Signed Distance Field Ray Differentials

Kevin Galim
TUM—Technical University of Munich
Boltzmannstrasse 3, 85748 Garching, Germany

April 12, 2018

Abstract

Ray differentials allow estimating the pixel footprint in the scene which can be used for filtering and anti-aliasing. This report introduces the use of ray differentials with signed distance functions, functions returning the distance to the closest point of the shape.

1 Introduction

Ray tracing is a powerful method for image synthesis and creating realistic looking images. Although this technique works really well for rendering any kind of 3D geometry and lighting can be simulated really realistic, it also suffers from issues regarding that only single rays with no thickness are traced though pixels. This could lead to aliasing if the resolution is not chosen high enough. To address this, supersampling could be used which traces multiple rays through one pixel and combines the results. This, however, comes with a high overhead considering the number of traced rays.

Another interesting approach is ray differentials [3]. This technique tries to estimate the extent of the whole footprint of ray inside the scene by keeping track of the differentials of a ray. The differentials of the ray give the change

of the ray's origin and direction when moving it on the x- and y-axis on the image plane. This can be used to determine filter regions and apply filtering in the respective area to consider the whole projected area of a pixel inside the scene.

The main focus for ray differentials are triangle meshes which are commonly used for rendering. An alternative approach of scene representation are signed distance fields which are useful for high detailed objects and exact representation of objects [4]. Here the shape is not represented as vertices connected to triangles, but as a mathematical function giving the distance to surface of the shape.

2 Ray Differentials

2.1 Definition

Using ray differentials, the footprint of a ray in the scene can be estimated. The basic concept is to regard an image plane ray as a function depending on the x and y offset on the image plane

$$R = P + tD \qquad R(x, y) = \{P(x, y) \quad D(x, y)\}. \tag{1}$$

To keep track of the ray change when shifting it on x- and y-direction on image plane, the ray function can be partially differentiated by the image plane coordinates x and y

$$\frac{\partial R}{\partial x} = \left\{ \frac{\partial P}{\partial x} \quad \frac{\partial D}{\partial x} \right\} \qquad \frac{\partial R}{\partial y} = \left\{ \frac{\partial P}{\partial y} \quad \frac{\partial D}{\partial y} \right\}. \tag{2}$$

By calculating these differentials for every ray propagation, the position and direction change of the ray when shifting on the image plane can be computed on every position along the ray path. This change rate can then be used to estimate the footprint extent of the pixel where the ray is shot through (Figure 1). For the differentiated ray, any arbitrary neighbor ray can be approximated which allows to approximate footprints into the scene by taking the difference of the approximated neighbor ray and the differentiated ray.

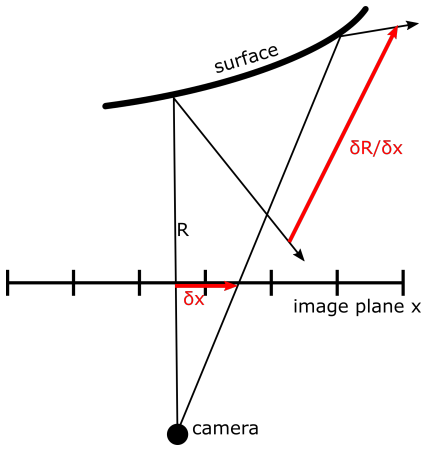


Figure 1: Ray Differentials on image plane x-axis.

2.2 Computation

Initially, every ray starts at the camera position and is directed through the image plane at coordinate (x, y)

$$\begin{aligned} P(x,y) &= \text{Camera} \\ D(x,y) &= \frac{d}{(d \cdot d)^{\frac{1}{2}}} \qquad d(x,y) = \text{View} + x\text{Right} + y\text{Up}. \end{aligned} \tag{3}$$

Differentiation by x then yields the following term

$$\begin{aligned} \frac{\partial P}{\partial x} &= 0 \\ \frac{\partial D}{\partial x} &= \frac{(d \cdot d)\text{Right} - (d \cdot \text{Right})d}{(d \cdot d)^{\frac{3}{2}}}. \end{aligned} \tag{4}$$

Differentiating by y would give a similar term.

For every change in position or direction by propagation, e.g. reflection and refraction, there is an equation to update the ray position and direction which can be again partially differentiated by x and y to update the ray differentials as well. Table 1 shows a quick overview of common ray operations and their derivate. For more details, see chapter 3.1 of "Tracing Ray Differentials" [3].

