GPU Ray Marching of Distance Fields

Lukasz Jaroslaw Tomczak

Supervised by
Professor Jeppe Revall Frisvad
Professor Jakob Andreas Bærentzen

Kongens Lyngby 2012
IMM-M.Sc-2012-58

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc-2012-58

Abstract

Computer Graphics since its existence has been striving to visualize its representations of surfaces and objects to accurately match their real life counterparts. We investigate the correctness of the ambient occlusion effect produced with the use of a distance to the closest surface metric. This value is obtained from a signed distance function/field. We evaluate the robustness of the method as well as its ease of use. We also test how this applies to a polygon mesh converted into a 3D texture of distance values and included into our scenes. For rendering images we use our GPU implementation of ray marching in the form of sphere tracing and we compare the results to the ones produced by a ray tracing framework.

Contents

List of Figures

List of Algorithms

CHAPTER 1

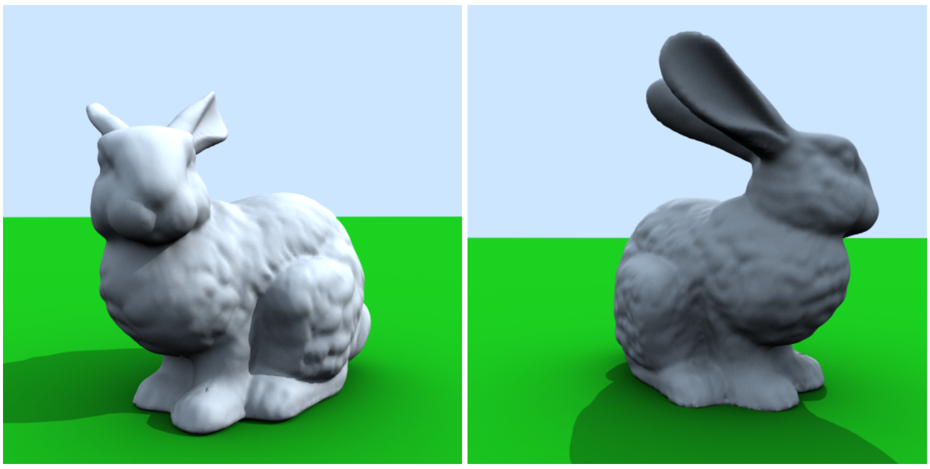
Introduction

In producing movies or games a lot of effort is put in making convincing images [IKSZ03, Lan02]. However, the correctness of a image in terms of its realism rarely overshadows the time cost of obtaining it. Methods like radiosity can require a great amount of time before they produce a correct effect. That is why computing global illumination with it is not a first choice for movie or game making.

In these areas of application, focus is on developing other techniques or modifying already existing ones that strive to achieve a certain degree of believable realism while at the same time producing images in an acceptable time. One of such influential works is the one by Zhukov et al. in [ZIK98] where the authors propose an ambient light illumination model based on the concept of *obscurance*. The authors work has been widely used and refined by the movie industry and production rendering community as it is stated in [KL05].

This thesis has a similar purpose. It will primary focus at the topic if the ambient occlusion effect computed with the use of a ray tracer implementation can be acquired with the use of distance fields and maintain a certain degree of acceptable realism.

As stated in [KL05] the ambient occlusion technique itself is a simplified version of the obscurance model proposed by Zhukov et al. Ambient occlusion can be



(a) Render time ∼ 47 minutes (b) Render time ∼ 46 minutes

Figure 1.1: Reference images of a Stanford Bunny on a ground plane

defined like it is by Klehm et al. in [KRES11] as a nonphysically-based approximation of environmental lighting. Its use can give the impression of having global illumination calculated for the rendered scene. As the main advantages of the method the authors give high performance in estimating the average occlusion of surfaces which influences the shading result. Moreover, the authors mention that, this estimation is done without taking into consideration the light sources in the scene, and that is why the result can lack of certain realism due to the missing influence of directional occlusion and lighting. The technique and its variations are quite popular in real-time and off-line rendering due to the high ratio of realism to image production time. The topic is still an active area of research notably in the fields for video games and movies as stated in [MFS09]. Some of the focus might be a specific case of occlusion, like self-occlusion or occlusion resulting between objects. For a broader overview of the concept the reader is referred to the aforementioned source.

The goal for this thesis is to be able to achieve results similar or as close as possible to the ones presented in Figure 1.1, or satisfying a certain degree of realism using distance fields and ray marching (or its variants). However, an additional condition is that scene visible in the aforementioned figure would be rendered in interactive frame rates by the implementation made for the purpose of this master thesis.

Images such as those in Figure 1.1 and similar (where it is stated) will be considered as reference ones and are obtained with the use of a modified (to include ambient occlusion) CPU implementation of a ray tracer framework used in the 02576 Physically Based Rendering course held at DTU [Fri12] which implementation was provided by Jeppe Revall Frisvad. The rendering is done with the use of an Intel® Core™ i5-750 Processor. The model used in the figures is that of the Stanford Bunny [Tur00]. It was chosen since of its availability as well as its fairly detailed surface structure.

The dimensions of the reference images as well as those generated for this thesis are of 512×512 pixels unless otherwise stated. Due to the formating of the thesis these are shrunk to fit the space provided for them in the layout. This might cause some artifacts or make some artifacts in the images (that resulted from using a specific technique/implementation method) less visible. In such cases areas of vital importance for an image will be displayed unscaled alongside the full image.

The structure of the thesis was chosen such that each chapter will bring the reader closer to the solution for the goal of this thesis. Moreover intermediate steps in the process will be accompanied by results for them. The reason for this decision was to allow the reader to see the effects of each vital step and allowing for a greater and easier understanding of the matter at hand.

1.1 Notes on implementation

We have implemented the sphere tracing algorithm on the GPU and we based it on the presented solution by Iñiqo Quilez in [nQ08b]. The implicit surface rendering is done in the pixel shader (also referred to as the fragment shader), which is applied to a screen representation made out of two triangles occupying the whole screen area.

The color value for each pixel is determined by the fragment shader program running for that pixel. Because of that, the implementation traces a ray for each pixel taking advantage of the parallel execution of fragment shader programs on the GPU. Thus many rays are traced at the same time for a group of pixels and that number depends on hardware capabilities of the GPU.

It is vital to state that the implementation requires a GPU with programmable fragment processing units, and that not all GPU architectures will be able to run the code. This is because some of the shader instructions or their number might not be adequate for older GPU architectures.

CHAPTER 2

Background

In computer graphics in order to visualize a surface one needs, first of all, to define its representation and based on that how the rendering process will be accomplished. This chapter will elaborate on that matter and define surface representations and methods used for rendering in the thesis.

2.1 Surface representations

As indicated in [Men96] there exist two main representations of surfaces that are used in geometric modeling and computer graphics: parametric and implicit.

In the parametric case the author mentions that an implicit surface can be defined as a set of points $\boldsymbol{x}$ in space that are represented by a set of coordinates (x, y, z) which have been parameterized by some variables s and t. This yields an equation of the form:

$$\boldsymbol{x}(s,t) = (x(s,t), y(s,t), z(s,t)) \tag{2.1}$$

where the parameters s and t belong to a certain domain. For example as it can be seen from [Wei] that a sphere centered at the origin of a coordinate system can be defined in a parametric form with the use of its spherical coordinates:

$$x(s,t) = r\cos(s)\sin(t) \tag{2.2}$$
$$y(s,t) = r\sin(s)\sin(t) \tag{2.3}$$
$$z(s,t) = r\cos(t) \tag{2.4}$$

where r is the radius of the sphere, s is a parameter representing the azimuthal coordinate such that $s \in [0, 2\pi]$ and t is a parameter representing the polar coordinate such that $t \in [0, \pi]$.

Menon in [Men96] as examples of parametric methods lists non-uniform rational B-splines or NURBS and briefly lists the motivations for such methods among which he states an important property for computer graphics which is as an efficient evaluation of points on the surface.

However, of interest for this master thesis are implicit surfaces. These, as mentioned by Menon have advantages over parametric surfaces. One of them being, that the surface to surface intersections calculations, as the author states: "are much more computationally (numerically as well as topologically) tractable". Menon mentions that an implicit surface can be defined as the zero contour of a function, that is:

$$f(\boldsymbol{x}) = f(x,y,z) = 0 \tag{2.5}$$

where as given by Hart in [Har93] $f : \mathbb{R}^3 \to \mathbb{R}$ and $\boldsymbol{x} \equiv (x,y,z) \in \mathbb{R}^3$. Moreover, by evaluating that function for any given point $\boldsymbol{x}$ in space, one can apart from assessing if the point is on the surface determine if the point is enclosed by or outside of it. Hart in his paper represents this as follows:

$$\boldsymbol{x} \in \mathring{A} \Leftrightarrow f(\boldsymbol{x}) < 0 \tag{2.6}$$
$$\boldsymbol{x} \in \partial A \Leftrightarrow f(\boldsymbol{x}) = 0 \tag{2.7}$$
$$\boldsymbol{x} \in \mathbb{R}^3 - A \Leftrightarrow f(\boldsymbol{x}) > 0 \tag{2.8}$$

where A represents a closed solid that is described by the implicit function f. Here the solid's interior is denoted as $\mathring{A}$, its surface as ∂A and exterior as $\mathbb{R}^3 - A$. Hart explains, that the above notational assumption regarding the assessment of a point's position with respect of the closed solid A can be understood "via point-set topology, that the implicit function is negative inside the solid, zero on its surface and positive outside". This description of an implicit function denoting a surface will be used in this thesis. Hart mentions that it is the most common one found in literature.

Menon in [Men96] amongst other advantages of implicit methods gives that these provide mathematical tractability, are useful for modeling operations like blending, boolean operations and other.

There also exist explicit methods as is stated by Menon and these for example can define the surface in the form:

$$y = g(x) \tag{2.9}$$
$$z = k(x) \tag{2.10}$$

However, the author explains that this form of representation is quite limiting. For example, for a given x it is impossible to get multiple values of y, thus in order to do so one must use multiple curve segments. Menon identifies three major areas of work based on implicit surfaces. These include algebraic surfaces, blobby objects and functional representations.

Hart in [Har96] states that the most common studied form of implicit surfaces are algebraic ones and when introducing the subject provides as an example the representation of a sphere, which can be given by the equation:

$$x^2 + y^2 + z^2 - r^2 = 0 \tag{2.11}$$

Where as before r denotes the radius of the sphere and its center is at the center of the coordinate system. In the above form the sphere is represented by a second degree polynomial function. As stated in [Men96] surfaces described by a second degree polynomial function are know as a quadric implicit surfaces. Menon also mentions that algebraic surfaces defined by equations of polynomial of order 3 and 4 are also typically used yet at the bulk of interest are mostly quadric surfaces.

In [Har96] Hart also states that the sphere can be represented in a geometric form using a distance metric. The general form of the implicit equation of (2.5) for the sphere then becomes:

$$f(\boldsymbol{x}) = \|\boldsymbol{x}\| - r \tag{2.12}$$

The $\|\boldsymbol{x}\|$ is defined as the Euclidean magnitude and equals to $\sqrt{x^2 + y^2 + z^2}$. It can be said that in this form the sphere is represented by a distance function. As the author states, these kind of functions "measure or bound the geometric distance to their implicit surfaces".

Hart defines the function f in its general form as a continuous mapping $f : \mathbb{R}^n \to \mathbb{R}$ by the use of which one can implicitly describes the set $A \subset \mathbb{R}^n$ as the locus of points fulfilling he condition:

$$A = \{\boldsymbol{x} : f(\boldsymbol{x}) \leq 0\} \tag{2.13}$$

As previously indicated the implicit surface is defined by the value of f being zero and by so denoting the boundary of ∂A, the negative value of f denotes the interior $\mathring{A}$, whereas the positive the exterior $\mathbb{R}^n - A$.

Hart in his paper also provides definitions 2.1 and 2.2 that help in the description of distance functions. These will be repeated here for the reader's convenience and because of their use in the later sections of this thesis.

Definition 2.1 The point-to-set distance defines the distance from a point $\boldsymbol{x} \in \mathbb{R}^3$ to a set $A \subset \mathbb{R}^3$ as the distance from $\boldsymbol{x}$ to the closest point in A,

$$d(\boldsymbol{x}, A) = \min_{y \in A} \|\boldsymbol{x} - \boldsymbol{y}\| \tag{2.14}$$

Definition 2.2 A function $f : \mathbb{R}^3 \to \mathbb{R}$ is a *signed distance bound* of its implicit surface $f^{-1}(0)$ if and only if

$$|f(\boldsymbol{x})| \leq d(\boldsymbol{x}, f^{-1}(0)) \tag{2.15}$$

If equality holds for (2.15), then f is called a *signed distance function*.

As indicated by [Har96], [Men96] and explained in [Gol83] for quadric surfaces the algebraic representation has some limiting disadvantages compared to the geometric representation. Among the main reasons given when considering implementation are the numerical errors resulting from operations on floating point data that have limited accuracy and which are used for describing the surface.

When considering distances to objects or surfaces usually one deals with the notion of a distance field. As indicated in [JBS06] a distance field is a representation of space where each point in that space can be described by a value of a distance from that point to the closest point on a surface belonging to that space. The authors discuss methods for generating distance fields, their applications and representations. One form of representation of a distance field is storing the distance values in a voxel grid. The distance based function described by (2.12) is another form of representation of a distance field which in this particular case contains only the distance information to the surface of a sphere.

2.2 Rendering solutions

As Hart indicates in [Har96] there exist several methods for rendering implicit surfaces. One of them is the polygonization of implicit surfaces which allows to take advantage of the graphical pipeline and rasterization techniques. However, this process of representing the surface by the use of polygons might result in the loss of detail of the surface and may not accurately detect disconnected sections of the surface.

As another method Hart mentions that ray tracing is a noteworthy technique for visualizing implicit surfaces. In ray tracing the ray equation is given by the form:

$$\boldsymbol{r}(t) = \boldsymbol{r}_o + t\boldsymbol{r}_d \tag{2.16}$$

where $\boldsymbol{r}_o$ denotes the point in space from which the ray originates, $\boldsymbol{r}_d$ represents the ray direction and is given by a unit vector, t is the distance traveled by the ray and finally $\boldsymbol{r}(t)$ is the point in space that the ray has reached after traveling the distance t from its origin.

In order to find the intersection point or points of the ray and the implicit surface the following procedure is carried out. For a three dimensional space of real numbers we insert the ray equation defined in the domain $\boldsymbol{r} : \mathbb{R} \to \mathbb{R}^3$ into the function f that describes the implicit surface defined in $f : \mathbb{R}^3 \to \mathbb{R}$ and the acquired result is a composite function $F = f \circ \boldsymbol{r}$ spanning other $F : \mathbb{R} \to \mathbb{R}$. The solution to the equation are all the distances t the given ray has traveled that satisfy the condition:

$$F(t) = f(\boldsymbol{r}(t)) = 0 \tag{2.17}$$

To solve that equation root finding methods are applied and the choice of the method depends on the type of function $F(t)$. As Hart states, for polynomials of degree four or less there exist analytical solutions, yet for an arbitrary function one needs to use a general robust root finder, which often requires more information about the function which can for instance be acquired from its derivative.

The root finding methods have a drawback of finding usually more than one intersection point of the ray and the implicit surface. From all the traversal t values that satisfy the equation (2.17) only the smallest one is taken into consideration. Ray marching and sphere tracing mitigate this by only focusing on determining the first root of the aforementioned equation. The process of doing so is elaborated in the following sections.

2.2.1 Ray marching

In standard constant step ray marching as indicated by[PH89] the ray equation of (2.16) takes the form of:

$$\boldsymbol{r}(k) = \boldsymbol{r}_o + \Delta t \; k \; \boldsymbol{r}_d \tag{2.18}$$

where Δt is the predefined step size and $k \in \{0, 1, 2, ...\}$ is the step number. Sampling the ray at fixed intervals is a basis for some of the solutions for volumetric

Algorithm 2.1 Ray marching

```
1: $k = 0$
2: $d = 0$
3: while $k < k_{max}$ do
4:     $d = f(\boldsymbol{r}(k))$
5:     if $d \leq 0$ then return $k\Delta t$                ▷ intersection
6:     $k = k + 1$
7: return 0                                              ▷ no intersection
```

rendering as Hart suggest in [HST89]. In particular the method is quite similar to the volumetric rendering technique of ray-casting which overview among other can be found in [Elv92].

In [PH89] the ray marching algorithm is used for rendering the *Hypertexture*. The ray marching process starts from the point identified as the first intersection (from the camera's side) of the ray with the parallelepiped bounding the Hypertexture volume. The ray is marched in fixed steps and the ray equation has the form of (2.18). For each such step the representation of the surface is sampled and the opacity values are accumulated. The condition for ceasing the sampling along the ray is when the accumulated opacity value reaches unity, or when the ray leaves the parallelepiped encapsulating the Hypertexture volume. Those are also the same termination conditions for standard ray-casting of volumetric data as described in [Elv92].

The incorporation of this algorithm to rendering of implicit surfaces defined by (2.7) is given in algorithm 2.1 and can be described as follows. The ray is marched in constant steps at which the function $F(k) = f(\boldsymbol{r}(k))$ is evaluated. That function is a modification of the one given by equation (2.17) by incorporating the new ray equation of (2.18). The marching ends when the first in the sequence of return values of the function $F(k)$ is zero or negative which respectively indicates that an intersection has taken place with the surface ∂A or the ray has marched into the interior $\mathring{A}$. For both cases the resulting ray traversal distance $k\Delta t$, being the number of steps times the step size is used to describe/identify the surface point. The ray marching also ends when the maximal predefined number of ray marching steps t_{max} has been reached without the ray intersecting the surface.

It should be noted, however, that the distance that the ray travels between steps ought to be as small as possible in order to better estimate the boundary region of A. If the steps are too large one can traverse too far into the interior $\mathring{A}$ and therefore loose accuracy of the overall surface shape. This is visible for the middle ray in figure 2.1. The surface here is depicted by the blue contour,

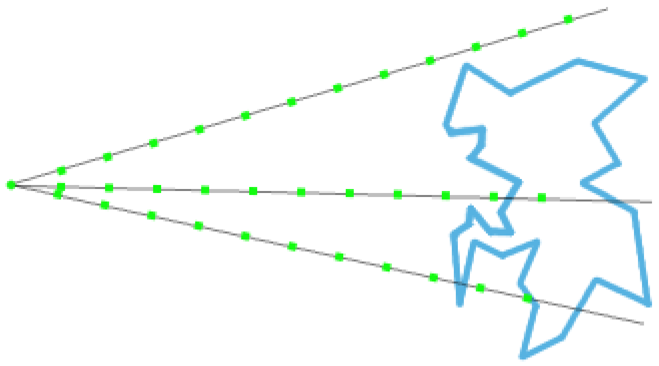


Figure 2.1: Ray marching. Surface represented by the blue contour, green dots are sampling points along the ray whose spacing is determined by the step size.

and steps taken along the ray are visible as green dots. Moreover, it is even possible to miss the surface completely or some parts of it if the step size is not properly determined as it is suggested in [Don05]. This situation is depicted by the bottom ray of the aforementioned figure. As indicated before, because of the step size the first intersection of the ray and the surface it is not detected, but only the second intersection when the sampled point happened to be taken on the surface.

