. The first step is computed on a volume $\mathcal{V}$ containing opacity and albedo information and a on set of light sources $\{l_1, \dots, l_m\}$. Based on $\mathcal{V}$, one voxelized volume $\mathcal{V}'_k$ is created for every light source l_k in the following manner: For a voxel $\vec{x}$ in $\mathcal{V}'_k$ and light source l_k the amount of light $\vec{x}$ receives from l_k is calculated by attenuating the emmited light εl_k by the transmittance $\tau(\vec{x}, l_k) = \int_{\vec{x}}^{l_k}(1-\alpha(\vec{x'}))d\vec{x'}$, which can be calculated as described above. The albedo $a(\vec{x})$ regulates how much of the arriving light is reflected at $\vec{x}$. Thus, the color of $\vec{x}$ is

$$c_k(\vec{x}) = a(\vec{x})\tau(\vec{x}, l_k)\varepsilon(l_k) \tag{18}$$

Thus, the volume $\mathcal{V}'_k$ contains information about the amount of light, that is reflected from l_k at every point in space.

In the second step works very similar to ray marching as described by equation 17, with the only difference being the calculation of $c(i)$. For this, the values of $c_k(p_i)$ needs to be known for all k, which can be calculated by sampling and interpolating in $\mathcal{V}'_k$. $c_k(p_i)$ is the amount of light reflected at p_i, but only a portion is reflected towards ω (the direction to the camera). Thus, by scaling all c_k by the phase function and summing up their contribution, we can define $c(i)$ as

$$c(i) = \sum_{1\leq k\leq m} f_p(\omega, \omega_k, p_i)c_k(p_i) \tag{19}$$

with ω_k being the direction from p_i to l_k (ω_k is different for every l_k, which is why the different c_k have to be stored seperately).

This approach could be generalized to calculate multi-scattering (that is light that need two or more scattering events to reach the camera), but this would very quickly become too costly. Therefore, when rendering images with global illumination, more sophisticated algorithms are necessary[REFORMULATE! + SOURCES]. [-expensive, mention optimization algorithms, transition to mcrt, mention bias]

4. Monte Carlo Ray Tracing and Global Illumination

A common technique in classical ray tracing to generate unbiased images with global illumination is the so-called monte carlo ray tracing (MCRT) approach which solves the rendering equation by stochastically sampling it (as opossed to uniform sampling used by the ray marching algorithm described above).

In this chapter, we describe a typical MCRT, explain certain techniques necessary for its formulation and describe further improvements to the MCRT.

4.1. General Monte Carlo Algorithm

We describe a typical MCRT such as used by Hofman et al. [CITE!]. The algorithm starts like[simiilar to?] the already mentioned bycasting a ray from the camera p_0 in direction $-\omega_0$. Then, a distance d_0 is sampled stochastically and a so called path vertex p_1 is created.From p_1, a new direction ω_1 is sampled. This process is repeated until a light source is hit or until the path gets terminated (ususally by Russian roulette). Furthermore, at every path vertex p_i, next event estimation is done by casting a shadow ray towards a randomly sampled point on a light source, called l_i.[ADD: Then, the light contributions along the path are properly weighted and summed up to compute the final result.] An illustration of the process can be seen in figure **??**. For the algorithm, following quantities need to be known: The transmittance τ between the various path vertices and light sources, the directions ω_i, and the locations of l_i and p_i. The sampling of ω_i and l_i works like in classical ray tracing, using multipe importance sampling[CITE] of the phase function and light distribution. Sampling p_i and estimating the transmittance require novel approaches not used in normal ray tracing, two of which are described below.

4.2. Delta Tracking

To calculate $p_{i+1} = p_i - \omega t_t$ from a given p_i and ω_i, the distance t_i between those two points must be sampled. Recall that $\tau(p_i, p_{i+1})$ describes the ratio of light arriving from p_{i+1} at p_i without colliding with a particle. Therefore, the probability that a collision occurs between p_i and p_{i+1} is

$$1-\tau p_i, p_{i+1} \tag{20}$$

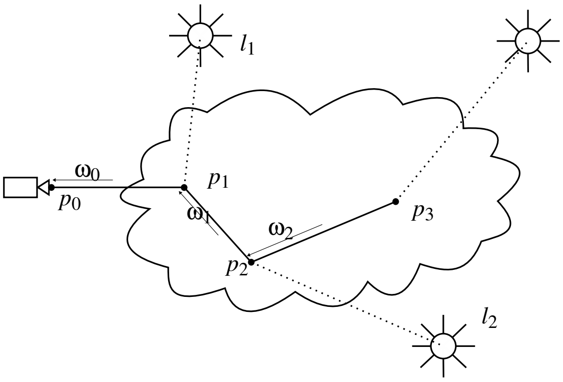


Figure 6: Projected cylinder.