	Operation	Derivative
Transfer	$P' = P + tD$	$\frac{\partial P'}{\partial x} = \left(\frac{\partial P}{\partial x} + t\frac{\partial D}{\partial x}\right) + \frac{\partial t}{\partial x}D$
	$D' = D$	$\frac{\partial D'}{\partial x} = \frac{\partial D}{\partial x}$
Reflection	$P' = P$	$\frac{\partial P'}{\partial x} = \frac{\partial P}{\partial x}$
	$D' = D - 2(D \cdot N)N$	$\frac{\partial D'}{\partial x} = \frac{\partial D}{\partial x} - 2\left((D \cdot N)\frac{\partial N}{\partial x} + \frac{\partial (D \cdot N)}{\partial x}N\right)$
Refraction	$P' = P$	$\frac{\partial P'}{\partial x} = \frac{\partial P}{\partial x}$
	$D' = \eta D - \mu N$	
	$\mu = (\eta(D \cdot N) - (D' \cdot N))$	$\frac{\partial D'}{\partial x} = \eta\frac{\partial D}{\partial x} - \left(\mu\frac{\partial N}{\partial x} + \frac{\partial \mu}{\partial x}N\right)$
	$D' \cdot N = -\sqrt{1 - \eta^2\left(1 - (D \cdot N)^2\right)}$	

Table 1: Updating ray differentials for common ray propagation operations. The left column shows how ray position and direction gets updated with the specific ray operation. The right column shows the exact same update for the derivate of the ray position and direction in x direction.

3 Signed Distance Fields

3.1 General Signed Distance Fields

The idea of a signed distance field of a shape S is a function which takes a 3D point in world space as input and yields a scalar value giving the minimal distance of the point to the closest surface of the shape. Generally the value of the signed distance function is negative if the point is inside the shape, 0 on the shape and positive outside of the shape.

$$SDF(P) = \begin{cases} -\text{distance}(P, S_{surface}) & \text{if } P \text{ inside shape } S \\ 0 & \text{if } P \text{ on the surface of shape } S \\ \text{distance}(P, S_{surface}) & \text{if } P \text{ outside shape } S \end{cases} \quad (5)$$

The surface of the shape is then defined as the set of points where the signed distance function yields 0.

$$S_{surface} = \{P \in \mathbb{R}^3 \mid SDF(P) = 0\} \quad (6)$$

For proper rendering, the definition of the surface by this function is not sufficient. On every surface point, a normal has to be defined for lighting and ray propagation like reflection and refraction. When dealing with signed distance fields, the normal at a surface point is equivalent to the value of the gradient of the signed distance function at that point [5].

$$n = \nabla SDF \qquad N = \frac{n}{(n \cdot n)^{\frac{1}{2}}} \tag{7}$$

3.2 Discrete Signed Distance Fields

3.2.1 Storing and evaluating the SDF

Since it is difficult to compute a specific continuous signed distance function describing the current scene, a discrete function can be used. This function can be seen as a discrete and uniform 3D grid over the cartesian coordinate system of the scene. Each grid vertex in the field stores the closest signed distance of the vertex to the surface of the scene.

The function, however, needs to be evaluated for any point in the continuous space, not only on points stored in the 3D field. In order to get the signed distance at an arbitrary point P, an interpolation is performed between the stored signed distance values of the function in the neighborhood of point P. The most common way is to use trilinear interpolation by linearly interpolating between the 8 vertices in the neighborhood of P [1].

3.2.2 Field Interpolation

A more general approach is to model the interpolation and determination of the signed distance at an arbitrary point P by applying an 1D filter kernel f over the 3D field neighborhood of P.