Hart mentions in [Har96] that ray marching suffers from performance issues since it requires fine sampling along the ray, in order to achieve adequate level of detail. In sphere tracing, however, the step size is not constant but it varies in size depending on the distance to the surrounding geometry as the next section will explain.

2.2.2 Sphere tracing

Sphere tracing as Hart defines it in [Har96] is a robust technique for ray tracing implicit surfaces. In this approach we still march along the ray, however, the step size is determined by the distance obtained from a distance function from a given sample point.

Sphere tracing has been first introduced in [HST89] where the authors used this technique to visualize 3-D Deterministic Fractals. In order to help visualize and describe this method the authors form a concept of *unbounding volumes*.

These are defined as volumes that are guaranteed not to contain any part of a surface we intend to render. For a given sampled point in space we are interested in finding an adequate volume representation centered at that sampled point. If one knows the distance from the sampled point to the closest point on the surface, then that distance can be treated as a radius of a sphere. This sphere then forms an unbounding volume which is guaranteed not to intersect with the surface (except for the point of the surface which lies at the length of the radius away from the center of the unbounding sphere) or contain any part of it. The radius of the unbounding sphere is obtained by evaluating the distance function for a given sampled point in space.

It is noteworthy to point out that in the authors' case the distance obtained is called a distance estimate. The choice for the name is given by the fact that the distance cannot be computed efficiently for the deterministic fractal objects they render, therefore they use methods to approximate the actual distance to the surface. However, if we deal with the rendering of an object such as a sphere represented by the implicit equation (2.12) we are able to accurately asses the closest distance to the surface for any point belonging to the domain.

To visualize the surface the authors use the camera model of an *eye*, which is a pinhole camera model with the image plane placed in front of it [PH10]. In [HST89] the ray traversal is described in the following way. The ray is marched from the eye through the image plane to the object. At each given point along the ray the radius of the unbounding sphere is estimated and used as the distance by which the ray will traverse. Several steps are taken since the actual distance towards the surface along the ray may be larger than the evaluated distance for a given point on the ray. This procedure ends when the surfaces is reached. The process of sphere tracing is visible in figure 2.2. Here the two bottom rays intersect the surface. Of note is that, as it can be seen sphere tracing does not suffer from the drawbacks indicated for ray marching in the previous section in the description of the intersections of the surface and the rays.

Moreover, the other condition for termination is when the ray has traversed the maximum predefined distance without having approached the surface [Har96]. This might mean that the ray does not intersect the surface (the top ray of the figure 2.2) or that the surface is farther away than the maximal allowed ray traversal distance. Hart also mentions that the name of *sphere tracing* took its origins "form the property that the ray intersections are determined by sequences of unbounding spheres".

In the process of rendering surfaces this thesis will mostly use the ray equation of (2.16) unless otherwise stated. It is the same one as given in [Har96]. However, in [HST89] it is provided in a different form (stated inductively) and

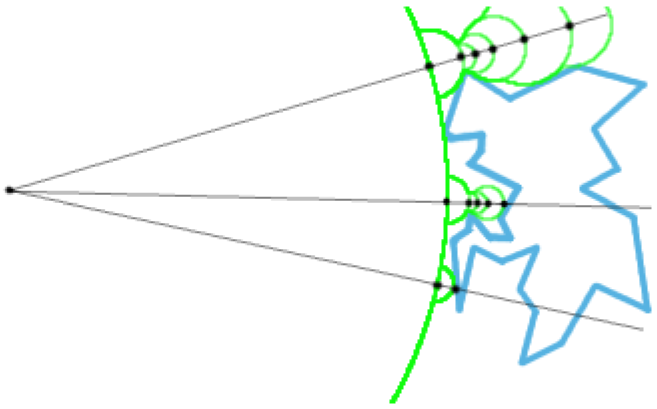


Figure 2.2: Sphere tracing. Green circles represent *unbounding spheres* and black dots their centers

includes a modification whose purpose is to speed up the ray traversal. This will be discussed in another section of this thesis. Nevertheless, the same method of determining the ray direction is used. The ray direction $\boldsymbol{r}_d$ is a unit vector (which is referred also by the authors as a unit slope vector of the ray) and determined from the equation:

$$\boldsymbol{r}_d = \frac{p_{x,y} - \boldsymbol{r}_o}{|p_{x,y} - \boldsymbol{r}_o|} \tag{2.19}$$

Here, as previously $\boldsymbol{r}_o$ denotes the ray origin. The variable $p_{x,y}$ represents a point on the image plane. The notation of ray direction has been changed to make it consistent with the one already used in this thesis.

As Hart explains in [Har96], and as it was indicated in section 2.2 of this thesis the equation $F(t)$ (2.17) is used to find the intersection point. However, now the function $f(\boldsymbol{x})$ is a distance estimating function like the geometric distance function for the sphere(2.12).

Evaluating that equation with respect to the procedure of the ray traversal described in this section allows us to find the first positive root (with respect to the smallest distance traveled by the ray form the eye to the surface) of $F(t)$. Hart defines that root as the "limit point of the sequence defined by the recurrence equation":

$$t_{i+1} = t_i + F(t_i) \tag{2.20}$$

The initial point is being defined here as $t_0 = 0$. The author states that the "sequence converges if and only if the ray intersects the implicit surface". The

Algorithm 2.2 Sphere tracing

```
1: $t = 0$
2: $d = 0$
3: **while** $t < D$ **do**
4:     $d = f(\boldsymbol{r}(t))$
5:     **if** $d < \epsilon$ **then return** $t$          ▷ intersection
6:     $t = t + d$
7: **return** 0                                ▷ no intersection
```

above allows Hart to state that “this sequence forms the kernel of the geometric implicit surface rendering algorithm”. This for the readers convenience is taken from [Har96] and shown as Algorithm 2.2. The t as previously defined, denotes the distance the ray has traversed, f is the distance function of the implicit surface. The variable D is the maximal distance the ray can traverse before the ray marching process terminates. By its use the far plane of the camera is also defined. The convergence test of equation (2.20) is visible in the **if** statement. Here ϵ denotes the desired precision of the convergence test and it is usually a very small number.

The rendered images of this thesis are mostly acquired by using the algorithm of 2.2 unless stated otherwise.

CHAPTER 3

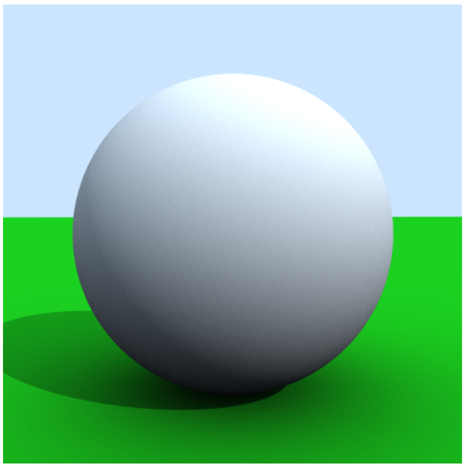
Rendering of implicit surfaces

Though the primary goal is rendering of the images resembling those of Figure 1.1, this chapter sets an intermediate goal of rendering shapes that can be easily defined analytically in terms of distance to the surface. This will allow for an simpler introduction of concepts that will lead to the end result.

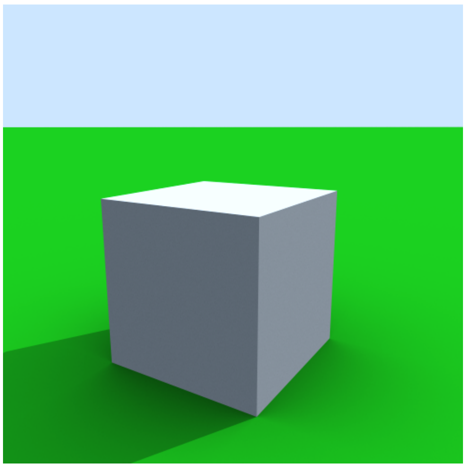
For serving the purpose of visualization of the intermediate goal the reference images of Figure 3.1 will be used. The models used there have been defined with the use of polygon meshes. Their rendering is acquired with the ray tracing framework mentioned in Chapter 1. The next sections will elaborate on how similar results can be achieved with implicit surfaces defined with the use of distance functions.

3.1 Lighting model

In order to visualize the implicit surfaces one needs to decide on a lighting model to use. The model chosen here is based on the Phong lighting model as described in Chapter 5 of [FK03]. The resulting color of the surface point $I(\boldsymbol{x})$ will be acquired from a sum of ambient, diffuse and specular lighting contributions.



(a) Sphere triangle count: 18240. Render time ∼ 20 minutes. View 1



(b) Box triangle count: 12. Render time ∼ 20 minutes.

Figure 3.1: Reference images of simple shapes

The emissive contribution normally being a part of the model is not considered here since we will not be rendering any emissive materials.

$$I(\boldsymbol{x}) = \mathit{ambient} + \mathit{diffuse} + \mathit{specular} \tag{3.1}$$

As a light source a single directional light was chosen and each term of the above equation is defined as follows. The ambient contribution being:

$$\mathit{ambient} = k_a(\boldsymbol{x})I_a \tag{3.2}$$

where $k_a(\boldsymbol{x})$ is the point's ambient reflectance and I_a is the color of the incoming global ambient light. The diffuse contribution can be defined as:

$$\mathit{diffuse} = k_d(\boldsymbol{x})I_d \max(\boldsymbol{n} \cdot \boldsymbol{l}, 0) \tag{3.3}$$

Where k_d is the surface point's diffuse color, I_d stands for the color of the incoming diffuse light. The dot product is between the surface point's unit length normal $\boldsymbol{n}$ and the unit vector $\boldsymbol{l}$ denoting the direction to the light source from the surface point. The specular term is given as follows

$$\mathit{specural} = \mathit{facing}\ k_s(\boldsymbol{x})\ I_s\ \max(\boldsymbol{n} \cdot \boldsymbol{h}, 0)^{\mathit{shininess}} \tag{3.4}$$

Here, k_s denotes the point's specular color, I_s defines the color of the incoming specular light. The $\boldsymbol{h}$ is a unit vector halfway between the vector denoting the direction towards the eye from the surface point and the light vector $\boldsymbol{l}$. The *shininess* exponent determines how much a given surface appears to be shiny. The *facing* parameter is defined as follows:

$$facing = \begin{cases} 1 & \text{if } \boldsymbol{n} \cdot \boldsymbol{l} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

In order to do the lighting calculations the information about the surface normal is required. As given in [HST89] the normal can be computed by using the gradient of the distance field for a given point on the implicit surface. The aforementioned method with the use of the distance function is presented in 3.1: Here $\boldsymbol{n}$ denotes the normal which is given by a three component vector, and as

Algorithm 3.1 Normal calculation for a point on the surface

```
$n_x = f(x + \epsilon, y, z) - f(x - \epsilon, y, z)$
$n_y = f(x, y + \epsilon, z) - f(x, y - \epsilon, z)$
$n_z = f(x, y, z + \epsilon) - f(x, y, z - \epsilon)$
```

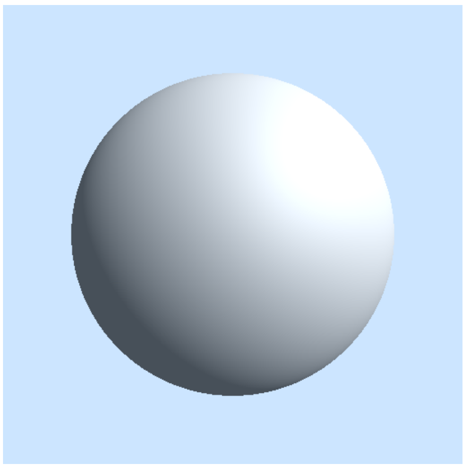
previously f is the distance function. The ϵ stands for a value by the use of which for a given surface point the neighbouring points along the coordinate axes are accessed. The correctness of the normal calculation will be affected based on the magnitude of ϵ. Because of that it is usually chosen to be a small number.

After computing the normal it should be normalized. The authors also mention that the 6-point gradient might be extended to versions where 18 or 26 samples are taken. This is usually done when we are of need of a better estimate of the normal and the 6-sample point method does not produce the expected result. Such a scenario might arise when the representation of a distance field is a "loosely" spaced voxel grid, from which we take our distance values. Therefore, taking more samples should approximate the surface point normal better.

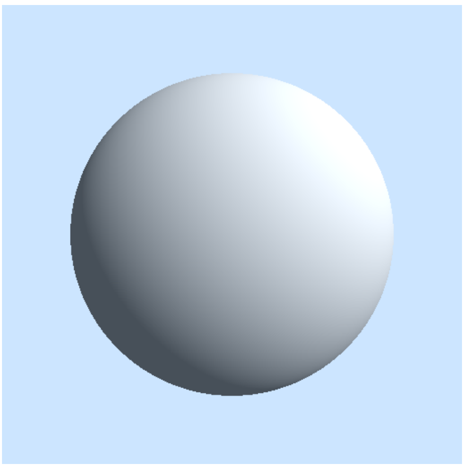
3.2 Initial rendering

Since the shape of the sphere is considered well known and whose mathematical description was used in the Chapter 2 of this thesis it was chosen as a candidate for an initial rendering of an implicit surface.

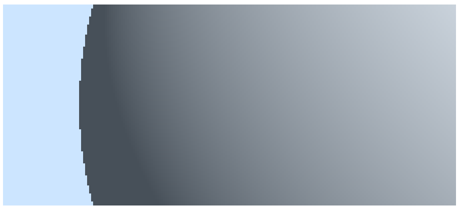
The results of rendering a sphere described by equation (2.12) and using the implicit surface rendering Algorithm 2.2 with the previously discussed lighting model are visible in Figure 3.2. The color of the sphere determined from the diffuse, ambient and specular contributions is visible in 3.2a, whereas in 3.2b



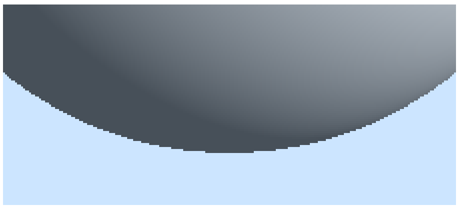
(a) Diffuse, ambient and specular term



(b) Diffuse and ambient term



(c) Aliasing: sphere's image left part



(d) Aliasing: sphere's image bottom part

Figure 3.2: Sphere

its color is determined only by the diffuse and ambient contribution. Unless otherwise stated, the latter will be the default contributing terms for rendered surfaces throughout this thesis since the reference images of Figure 1.1 and Figure 3.1 do not have a specular contributing component.

Like ray tracing, sphere tracing is prone to aliasing. This can be clearly seen in the bottom and left part of the rendered sphere's image. The unscaled aforementioned parts are given by Figures 3.2c and 3.2d respectively.

3.3 Basic operations on implicit surfaces

Hart in [Har96] and Quilez in [nQ08a] provide signed distance functions for common geometric objects like the sphere, plane, cone and other along with the operations that can be performed on them. Hart provides these in a form of mathematical notation whereas Quilez in already coded form. Since this thesis is greatly influenced by Hart's paper the mathematical notation will be also used here.

As Hart defines it, implicit surfaces can be transformed in a procedure involving two steps. The first one is the application of the inverse transformation to the space the implicit surface is said to reside in (that is the domain of the implicit function). Then the points in that transformed space are queried with the distance function of the implicit surface. The transformed implicit surface can be written as:

$$f(\boldsymbol{T}^{-1}(\boldsymbol{x})) = 0 \tag{3.6}$$

where $f(\boldsymbol{x})$ is the distance function of the implicit surface and $\boldsymbol{T}^{-1}(\boldsymbol{x})$ is defined as the inverse of the transformation $\boldsymbol{T}(\boldsymbol{x})$ that we want to apply to the implicit surface.

Not all transformations preserve the distance that would be returned from the distance function of the transformed surface. That is, it may happen that the distance returned is not an actual distance from a given point in space to the closest point on any surface. However, a group of transformations called isometries after their application provide the correct distance to the transform surface. In this group are transformation such as translations, rotations and reflections. After applying these transformations the distance value returned form the distance function need not be altered to compensate the effects of the transformation. This as Hart defines it can be written as:

$$d(\boldsymbol{x}, \boldsymbol{I} \circ f^{-1}(0)) = d(\boldsymbol{I}^{-1}(\boldsymbol{x}), f^{-1}(0)) \tag{3.7}$$

where the $\boldsymbol{I}$ stands for an isometry transformation, and $f^{-1}(0)$ for a given implicit surface.

Scaling an implicit surface on the other hand does not preserve the distance and the result from a distance function of the scaled surface needs to be adjusted. Hart gives, the transformation of a uniform scale as

$$\boldsymbol{S}(\boldsymbol{x}) = s\boldsymbol{x} \tag{3.8}$$

where s is the scale factor. Its inverse is given by:

$$\boldsymbol{S}^{-1}(\boldsymbol{x}) = \frac{1}{s}\boldsymbol{x} \tag{3.9}$$

which allows for defining the distance to the scaled implicit surface as:

$$d(\boldsymbol{x}, \boldsymbol{S}(f^{-1}(0))) = sd(\boldsymbol{S}^{-1}(\boldsymbol{x}), f^{-1}(0)) \tag{3.10}$$

The distance returned form the distance function of the scaled implicit surface is multiplied by the scale factor s in order to preserve the accurate distance information. For more examples and information about the transformations that can be carried out on implicit surfaces defined by the distance metric the reader is referred to the aforementioned references.

Implicit surfaces can also be created for example with the use of operations like union, subtraction and intersection on other implicit surfaces for which their signed distance functions are known.

