Tracing Ray Differentials

Homan Igehy

Computer Science Department

Stanford University

Abstract

Antialiasing of ray traced images is typically performed by super-sampling the image plane. While this type of filtering works well for many algorithms, it is much more efficient to perform filtering locally on a surface for algorithms such as texture mapping. In order to perform this type of filtering, one must not only trace the ray passing through the pixel, but also have some approximation of the distance to neighboring rays hitting the surface (i.e., a ray's footprint). In this paper, we present a fast, simple, robust scheme for tracking such a quantity based on *ray differentials*, derivatives of the ray with respect to the image plane.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – color, shading, shadowing, and texture; raytracing.

1 INTRODUCTION

Ray tracing [18] is an image generation technique that is able to accurately model many phenomena which are difficult or impossible to produce with a traditional graphics pipeline. As with any image synthesis algorithm, ray tracing is prone to aliasing, and antialiasing is typically performed by tracing rays at multiple sub-pixel offsets on the image plane (e.g., [12, 18]). By stochastically point sampling each of many variables per ray [6], one may filter over multiple phenomena simultaneously. For some algorithms, however, it is much more efficient to filter over a more local domain. For example, in texture mapping, a texture will often be viewed under high minification so that the entire texture falls under a small number of pixels. Filtering such a texture by taking bilinear samples for a set of stochastic rays requires the tracing of a correspondingly large number of rays. This is problematic because the cost of tracing a ray is relatively high, and the minification factor can be arbitrarily high. On the other hand, if we knew the distance between a ray and the rays for a neighboring pixel when the ray hit the texture map (i.e., a ray's footprint), then we could efficiently filter the texture by using a fast, memory coherent algorithm such as mip mapping [19]. Tracing such a quantity in a polygon rendering pipeline is straightforward because primitives are drawn in raster order, and the transformation between texture space and image space is described by a linear projective map [10]. In a ray tracer, however, the primitives are accessed according to a pixel's ray tree, and the transformation between texture space and image space is non-linear (reflection and refraction can make rays converge, diverge, or both), making the problem substantially more difficult.

Tracing a ray's footprint is also important for algorithms other than texture mapping. A few of these algorithms are listed below:

- Geometric level-of-detail allows a system to simultaneously antialias the geometric detail of an object, bound the cost of tracing a ray against an object, and provide memory coherent access to the data of an object. However, to pick a level-of-detail, one must know an approximation of the ray density when a ray is intersected against an object.
- When ray tracing caustics, the intensity attributed to a sampled light ray depends upon the convergence or divergence of the wavefront [13]. Similarly, in illumination mapping, the flux carried by a ray from a light source must be deposited over an area on a surface's illumination map based on the density of rays [5].
- Dull reflections may be modeled by filtering textures over a kernel which extends beyond the ray's actual footprint [2].

In this paper, we introduce a novel approach for quickly and robustly tracking an approximation to a ray's footprint based on *ray differentials*. Ray tracing can be viewed as the evaluation of the position and direction of a ray as the ray propagates through the scene. Because a ray is initially parameterized in terms of image space coordinates, in addition to tracing the ray, we can also trace the value of its derivatives with respect to the image plane. A first-order Taylor approximation based on these derivatives gives an estimate of a ray's footprint. A few techniques for estimating ray density have been presented in the literature [2, 5, 13, 16], but ray differentials are faster, simpler, and more robust than these algorithms. In particular, because ray differentials are based on elementary differential calculus rather than on differential geometry or another mathematical foundation that has an understanding of the 3D world, the technique may be readily applied to non-physical phenomena such as bump mapping and normal-interpolated triangles. General formulae for tracing ray differentials are derived for transfer, reflection, and refraction, and specific formulae are derived for handling normal-interpolated triangles. Finally, we demonstrate the utility of ray differentials in performing texture filtering.