$$f(v_1, \ldots, v_n, t) \tag{8}$$

This filter function with width n and takes n values and interpolates among them based on the filter weight $t \in [0, 1]$. A linear filter kernel for example would look like

$$f(v_1, v_2, t) = (1 - t)v_1 + tv_2. \tag{9}$$

In order to filter with a 1D kernel in the 3D field, a neighborhood of n^3 vertices around P is selected, resulting in a cube of vertices with size n. Using this

area, the 1D kernel is applied n^2 times over all vertices in x direction. Over this resulting square of values, the kernel is applied n times over the vertices in y direction resulting in n values. Over these final values in z direction the kernel is applied once which gives the final interpolation value. Formulating as function, we get

$$v(V,t) = f\left(\begin{array}{c} f\left(\begin{array}{c} f(v_{111},\ldots,v_{n11},t_x), \\ \vdots \\ f(v_{1n1},\ldots,v_{nn1},t_x), \\ t_y \end{array}\right), \\ \vdots \\ f\left(\begin{array}{c} f(v_{11n},\ldots,v_{n1n},t_x), \\ \vdots \\ f(v_{1nn},\ldots,v_{nnn},t_x), \\ t_y \end{array}\right), \\ t_z \end{array}\right). \tag{10}$$

$v_{xyz} \in V$ denotes the discretely stored signed distance value at index $\{x, y, z\}$ in the neighborhood interpolation area V of P. Figure 2 shows P inside the filter area of 8 vertices which is, for example, used for linear interpolation.

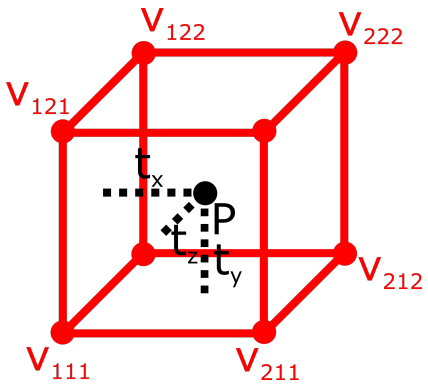


Figure 2: Interpolation to determine value at P. v_{111} is the lower vertex and v_{222} is the upper vertex of the filter area consisting of a total of 8 vertices. t is the filter weight which is calculated based on the distance to the lower vertex.

3.3 Building the discrete Signed Distance Function

3.3.1 Definition of the discrete Signed Distance Function

Considering the general definition of signed distance functions (5) and interpolation of discrete grid values (10), a concrete signed distance function can now be build using these two components.

First of all, it is assumed that the signed distance function takes positions P in world space as input. To determine the neighborhood of P inside the discrete SDF grid and to calculate the filter weight t, this position must first be transfered to absolute grid coordinates P_{grid} (Figure 3). Assuming that the SDF grid in the world coordinate system is centered at V_{origin} with an extent of V_{scale} in all 6 axis directions and a grid resolution $V_{\text{resolution}}$, then the absolute grid position can be calculated as

$$P_{\text{grid}} = V_{\text{resolution}} \left(\frac{(P - V_{\text{origin}})}{2V_{\text{scale}}} + \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}^T \right). \tag{11}$$

The resulting P_{grid} ranges between $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ and $V_{\text{resolution}} - \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$. A cell in absolute grid space has width 1. All vertices of the grid storing the SDF values are located on the integer grid positions. To now determine arbitrary signed distance values, interpolation is performed using the appropriate stored neighbor grid values. The vertex values are weighted according to the position of the value to lookup inside the neighborhood.

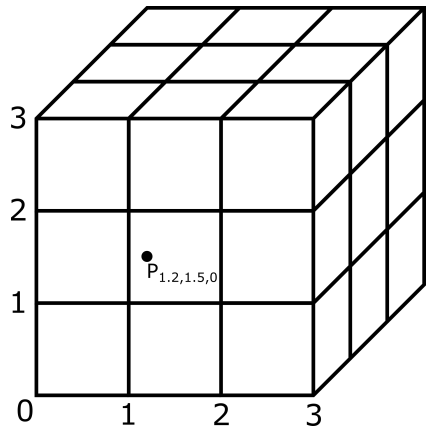


Figure 3: P in absolute grid coordinates.

Thus, a three dimensional filter weight t (Figure 2) is calculated by taking

the distance between P and its nearest lower integer vertex

$$t = P_{grid} - \lfloor P_{grid} \rfloor \tag{12}$$

t ranges between $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ and $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$. This weight is then used for the interpolation function (10) to formulate the final discrete signed distance function

$$SDF(P) = v\left(N_{fP}, t\left(P_{\text{grid}}\left(P\right)\right)\right). \tag{13}$$

N_{fP} here is the set of values in the neighborhood of P which the filter kernel f interpolates on.