The union of two implicit surfaces A and B is defined by Hart as the minimum distance from either of their distance functions f_A and f_B respectively, and it is written as:

$$d(\boldsymbol{x}, A \cup B) = min(f_A(\boldsymbol{x}), f_B(\boldsymbol{x})) \tag{3.11}$$

As previously the $\boldsymbol{x}$ stands for the sampled point in space. On the other hand, the distance to the intersection of two implicit surfaces the author states as:

$$d(\boldsymbol{x}, A \cap B) \geq max(f_A(\boldsymbol{x}), f_B(\boldsymbol{x})) \tag{3.12}$$

The subtraction operation uses the description of the signed distance function to the complement of an implicit surface. The distance to the complement of an implicit surface A is given by Hart as:

$$d(\boldsymbol{x}, \mathbb{R}^3 \setminus A) = -f_A(\boldsymbol{x}) \tag{3.13}$$

The subtraction of the implicit surface B from A based on the one given by Quilez can be written as:

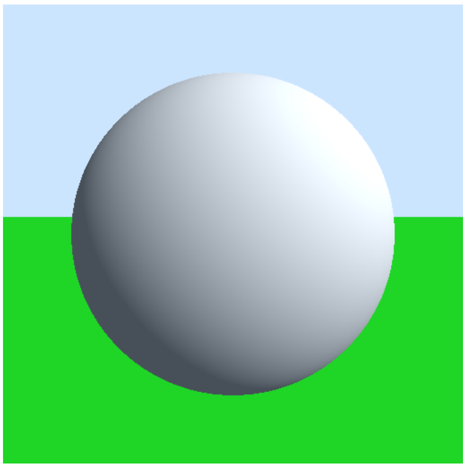
$$d(\boldsymbol{x}, A \cap B) \geq max(f_A(\boldsymbol{x}), -f_B(\boldsymbol{x})) \tag{3.14}$$

As it is stated by Hart the distance to a list of implicit surfaces is the smallest distance originating from their respective distance functions. Therefore, in addition to combining surfaces the union operation allows for inclusion of more implicit surfaces which do not have to be in contact with each other in the scene.

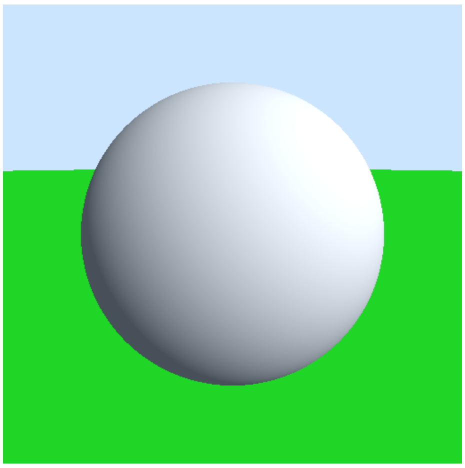
For instance the inclusion of flat ground to the image 3.2b is achieved by first defining the distance function for it. It can be represented by a plane P which distance function is given by Hart as (3.15).

$$f(\boldsymbol{x}) = d(\boldsymbol{x}, P) = \boldsymbol{x} \cdot \boldsymbol{n} - h \tag{3.15}$$

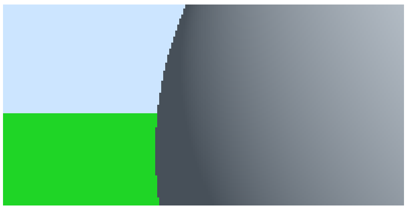
It is defined with the use of the unit normal $\boldsymbol{n}$ with the point of intersection $h\boldsymbol{n}$. The h defines the relative placement of the plane with respect to the plane perpendicular to the normal n and intersecting the origin of the coordinate



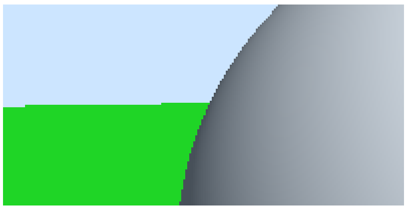
(a) Initial camera position



(b) Another camera position



(c) Left part of figure 3.3a



(d) Left part of figure 3.3b

Figure 3.3: Sphere and ground

system. The operation $\boldsymbol{x} \cdot \boldsymbol{n}$ stands for the dot product of the sampled point and the unit normal of the plane P. Then a scene consisting of a sphere and the plane has its distance function used in sphere tracing Algorithm 2.2 given as:

$$f(\boldsymbol{r}(t)) = min(f_G(\boldsymbol{r}(t)),\ f_S(\boldsymbol{r}(t))) \tag{3.16}$$

Where $f_G(\boldsymbol{r}(t))$ stands for the singed distance function of the plane given by equation (3.15) and $f_G(\boldsymbol{r}(t))$ stands for the signed distance function of the sphere given by (2.12).

The result of placing the plane beneath the sphere is depicted in Figure 3.3a. The relative placement is achieved by adjusting the h variable of equation (3.15).

In the image of 3.3a the ground does not suffer from any aliasing as it is more clearly visible in the unscaled left part of it as presented in 3.3c. This is because of a quite large maximal ray traversal distance set for producing this figure, as well as the parallel placement of the camera with respect to the ground plane.

Algorithm 3.2 Hard shadows

```
1: $t = t_{min}$
2: $d = 0$
3: while $t < D$ do
4:     $d = f(\boldsymbol{r}_S(t))$
5:     if $d < \epsilon$ then return 0                ▷ in shadow
6:     $t = t + d$
7: return 1                                           ▷ not in shadow
```

On the other hand when the same maximal ray traversal distance has been set and the camera position has changed the aliasing of the ground is now apparent in as seen in Figure 3.3d which is the unscaled left part of Figure 3.3b. The images in this section seem to lack depth this can be mitigated by adding shadows as described in the next section.

3.4 Shadows

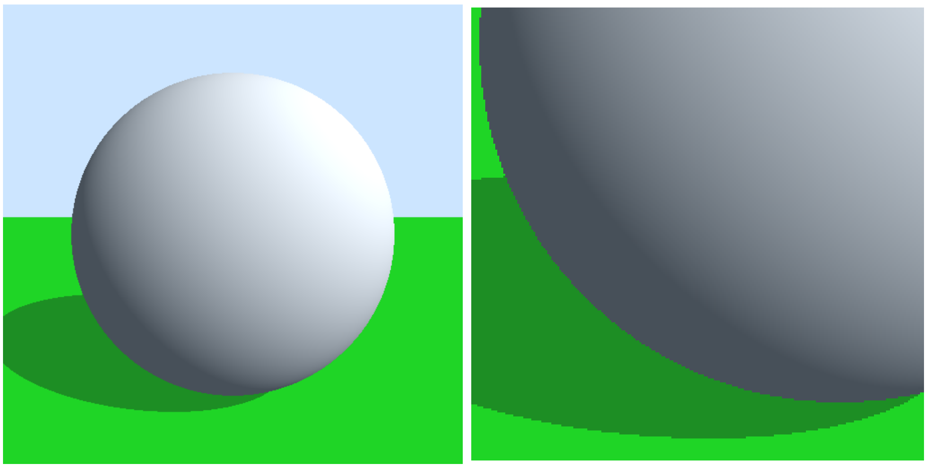
Hard shadows are easily obtained much like in ray tracing. The procedure involves sphere tracing from a surface point towards the direction of the light source until a surface intersection is determined or the maximal ray traversal distance is surpassed. The distance to the light source for a given surface point is usually that maximal ray traversal distance or it might be a minimal arbitrary value that is adequate for the scene geometry like for instance in the case of a directional light source.

The traversal here will be done on the shadow ray which is described by equation (3.17) that differs slightly from the standard ray equation of (2.16) by using a different starting point and direction of traversal.

$$\boldsymbol{r}_S(t) = \boldsymbol{x} + t\ \boldsymbol{r}_{Ld} \tag{3.17}$$

Here, the origin of the ray is now the surface point $\boldsymbol{x}$ which has been previously determined by sphere tracing Algorithm 2.2. The direction $\boldsymbol{r}_{Ld}$ of the shadow ray is a unit vector having the direction from the surface point $\boldsymbol{x}$ towards the light source.

In [nQ11] Quilez provides method for obtaining hard shadows, which is presented as Algorithm 3.2. The reader might notice that this algorithm looks very similar to the standard sphere tracing Algorithm 2.2. There are however, some differences. For instance the ray is defined now as the shadow ray $\boldsymbol{r}_S(t)$ of equation (3.17) and that the distance function f is the distance function



(a) Whole image **(b)** Hard shadow aliasing

Figure 3.4: Sphere and ground scene, hard shadows

of the scene itself. Moreover, the algorithm's return value denotes something else. Here the value 1 indicates that the shadow ray $r_S(t)$ did not intersect any surface when the maximal ray traversal distance D has been reached. On the other hand, the value 0 means that the ray has intersected with a surface and therefore the surface point $\boldsymbol{x}$ is in shadow.

There is one additional change that might not be noticeable at first. The variable t is initialized to a t_{min} value (determined per rendered scene) which should be defined with the relation to ϵ as $t_{min} \gg \epsilon$. If it were otherwise, the considered point $\boldsymbol{x}$ would be deemed in shadow, since of the initial value of d fulfilling the **if** branch condition of $d < \epsilon$. The reader might remember that the **if** condition was responsible for the determination if the sampled point along the ray was of the surface in the sphere tracing Algorithm 2.2. In the case of the hard shadows Algorithm 3.2 when the initial traversal t would be zero then the sampled point returned by the shadow ray equation would be the surface point $\boldsymbol{x}$ resulting in a distance d of 0 and triggering the **if** branch. Therefore, setting the initial shadow ray traversal to a value t_{min} allows the point $\boldsymbol{x}$ not to "shadow itself".

Applying hard shadows to the scene containing the sphere and the ground results in an image visible in Figure 3.4a.

As it can be seen in the unscaled Figure 3.4b the hard shadows also suffer from aliasing.

Algorithm 3.3 Soft shadows

```
1: $t = t_{min}$
2: $d = 0$
3: $shadow = 1$
4: while $t < D$ do
5:     $d = f(\boldsymbol{r}_S(t))$
6:     if $d < \epsilon$ then return 0                      ▷ in shadow
7:     $shadow = min(shadow, scale * d/t)$
8:     $t = t + d$
9: return $shadow$                                         ▷ not in shadow
```

Additionally, Quilez proposes a technique for obtaining soft shadows with a penumbra region. He argues that since distance fields provide global information of the surrounding scene geometry by inspecting a point in space with a distance function they become quite useful at obtaining some of the more realistic illumination and shading techniques. The method described by Quilez for obtaining soft shadows with a penumbra region can be written as shown in Algorithm 3.3.

Comparing the soft shadow Algorithm 3.3 with the one for producing hard shadows 3.2 shows that the former is changed by an addition of line 7. This allows for inspecting the closeness of the ray to the surrounding geometry of the scene given by the distance d, along with the traversal t of the ray at the time of inspection during the ray marching process. As Quilez points out these two variables help us in assessing and forming the penumbra region along with the soft shadows. The reasoning for their use in the form of equation of line 7 is the following.

When the ray was close to intersecting a surface but failed to do so the minimum distance d recorded along the traversal of the ray will be quite small. In this case and depending on its value one can say that the point from which the ray originated from is in the penumbra region and adjust its shading accordingly. That is, the closest the ray was to intersecting an surface the darker the point should be.

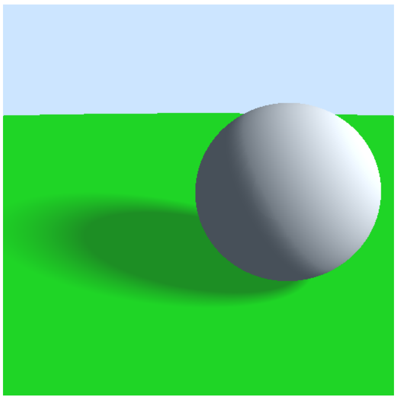
Additionally, what influences the shading is taking into account the distance the ray has traversed when testing the distance to a surface. For instance, for two rays originating from two different surface points and having the same minimum recorded distance to a surface, the one having traversed less when the minimum distance has been recorded should be made darker.

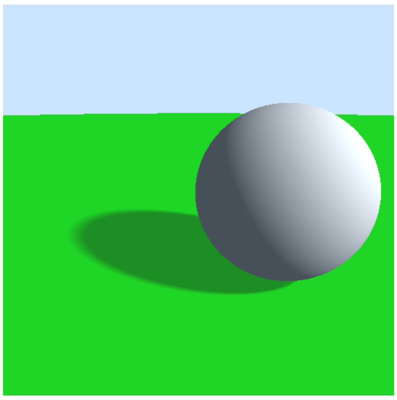
Keeping track of the minimum *penumbra factor* of d/t (as Quilez calls it) allows for assessing the shading of a point with respect to the above considerations.

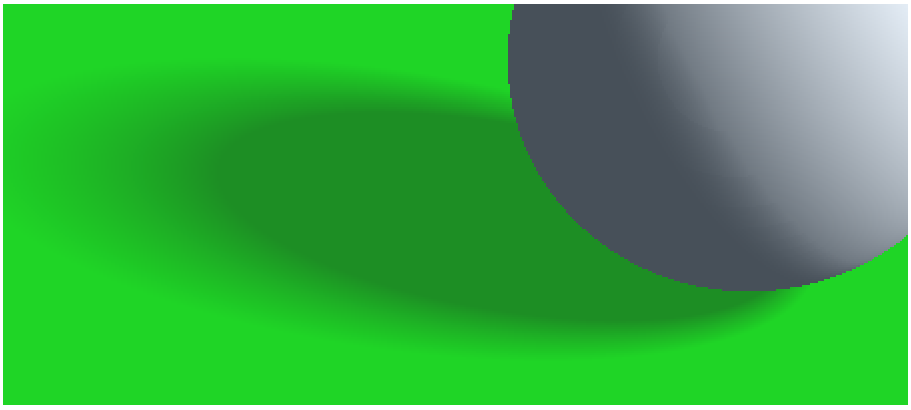
As Quilez mentions the *scale* variable affects the visualization of the penumbra region making the shadows more hard or soft. The application of the soft shadow Algorithm 3.3 results in the images of Figures 3.5 and 3.6. The camera position was changed to better illustrate the results. These also show the effects of setting the penumbra factor's *scale* to different values.

By comparing Figures 3.5c and 3.5d it is also visible that the penumbra scale factor also has an effect on the sphere's shading. This is mostly visible in the transitional region from the sphere's surface being illuminated to going into self shadow.

What is noticeable is that the shadows do not have the aliasing artifacts as it was in the case with hard shadows of Figure 3.4b. Moreover, one can try to set the *scale* parameter to such a value that will mimic the effects of anti-aliasing for hard shadows as Figure 3.6 illustrates.


(a) Penumbra factor scale 4


(b) Penumbra factor scale 16


(c) Shadow region of figure 3.5a

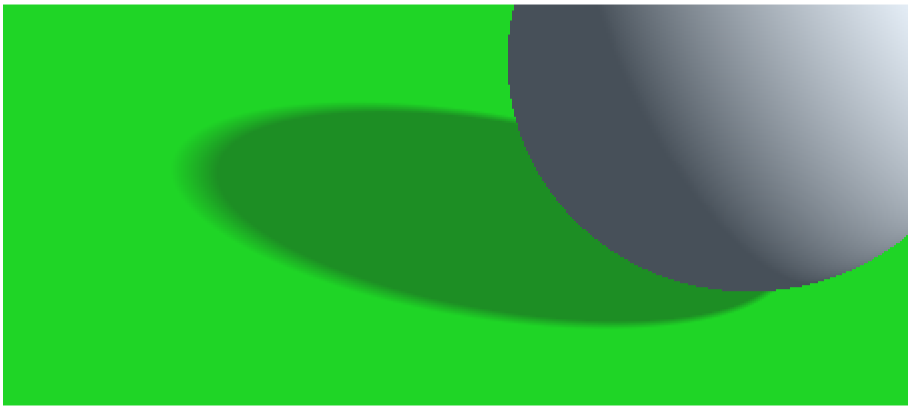
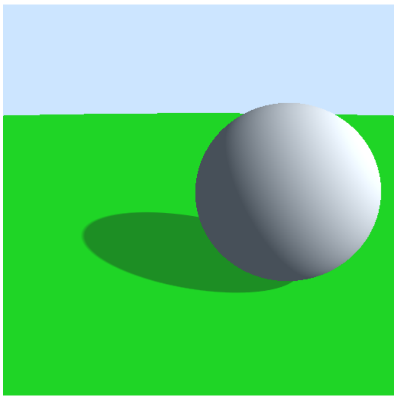
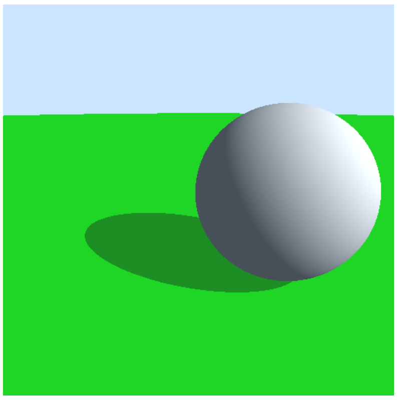
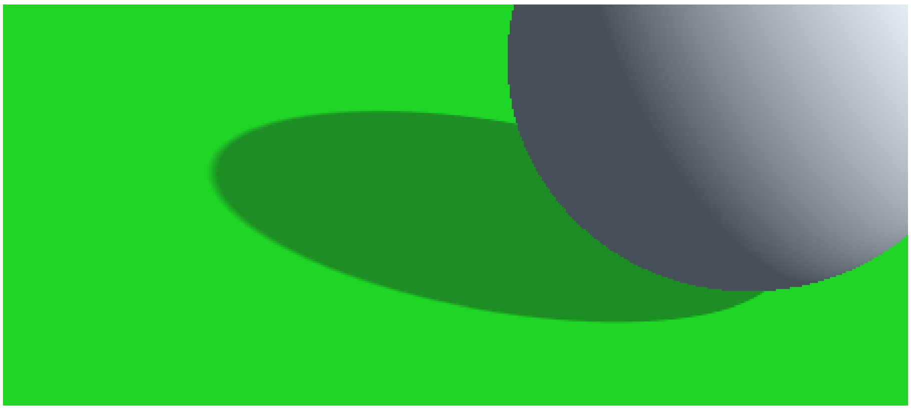

(d) Shadow region of figure 3.5b

Figure 3.5: Sphere and ground scene, soft shadows, penumbra scale 4 and 16


(a) Penumbra factor scale 64


(b) Hard shadows


(c) Shadow region of figure 3.6a

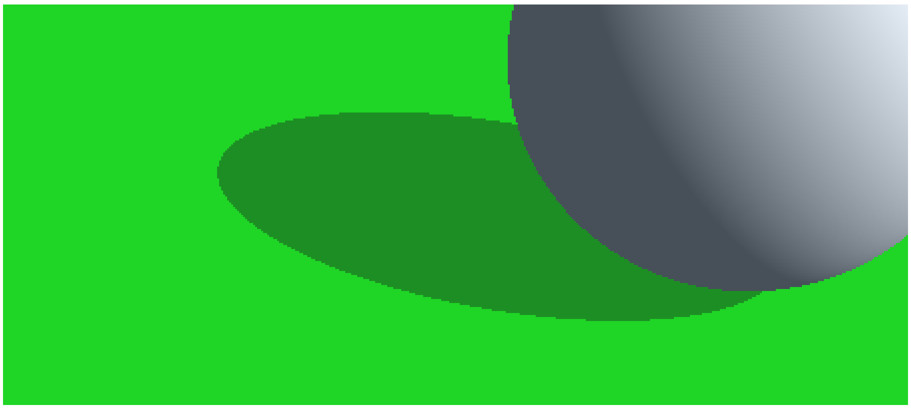

(d) Shadow region of figure 3.6b

Figure 3.6: Sphere and ground scene, soft and hard shadows

3.5 Ambient occlusion

3.5.1 Background

Many may state that to produce convincing photorealistic images one has to also take into account the global illumination effects that are the result of diffuse interreflections. Such view among other, is taken into consideration in [IKSZ03]. In order to achieve such images the authors state that one usually uses methods like radiosity or ray tracing. However, adequate results are obtained after a lengthy time period and the processes themselves do require significant amounts of memory. Moreover, as the authors mention setting up such methods and their use is not considered to be trivial, therefore they are not used by artists very often. In movies or games for that matter more stress is put on the general vision of the director of how the scene should look like than its realism and if it corresponds to the general vision of the artistic style of the given work. Therefore, it is more often vital to produce images that are rendered quickly, which may not be physically correct, but are aesthetically pleasing.