This can be used to importance sample the distance t_i by inverting equation **??** and applyingthe resulting function to ξ (a random variable uniformly sampled from $[0,1]$). However, τ is defined as $e^{\int_0^{t_i} \mu_t(s)ds}$, which can't be easily inverted if the medium is heterogenous (if μ_t is not constant). Delta tracking (alos known as woodcock tracking) solves this problem by introducing so called fictitious particles, which neither scatter nor absorb light rays. Their distribution μ_f is chosen so that the sum of μ_t and μ_f add up to a globaly constant value, the so called majorant $\bar{\mu}$. The orignal problem of finding a real collision is then reformulated to finding any collision (fictitious or real). Since $\bar{\mu}$ is a constant, equation 20 can easily be inverted and we get

$$t_i = -\frac{ln(1-\xi)}{\bar{\mu}} \tag{21}$$

as the sampled distance to the next collision, which might be with a real of a fictitious particle (a so called null collision). At position $p_i - \omega_i t_i$, the probability of hitting a real particle is $\frac{\mu_t(p_i - \omega_i t_i)}{\bar{\mu}}$. Thus, we can decide if a real particle has been hit by sampling another ξ and comparing it $\frac{\mu_t(p_i - \omega_i t_i)}{\bar{\mu}}$. If ξ is smaller, a real particle has been hit, if not, there is a null collision. In this case, we repeat the process (illustrated in figure 7) from the point $p_i - \omega_i t_i$ until a real collision is detected. The distance to this real collision is then the sum of all distances previously computed. The position of the real collision is then assumed to be the path vertex p_{i+1}.

Delta tracking can also be used to estimate the transmmittance τ between two points p' and $p + \omega t$. This is done by using delta tracking to find a real collision on the ray from p to p'. If this collision occurs before p', τ is assumed to be 0, if the collision occurs after p', τ is estimated as 1 (see figure 8). However, there exists other, more sophisticated methods for etimating the transmittance, such as ratio tracking.

4.3. Ratio Tracking

Ratio tracking is used to estimate the transmittance between two already known points p and p'. This is done by calculating various collision points between p and p', as described above, and stops once a collision point behind p' is reached. At each collision point p^i, the value $\frac{\mu_f}{\bar{\mu}} = 1 - \frac{\mu_t}{\overline{mu}}$ is saved. This fraction between fictious

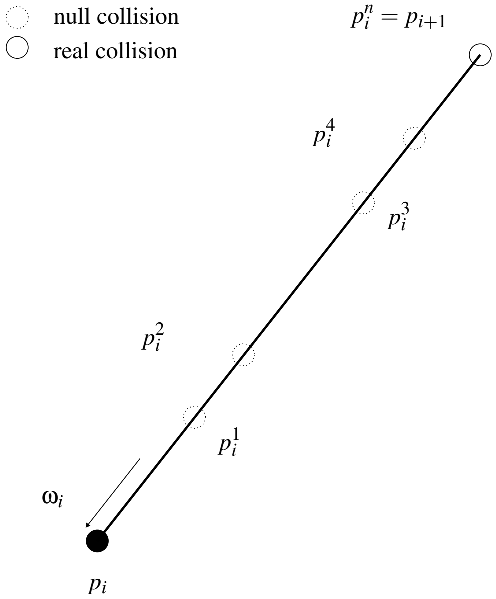


Figure 7: Projected cylinder.

particles and all particles describes the probability, that a light ray can pass p^i without real collision. The transmittance τ can then be estimated as

$$\tau(p,p') = \prod_{1 \leq i \leq} (1 - \frac{\mu_t}{\bar{\mu}}) \tag{22}$$

where n is the index of the last collision found before p'.

5. Selection of Various Monte Carlo Ray Tracing Algorithms

5.1. Metropolis Lighting

5.2. Bidirectional Path Tracing

6. Conclusion

7. REPETITION

7.1. Underlying Volume Data

In this paper, we consider the topic of synthesizing a 2D image from a 3D volume. A volume can either be some continuous, ternary function (such as perlin or simplex noise [**?**], which can be used to render volumetric clouds [**?**]), or a discrete 3D array [**?**]. Similar to a 2D image that consists of pixels that can be addressed by a 2D vector $\vec{x} \in \mathbb{N}^2$, a 3D array consists of voxels that are addressed by a 3D vector $\vec{x} \in \mathbb{N}^3$. [REFORMULATE?]The color of a voxel at position $\vec{x}$ is called $c(\vec{x})$, the opacity $\alpha(\vec{x})$ and the complete vector $v(\vec{x})$. The opacity is a scalar value, the color may also be a scalar value (for grayscale volumes) or a 3D vector (for colorful volumes). To address the components of a vector $\vec{v}$, following notation is used:

$$\vec{v}_x, x \in sr, g, b, \alpha \tag{23}$$

In the following, all values are assumed to be normalized to between 0 and 1. To project this volume to a image, for each pixel in