3.3.2 Definition of the surface normal

In order to define a normal on the surface of the discrete signed distance function, the gradient has to be determined ($\odot$ stands for the component-wise product of two vectors)

$$n = \nabla SDF = \begin{pmatrix} \frac{\partial SDF}{\partial P_x} \\ \frac{\partial SDF}{\partial P_y} \\ \frac{\partial SDF}{\partial P_z} \end{pmatrix} = \begin{pmatrix} \frac{\partial v}{\partial t_x} \\ \frac{\partial v}{\partial t_y} \\ \frac{\partial v}{\partial t_z} \end{pmatrix} \odot \begin{pmatrix} \frac{\partial t_x}{\partial P_x} \\ \frac{\partial t_y}{\partial P_y} \\ \frac{\partial t_z}{\partial P_z} \end{pmatrix} \tag{14}$$

which results in the component-wise product of the interpolation and the derivate of t for the unnormalized surface normal

$$n = \nabla v \odot \frac{\partial t}{\partial P}. \tag{15}$$

Since the $floor$ function occurs in the definition of t (12), the derivate t' has singularities at all integer grid points $P_{\text{grid}} \in \mathbb{Z}$. Assuming the SDF grid is axis aligned in the world coordinate system, the differentiation yields

$$\frac{\partial t}{\partial P} = \frac{V_{\text{resolution}}}{2V_{\text{scale}}} \qquad P_{\text{grid}} \notin \mathbb{Z}. \tag{16}$$

However, this derivative is constant and all singularities are removable using its constant value. Thus the singularities can be removed by setting

$$\frac{\partial t}{\partial P} := \frac{V_{\text{resolution}}}{2V_{\text{scale}}} \qquad P_{\text{grid}} \in \mathbb{Z}. \tag{17}$$

4 Ray Differentials with Signed Distance Fields

4.1 Calculating the Normal Differential

In order to compute the ray differentials, the differential of the surface normal $\frac{\partial N}{\partial x}$ has to be calculated to compute the derivate of the ray reflection and refraction (Table 1). The differential in x-direction can be computed as

$$\begin{aligned}
\frac{\partial n}{\partial x} = \frac{\partial \left(\nabla SDF\right)}{\partial x} &= \begin{pmatrix} \frac{\partial^2 SDF}{\partial P_x \partial x} \\ \frac{\partial^2 SDF}{\partial P_y \partial x} \\ \frac{\partial^2 SDF}{\partial P_z \partial x} \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial^2 v}{\partial t_x^2}\left(\frac{\partial t_x}{\partial P_x}\frac{\partial P_x}{\partial x}\right)\frac{\partial t_x}{\partial P_x} + \frac{\partial^2 v}{\partial t_x \partial t_y}\left(\frac{\partial t_y}{\partial P_y}\frac{\partial P_y}{\partial x}\right)\frac{\partial t_x}{\partial P_x} + \frac{\partial^2 v}{\partial t_x \partial t_z}\left(\frac{\partial t_z}{\partial P_z}\frac{\partial P_z}{\partial x}\right)\frac{\partial t_x}{\partial P_x} \\ \frac{\partial^2 v}{\partial t_y \partial t_x}\left(\frac{\partial t_x}{\partial P_x}\frac{\partial P_x}{\partial x}\right)\frac{\partial t_y}{\partial P_y} + \frac{\partial^2 v}{\partial t_y^2}\left(\frac{\partial t_y}{\partial P_y}\frac{\partial P_y}{\partial x}\right)\frac{\partial t_y}{\partial P_y} + \frac{\partial^2 v}{\partial t_y \partial t_z}\left(\frac{\partial t_z}{\partial P_z}\frac{\partial P_z}{\partial x}\right)\frac{\partial t_y}{\partial P_y} \\ \frac{\partial^2 v}{\partial t_z \partial t_x}\left(\frac{\partial t_x}{\partial P_x}\frac{\partial P_x}{\partial x}\right)\frac{\partial t_z}{\partial P_z} + \frac{\partial^2 v}{\partial t_z \partial t_y}\left(\frac{\partial t_y}{\partial P_y}\frac{\partial P_y}{\partial x}\right)\frac{\partial t_z}{\partial P_z} + \frac{\partial^2 v}{\partial t_z^2}\left(\frac{\partial t_z}{\partial P_z}\frac{\partial P_z}{\partial x}\right)\frac{\partial t_z}{\partial P_z} \end{pmatrix} \\
&= \left(\begin{pmatrix} \frac{\partial^2 v}{\partial t_x^2} & \frac{\partial^2 v}{\partial t_x \partial t_y} & \frac{\partial^2 v}{\partial t_x \partial t_z} \\ \frac{\partial^2 v}{\partial t_y \partial t_x} & \frac{\partial^2 v}{\partial t_y^2} & \frac{\partial^2 v}{\partial t_y \partial t_z} \\ \frac{\partial^2 v}{\partial t_z \partial t_x} & \frac{\partial^2 v}{\partial t_z \partial t_y} & \frac{\partial^2 v}{\partial t_z^2} \end{pmatrix} \left(\frac{\partial t}{\partial P} \odot \frac{\partial P}{\partial x}\right) \right) \odot \frac{\partial t}{\partial P}
\end{aligned} \tag{18}$$