In [ZIK98] a method aiming to achieve illumination effects of occlusion similar or correct up to a certain degree to those acquired from radiosity but in a shorter time period is presented. There the authors Zhukov et al. devised the concept of obtaining these effects by computing the *obscurance* of a given point in the scene. As it is explained the property tells how much a given surface point is open and by its use the authors introduce their “empirical ambient light illumination model”. It accounts for the lack of secondary light ray reflections that would be contributing to a point’s illumination if it were not for occluding surfaces and the degree of that occlusion is the focus of the measurement. As the authors point out this is opposite to radiosity where the contribution to the illumination is the measured intensity from secondary reflections.

As it is stated in the paper most local illumination models use ambient light to account for secondary diffuse reflections and treat it as a value that is constant for the whole scene. This is also true for the Phong lighting model used so far in this thesis for rendering implicit surfaces and as it can be seen these lack a certain degree of realism. With the use of ambient light Zhukov et al. in the paper simulate indirect illumination by combining it with a model that reproduces the darkening effects of obscured areas by taking into account the "geometric obscurance in these areas".

The paper by Zhukov et al. has been a basis for many others like [IKSZ03] and [KL05]. In the latter Kontkanen et al. mentions that the ambient occlusion idea is considered to be a special case, a simplification of the model proposed

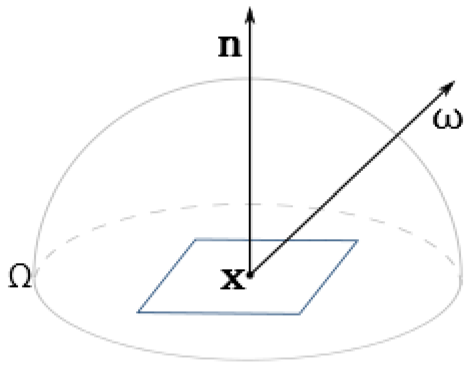


Figure 3.7: Hemisphere sampling for Ambient Occlusion

by Zhukov et al. and that this technique has become quite popular in the movie industry under the influence of which it has undergone some refinement. Since of it being the simpler model it will be explained first. As the authors define it, ambient occlusion is linked to the attenuation of ambient light which is the result of the occlusion of nearby geometry and can be given by the following formula:

$$O_A(\boldsymbol{x}, \boldsymbol{n}) = \frac{1}{\pi} \int_{\Omega} V(\boldsymbol{x}, \boldsymbol{\omega}) \lfloor \boldsymbol{\omega} \cdot \boldsymbol{n} \rfloor d\boldsymbol{\omega} \tag{3.18}$$

As before $\boldsymbol{x}$ defines the surface point and $\boldsymbol{n}$ its normal as ilustrated in Figure 3.7. The function $V(\boldsymbol{x}, \boldsymbol{\omega})$ determines the visibility in the direction defined by the direction vector $\boldsymbol{\omega}$. The visibility function is said to be equal to unity if in the direction $\boldsymbol{\omega}$ a surface is found (ergo an occluder) and zero otherwise. The integration takes over a hemisphere of directions Ω which orientation is with accordance to the normal $\boldsymbol{n}$ and centered at the point $\boldsymbol{x}$. The authors unfortunately do not provide explicitly the explanation why in this case the floor operator is used in the expression: $\lfloor \boldsymbol{\omega} \cdot \boldsymbol{n} \rfloor$. Yet after consultation in [PH10] it is assumed that the authors might have meant the absolute value of the dot product between vectors $\boldsymbol{\omega}$ and $\boldsymbol{n}$. This will be left here as it is in its current form for consistency reasons.

$O_A(\boldsymbol{x}, \boldsymbol{n})$ will take on a value of between 1 and 0. The full occlusion of a point is repressed by value 1, whereas the value 0 denotes no occluding geometry influencing that point. As the authors state multiplying the constant ambient term of (3.2) in the Phong's model by $1 - O_A(\boldsymbol{x}, \boldsymbol{n})$ gives considerably better results and allows for an approximation of some of the effects received from computing the full global illumination. There the computation of ambient occlusion in production rendering as the authors indicate is usually done on the surfaces of each object and the results are stored as vertex attributes or texture maps for later use.

The authors also relate equation (3.18) to the Zhukov et al. obscurance illumination model, in which the obscurance W can be given as:

$$W(\boldsymbol{x}, \boldsymbol{n}) = \frac{1}{\pi} \int_{\Omega} \rho(d(\boldsymbol{x}, \omega)) \lfloor \omega \cdot \boldsymbol{n} \rfloor d\omega \tag{3.19}$$

Kontkanen et al. point out that as it can be seen from the above the place of the visibility function was taken by $\rho(d(\boldsymbol{x}, \boldsymbol{\omega}))$ which can be understood as follows. The purpose of the distance function d is to provide the distance of the first intersection of the ray shot from the surface point $\boldsymbol{x}$ in the direction $\boldsymbol{\omega}$ with an occluding surface. The function ρ on the other hand as the authors define it is a mapping of a distance suitability.

That distance suitability is elaborated more by Zhukov et al. in [ZIK98] and by Iones et al. in [IKSZ03]. In the latter the authors there aim to achieve an distance ambient effect based on the values of the function ρ being in the range between and including 0 and 1. This will measure the magnitude of ambient light coming from the direction ω. Moreover, the properties that the function should fulfil have been defined by the authors as follows:

$$\rho(d) = \begin{cases} 0 & \text{for } d = 0 \\ 1 & \text{for } d = +\infty \end{cases} \tag{3.20}$$

$$\rho'(d) = \begin{cases} > 0 & \\ \quad 0 & \text{for } d = +\infty \end{cases} \tag{3.21}$$

$$\rho''(d) < 0 \tag{3.22}$$

The above, as the authors state, describes a function that is monotonous and increasingly smooth. The function produces the value 0 for the intersection distance being 0 which means that the given surface point is fully occluded. The highest value of the function is bounded by the asymptotical value 1 and the function is characterised by a monotonous decreasing derivative. To summarise, the above reflects a function that describes the increase of ambient light contribution the further the intersection point is with the maximal value 1 if no such point is detected. Iones et al. suggest that for the ρ function a family of functions given by the equation:

$$\rho(d) = 1 - e^{-\tau d} \tag{3.23}$$

would be fitting. The τ is defined as a positive parameter.

The obscurance value W given by equation (3.19) for any surface point $\boldsymbol{x}$ is in the range $0 \leq W \leq 1$ and depicts the local geometric properties of the surroundings of the surface point. The value 0 denotes that the point is fully occluded, whereas the value 1 means that there in no occlusion by surrounding geometry, and as the authors state the point is "fully open". Iones et al. assume that the openness of a point measures its brightness, that is, it is a measure of the intensity of the ambient light reaching that point, and by doing that form a reflected intensity equation for a surface point:

$$I(\boldsymbol{x}) = I_a k_a(\boldsymbol{x}) W(\boldsymbol{x}) \tag{3.24}$$

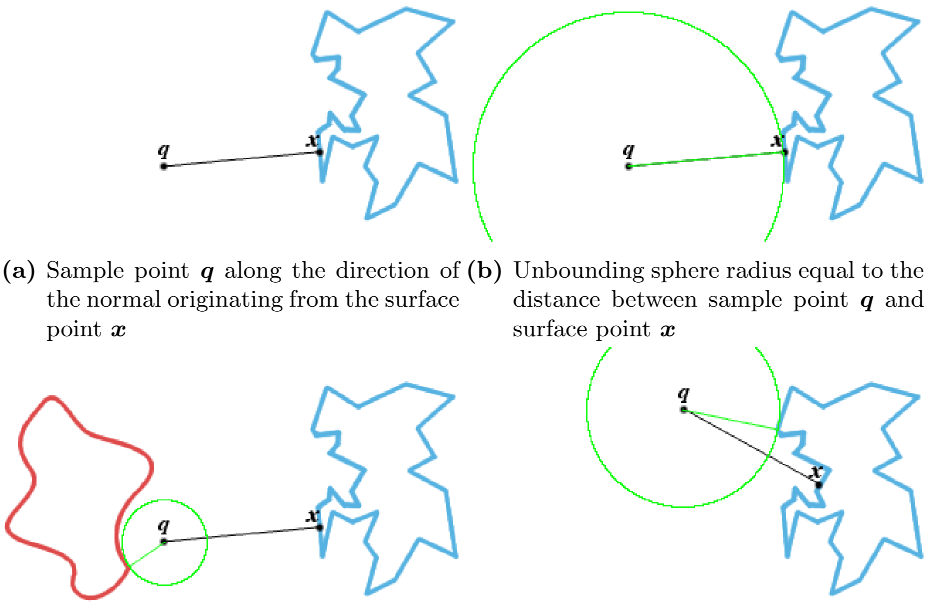
The $k_a(\boldsymbol{x})$ is the diffuse reflectance for ambient light. The above equation does not take into account any light sources just the ambient contribution part. If the surface point is not occluded the equation then becomes $I(\boldsymbol{x}) = I_a k_a(\boldsymbol{x})$ which is the same as the ambient contributing part for the Phong lighting model and same as the ambient contribution of equation 3.2 used in the lighting model of this thesis. The authors mention that obscurance can be computed for static scenes without the dependence on light sources and stored in obscurance maps for future use.

Since the obscurance concept relies on the geometric properties of the scene the idea of it has been further revised by Evans in [Eva06a] where the author uses it with the conjunction of distance fields. Inquiring the signed distance function about a point surroundings can lead to a general idea about its surroundings which are of use in estimating the amount of ambient light reaching that surface point.

Evans in his paper is interested in finding the amount of sky visibility for any point in the scene that would undergo lighting computations and based on that the shading of the point itself. He defines features that his algorithm for approximating global illumination should have. The key ones are the following. There exist “dark contact shadows” of objects which are in contact with each other, thus an object can be an occluder and/or an occludee. Another thing is to be able to register darker regions in creases and valleys of complex surfaces. Moreover, what is of contrast to other methods is that the algorithm will support a scene and the multiple objects in it that do not have to be static and can move freely or even deform.

As Evans explains the distance values returned from a signed distance function allow us to asses how much an object/surface is occluded, or more precisely as the author defines it: to measure the visibility of the sky from any point in the scene considered for shading. This, relates to the obscurance notion that was mentioned in previous paragraphs, and its influence in devising this method.

Before Evans method is introduced it is noteworthy to mention about the convention he uses for representing distance fields. The author uses the opposite



(a) Sample point $\boldsymbol{q}$ along the direction of the normal originating from the surface point $\boldsymbol{x}$

(b) Unbounding sphere radius equal to the distance between sample point $\boldsymbol{q}$ and surface point $\boldsymbol{x}$

(c) Unbouding sphere radius influenced by the red surface

(d) Unbounding sphere radius influenced by the geometry of the blue surface

Figure 3.8: Signed distance function as an occlusion measure

of those for surface interior and exterior that were defined by equations (2.6) and (2.8). That is, Evans considers the return value for the distance function inside the surface as a positive value, whereas the return value for the distance function to be negative for the space enclosing the surface.

However, in order to be consistent with the representation already used here in this thesis, Evans reasoning concerning values of distance fields and their use in equations will be adjusted accordingly. Moreover, the terminology introduced so far in this thesis will also be incorporated into the description of Evans method.

The explanation for using distance fields as a source of information about the amount of ambient light reaching a surface point is justified as follows. Consider sampling the distance field in the direction of the normal of a surface point $\boldsymbol{x}$. The sampling distance is defined beforehand and is given to be d_q. The sample is taken at point $\boldsymbol{q}$ that is of distance d_q away from the surface point $\boldsymbol{x}$ as it is illustrated in Figure 3.8a. If the distance function is queried for the point $\boldsymbol{q}$ it returns an distance which is the unbounding sphere radius and defined as $r_{\boldsymbol{q}}$, which is visible in Figure 3.8b. If it happens that the surface point $\boldsymbol{x}$ is not occluded by either other surfaces or the surface itself as it is in this case then there exist an equality $d_q = r_q$.

On the other hand if in the vicinity of sampled point q there exists another surface point which is closer than the point $\boldsymbol{x}$ the situation is different. In the case of Figure 3.8c that closer point is of another surface, whereas in Figure 3.8d it is an another point on the same surface. Then the following inequality holds for both cases $d_q > r_q$. In his paper Evans suggests that the difference of values of d_q and r_q can be used in the measurement of the occlusion of surface point $\boldsymbol{x}$.

In order to better estimate the amount of occlusion usually several samples are taken. As Evans explains it in the absence of occluders one expects the following equation to hold true:

$$\sum_{i=0}^{i_{max}} f(\boldsymbol{x} + d_i\boldsymbol{n}) = \sum_{i=0}^{i_{max}} d_i \tag{3.25}$$

Where f is the singed distance function describing the scene, and $\boldsymbol{n}$ is the surface normal originating from point $\boldsymbol{x}$. The maximal number of sampled points in the direction of $\boldsymbol{n}$ away from $\boldsymbol{x}$ is given by i_{max}. The distance relations of the samples are such that: $d_i < d_{i+1}$ with $d_0 = 0$.

As the author states, any difference between the two sides of equation (3.25) can be used as an estimate of a point's occlusion or the openness of the space around it and the author chooses to measure that by the use of the following equation:

$$C = e^{\tau \sum_{i=0}^{i_{max}} (d_i - f(\boldsymbol{x} + d_i\boldsymbol{n}))} \tag{3.26}$$

The τ variable is set to define the overall contrast. Evans elaborates that each sample at a distance d_i is used to asses the occlusion effects of objects at that distance. The C stands for the approximation of ambient occlusion and is the output of the shader as the author defines it.

In Evans approach the distance flied is computed for the entire scene and stored in a volume texture and then the information stored there is used for assessing the global illumination. The scene itself can be constructed out of objects represented as polygons. Because of this the author states that his algorithm is rather limited to small scenes. Moreover, Evans sums up in his presentation [Eva06b] that the unphysical nature of these lighting effects by using approximations of the occluding geometry is somewhat difficult to apply in complex scenes. The author also mentions that though these might not reflect how these effects would look like in real life they give the viewers adequate approximations in helping them visualize and understand the scene.

In this thesis the current approach was to represent surfaces by their distance functions and not to compute the distance field based on the scene geometry and store it in a 3D texture. Therefore, no mater how large the procedural scene will be the technique for using the distance from the distance functions can still be used for obtaining the effects of ambient occlusion on an arbitrary procedural scene. Moreover, some of the techniques that Evans describes later in his paper do not apply here. For example, the determination of the distance of sample points, since it is influenced by the dimensions of the texture volume representation that stores the signed distance function (distance field).

On the other hand Quilez in his presentation [nQ08b] modifies slightly Evans approach. The sampling method described previously can be written in another form using the ray equation. A ray cast from the surface point $\boldsymbol{x}$ in the direction of the normal $\boldsymbol{n}$ at that surface point, is defined as:

$$\boldsymbol{r}_n(k) = \boldsymbol{x} + \Delta t\ k\ \boldsymbol{n} \tag{3.27}$$

The ray equation is similar to the one used for constant step marching of (2.18). The only difference except for notation, is the ray's direction. The variable $k \in \{0, 1, 2, ...\}$ denotes the sample number and Δt is the step size of the samples. Then in the case of no occlusion as in Figure 3.8b the following holds:

$$f(\boldsymbol{r}_n(k)) = \Delta t\ k \tag{3.28}$$

The above, for that case can be generalized for a number of samples (as it was respectively in the equation (3.25)):

$$\sum_{k=1}^{k_{max}} f(\boldsymbol{r}_n(k)) = \sum_{k=1}^{k_{max}} \Delta t\ k \tag{3.29}$$

The zero has been omitted from the k domain since the result there will always be the same. Samples are taken a certain amount of times defined by k_{max}. In the case of scenarios depicted by Figures 3.8c and 3.8d the above equation will turn into an inequality.

$$\sum_{k=1}^{k_{max}} f(\boldsymbol{r}_n(k)) < \sum_{k=1}^{k_{max}} \Delta t\ k \tag{3.30}$$

For obtaining the ambient occlusion Quilez defines the equation:

$$A_O = 1 - s \sum_{k=1}^{k_{max}} \frac{1}{2^k}(k\ \Delta t - f(r_n(k))) \tag{3.31}$$

Algorithm 3.4 Ambient Occlusion

```
1: $k = 1$
2: $d = 0$
3: $occlusion = 0$
4: while $k < k_{max}$ do
5:     $d = f(\boldsymbol{r}_n(k))$
6:     $occlusion = \frac{1}{2^k}(k\Delta t - d)$
7:     $k = k + 1$
8: return $1 - \text{clamp}(s\ occlusion, 0, 1)$
```

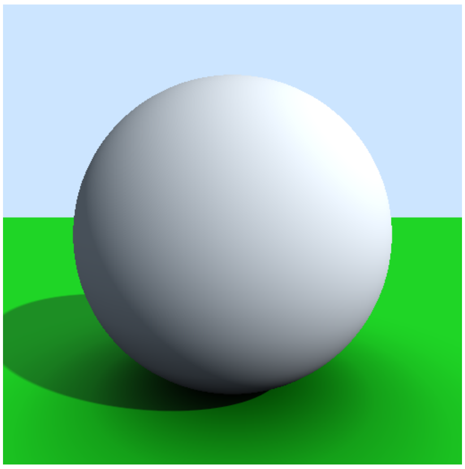
Where s is a scale variable determining the darkening effects of the occlusion. For A_0 value range is between 0 and 1 and similarly to obscurance W of equation (3.19) 0 stands for full occlusion of a surface point, whereas 1 means no occlusion. The A_O behaves similarly to the function ρ of equation (3.23) in the sense that, as Quilez states it, the further away surfaces contribute less to the occlusion than the nearby ones.

Based on the solution presented by Quilez the ambient occlusion algorithm is given in Algorithm 3.4. The procedure is based on the concept of constant step ray marching of Algorithm 2.1. One of the notable differences is the change of the ray equation to the one of 3.27 and that there are no explicit intersection checks. As previously stated the ray now is marched from the surface point in the direction of its normal for a maximal number of steps defined by k_{max}. At each step the occlusion is estimated and the result accumulated. Usually for the value of k_{max} a couple of steps are sufficient. Quilez in his presentation mentions that around 5 steps will be adequate. This of course varies with respect to the scene complexity as well the step size yet it barely reaches high values. This might be attributed to the observation by Zhukov et al. when they described their model that the lighting of a point is mostly affected by the geometry in its vicinity.