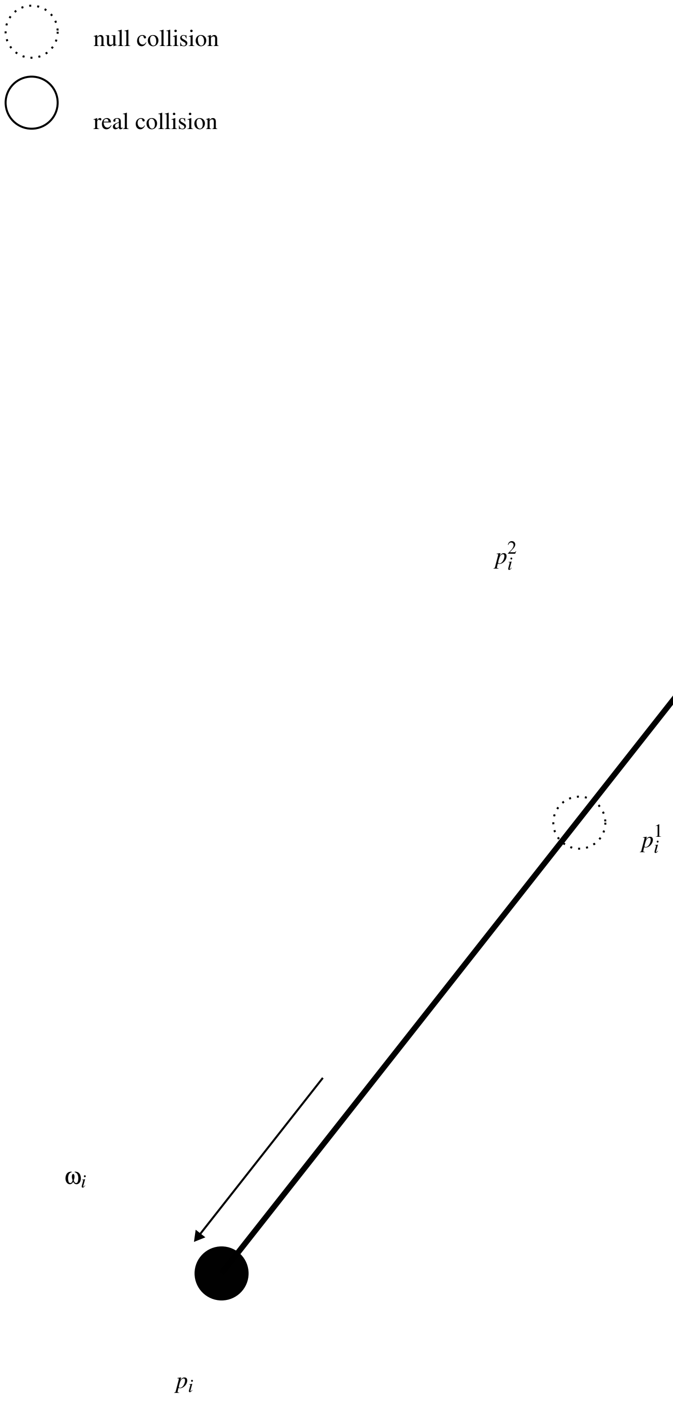


Figure 8: A figure that contains three subfigures

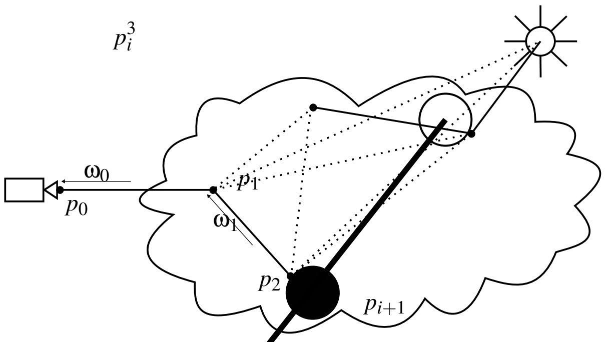


Figure 9: Projected cylinder.

the image a ray is cast through the volume and sampled at multiple, evenly spaced points on the ray (see figure 1) [**?**].

7.2. Interpolation

The sample points on the ray are in general part of $\mathbb{R}^3$. This is no problem if the volume is described by a continuous function, which is defined everywhere. However, if it is a discrete 3D array only defined for points in $\mathbb{N}^3$, the value of the volume at the sample point must be interpolated. This interpolation is done over the 8 closest voxels to the sample point (usually with a trilinear interpolation). The opacity values can be interpolated regularly, but the color values must be weighted with the respective opacity values before interpolation [**?**]. To see why this is necessary, consider figure 2, which presents a simplified 2D example of a completely opaque, white object behind completely transparent empty space. Naively interpolating in this volume would result in two sample points, one completely white and opaque, the other gray and semitransparent. This gray point is not present in the original data and is an unwanted artifact. Weighting the colors with their opacity prevents this from happening.

8. Optimization Strategies

In this chapter, we will shortly desribe several optimizations for the basic ray tracing algorithm.

8.1. Adaptive Termination Of Ray Tracing

When looking at the compositing process described above, it becomes apparent that the contribution of a sample point to the final result decreases, the more other opaque sample points lie before it [**?**]. For an extreme example, consider figure 3. Here, all sample points behind the fully opaque black wall contribute nothing to the final value and can therefore be ignored. In other words, going back to equation (2)[CHANGE REFERENCE], the ray casting can be terminated after the accumulated opacity reaches 1, without changing the image quality. If some errors are acceptable, the threshold value might be chosen to be somewhat lower.