which results in the following term for the normal differential in x-direction:

$$\frac{\partial n}{\partial x} = \left(\text{Hess}(v)\left(\frac{\partial t}{\partial P} \odot \frac{\partial P}{\partial x}\right)\right) \odot \frac{\partial t}{\partial P} \tag{19}$$

The final step is derive the vector normalization (7) to get the derivate for the normalized normal with length 1:

$$\frac{\partial N}{\partial x} = \frac{(n \cdot n)\frac{\partial n}{\partial x} - (n \cdot \frac{\partial n}{\partial x}) \cdot n}{(n \cdot n)^{\frac{3}{2}}} \tag{20}$$

4.2 Choosing the interpolation kernel

For simplicity and performance reason, linear interpolation is usually used in computer graphics (9). However, the derivate of the linear kernel is constant and yields flat normals when calculating the gradient (15) which can

be a problem for realizing curved shapes (Figure 4). All values in the same neighborhood, enclosed by the same 8 neighborhood values, are seen as lying on the same flat plane whereby hard edges arise at the border of each neighborhood region . Therefore, choosing a cubic interpolation instead is a reasonable approach. A common cubic interpolation technique are Catmull-Rom splines [6][7]. The filtering can be expressed as:

$$f(v_1, v_2, v_3, v_4, t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} \quad (21)$$

τ is called "tension" and controls how strong the curve bends at the interpolation points. It is often chosen as $\frac{1}{2}$. This value will also be used in this report.

This filter has a width of 4 and operates on a total of 64 values in 3D space. The interpolation curve always passes through all input values and produces a soft curve among all interpolation data. It yields v_2 for $t = 0$ and v_3 for $t = 1$. It can simply be plugged into the 3D kernel interpolation (10) and act as tricubic interpolation between grid values.

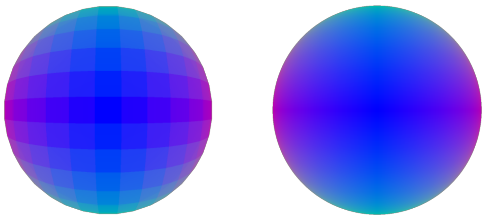


Figure 4: Linear and cubic filter. Image shows a sphere with a grid resolution of 16, filtered with a linear filter (left) and a cubic filter (right). Sphere is colored by normals.

4.3 Texture filtering

4.3.1 Texture Coordinate Differentials

A practical application of ray differentials is texture filtering. To map textures to the signed distance grid, we store additionally a pair of UV coordinates at each grid vertex and interpolate among them similarly to SDF

values. A similar expression to (13) can be derived for the texture coordinates by taking the UV coordinates instead of the SDF values:

$$T(P) = \begin{pmatrix} v_u\left(N_{fP}, t\left(P_{\text{grid}}\left(P\right)\right)\right) & v_v\left(N_{fP}, t\left(P_{\text{grid}}\left(P\right)\right)\right) \end{pmatrix}^T. \tag{22}$$

Now this equation can be derived similarly to get the change of the texture coordinate when shifting the ray on the image plane for the x-direction:

$$\frac{\partial T}{\partial x} = \begin{pmatrix} \left(\nabla v_u \odot \frac{\partial t}{\partial P}\right) \cdot \frac{\partial P}{\partial x} & \left(\nabla v_v \odot \frac{\partial t}{\partial P}\right) \cdot \frac{\partial P}{\partial x} \end{pmatrix}^T \tag{23}$$