2 RELATED WORK

Several algorithms have been developed for estimating a texture filtering kernel in polygon rendering pipelines (e.g., [1, 10]). Similar algorithms have been used in ray tracers that calculate the projection of texture coordinates onto the image plane [8], but

homan@graphics.stanford.edu

such a projection is only valid for eye rays. This technique can be extended to reflected and refracted rays by computing the total distance traveled, but such an approximation is invalid because curved surfaces can greatly modify the convergence or divergence of rays. A few algorithms have been developed that take this into account, and we will review them briefly.

Within the context of a ray tracer, finite differencing has been used to calculate the extent over which a light ray's illuminance is deposited on an illumination map by examining the illumination map coordinates of neighboring rays [5]. The main advantage of finite differencing is that it can easily handle any kind of surface. The main disadvantages, however, are the difficult problems associated with robustly handling neighboring eye rays whose ray trees differ. The algorithm must detect when a neighboring ray does not hit the same primitives and handle this case specially. One plausible method of handling such a case is to send out a special neighboring ray that follows the same ray tree and intersects the plane of the same triangles beyond the triangle edges, but this will not work for spheres and other higher-order primitives. Additionally, this special case becomes the common case as neighboring rays intersect different primitives that are really part of the same object (e.g., a triangle mesh). Our algorithm circumvents these difficulties by utilizing the differential quantities of a single ray independently of its neighbors.

Cone tracing [2] is a method in which a conical approximation to a ray's footprint is traced for each ray. This cone is used for edge antialiasing in addition to texture filtering. The main problem with a conical approximation is that a cone is isotropic. Not only does this mean that pixels must be sampled isotropically on the image plane, but when the ray footprint becomes anisotropic after reflection or refraction, it must be re-approximated with an isotropic footprint. Thus, this method cannot be used for algorithms such as anisotropic texture filtering. In addition, extending the technique to support surfaces other than planar polygons and spheres in not straightforward.

Wavefront tracing [9] is a method in which the properties of a differentially small area of a ray's wavefront surface is tracked, and it has been used to calculate caustic intensities resulting from illumination off of curved surfaces [13]. Wavefronts have also been used to calculate the focusing characteristics of the human eye [11]. Although highly interrelated, the main difference between wavefronts and our method of ray differentials is that our method tracks the directional properties of a differentially small distance while wavefront tracing tracks the surface properties of a differentially small area. Thus, wavefronts cannot handle anisotropic spacing between pixel samples. Additionally, because a differential area is an inherently more complex quantity, the computational steps associated with wavefront tracing are more complicated. Wavefront tracing is based on differential geometry while our algorithm is based on elementary differential calculus, and a technical consequence of this is that we can readily handle non-physically based phenomena such as normal-interpolated triangles, bump mapped surfaces, and other algorithms that are self-contradicting in the framework of differential geometry. A practical consequence of being based on the simpler field of elementary differential calculus is that our formulation is easier to understand, and extending the technique to handle different phenomena is straightforward.

Paraxial ray theory [4] is an approximation technique originally developed for lens design, and its application to ray tracing is known as pencil tracing [16]. In pencil tracing, paraxial rays to an axial ray are parameterized by point-vector pairs on a plane perpendicular to the axial ray. The propagation of these paraxial rays is approximated linearly by a system matrix. As with wavefront tracing, the computational complexity of pencil tracing is significantly higher than our method. Additionally, pencil tracing makes a distinct set of simplifying assumptions to make each phenomenon linear with respect to the system matrix. An approximation is made even on transfer, a phenomenon that is linear with respect to a ray. Furthermore, the handling of non-physical phenomena is unclear. By comparison, the single unified approximation we make is simply that of a first-order Taylor approximation to a ray function.