The initial step is for $k = 1$. The 0 value is removed form the k domain since we will not have any use for it in our computations by measuring the distance to the surface point itself. The s as stated previously is a scale influencing the darkening effects of surface points being occluded. The end result is clamped to the range between 0 and 1 since that is the result we expect to use in for the variable A_0 in the ambient term of equation (3.32).

Similarly as in the obscurance case the ambient term previously given by equation(3.2) now becomes.

$$ambient = I_a k_a(\boldsymbol{x}) A_O \tag{3.32}$$


(a) Sphere traced: sphere and ground plane with AO addition

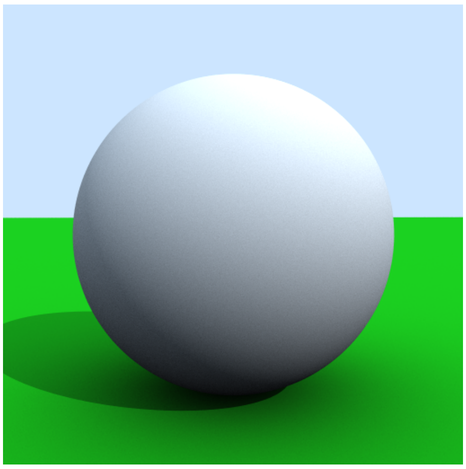

(b) Reference image, render time $\sim$ 20 minutes

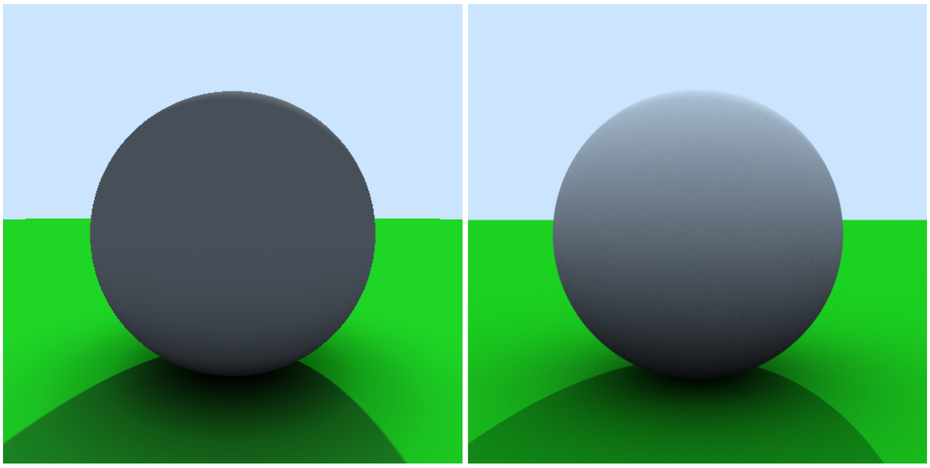
Figure 3.9: Addition of ambient occlusion. Sphere, ground scene. View 1

3.5.2 Rendering results

Continuing in the process of achieving our goal of rendering images as closely resembling the referenced ones, the results of application of ambient occlusion Algorithm 3.4 are shown in Figures 3.9a and 3.10a. The reference images for these are 3.9b and 3.10b respectively.

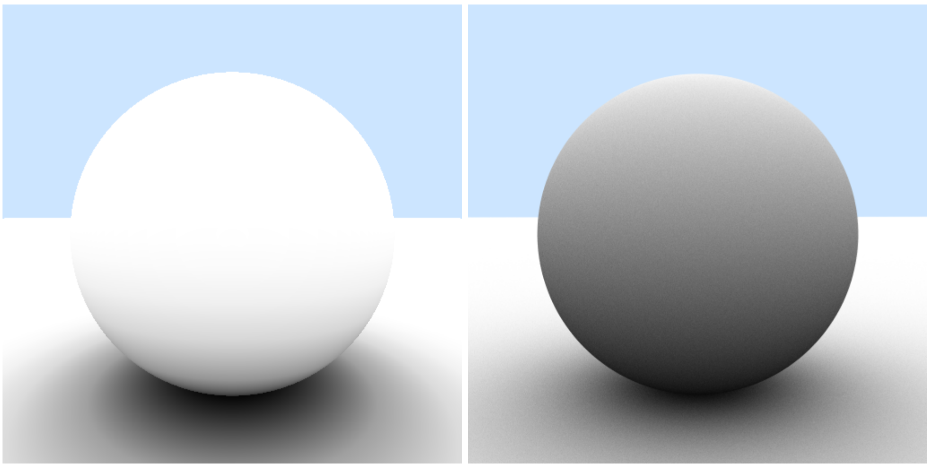
The first view of the scene depicted in Figure 3.9 illustrates that by an appropriate selection of color values and other parameters in the previously presented algorithms and equations it is possible to achieve quite similar results. However, while keeping the same settings, the differences become more apparent when the scene is displayed in another view depicted in Figure 3.10. Of notice is the shading of the sphere. The reference image displays a gradual change in color from the sphere's top to its bottom. However, in our image that is not the case. A change in the color is only noticeable near the bottom of the sphere. This is due to the fact that from a certain height of the sphere the geometry that contributed to the occlusion (in this case the ground) is no longer regarded as an occluder. In these cases the equation (3.29) holds.

The difference is more apparent when viewing just the ambient occlusion contribution like the one in Figure 3.11.



(a) Sphere traced: sphere and ground plane with AO addition **(b)** Reference image, render time ~ 20 minutes

Figure 3.10: Addition of ambient occlusion. Sphere, ground scene. View 2.



(a) Sphere traced: sphere and ground. Ambient occlusion influence **(b)** Reference image. Ambient occlusion influence. Render time ~ 20 minutes

Figure 3.11: Sphere, ground scene. Ambient occlusion contribution

Algorithm 3.5 Distance function for a signed box, version 1

```
1: $\boldsymbol{di} = |\boldsymbol{x}| - \boldsymbol{b}$
2: $c = \max(\boldsymbol{di}_x, \max(\boldsymbol{di}_y, \boldsymbol{di}_z))$
3: return $\min(c, \|\max(\boldsymbol{di}, 0)\|)$
```

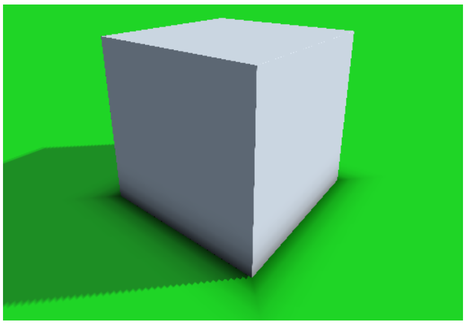
The occlusion contribution is depicted as darkened areas. The sky background has been left here as it was, in order to better visualize the contours of the ground and the sphere. The difference in both images is quite noticeable. Our image also displays some banding effect on the ground. That is, the occlusion does not uniformly change in color as result of which some "rings" of different shades of gray are noticeable. This is attributed to taking steps of finite length along the ray when determining the occlusion. The result of the reference image in Figure 3.11b is achieved by a cosine-weighted hemisphere sampling and checking the visibility of the sky.

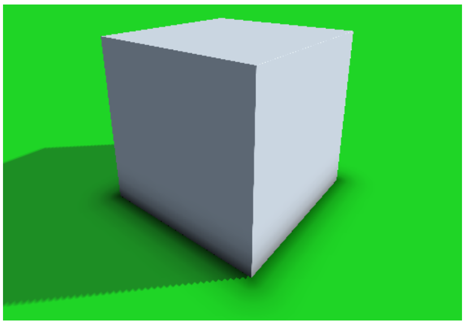
The reader might wonder why the sphere in the reference image 3.10b appears brighter than the one sphere traced in the Figure 3.10a, considering that the opposite is true in Figure 3.11 for the ambient occlusion contributions of the respective methods. The reason for this is that the ambient reflectance $k_a(\boldsymbol{x})$ is of a higher value for the reference images.

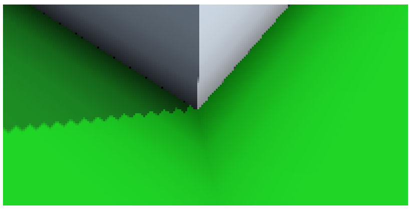
Both views in Figures 3.9 and 3.10 have the same respective (for each method) ambient contribution as depicted in 3.11. The reason for that is that no lighting is taken into consideration in determining the occlusion and it only depends on the geometry of the scene, which for both views in this particular case the contributing geometry is the same.

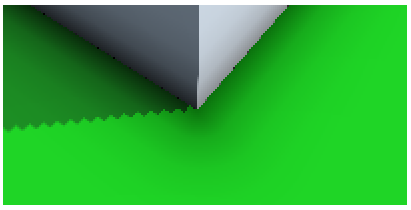
For producing the image like the one with the box in Figure 3.1b we first need to define a distance function for it. Quilez gives one in [nQ08a] and it is repeated here in algorithm 3.5: The box is centered at the origin of the world space. As previously $\boldsymbol{x}$ denotes a point in space. Here $\boldsymbol{b}$ stands for the bounds of the box. For instance $\boldsymbol{b} = (0.5, 0.5, 0.5)$ would result in creation of a box having dimensions $1 \times 1 \times 1$ that is $\boldsymbol{di} = (1, 1, 1)$. The operator $|\cdot|$ stands for absolute value, $\|\cdot\|$ for the length of the vector. The max and min operators are respectively the maximum and minimum operators. The $\max(\boldsymbol{di}, 0)$ is understood as: apply the maximum operator to each component of the vector $\boldsymbol{di}$ along with 0 as the second argument and save the results as a vector having the same number of components as the original vector $\boldsymbol{di}$.

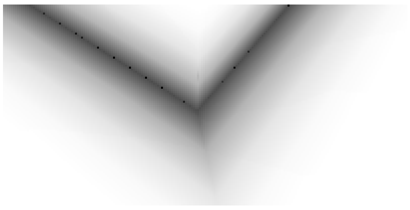
The result of rendering the box is given in Figure 3.12a. The aliasing here is more apparent than in the case of rendering the sphere, though the only change was the use of a different implicit surface (that is the box). All the parameters


(a) Box rendered with algorithm 3.5


(b) Box rendered by with algorithm 3.6


(c) Bottom corner part of the box from 3.12a


(d) Bottom corner part of the box from 3.12b


(e) Ambient occlusion contribution of 3.12c

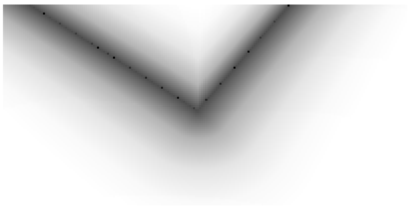

(f) Ambient occlusion contribution of 3.12d

Figure 3.12: Box and ground scene version 1

determining how the scene looks like have been left unchanged, including the ones for "soft shadows". It can be seen that the box edges and the shadows are affected as it is visible in Figure 3.12a. Also the determination of ambient occlusion is affected by this as it can be seen by apparent black dots in the aforementioned figure and in Figure 3.12e. Additionally, although we are able to render the shape itself correctly ambient occlusion contribution in the outside corner areas of the box does not seem to look right with respect to other regions. Upon careful inspection the reader might notice that the Algorithm 3.5 does not produce correct distance values around the corners of the box.

This is taken into account in Algorithm 3.6 with the use of which Figure 3.12b

Algorithm 3.6 Distance function for a signed box, version 2

```
1: $\boldsymbol{di} = |\boldsymbol{x}| - \boldsymbol{b}$
2: $res = \max(\boldsymbol{di}_x, \max(\boldsymbol{di}_y, \boldsymbol{di}_z))$        ▷ Distance for inside the box
3: if $res > 0$ then
4:     $res = \| \max(\boldsymbol{di}, 0) \|$        ▷ Distance for outside the box
5: return $res$
```

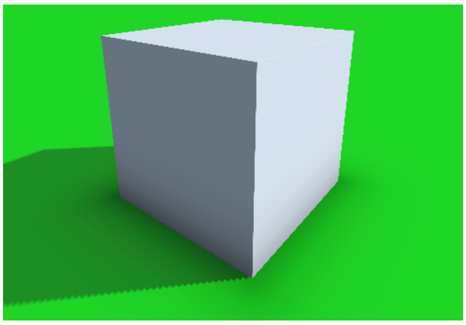
is rendered. The ambient occlusion contribution is visible in Figure 3.12f. The corner region now looks more plausible than the one from Figure 3.12e. The Figures 3.12a and 3.12b have been cropped to show only the vicinity of the box on the ground and that is why no sky part is visible in these images.

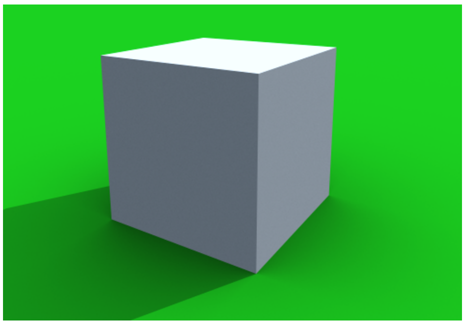
What is of interest here that can be taken form Algorithm 3.5 is that we are able to render shapes for which we "underestimate" the distance to its surface. In this particular case the underestimation happened just in the corner regions. The steps along the ray will be smaller in those regions as the result of the undounding sphere radius not being correctly estimated and leading to a slower convergence of the sequence of equation (2.20). Though it might not directly influence the "spacial" visualization of the surface, it has an impact on the computation of occlusion as it was illustrated by the previous Figure 3.12.

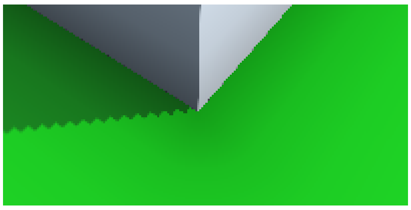
It is interesting to note that if we had overestimated the distance then we might have had similar issues as the ones that were discussed for the constant step ray marching Algorithm 2.1. That is: we might have gone through the surface (or some parts of it) or not precisely have estimated the surface point position in space. These both cases would have consequences for later computation of ambient occlusion.

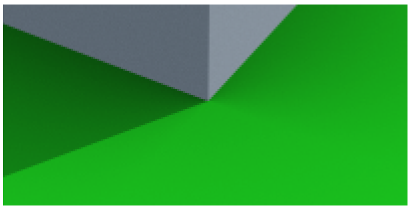
The results of rendering the box and ground scene along with the reference images are visible in Figure 3.13. The ambient occlusion contributions for these images are also provided. On the left side are images obtained by sphere tracing and on the right side are their reference counterparts. The images have been cropped (removing the sky background) in order for them to fit on one page. It is still quite visible how the ambient occlusion contribution differ from the sphere traced images to the reference ones. Of notice is the result that estimated the amount of occlusion in the region in the vicinity of the box corners depicted in Figures 3.13g and 3.13h. The latter image determines the surface points there to be less occluded than the points near the sides of the box. The situation is quite different for the former image, the occlusion in these areas is determined to be the same.

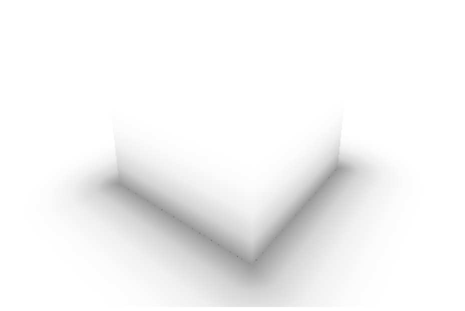
Moreover, for producing these images the ambient occlusion parameter values

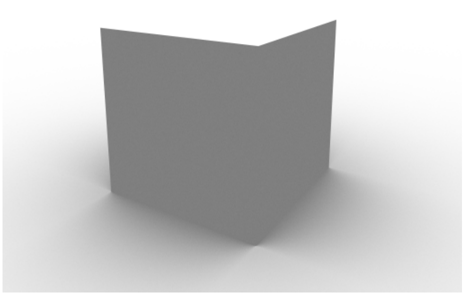

(a) Box rendered by the use of Algorithm 3.6

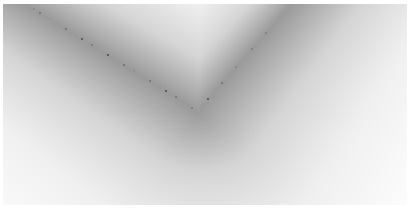

(b) Reference image: render time $\sim$ 20 minutes


(c) Bottom of the box from 3.13a


(d) Bottom of the box from 3.13b


(e) Ambient Occlusion influence


(f) Ambient Occlusion influence in reference image: render time $\sim$ 20 minutes


(g) Bottom of the box from 3.13e

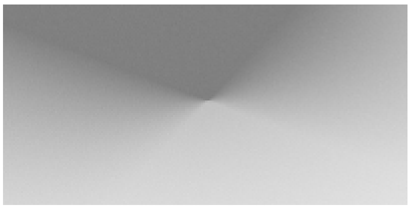

(h) Bottom of the box from 3.13f

Figure 3.13: Box and ground scene comparison

Algorithm 3.7 Ambient Occlusion with normal modification

```
1: $k = 1$
2: $d = 0$
3: $occlusion = 0$
4: while $k < k_{max}$ do
5:     $\boldsymbol{r} = \boldsymbol{x} + \Delta t\ k\ \boldsymbol{n}$
6:     $d = f(\boldsymbol{r})$
7:     $occlusion = \frac{1}{1.1^k}(\|\boldsymbol{n}k\Delta t\| - d)$
8:     $\boldsymbol{n} = \boldsymbol{n} + \alpha\boldsymbol{u}$
9:     $k = k + 1$
10: return $1 - \text{clamp}(s\ occlusion, 0, 1)$
```

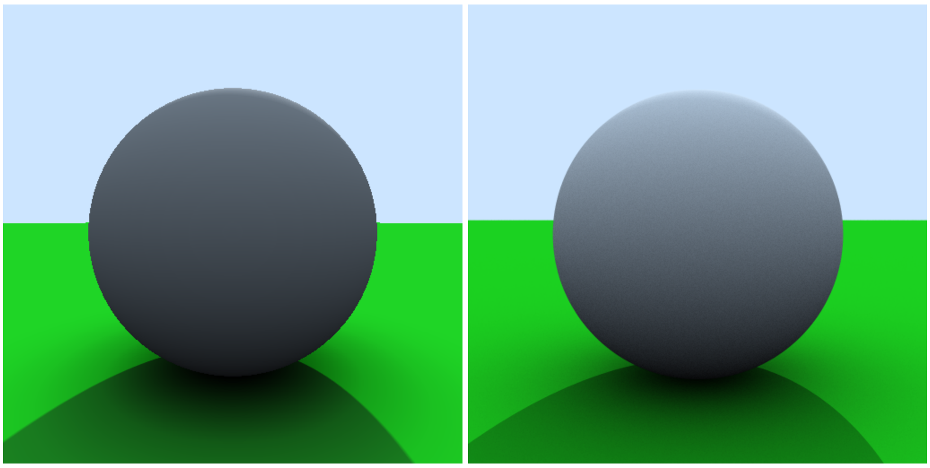
in Algorithm 3.4 have been changed with respect to those that helped to achieve the images of the sphere and ground of Figures 3.9 and 3.10 and those of box and ground of Figure 3.12. This was done in order to show that this method is not as robust as the one from the reference images and some adjusters usually are made when introducing new geometry to the scene. Moreover, the new settings make the rendered images of 3.13 slightly more in accordance with the reference images than those of 3.12. As it is mentioned in [Chr11] the parameters for the ambient occlusion Algorithm 3.4 do not have an "obvious default settings". Ergo they usually change depending on the scene complexity or what type of occlusion the user would like to make more apparent.