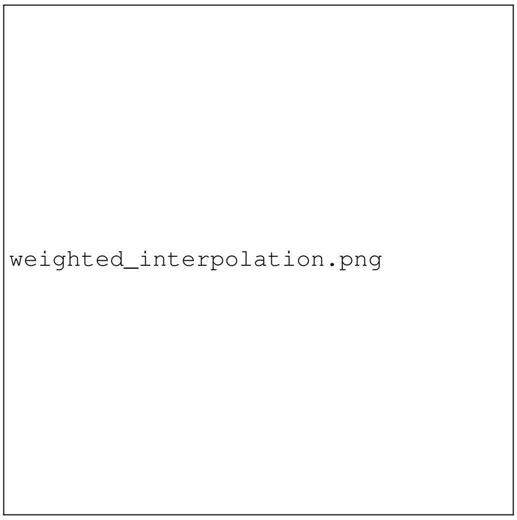


Figure 10: Simplified visualization of naive interpolation in 2 dimensions. The black dots represent the voxels, the red dots the sample points. Notice that the second sample point has a gray color, even though the original data only has white voxels and blck, transparent voxels, which shouldn't affect the final color. Adapted from Wittenbrink et al. [**?**]

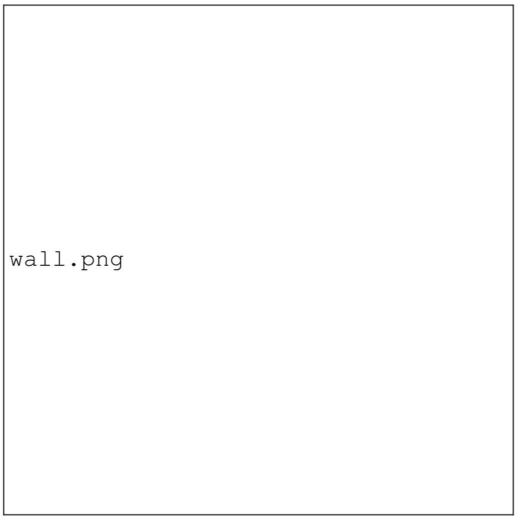


Figure 11: The black, opaque object blocks parts of the volumes behind it. The samples behind the object (from the cameras perspective) are not visible and therefore irrelevant.

8.2. Ray Termination With Russian Roulette

The above described termination creates a bias in the image [**?**]. One way to avoid this is to use Russian Roulette to decide when to terminate a ray [**?**]. Unlike the above described approach, Russian Roulette terminates a ray only with a certain likelihood once the accumulated opacity reaches the threshold. The weight of the surviving rays is increased proportionally to compensate for the terminated rays.

8.3. Pyramid Data Structures

The following optimizations use data structures called pyramids, which are created from 3D arrays containig color and opacity, if the original volume is defined differently, an intermediate representation must be created first. Analogous to a mip-map which is a set of 2D arrays with decreasing resolution, a pyramid is a set of 3D arrays with decreasing resolution. The different volumes of which the pyramid consits are called the levels of the pyramid. The lowest level has the highest resolution, each succesive level has half the resolution than the preceeding one. To ensure that each level covers the same region, the distance between the voxels are doubled each level.

In the following, we will use average, maximum and range pyramids.

Each voxel v in the average pyramid stores the average value of the original volume within its area of influence (that is the region of space closer to v than any other voxel). The pyramid can be sampled at every level n similarly to original volume by interpolation, but the distance between the sample points is 2^n times greater than in the original volume, speeding up the sampling. Since 1 sample point in the average pyramid needs to approximate 2^n original sample points, the sampled value is blended with itself 2^n times.

Each voxel v in the maximum pyramid stores the maximum value of the original volume within its area of influence. The maximum pyramid is sanpled with nearest neighbor interpolation (unlike in the average pyramid, the sample does not need to be blended with itself). The minimum pyramid works analogously.

The range pyramid is calculated by taking the difference between the maximum and minimum pyramid. If those pyramids store vector values, the average of the difference vector is stored in the range pyramid instead. The range pyramid is sampled like the maximum pyramid and provides a measure of the homogenety of the original data.

8.4. Fixed Step Multiresolution

This optimization uses the average pyramid, and works by casting the rays through a level of the pyramid, instead of the original data. The level through which the rays are cast can be freely choosen. Similar to mip-maps, it is also possible to select a real number as a level, in this case, the levels above and below the chosen level are rendered, and the two results are interpolated together. Since this would require rendering the pyramid at two levels, this is slower than just choosing a natural number as a level and recommendable only if a smooth transition between two levels is needed, like in gaze directed rendering as desribed by Levoy(CITE!!). The higher the level the faster the algorithms works, but the lower the resolution becomes.