3 TRACING RAY DIFFERENTIALS

One way to view ray tracing is as the evaluation of the position and direction of a ray function at discrete points as it propagates through the scene. Any value v that is computed for the ray (e.g., luminance, texture coordinate on a surface, etc.) is derived by applying a series of functions to some initial set of parameters, typically the image space coordinates x and y:

$$v = \mathrm{f}_n\left(\mathrm{f}_{n-1}\left(\ldots \mathrm{f}_2\left(\mathrm{f}_1(x, y)\right)\right)\right) \tag{1}$$

We can compute the derivative of this value with respect to an image space coordinate (e.g., x) by applying the Chain Rule:

$$\frac{\partial v}{\partial x} = \frac{\partial \mathrm{f}_n}{\partial \mathrm{f}_{n-1}} \cdots \frac{\partial \mathrm{f}_2}{\partial \mathrm{f}_1} \frac{\partial \mathrm{f}_1}{\partial x} \tag{2}$$

As transformations are applied to a ray to model propagation through a scene, we are just applying a set of functions f_i to keep track of the ray. We can also keep track of *ray differentials*, derivatives of the ray with respect to image space coordinates, by applying the derivatives of the functions. These ray differentials can then be used to give a first-order Taylor approximation to the ray as a function of image space coordinates. In the forthcoming derivations, we express scalars in italics and points, directions, and planes with homogeneous coordinates in bold.

A ray can be parameterized by a point representing a position on the ray and unit vector representing its direction:

$$\vec{\mathbf{R}} = \left\langle \mathbf{P} \quad \mathbf{D} \right\rangle \tag{3}$$

The initial values for a ray depend on the parameterization of the image plane: a pinhole camera is described by an eye point, a viewing direction, a right direction, and an up direction. The direction of a ray going through a pixel on the image plane can be expressed as a linear combination of these directions:

$$\mathbf{d}(x, y) = \mathbf{View} + x\mathbf{Right} + y\mathbf{Up} \tag{4}$$

Thus, the eye ray is given by:

$$\begin{aligned} \mathbf{P}(x, y) &= \mathbf{Eye} \\ \mathbf{D}(x, y) &= \frac{\mathbf{d}}{(\mathbf{d} \cdot \mathbf{d})^{1/2}} \end{aligned} \tag{5}$$

We can now ask the question, given a ray, if we were to pick a ray slightly above or to the right of it on the image plane, what ray would result? Each of these differentially offset rays can be represented by a pair of directions we call *ray differentials*:

$$\begin{aligned} \frac{\partial \vec{\mathbf{R}}}{\partial x} &= \left\langle \frac{\partial \mathbf{P}}{\partial x} \quad \frac{\partial \mathbf{D}}{\partial x} \right\rangle \\ \frac{\partial \vec{\mathbf{R}}}{\partial y} &= \left\langle \frac{\partial \mathbf{P}}{\partial y} \quad \frac{\partial \mathbf{D}}{\partial y} \right\rangle \end{aligned} \tag{6}$$

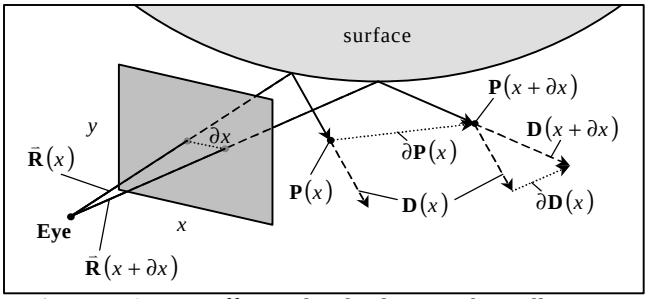

Figure 1: A Ray Differential. The diagram above illustrates the positions and directions of a ray and a differentially offset ray after a reflection. The difference between these positions and directions represents a ray differential.