Evans in [Eva06a] mentions that in order to better approximate the sky effect one can influence the surface point normal. He gives the following formula for changing the normal per each sample step:

$$\boldsymbol{n} = \boldsymbol{n} + \alpha\boldsymbol{u} \tag{3.33}$$

Here α denotes a parameter for which Evans used the value of 0.5 in his paper. The vector $\boldsymbol{u}$ denotes the upward pointing unit vector of the scene. The Algorithm 3.4 has been modified with this addition and its new version is given in Algorithm 3.7. The changes with reference to the previous version of the algorithm include as follows. The addition of line 8 to include equation (3.33). Line 7 incorporates the occlusion estimate from a sample point along the new normal. And also changes the base of the power from to 2 to 1.1 (this number seemed to work for the case described in the following paragraphs). And finally the ray equation in line 5 has been given explicitly for better illustration of the influence of the normal there in that equation.

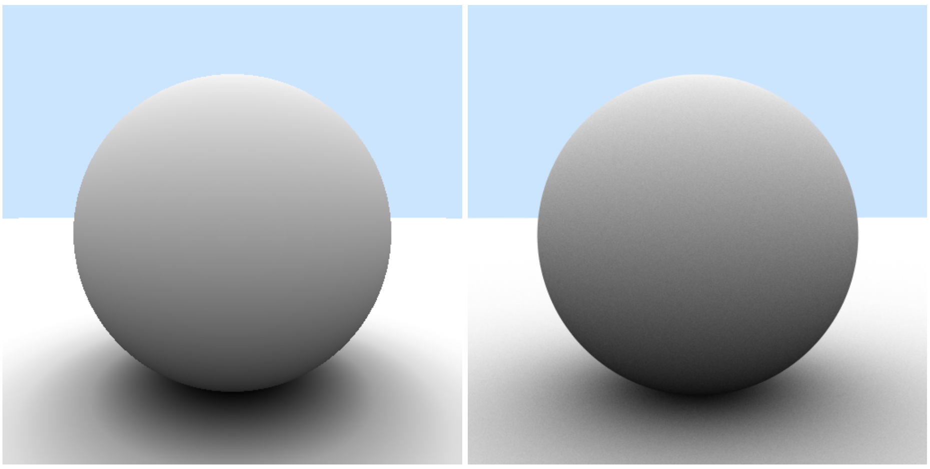
When applying new algorithm to render again the sphere and ground scene of 3.10 we receive the following result in figures 3.14a and 3.15a.



(a) Sphere and ground plane with AO addition of Algorithm 3.7

(b) Reference image, render time ∼ 20 minutes

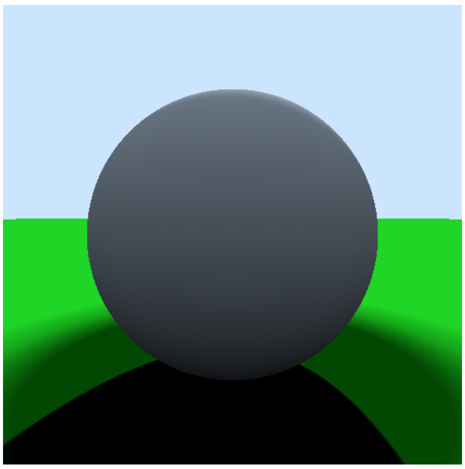
Figure 3.14: Addition of ambient occlusion. Sphere, ground scene. View 2. Version 2



(a) Ambient occlusion influence using Algorithm 3.7.

(b) Reference image. Ambient occlusion influence. Render time ∼ 20 minutes

Figure 3.15: Sphere, ground scene. Ambient occlusion contribution. Version 2


(a) Ground and sphere scene using the modified algorithm

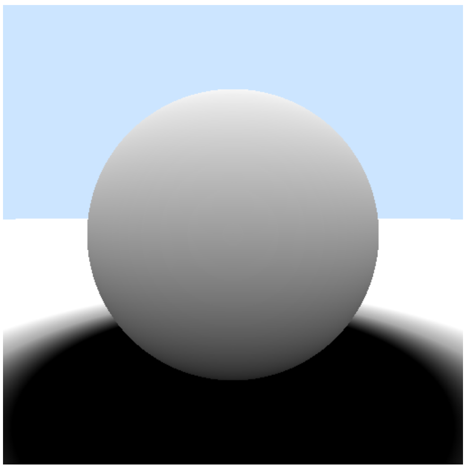

(b) Ambient occlusion influence of figure 3.16a.

Figure 3.16: Sphere, ground scene. Ambient occlusion contribution. Version 3. Depicting the differences in assessment of the magnitude of occlusion for surface points of the sphere and ground when both are determined by Algorithm 3.7 with the same values of parameters.

The reference images are the same as those of 3.10b and 3.11b and have been copied here for the readers convenience. As it can be seen from the images in 3.14a and 3.15b the application of the modified ambient occlusion Algorithm 3.7 makes the sphere traced images more closely resemble the reference ones. The reader must wonder at this point why this change has not been introduced earlier in this section. This is because of two reasons, presented below.

The first reason being that in order to produce the images of 3.14a and 3.15b both versions versions of the ambient occlusion algorithm are need to be used. The Algorithm 3.4 is used for determining the occlusion of the points belonging to the ground while the modified ambient occlusion Algorithm 3.7 just to determine the occlusion of points belonging to the sphere. The reason for this can be depicted in Figure 3.16 when only the new algorithm is used for both the sphere and the ground.

What can be seen here is that when making the sphere's shading resemble the one from the reference images by setting appropriate values in the modified ambient occlusion algorithm the shading of the ground is then considerably influenced.

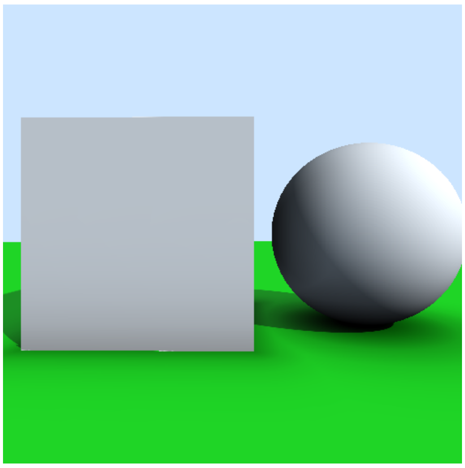
The second reason is the algorithm result when other surfaces are in the vicinity of the sphere as it will be illustrated in the next paragraphs.

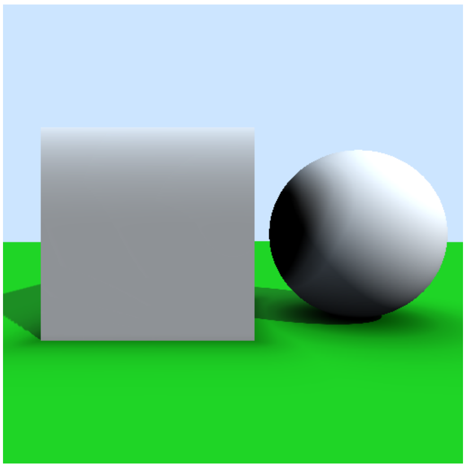
Having presented distance functions for a box and a sphere one can also test how the effects of ambient occlusion look like when those shapes are in near vicinity of each other. These are presented in Figure 3.17. In Figure 3.17a the scene is rendered with the use of the previous ambient occlusion Algorithm 3.4. However, in the Figure 3.17b both the sphere and the box use the modified ambient occlusion algorithm, whereas the ground uses the previous one. Figure 3.17c shows the reference image.

It can be clearly seen that the occlusion using the new algorithm in Figure 3.17b is quite exaggerated when keeping the same parameter values that were used for rendering Figure 3.14a. Moreover, the box seems to have the effect of being highlighted near its top. On the other hand Figure 3.17a was rendered with the same parameter values as Figure 3.10a and the ambient occlusion effects are very close to the ones for the reference image.

The above cases hopefully show, how hard it is to set the correct parameters that would work for a generic scene. Usually, each scene, or scene configuration will have specific values by the use of which a specific effect would be tried to be achieved. This is unlike the case in the reference images where the method for obtaining these there is robust.

The sphere traced rendered scenes of this chapter are all displayed in interactive frame rates for the resolution 512×512 and these have been rendered on a NVIDIA GeForce GTX 560 Ti graphic card. The frame rates for all cases greatly exceeded 60 frames per second.


(a) Use of ambient occlusion Algorithm 3.4 for the whole scene of box, sphere and ground


(b) Use of ambient occlusion Algorithm 3.4 for the ground, and Algorithm 3.7 for the sphere and box

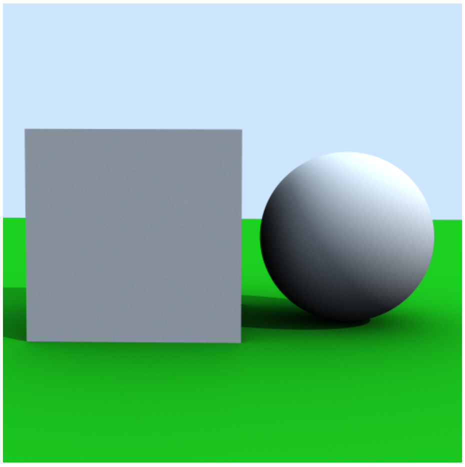

(c) Reference image, render time $\sim$ 20 minutes

Figure 3.17: Sphere, Box and ground scene

Chapter 4

3D textures as implicit surfaces

It may seem at first that the Stanford Bunny might be quite challenging to render considering that this thesis deals with surfaces defined with the use of their distance functions, whereas the bunny is given by a representation of a mesh of 69.451 triangles [Tur00]. However, there exist a solution in defining the distance function for that model. One might think of it being the manipulation of distance functions of known implicit surfaces in order to obtain the bunny shape. This however, would be quite hard to achieve since of the very cumbersome process of having to deal with many implicit shapes along with the operations and deformations applied to them.

Therefore, another method for obtaining the distance function was chosen. The process involves placing the model in a bounded three dimensional grid and sampling the distance values to the surface for every point in that grid. The raw floating point data representing the distance values (minimum distance to the surface for a given grid point) is then read in and stored as a 3D distance texture of appropriate size. This will serve as a representation of a distance function describing the surface of the bunny. It is vital to state that the domain of the distance function is no longer $\boldsymbol{x} \in \mathbb{R}^3$ but bounded by the texture coordinates and these being in the [0, 1] range. The tool *MeshDistance* with the help of which the triangle mesh was transformed to a grid of distance values was of the *GEL* (GEometry and Linear algebra) framework [GEL].

Algorithm 4.1 Distance function for a cube sized 3D texture

```
1: textureBoundary = $(0.5, 0.5, 0.5)$
2: textureShell = $(0.05, 0.05, 0.05)$
3: $\boldsymbol{di} = |\boldsymbol{x}| -$ textureBoundary + textureShell
4: $res = \|\max(\boldsymbol{di} + \text{textureShell}, 0)\|$        ▷ for $\boldsymbol{x}$ outside 3D texture space
5: $c = \max(\boldsymbol{di}_x, \max(\boldsymbol{di}_y, \boldsymbol{di}_z))$
6: if $c < 0$ then
7:     $\boldsymbol{x} = \boldsymbol{x} +$ textureBoundary
8:     $res = \text{lookupTexture}(\text{texture}, \boldsymbol{x})$        ▷ for $\boldsymbol{x}$ in 3D texture space
9: return $res$
```

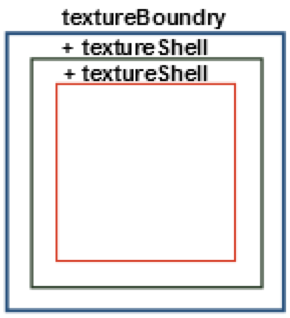


Figure 4.1: Texture boundary concept

However, before the conversion process commences from 3D mesh to the 3D distance texture the Stamford Bunny mesh has to be altered in some areas. This is because the model itself contains 5 holes in the mesh [Tur00] and these need to be closed in order to obtain the correct representation from the MeshDistance tool.

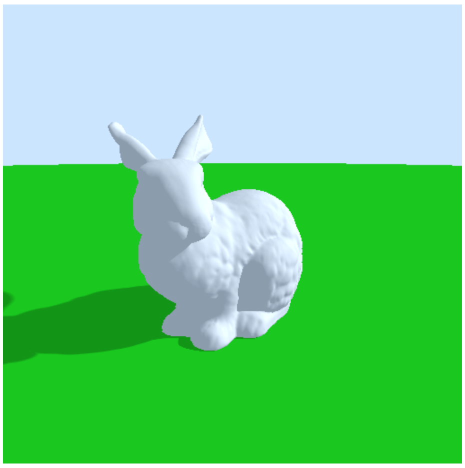
Upon obtaining the representation of the bunny as a 3D texture distance field its distance function can be constructed as we have done in algorithm 4.1. This distance function will allow for placing the texture inside the scene. The addition to the scene is done like previously described using the union operator of equation 3.11.

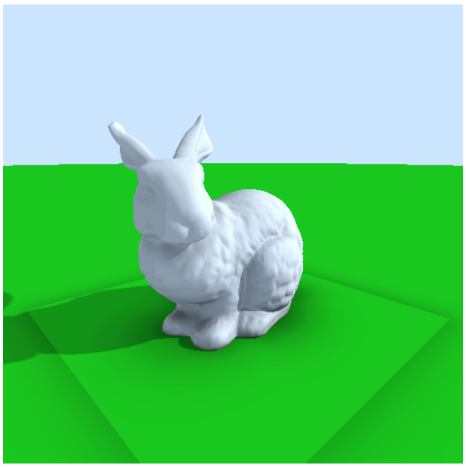
The procedure in the algorithm in general terms involves a creation of a box like in algorithm 3.6 which is centered at origin and bounded by the dimensions satisfying the property $|\boldsymbol{x}| \leq (0.5, 0.5, 0.5)$ (given by the variable *textureBoundary*). This is because the 3D texture is placed inside that box and the textures considered here are represented as cubes of dimensions $1 \times 1 \times 1$. The Figure 4.1 will serve as a help in explaining what the algorithm does. As it was said previously the texture is bounded by a cube/box that is defined by *textureBoundary* and in the figure illustrated by the blue box. Texture samples of interest to us are those bounded by the green box. Those outside the green box but still bounded by the blue one, are of the texture, yet since they are on the boundary region of the texture they may contain distance values that might be invalid. In this way we are remedying this by disregarding that region for texture lookups. The *textureShell* value given here in the algorithm and used in the process of estimating that boundary region was deduced by try and error and may change for a different texture. However, of interest is that that value would be as small as possible to get the largest region in which texture data can be looked up.

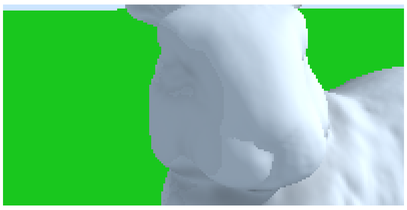
The process in the algorithm commences as follows. At first we estimate a point's distance to the red box in the Figure 4.1. The points location here is not checked relative to the red box, but computation in lines 3 and 4 assumes that the point is outside of the red box and calculates the distance from it to the nearest point on the red box. By what was already seen with sphere tracing is that using this distance value the ray will be traced up to the red surface/box if an intersection is determined and won't be able to pass into the interior of the surface. The overall spacing of the red box relative to the green one is such that the space of potential texture lookup values bounded by the red box would be as large as possible while allowing the ray to pass through the green surface and at the same not pass through the red one. In this case the "relative" spacing of the red box and the green one was determined by the *textureShell* variable, but can be set to any other variable that is found to work for a particular texture.

If the ray has passed through the green surface which is determined by lines 5 and 6 then we know that the ray is in the texture region we are interested in. Therefore, we can now asses the distance to the implicit surface represented by distance values in the texture by looking it up. Before that, however, we need to account the point's relative position to the texture coordinates, this is done in line 7. The distance lookup then is done in line 8 with the *lookupTexture* function. Ergo, this distance is returned by the distance function if the tray has entered the green bounding region. Otherwise, the distance to the red bounding region is returned as was indicated previously by line 4.

What is of importance is that when creating the 3D texture there exist some "outer" region containing distance values but in which the surface itself is not stored there. In the spacial representation this would be the space surrounding the red surface but bounded by the blue one. Moreover, the surface itself should be contained in the red bounding region, the "inner" region, in order for the distance to it to be assessed correctly. Additionally, it is assumed that the distance returned by the texture lookup is in accordance with the scale of the scene otherwise an appropriate bound needs to be applied to it.


(a) No ambient occlusion computation


(b) Initial ambient occlusion computation


(c) Occlusion artefacts on the bunny's face

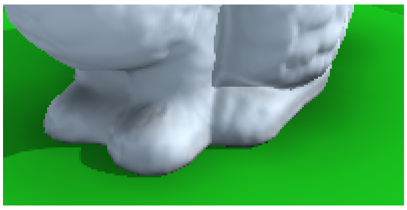

(d) Occlusion near the bunny's feet

Figure 4.2: Stanford Bunny as a 3D distance texture

In this distance function form of Algorithm 4.1 we are able to perform operations on it like the ones from section 3.3. Using the union operator of equation 3.11 we include the bunny representation into our scene. The rendered result is visible in Figure 4.2a in which no ambient occlusion is assessed. Of note is that the ϵ value in the normal estimation Algorithm 3.1 needs to be adjusted to account for for the voxel spacing in the texture. If it isn't the bunny representation might have a "grainy" look if the value is too small.

The texture used here is of $192 \times 192 \times 192$ dimensions, with the inner region's dimensions being $128 \times 128 \times 128$. These dimensions proved to be adequate for our purpose of visualizing the model of the bunny. The inner region was chosen such, so that when the ambient occlusion sampling commences, we are still able to access the texture region for obtaining distance values. Therefore this sampling should not leave past the texture bound. However, when the Ambient occlusion calculation is turned on the following result is achieved as visible in Figure 4.2b.