This can now be used to estimate the projection of the pixel onto the texture, the ray footprint, since this describes the change of the texture coordinate when shifting the ray on the image plane in x- and y-direction [3]. By multiplying the differentials by the texel pixel ratio, an estimate about the extent of the footprint in texture space can be calculated

$$\begin{aligned} \Delta T_x &\approx \frac{\partial T}{\partial x} \odot \text{texel/pixel} \\ \Delta T_y &\approx \frac{\partial T}{\partial y} \odot \text{texel/pixel} \\ T_{\text{size}} &= \begin{pmatrix} \left(\Delta T_x \cdot \Delta T_x\right)^{\frac{1}{2}} & \left(\Delta T_y \cdot \Delta T_y\right)^{\frac{1}{2}} \end{pmatrix} \end{aligned} \tag{24}$$

The texture should now be filtered in the footprint region. A common approach for texture filtering is mipmapping [8] where the texture gets stored in progressively smaller resolutions. According to an level of detail, a certain texture resolution is then chosen where the texture is finally sampled. The level of detail should be selected regarding the extent of the footprint inside the texture. An algorithm for choosing a suitable level of detail is to take the logarithm of the longer side of the footprint extent.

$$\text{LoD} = \log_2\left(\max\left(T_{\text{size}_x}, T_{\text{size}_y}\right)\right) \tag{25}$$

If the LoD lies between two detail representations, linear interpolation is performed between the two textures.

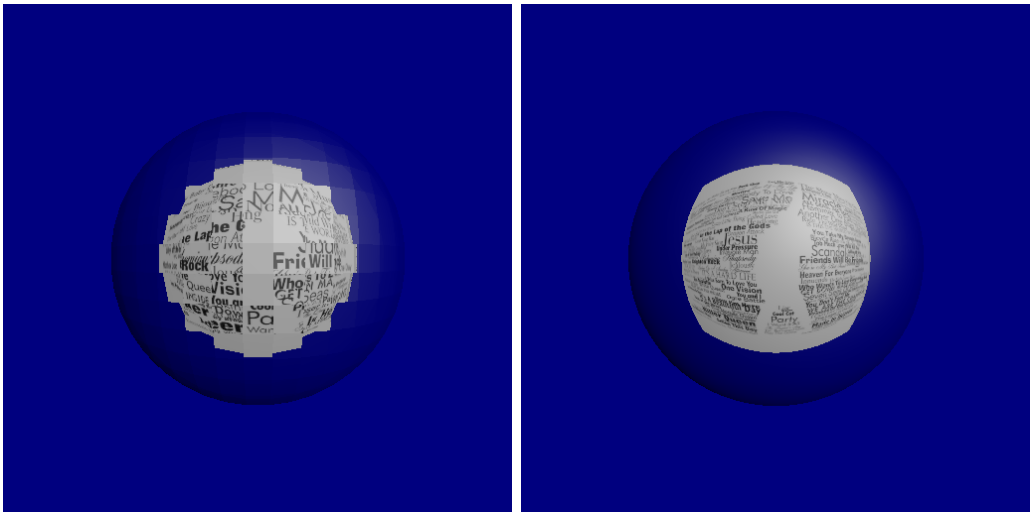
4.3.2 Results

To test the texture filtering, a scene consisting of a reflective sphere and a paper sheet containing text is used. The reflective sphere is placed directly in

front of the camera while the paper is placed behind it and not directly visible by the camera, only in the reflection on the sphere. The whole scene is stored inside a signed distance grid with a resolution of either 64 or 512 in all three dimensions. The scene is rendered using raytracing with a screen resolution of 512x512. The gradient and hessian matrix for the normal differential are calculated using automatic differentiation [2]. The effect of mipmapping will be tested as well as the influence of the grid dimension and the interpolation type linear and cubic as presented above.