8.5. Presence And Homogenity Acceleration

Presence acceleration uses an average and a maximum pyramid. The maximum pyramid is used to quickly find regions with opacity lower than a user provided threshold, where the average pyramid is used to sample at a lower resolution. The idea behind this algorithm

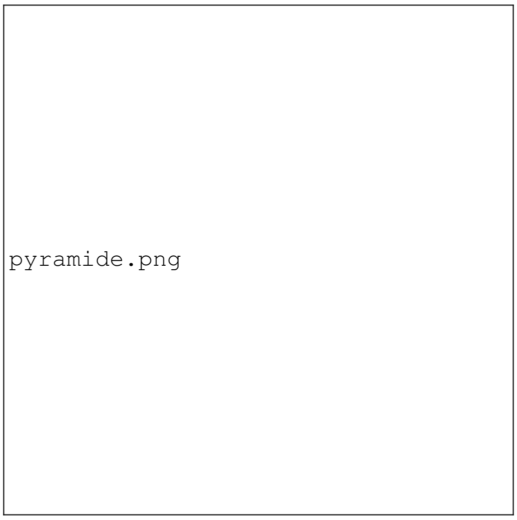


Figure 12: Simplified 2 dimensional example of 2 levels of a pyramid. Each pixel in level n +1 contains 4 pixel of the lower level. In 3 dimensions, 8 voxels of the lower level are contained in one voxel of the upper level.

is that low opacity regions do not contribute much to the final result, and can be therefore be sampled more sparsely.

Homogenity acceleration works conceptually the same as presence acceleration. The difference is that homogenity acceleration takes fewer samples in regions with high homogenity and not in regions of low opacity. The idea behind the algorithm is that in a region where all voxels are very similar to each other, accumulating x different samples and accumulating the average of the region x times yields a very similar result.

8.6. Presence Acceleration

This algorithm uses an average and a maximum pyramid. The maximum pyramid is used to quickly find regions with opacity lower than a user provided threshold, where the average pyramid is used to sample at a lower resolution. The idea behind this algorithm is that low opacity regions do not contribute much to the final result, and can be therefore be sampled more sparsely.

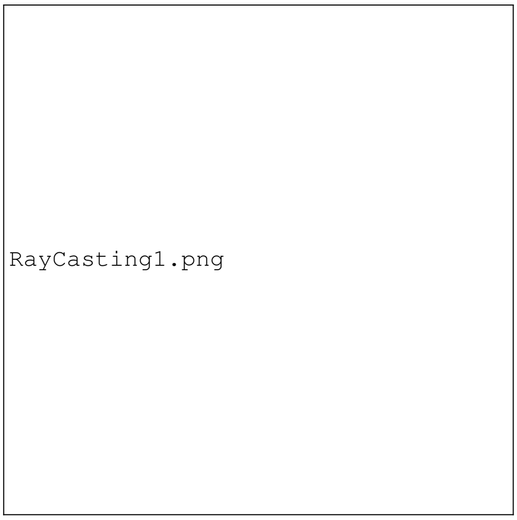
This algorithm uses an average and a maximum pyramid. The maximum pyramid is used to quickly find regions with low opacity, where the average pyramid is used to sample at a lower resolution. The idea behind this algorithm is that low opacity regions do not contribute much to the final result, and can be therefore be sampled more sparsely. The algorithm works by casting each ray through the maximum pyramid at the highest level. For each cell that is intersected, it checks if the opacity value of that cell is larger than some user provided treshold $k \in [0,1]$. If so, that means that the average opacity in that region is not small enough, and we move down one level in the pyramid in the hope of finding such a region at a lower resolution. However, if the opacity is smaller the average pyramid is used to approximate that cell. [MORE DETAILS?] Once a cell in the maximum pyramid has been checked, the algorithm advances to the next cell and checks if the new cell has the same parent as the old one. If not, we move up one level in the pyramid. This is done to ensure that the algorithm always advances with the largest possible step size. After that, the same procedure is repeated until the ray has moved through the entire pyramid. Further improvements to this algorithm can be madeby considering two observations: Firstly, it is very rare for the cell in the highest level of the maximum pyramid to have a α lower than k (this would mean that the entire volume is almost completly transparent). It is therefore preferable not to start at the highest level but samewhat lower. XXX suggests in [CITE] to start two levels lower. Secondly, finding the cells that the ray intersects is a computationally relatively expensive operation that might not amortize itself at lower levels. Therefore, it might be more effiecient to start sampling at the lowest level of the average pyramid before the lowest level of the maximum pyramid is reached. XXX suggests in [] to sample level 0 of the average pyramid once level 2 of the maximum pyramid is reached. [REFORMULATE?]

8.7. Homogenity Acceleration

This algorithm works conceptually the same as presence acceleration. The difference is that homogenity acceleration takes fewer samples in regions with high homogenity and not in regions of low opacity. The idea behind the algorithm is that in a region where all voxels are very similar to each other, accumulating x different samples and accumulating the average of the region x times yields a very similar result. The algorithm works in the same way as presence acceleration, only the maximum pyramid is replaced with the range pyramid.

8.8. β Acceleration

9. REPETITION



[?]

Figure 13: Illustration of the ray casting process.