Algorithm 4.2 Distance function for a cube sized 3D texture. Version for ambient occlusion

```
1: textureBoundary = $(0.5, 0.5, 0.5)$
2: textureShell = $(0.05, 0.05, 0.05)$
3: $\boldsymbol{di} = |\boldsymbol{x}| -$ textureBoundary + textureShell
4: $res = k\Delta t$                                ▷ for $\boldsymbol{x}$ outside 3D texture space
5: $c = \max(\boldsymbol{di}_x, \max(\boldsymbol{di}_y, \boldsymbol{di}_z))$
6: if $c < 0$ then
7:     $\boldsymbol{x} = \boldsymbol{x} +$ textureBoundary
8:     $res$ = lookupTexture(texture, $\boldsymbol{x}$)          ▷ for $\boldsymbol{x}$ in 3D texture space
9: return $res$
```

It can be seen that the bounding red box contributes to the ambient occlusion result. Also there exist some artifacts on the bunny's face from the left side as it can be seen in Figure 4.2c. Apart from that the area near the bunny's feet seems to have the occlusion correctly determined as it is depicted if Figure 4.2d.

To mitigate this bounding red box influence a special distance function for the bunny texture needs to be created and provided to the ambient occlusion algorithm. We do so as explained in the following paragraphs.

Currently for $f(\boldsymbol{x})$ in the of sphere tracing Algorithm 2.2 and ambient occlusion Algorithm 3.4 the following function is used:

$$f(\boldsymbol{x}) = min(f_G(\boldsymbol{x}),\ f_B(\boldsymbol{x})) \tag{4.1}$$

Where, $f_G(\boldsymbol{x})$ and $f_G(\boldsymbol{x})$ respectively denote the distance functions of the ground (equation (3.15)) and the bunny (of Algorithm 4.1). This above equation will remain the same for the sphere tracing algorithm, yet it will be changed to:

$$f(\boldsymbol{x}) = min(f_G(\boldsymbol{x}),\ f_{BA}(\boldsymbol{x})) \tag{4.2}$$

in the ambient occlusion algorithm. Where $f_{BA}(\boldsymbol{x})$ denotes the ambient occlusion version of the signed distance function for the bunny 3D texture, and it is given in Algorithm 4.2. With respect to Algorithm 4.1 the only change is made in line 4. Here, the $k\Delta t$ is the value from the ambient occlusion algorithm that will be passed to the distance function for each step an determines the sample point along the ray equation of (3.27). What this means is that when we have left the bounds of the texture (or more precisely the values of interest enclosed by the green box) we want the bunny's surface not to influence the occlusion computation. On the other hand if the sample point is in the green region then the distance value stored in the 3D texture is returned. This simple change allows for producing the image in Figures 4.3a and 4.3c. These are contrasted with the their respective reference images given by Figures 4.3b and 4.3d. The

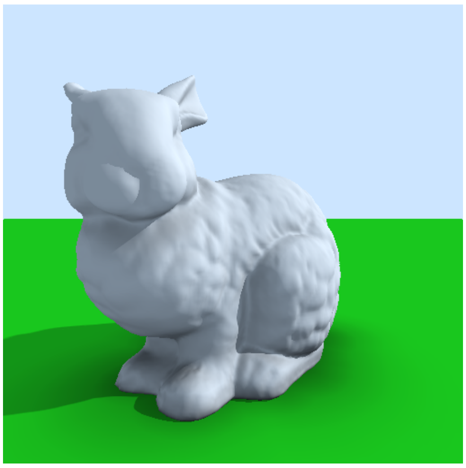
Algorithm 4.3 Fake Ambient Occlusion effect

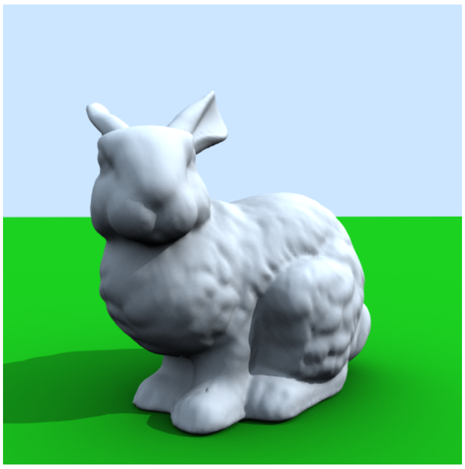
```
1: $occlusion = (1 - (\boldsymbol{n} \cdot \boldsymbol{u}))$
2: return $1 - \text{clamp}(s\ occlusion, 0, 1)$
```

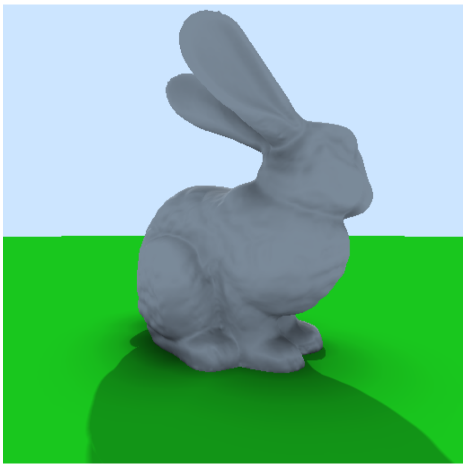
differences in shading are quite noticeable. This is attributed to the methods of estimation of the ambient occlusion. The algorithm used for the sphere traced rendered images is the one of Algorithm 3.4. If on the other hand we use Algorithm 3.7 where we added the changing normal the results are not better as it can be seen in the Figures 4.4a and 4.4c. A similar effect to the reference images is achieved in Figures 4.4b and 4.4d. However, this is a result of a trick in determining the ambient occlusion for each surface point of the bunny which is given in Algorithm 4.3. What is exploited here is the observation that the lesser the angle between the point's normal $\boldsymbol{n}$ and the scene's unit upwards vector $\boldsymbol{u}$ the brighter the point is. The effect can be adjusted by having the *occlusion* variable multiplied by the scale value s. As in the case of each ambient occlusion value it is clamped between 0 and 1. Using this algorithm will mean that the bunny's surface ambient occlusion value will not be influenced by a nearby occluder.

The effects of an occluder in the form of box can be seen in Figure 4.5. For the sphere traced images of Figures 4.5a and 4.5c the standard ambient occlusion Algorithm 3.4 was used. Of interest is the fact that the bunny's face is determined to be more occluded in the reference images than in the sphere traced ones. However, the opposite happens for the degree of influence the bunny has on the box. In the sphere traced Figure 4.5c the box is much more occluded by the bunny than in the reference Figure 4.5d.

The setting of the ambient occlusion for this chapter proved to be a challenge. Only the result from Figures 4.4b and 4.4d were close to match the reference images of Figures 4.3b and 4.4d. Unfortunately, the method for producing these images used a form of a trick to determine the possible occlusion of the surface of the bunny. Moreover, its drawback is that it does not reflect the influence of nearby occluders. Probably, if one invests enough time it would be possible to find a function that would in a better way determine the self occlusion of the bunny and the occlusion caused by other surfaces. Nevertheless, we were able to render the representation of the bunny stored in a distance field 3D texture and include it to our scene consisting of analytic implicit surfaces. The sphere tracing like in the previous chapter was also done in interactive frame rates.


(a) Sphere traced: bunny as a 3D distance texture


(b) Reference image, render time ∼ 45 minutes


(c) Sphere traced: bunny as a 3D distance texture

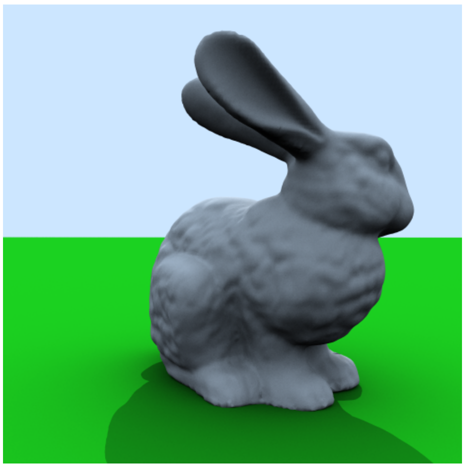
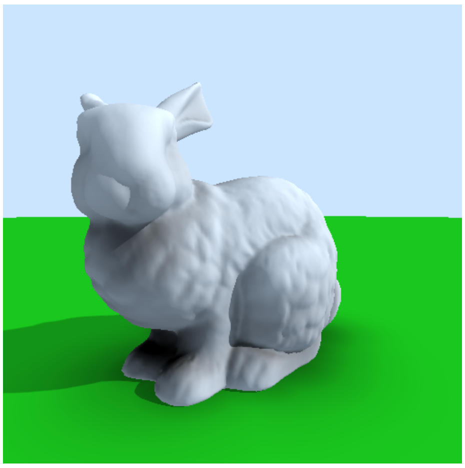
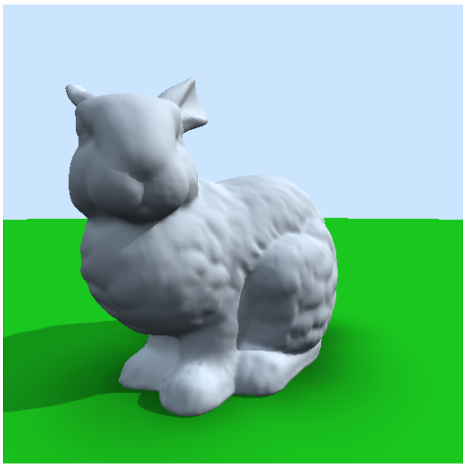
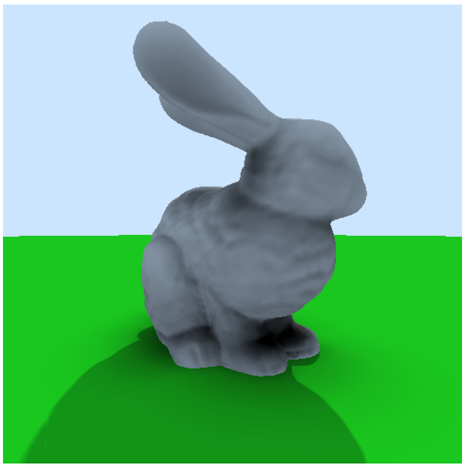

(d) Reference image, render time ∼ 45 minutes

Figure 4.3: Stanford Bunny as a 3D distance texture, comparison with the reference images


(a) Result from Algorithm 3.7


(b) Fake ambient occlusion effect


(c) Result from Algorithm 3.7

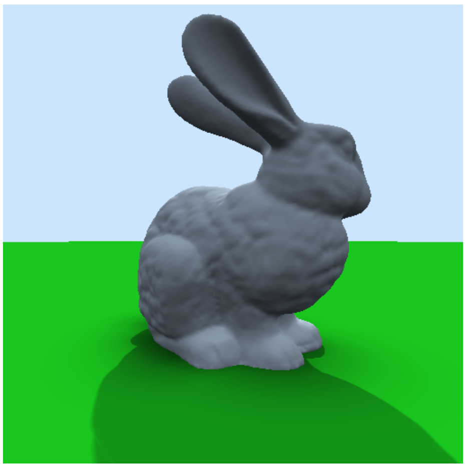
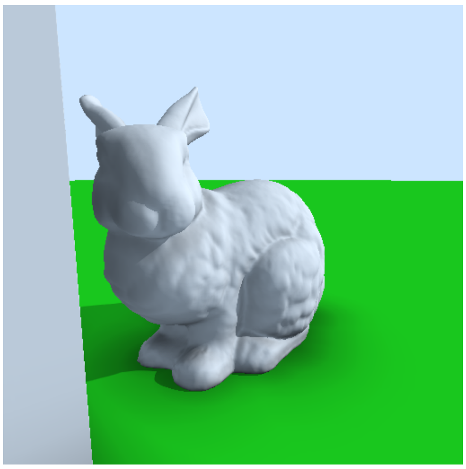
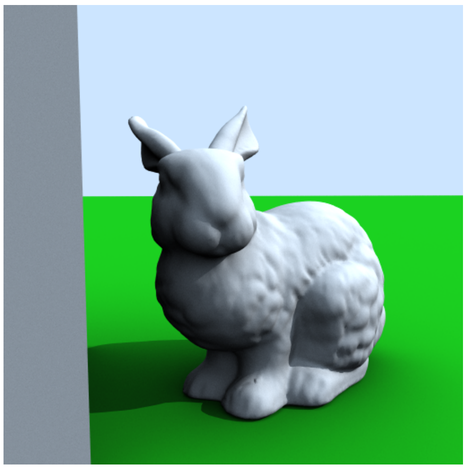

(d) Fake ambient occlusion effect

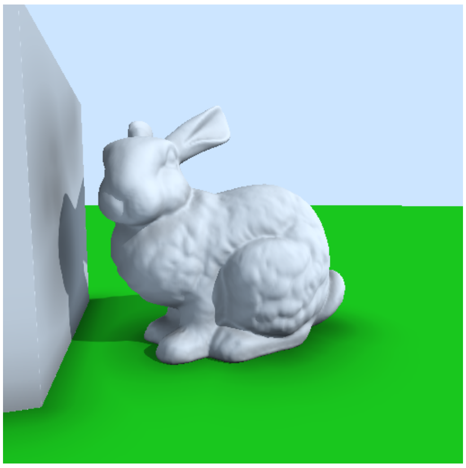
Figure 4.4: Stanford Bunny as a 3D distance texture, other ambient occlusion results



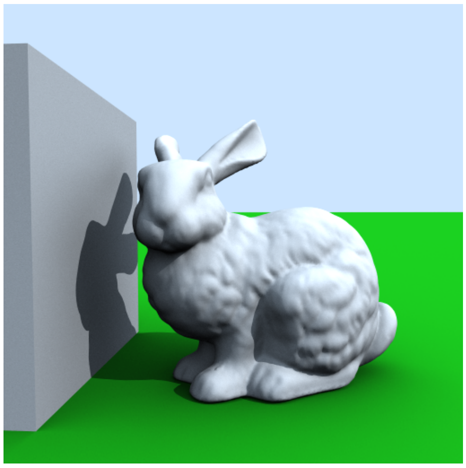
(a) Result with using Algorithm 3.4



(b) Reference image, render time ∼ 40 min



(c) Result with using Algorithm 3.4



(d) Reference image, render time ∼ 40 min

Figure 4.5: Bunny, box and ground scene

Chapter 5

Antialiasing

Sphere tracing as it was seen in the figures in the previous chapters is prone to the effects of aliasing. Taking from ray tracing we can apply similar techniques in order to remove these aliasing artifacts, such as symbolic prefiltering methods or those based on resampling [Mit90]. In the latter the increase of the sampling rate can be computationally expensive since each sample requires a ray casting calculation to be made. However, we would like to refrain from generating more primary rays (those that determine ray-surface intersection) since they can be quite computationally expensive in sphere tracing [nQ08b], especially on surface edges/boundaries.

Of interest to us are however, the symbolic prefiltering techniques that allow us to achieve antialiasing with the use of only one ray per pixel like cone tracing and the use of covers. Hart in [Har96] uses both of these methods in conjunction with sphere tracing to achieve antialiasing.

In standard ray tracing a pixel is represented as a point on the screen. The motivation for developing cone tracing is that the pixel could be represented by an area on the screen as Amanatides mentions in [Ama84]. The ideal solution is to have the ray repressed as a pyramid with its apex at the eye. Its base would then be defined with the use of the area on the screen enclosed by the pixel borders that the sides of the pyramid intersect. This representation, however, would make intersection calculations not trivial, and therefore Amanatides suggest an approximation in representing the ray with a cone.

When the cone intersects a surface it bounds a surface area. We now need to determine which point on it will contribute to the shading. In order to estimate that "representative", Hart in [Har96] uses among other the concept of covers. These are defined as offsets of a pixel length bounding the implicit surface from the outside and inside. Moreover, the author states that "a ray-cover intersection indicates a cone-object intersection". In terms of the implicit surface represented by its signed distance function $f(\boldsymbol{x})$, its outer cover can be defined as $f(\boldsymbol{x}) - r_p$, whereas, its inner cover as $f(\boldsymbol{x} + r_p)$. The r_p is the radius of the pixel and as the authors suggest in [HD91], it can be approximated with its maximal value for a given ray traversal distance t as:

$$p(t) = \frac{2\tan\theta/2}{N_h}t \tag{5.1}$$

Here θ is the field of view and N_h stands for the horizontal resolution. The r_p in fact is the approximation of "the horizontal extent of a pixel projected at a distance t from its ray origin". Taking into account the suggestion in [Bar86] we will modify the above equation to take into account the possibility of changing the resolution of the window. The above equation then becomes:

$$p(t) = \frac{2\tan\theta/2}{\max(N_h, N_v)}t \tag{5.2}$$

where we take now the highest value being either the horizontal N_h or vertical N_v resolution for determining the pixel size.

Hart states in [Har96] that, the antialiazing algorithm will sphere trace the inner cover. In Figure 5.1 it can be visible what happens when a cone intersects a surface. Near the intersection point the sequence of unbounding spheres comes within the bounds of the cone. The notion of covers allows to determine that the best approximation for the "representative" point, as Hart defines it, is "the point along the section of the ray closest to the surface". This being the center of the smallest sphere (with respect to pixel size) that is bounded by the cone.

The condition defining the cone intersection with the surface can be given as:

$$f(\boldsymbol{x}) \leq r_p \tag{5.3}$$

This will be usually fulfilled by a sequence of unbounded spheres from which the one with the smallest radius is taken as the representative sample, as it can be seen in Figure 5.1. The radius of the unbounding sphere is now defined as $r_p + f(\boldsymbol{x})$ which is also the step of the ray traversal. It is vital to note that this may take the ray traversal through the surface as it was the case with constant step ray marching.