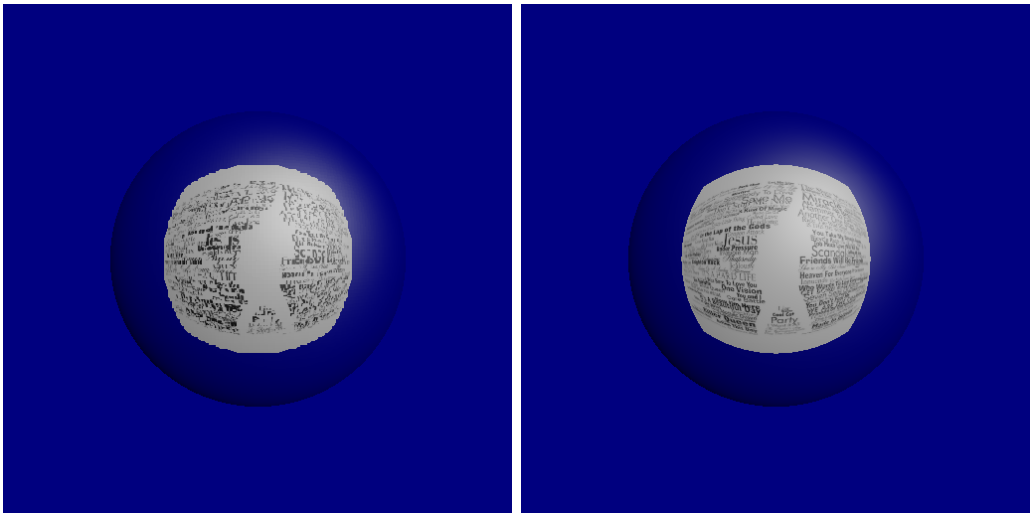
Figure 5 shows the render result of the scene when no mipmapping is applied. The paper reflection in the sphere mirror shows clear aliasing since the footprint of the ray is not considered, only the intersection point of the ray is used. The next test shows the same scene, but this time the mipmapping technique (25) is used with linear and cubic interpolation at a grid resolution of 64 (Figure 7). Both images show a clear improvement considering texture aliasing although the cubic interpolation here yields superior results. The linear interpolation produces visible flat surfaces on the sphere. Another rendering is performed at the higher grid resolution of 512 (Figure 6). The higher resolution makes the sphere in the linear interpolation test seem more smooth. However, the reflection in the linear result is distorted and not smooth. The cubic result is better and similar to the lower resolution cubic result. The last image (Figure 8) shows the cubic interpolation with 2x supersampling which produces a small improvement by additional smoothing of the texture.

Figure 5: Reflective sphere scene without mipmapping, 512^3 grid dimension, cubic interpolation.



(a) Linear Interpolation. (b) Cubic Interpolation.

Figure 6: Reflective sphere scene with mipmapping, 64^3 grid dimension, linear vs cubic interpolation comparison.



(a) Linear Interpolation. (b) Cubic Interpolation.

Figure 7: Reflective sphere scene with mipmapping, 512^3 grid dimension, linear vs cubic interpolation comparison.

Figure 8: Reflective sphere scene with mipmapping, 512^3 grid dimension, cubic interpolation, 2x SSAA.

5 Conclusion

Summarizing, a general definition of discrete signed distance fields can be formulated to successfully make use ray differentials to estimate the footprint of a ray in the scenes. Arbitrary interpolation kernels can be used for the signed distance function although higher order kernels are recommendable to avoid blocky looking curved shapes. The use of higher order kernels may also produce good results with a low grid resolution which lowers the memory consumption. Using ray differentials for estimating the footprint on a texture is an effective way to reduce aliasing when mapping textures to signed distance fields. Another practical use of ray differentials with signed distance functions may be choosing LoD levels of the voxel grid itself to filter among different detail representations of the scene or certain objects.

References

[1] F. Calakli and G. Taubin. "SSD: Smooth Signed Distance Surface Reconstruction". In: *Computer Graphics Forum* 30.7 (), pp. 1993–2002.

[2] Robert Mansel Gower and Margarida P Mello. *Hessian matrices via automatic differentiation*. Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica, 2010.

[3] Homan Igehy. "Tracing Ray Differentials". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 179–186.

[4] M. W. Jones, J. A. Baerentzen, and M. Sramek. "3D distance fields: a survey of techniques and applications". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (July 2006), pp. 581–599.

[5] Stanley Osher and Ronald Fedkiw. "Implicit Functions". In: *Level Set Methods and Dynamic Implicit Surfaces*. New York, NY: Springer New York, 2003, pp. 3–16.

[6] Christopher Twigg. "Catmull-rom splines". In: *Computer* 41.6 (2003), pp. 4–6.

[7] Stephen Weston. "An introduction to the mathematics and construction of splines". In: *Addix software consultancy limited, Version* 1 (2002).

[8] Lance Williams. “Pyramidal Parametrics”. In: *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’83. Detroit, Michigan, USA: ACM, 1983, pp. 1–11.