9.1. Terminology

To enable a clear understanding in this section we will shortly define key terminology used in this paper. A voxel is an infinitesimally small point in 3D space with an associated vector value, that represents the color of that voxel. In this paper, these voxels are arranged in a regular cube lattice (although other arrangements are possible as well). A cell or area of influence of a voxel v is defined as the region of space which is closer to v than to any other voxel (meaning a cube in space with v in the center). A voxel cube is defined as a group of 8 voxels that form a cube in space. The neighborhood of a voxel v is defined as the set that includes v as well as its 26 neighbors in 3D space. Since different authors use different notations, this terminology might be used slightly different in other papers.

9.2. Underlying Volume Data

In this paper, we consider the topic of synthesizing a 2D image from a 3D volume. A volume can either be some continuous, ternary function (such as perlin or simplex noise [**?**], which can be used to render volumetric clouds [**?**]), or a discrete 3D array [**?**]. Similar to a 2D image that consists of pixels that can be addressed by a 2D vector $\vec{x} \in \mathbb{N}^2$, a 3D array consists of voxels that are addressed by a 3D vector $\vec{x} \in \mathbb{N}^3$. [REFORMULATE?]The color of a voxel at position $\vec{x}$ is called $c(\vec{x})$, the opacity $\alpha(\vec{x})$ and the complete vector $v(\vec{x})$. The opacity is a scalar value, the color may also be a scalar value (for grayscale volumes) or a 3D vector (for colorful volumes). To address the components of a vector $\vec{v}$, following notation is used:

$$\vec{v}_x, x \in sr, g, b, \alpha \tag{24}$$

In the following, all values are assumed to be normalized to between 0 and 1. To project this volume to a image, for each pixel in the image a ray is cast through the volume and sampled at multiple, evenly spaced points on the ray (see figure 1) [**?**].

9.3. Volumetric Ray Tracing

Like in the classical ray tracing algorithm, volumetric ray tracing works by casting a ray (or multiple rays if multisampling is used) for each pixel in the image that is to be created.

These rays are described by the equation $\vec{o} + t \cdot \vec{d}$, where $\vec{o}$ is the origin of the ray and $\vec{d}$ the direction. On each ray evenly spaced sample points are placed whose position is described by

$$\vec{o} + n \cdot s \cdot \vec{d} \tag{25}$$

for the n-th sample point. s is a scale factor, determining how far apart the sample points are. s should be roughly equal to the distance between voxels (this distance is assumed to be 1 in the volume model, but depending on the volumes world matrix this might be different in world coordinates), since a too great mismatch between sampling and voxel frequency woul lead to aliasing. In the follwing, only sample points within the volume are considered. Those sample points then are sampled and composited together, resulting in a final color value for the image pixel.

9.4. Interpolation

The sample points on the ray are in general part of $\mathbb{R}^3$. This is no problem if the volume is described by a continuous function, which is defined everywhere. However, if it is a discrete 3D array only defined for points in $\mathbb{N}^3$, the value of the volume at the sample point must be interpolated. This interpolation is done over the 8 closest voxels to the sample point (usually with a trilinear interpolation). The opacity values can be interpolated regularly, but the color values must be weighted with the respective opacity values before interpolation [**?**]. To see why this is necessary, consider figure 2, which presents a simplified 2D example of a completely opaque, white object behind completely transparent empty space. Naively interpolating in this volume would result in two sample points, one completely white and opaque, the other gray and semitransparent. This gray point is not present in the original data and is an unwanted artifact. Weighting the colors with their opacity prevents this from happening.

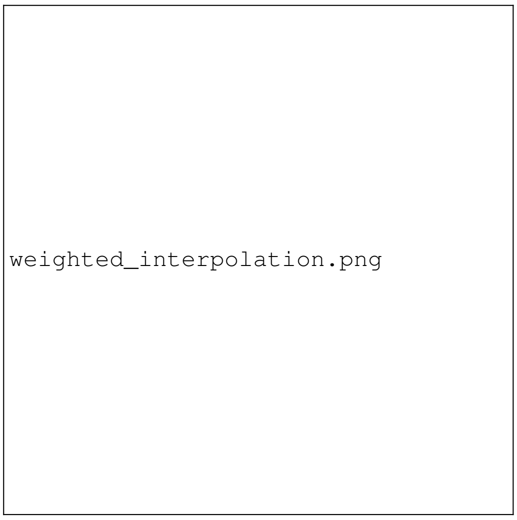


Figure 14: Simplified visualization of naive interpolation in 2 dimensions. The black dots represent the voxels, the red dots the sample points. Notice that the second sample point has a gray color, even though the original data only has white voxels and blck, transparent voxels, which shouldn't affect the final color. Adapted from Wittenbrink et al. [**?**]

9.5. Compositing Of Multiple Sample Points

To compose the various sampled points on the ray to a single pixel, the points one after the other are alpha blended together. This compositing can be done front to back [**?**, **?**] (starting with the sample point closest to the camera, blending it with the second, then the third, and so on), or back to front [**?**, **?**] (vice versa). Usually, the back to front approach is chosen since it allows to optimize the computation [**?**] (see below). In the following, $c(i)$ is the color of the i-th sample point, and $\alpha(i)$ the opacity. The accumulated opacity

$$\beta(i) = 1 - \prod_{j=1}^{i} (1 - \alpha(j)) \tag{26}$$

is the fraction of light absorbed between the first and i-th sample point. The final color c_f, which is displayed in the projected image, is then

$$c_f = \sum_{i=1}^{n}((1-\beta(i)) * c(i)) \tag{27}$$

if the ray has n sample points [?].