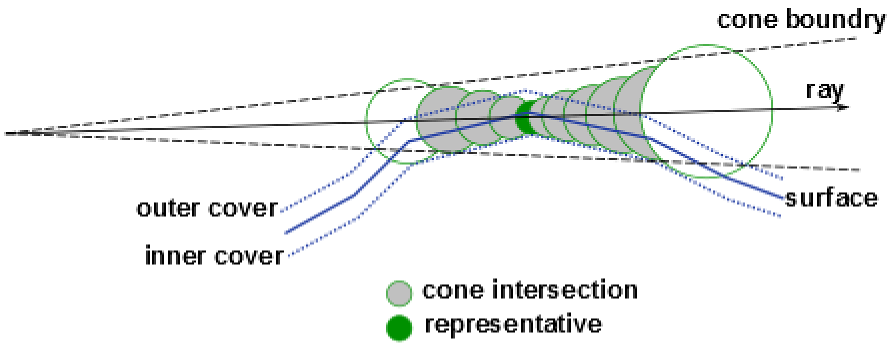


Figure 5.1: Cone surface intersection. Based on Figure 7 from [Har96]

Having chosen the representative sample we now determine what is the area of the cone surface intersection. Hart, determines that with the assumption that smooth implicit surfaces exhibit local planarity. The problem is brought down to calculating the "fraction of coverage of a disk of radius r_p by an intersecting half-plane of signed distance $f(\boldsymbol{x})$ from its center". This is given by Hart as:

$$\alpha = \frac{1}{2} - \frac{f(\boldsymbol{x})\sqrt{r_p^2 - f(\boldsymbol{x})^2}}{\pi r_p^2} - \frac{1}{\pi}\arcsin\frac{f(\boldsymbol{x})}{r_p} \tag{5.4}$$

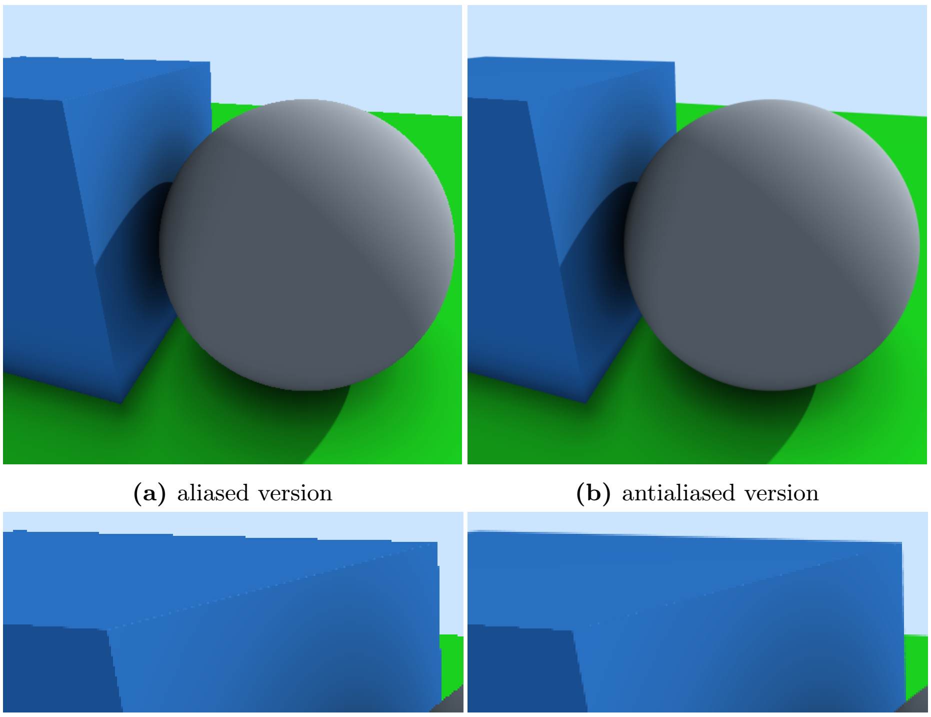
Thompson in [Tho90] gives two variants of an equation for calculating the intersection of a circle and a half plane depending if the circle's center is inside or outside of the half plane. However, because of the sign information that the signed distance function provides both cases are covered by the above equation.

The parameter α is defined here as the percentage of coverage and represents the degree of the cone intersection. It is treated as an opacity and it is accumulated. With its use, the blending of the current representative sample's color and the blended color of other samples can be determined. For this we have used the *over* operator as given by [PD84]:

$$c_O = c_A + c_B(1 - \alpha_A) \tag{5.5}$$

It is used for back to front image composition process. Here the surface A is over the surface B, with α_A denoting the opacity of the surface A. The values of c_A and c_B are color values already multiplied by their respective alpha values.

Using this operator requires from us to keep track of all the representative points indicated by the minimum unbounding sphere radius of a cone intersection. The color value of the pixel is then computed by compositing the color



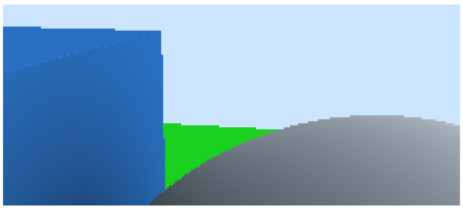
(a) aliased version **(b)** antialiased version

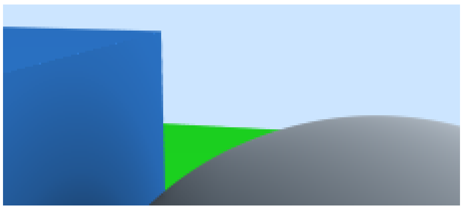
(c) aliased version **(d)** antialiased version

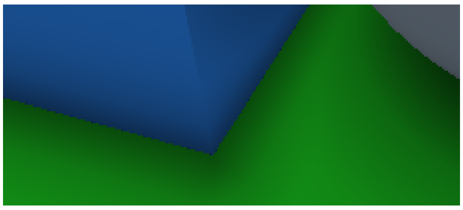
Figure 5.2: Aliased and antialiased basic implicit surfaces, part 1

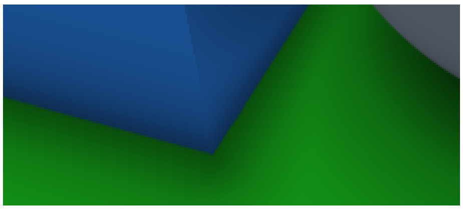
values of these representative samples in the back to front order. Amanatides mentions of keeping a list of eight samples, whereas we impose a limit of four in our implementation. This is because, keeping track of and storing the minimum unbounded sphere radius in a sequence proved to be computationally expensive in our case. The frame rates achieved are around 60 frames per second for images of 512×512 resolution, however, for larger resolutions they drop considerably. The aliased images and their antialiased counterparts are displayed in Figures 5.2 and 5.3. The ground in both figures has been replaced by "flattened" box.

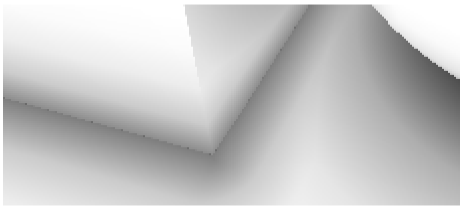
The antialiased solution that we implemented does not take into account the antialiasing of edges which if projected from the cameras perspective would lie on the object they belong to. This was done in order to maintain the interactive frame rate. This can be visible in Figure 5.2d and can be compared to the aliased version of Figure 5.2c. However, blending with the background or with other surfaces is done properly as it can also be seen in Figure 5.3b.


(a) aliased version


(b) antialiased version


(c) aliased version


(d) antialiased version


(e) ambient occlusion contribution, aliased version

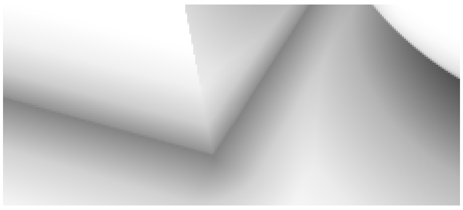
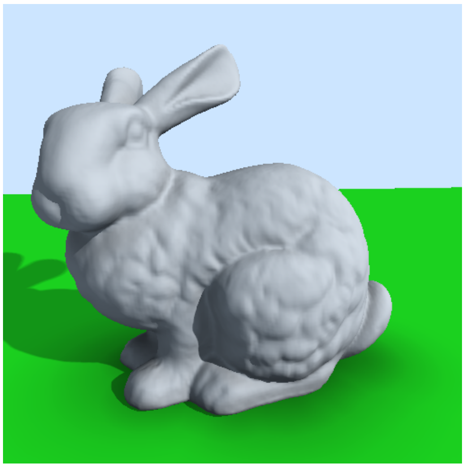
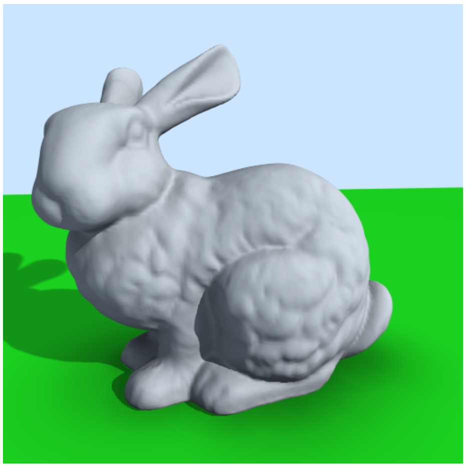

(f) ambient occlusion contribution, antialiased version

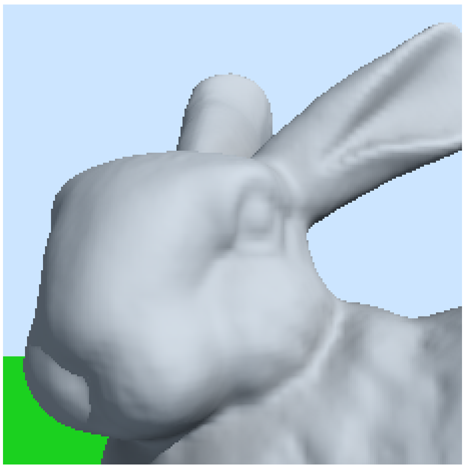
Figure 5.3: Aliased and antialiased basic implicit surfaces, part 2

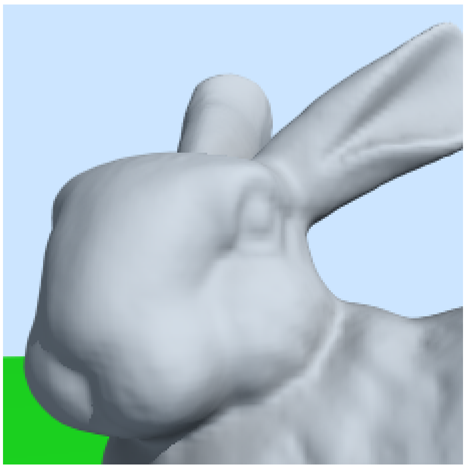
In Section 3.5.2 it was indicated that the black dots visible previously among other in Figures 3.12f and 3.13g existed there because no antialiasing was implemented. The aliased version of the ambient occlusion contribution in Figure 5.3e show their existence, whereas the antialiased version does not have these as shown in Figure 5.3f. Figures 5.3c and 5.3d show how these areas look when illuminated with the rest of the lighting model.

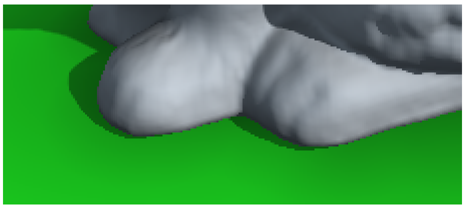
The effects of applying the antialiasing algorithm on the bunny 3D distance texture can be seen in Figure 5.4. As it can be seen these improve the rendering result.


(a) aliased version


(b) antialiased version


(c) aliased version


(d) antialiased version


(e) aliased version

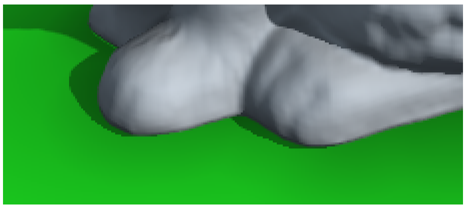

(f) antialiased version

Figure 5.4: Aliased and antialiased 3D Stanford Bunny distance texture

Chapter 6

Conclusions

In this thesis we have focused on producing images with the use of sphere tracing and distance fields. Our main motivation was to render images resembling the ones acquired from the ray tracing framework.

We have started with the introduction of implicit surfaces and the methods on how to render them. Following that, we defined the lighting model, and showed how to add and render more surfaces in the scene. Moreover, we have described how shadows are obtained and how to produce the effect of ambient occlusion. This allowed us to render results which we could then compare to the ones in the reference images.

The comparisons mainly revolved around the topic of how the ambient occlusion effect was depicted by each method for a given object representation or scene configuration. In this process we noted that for shapes like the sphere it is possible to achieve similar looking results in both methods. However, in general the results produced by both methods differed. This is mainly because of what is considered to be the indicator of occlusion for these methods. Based on it their limitations can be determined.

Though the distance from the distance function was used to estimate the geometry surrounding a given point it proved itself not that accurate with respect to method used for producing the reference images. Some additions to the ambient occlusion algorithm were proposed to better asses the sky visibility yet these produced adequate results in only certain cases. Moreover, we found out that the same parameter configurations of the ambient occlusion algorithm are not necessarily suitable for other scenes or rendered surfaces.

In the process of rendering our results, we have found it also quite difficult to accurately determine the correct parameters of the ambient occlusion algorithms that we have presented. This is because they do not take from any physical model.

We have also, introduced a method for producing a distance function for a object represented by a 3D distance field. This allowed us to render an object like the Sanford Bunny using our sphere tracing implementation. Though the surface of the bunny was rendered correctly the ambient occlusion effect differed from the one in the reference image.

Finally, we have addressed the issue with aliasing by implementing a solution based on cone tracing with the use of covers.

The overall comparison has made us believe that relying on just the distance value of the distance function in the way presented in the ambient occlusion algorithms is not a reliable indication of occlusion. However, we base this observation on the comparison with the results obtained from the ray tracing framework. It is vital to note that in some cases, if it were not for reference images if would be quite hard to asses the correctness of the results, since some of them looked very plausible. Moreover, the advantage of the sphere traced approach is that it allows us to view the scene in interactive frame rates. One of the possible applications of this is the ability to create a quick approximation of the results of the reference ray tracing method.

6.1 Future work

The *over* operator used for image composition in the antializing Chapter 5 required from us the storage of the smallest undbounding sphere radius for a given cone intersection. The this storage requirement for computing antialiasing might be mitigated by the use of the *under* operator for front to back image composition as given in [IKLH04]. Its application could bring performance benefits which would be worth investigating.

Another topic that we would like to investigate is the rendering of fractals, and sphere tracing is quite capable of doing that [HST89].

The addition of fur to the Bunny would also be an interesting choice. This can be achieved by the use of the “Hypertexture” model [PH89]. The acquisition of distance values to the “Hypertexture” is also elaborated in [Har96].

Bibliography

[Ama84] John Amanatides. Ray tracing with cones. *SIGGRAPH Comput. Graph.*, 18(3):129–135, January 1984.

[Bar86] Alan H. Barr. Ray tracing deformed surfaces. *SIGGRAPH Comput. Graph.*, 20(4):287–296, August 1986.

[Chr11] Mikael Hvidtfeldt Christensen. Distance Estimated 3D Fractals (II): Lighting and Coloring. http://blog.hvidtfeldts.net/index.php/2011/08/distance-estimated-3d-fractals-ii-lighting-and-coloring/, 2011.

[Don05] William Donnelly. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter Per-pixel Displacement Mapping with Distance Functions, pages 123–136. Addison-Wesley Professional, 2005.

[Elv92] T. Todd Elvins. A survey of algorithms for volume visualization. *SIGGRAPH Comput. Graph.*, 26(3):194–201, August 1992.

[Eva06a] Alex Evans. Fast approximations for global illumination on dynamic scenes. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 153–171, New York, NY, USA, 2006. ACM.

[Eva06b] Alex Evans. Fast Approximations for lighting of Dynamic Scenes. http://developer.amd.com/media/gpu_assets/Evans-Fast_Approximations_for_Lighting_of_Dynamic_Scenes-print.pdf, 2006.

[FK03] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Fri12] Jeppe Revall Frisvad. 02576 Physically Based Rendering, DTU course page. http://www2.imm.dtu.dk/courses/02576/, 2012.

[GEL] GEL (GEometry and Linear algebra) framework. http://www2.imm.dtu.dk/projects/GEL.

[Gol83] R. N. Goldman. Two Approaches to a Computer Model for Quadric Surfaces. *Computer Graphics and Applications, IEEE*, 3(6):21 –24, Sept. 1983.

[Har93] John C. Hart. Ray Tracing Implicit Surfaces. *ACM SIGGRAPH 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

[Har96] John C. Hart. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[HD91] John C. Hart and Thomas A. DeFanti. Efficient antialiased rendering of 3-D linear fractals. *SIGGRAPH Comput. Graph.*, 25(4):91–100, July 1991.

[HST89] John C. Hart, Daniel J. S, and Louis H. Kauffman T. Ray Tracing Deterministic 3-D Fractals. *Computer Graphics*, 23:289–296, 1989.

[IKLH04] Milan Ikits, Joe Kniss, Aaron Lefohn, and Charles Hansen. *GPU Gems*, chapter 39. Volume Rendering Techniques. Addison-Wesley Professional, 2004.

[IKSZ03] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Comput. Graph. Appl.*, 23(3):54–64, May 2003.

[JBS06] M.W. Jones, J.A. Baerentzen, and M. Sramek. 3D Distance Fields: A Survey of Techniques and Applications. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):581 –599, july-aug. 2006.

[KL05] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 41–48, New York, NY, USA, 2005. ACM.

[KRES11] Oliver Klehm, Tobias Ritschel, Elmar Eisemann, and Hans-Peter Seidel. Bent Normals and Cones in Screen-space. In Peter Eisert, Joachim Hornegger, and Konrad Polthier, editors, *VMV*, pages 177–182. Eurographics Association, 2011.

[Lan02] Hayden Landis. Production-ready global illumination. *ACM SIGGRAPH 2002 Course 16 Notes: RenderMan in Production*, pages 87–101, 2002.

[Men96] J. Menon. An Introduction to Implicit Techniques. *ACM SIGGRAPH 96 Course Notes: Implicit Surfaces for Geometric Modeling and Computer Graphics*, pages A1–13, 1996.

[MFS09] Àlex Méndez-Feliu and Mateu Sbert. From obscurances to ambient occlusion: A survey. *Vis. Comput.*, 25(2):181–196, January 2009.

[Mit90] Don Mitchell. The Antialiasing Problem in Ray Tracing. *ACM SIGGRAPH 90 Course Notes: Advanced Topics in Ray Tracing*, 1990.

[nQ08a] Iñigo Quilez. Modeling with distance functions. `http://iquilezles.org/www/articles/distfunctions/distfunctions.htm`, 2008.

[nQ08b] Iñigo Quilez. Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. `http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf`, August. 2008.

[nQ11] Iñigo Quilez. Free penumbra shadows for raymarching distance fields. `http://iquilezles.org/www/articles/rmshadows/rmshadows.htm`, 2011.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *SIGGRAPH Comput. Graph.*, 18(3):253–259, January 1984.

[PH89] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3):253–262, July 1989.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation.* Morgan Kaufmann, 2 edition, 2010.

[Tho90] Kelvin Thompson. Area of intersection: circle and a half-plane. In Andrew S. Glassner, editor, *Graphics gems*, pages 38–39. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[Tur00] Greg Turk. The Stanford Bunny. `http://www.cc.gatech.edu/~turk/bunny/bunny.html`, 2000.

[Wei] Eric W. Weisstein. "Sphere". From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/Sphere.html`.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson L. Max, editors, *Rendering Techniques*, pages 45–56. Springer, 1998.