10. Optimization Strategies

10.1. Adaptive Termination Of Ray Tracing

When looking at the compositing process described above, it becomes apparent that the contribution of a sample point to the final result decreases, the more other opaque sample points lie before it [?]. For an extreme example, consider figure 3. Here, all sample points behind the fully opaque black wall contribute nothing to the final value and can therefore be ignored. In other words, going back to equation (2), the ray casting can be terminated after the accumulated opacity reaches 1, without changing the image quality. If some errors in the image are acceptable, the threshold value for terminating the ray tracing might be chosen to be somewhat lower.

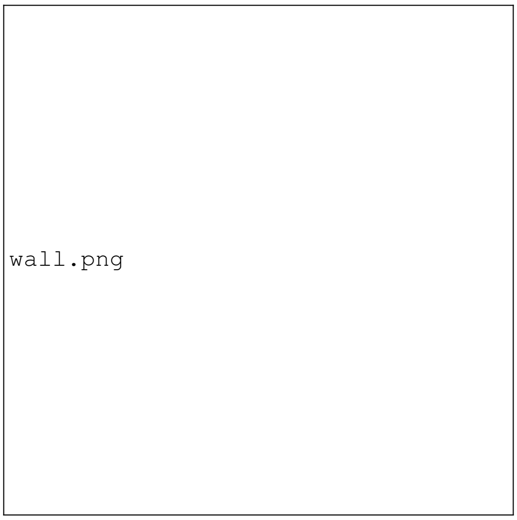


Figure 15: The black, opaque object blocks parts of the volumes behind it. The samples behind the object (from the cameras perspective) are not visible and therefore irrelevant.

10.2. Ray Termination With Russian Roulette

The above described termination after a certain threshold is reached creates a bias in the synthesized image [?]. One way to avoid this is to use Russian Roulette to decide when to terminate a ray [?]. Unlike the above described approach, Russian Roulette terminates a ray only with a certain likelihood once the accumulated opacity reaches the threshold. The weight of the surviving rays is increased proportionally to compensate for the terminated rays.

10.3. Pyramid Data Structures

The follwing optimizations require a data structure called a pyramid. Pyramids are created from 3D arrays, if the volume to be rendered is defined as a function an intermediate array representation must be created. Analogous to a mip-map which is a set of 2D arrays with decreasing resolution, a pyramid is a set of 3D arrays with decreasing resolution. The different volumes of which the pyramid consits are called the levels of the pyramid. The lowest level (called level 0) has the highest resolution, each succesive level has half the resolution than the preceeding one. To ensure that each level covers the same region in 3D space, the distance between the voxels are doubled each level.

–FIGURE! In the follwing, we will make use several different kinds of pyramids, namely average, maximum and range pyramids.

10.4. Average Pyramid

The lowest level of the average pyramid is equal to the original 3D array, padded with zeros so that its size is a power of 2 in all dimensions. The next level is created by aggregating the average of a cube of 8 voxels in the lower level to one voxel in the higher level. The voxel in the higher level is called the parent, the 8 lower voxels are called the children. This process is repeated until the last level, consisting of only one voxel, is reached. Similarily to the original volume, the pyramid can be sampled by interpolating the 8 closest voxels to the sample point. Consider however, that the primary purpose of the average pyramid is to enamble an approximation of the original data that is computationally faster than to render the original data. The time complexity of rendering 3D data however does not depend on the number of voxels, but rather the number of samples along the ray (at first glance, this seems to imply that the average pyramid is superfluous and that the same could be accomplished by taking fewer sampling points in the original volume. This however would lead to a sample frequency significantly lower than the voxel frequency and thus to aliasing). If we therefore wish to cut the rendering time in half for each level we move up the pyramid, the amount of sampling points must be halfed and the distance between the sampling points must be doubled. This means that one sample in level $n+1$ should approximate two samples at level n. To achieve this, the sample at level $n+1$ must be blended with itself, since in level n two samples are blended as well.

10.5. Maximum And Minimum Pyramids

A maximum pyramid works similar to an average pyramid, but the construction of level 0 is somewhat different. A voxel in level 0 at position $\vec{x}$ is created by taking the maximum of the 27 voxels in the original dataset, that form a 3 by 3 cube around $\vec{x}$. Once the lowest level is created the succeding levels are constructed llike in the average pyramid, only that the maximum is used as the aggregation method, and not the maximum. Sampling a maximum pyramid is different from sampling an average pyramid. Since the maximum pyramid stores maxima, interpolating between different maxima would lead to wrong results. Instead, nearest neighbor interpolation is used (unlike in the average pyramid, the sample does not need to be blended with itself). The minimum pyramid works analogously. In the following, the sample taken at location $\vec{x}$ in the maximum pyramid at level n is called $pyr_{max}(\vec{x}, n)$, and the sample in the minimum pyramid is called $pyr_{min}(\vec{x}, n)$.

10.6. Range Pyramid

The range pyramid is calculated by taking the difference between the maximum and minimum pyramid. Both the maximum and minimum pyramid store vectors (2D or 4D, depending on wether the underlying data is grayscale or colorful). To express the difference in a scalar, the diffrence of the components is summed up and normalized to the range of 0 and 1:

$$\frac{pyr_{range}(\vec{x},n) = 1}{4\cdot((pyr_{max}(\vec{x},n)_r - pyr_{min}(\vec{x},n)_r) + (pyr_{max}(\vec{x},n)_g - pyr_{min}(\vec{x},n)_g) + (pyr_{max}(\vec{x},n)_b - pyr_{min}(\vec{x},n)_b) + (pyr_{max}(\vec{x},n)_\alpha - pyr_{min}(\vec{x},n)_\alpha))} \quad (28)$$

The range pyramid is sampled like the maximum and minimum pyramids and provides a measure of the homogenety of the original data. If a voxel in the range pyramid has the value 0, this means that the region of space covered by this pixel is completly homogenous in the original data, if the voxel is 1, this means at least 2 voxels in the original data are completly different.

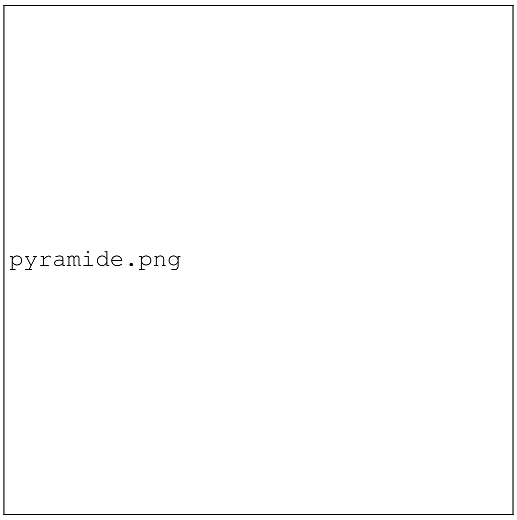


Figure 16: Simplified 2 dimensional example of 2 levels of a pyramid. Each pixel in level n +1 contains 4 pixel of the lower level. In 3 dimensions, 8 voxels of the lower level are contained in one voxel of the upper level.

10.7. Fixed Step Multiresolution

This optimization makes use of the above described average pyramid, and works by casting the rays through a level of the pyramid, instead of the original data. As previously mentioned, when casting a ray trhough level n, the distance between the sampling points is 2^n times larger than the normal distance. The level through which the rays are cast can be freely choosen. Similar to mip-maps, it is also possible to select a real number as a level, in this case, the levels above and below the chosen level are rendered, and the two results are interpolated together. Since this would require rendering the pyramid at two levels, this is slower than just choosing a natural number as a level and recommendable only if a smooth transition between two levels is needed, like in gaze directed rendering as desribed by Levoy(CITE!!). The higher the level the faster the algorithms works, but the lower the resolution becomes.

10.8. Presence Acceleration

This algorithm uses an average and a maximum pyramid. The maximum pyramid is used to quickly find regions with low opacity, where the average pyramid is used to sample at a lower resolution. The idea behind this algorithm is that low opacity regions do not contribute much to the final result, and can be therefore be sampled more sparsely. The algorithm works by casting each ray through the maximum pyramid at the highest level. For each cell that is intersected, it checks if the opacity value of that cell is larger than some user provided treshold $k \in [0,1]$. If so, that means that the average opacity in that region is not small enough, and we move down one level in the pyramid in the hope of finding such a region at a lower resolution. However, if the opacity is smaller the average pyramid is used to approximate that cell. [MORE DETAILS?] Once a cell in the maximum pyramid has been checked, the algorithm advances to the next cell and checks if the new cell has the same parent as the old one. If not, we move up one level in the pyramid. This is done to ensure that the algorithm always advances with the largest possible step size. After that, the same procedure is repeated until the ray has moved through the entire pyramid. Further improvements to this algorithm can be madeby considering two observations: Firstly, it is very rare for the cell in the highest level of the maximum pyramid to have a α lower than k (this would mean that the entire volume is almost completly transparent). It is therefore preferable not to start at the highest level but samewhat lower. XXX suggests in [CITE] to start two levels lower. Secondly, finding the cells that the ray intersects is a computationally relatively expensive operation that might not amortize itself at lower levels. Therefore, it might be more effiecient to start sampling at the lowest level of the average pyramid before the lowest level of the maximum pyramid is reached. XXX suggests in [] to sample level 0 of the average pyramid once level 2 of the maximum pyramid is reached. [REFORMULATE?]

10.9. Homogenity Acceleration

This algorithm works conceptually the same as presence acceleration. The difference is that homogenity acceleration takes fewer samples in regions with high homogenity and not in regions of low opacity. The idea behind the algorithm is that in a region where all voxels are very similar to each other, accumulating x different samples and accumulating the average of the region x times yields a very similar result. The algorithm works in the same way as presence acceleration, only the maximum pyramid is replaced with the range pyramid.

10.10. β Acceleration