A ray differential is illustrated in Figure 1. If we evaluate the ray differentials as a ray propagates in addition to the position and direction of the ray, then the distance between neighboring rays (and hence the ray's footprint) can be estimated with a first-order differential approximation:

$$\begin{aligned}\left[\vec{\mathbf{R}}(x+\Delta x, y)-\vec{\mathbf{R}}(x, y)\right] &\approx \Delta x \frac{\partial \vec{\mathbf{R}}(x,y)}{\partial x} \\ \left[\vec{\mathbf{R}}(x, y+\Delta y)-\vec{\mathbf{R}}(x, y)\right] &\approx \Delta y \frac{\partial \vec{\mathbf{R}}(x,y)}{\partial y}\end{aligned} \tag{7}$$

We can compute the initial value of the ray differential in the ***x*** direction by differentiating (5) with respect to ***x***:

$$\begin{aligned}\frac{\partial \mathbf{P}}{\partial x} &= \mathbf{0} \\ \frac{\partial \mathbf{D}}{\partial x} &= \frac{(\mathbf{d}\cdot\mathbf{d})\ \mathbf{Right}-(\mathbf{d}\cdot\mathbf{Right})\ \mathbf{d}}{(\mathbf{d}\cdot\mathbf{d})^{3/2}}\end{aligned} \tag{8}$$

A similar expression can be derived for the *y* direction. Although we only track first-order derivatives, higher-order derivatives could be computed as well for a better approximation or for error bounding. However, we have found that discontinuities limit the effectiveness of higher-order approximations and that a first-order approximation is sufficient in practice.

3.1 Propagation

Given an expression for the propagation of a ray for any phenomenon (i.e., how a phenomenon affects the ray's value), then we can find the expression for the propagation of a ray differential by simply differentiating the expression. Here, we will derive the formulae for the three common ray tracing operations: transfer, reflection, and refraction. We will express our formulae as derivatives with respect to *x* without any loss in generality.

3.1.1 Transfer

Transfer is the simple operation of propagating a ray through a homogenous medium to the point of intersection with a surface. The equation for transfer onto a surface at a distance *t* is given by:

$$\begin{aligned}\mathbf{P}' &= \mathbf{P} + t\mathbf{D} \\ \mathbf{D}' &= \mathbf{D}\end{aligned} \tag{9}$$

For a ray differential, we differentiate (9) to get:

$$\begin{aligned}\frac{\partial \mathbf{P}'}{\partial x} &= \left(\frac{\partial \mathbf{P}}{\partial x} + t\frac{\partial \mathbf{D}}{\partial x}\right) + \frac{\partial t}{\partial x}\mathbf{D} \\ \frac{\partial \mathbf{D}'}{\partial x} &= \frac{\partial \mathbf{D}}{\partial x}\end{aligned} \tag{10}$$

For a planar surface **N** (defined as the locus of points **P′** such that **P′** • **N** = 0), *t* is given by:

$$t = -\frac{\mathbf{P}\cdot\mathbf{N}}{\mathbf{D}\cdot\mathbf{N}} \tag{11}$$

Differentiating this and re-expressing it in terms of *t*, we get:

$$\frac{\partial t}{\partial x} = -\frac{\left(\frac{\partial \mathbf{P}}{\partial x}+t\frac{\partial \mathbf{D}}{\partial x}\right)\cdot\mathbf{N}}{\mathbf{D}\cdot\mathbf{N}} \tag{12}$$

Note that the fourth component of **N** is irrelevant in this equation (its dot product is taken with directions only), and it can thus be viewed as the normal of the surface. Equations (10) and (12) actually have a geometric interpretation: the first two terms of the first equation in (10) express the fact that as a ray travels through homogeneous space, the positional offset of a differentially offset ray changes according to its directional offset and the distance traveled. Then, the third term orthographically projects this positional offset in the direction of the ray onto the plane.

Although a formal proof is beyond the scope of this paper, (12) is not only valid for a plane, but is also correct for an arbitrary surface. In the case of an arbitrary surface, **N** is just the normal of the surface at the point of intersection. The intuition behind this is that a surface has a fixed shape and curvature at an intersection point. As we intersect an offset ray against this surface by smaller and smaller offsets, the surface will look more and more like the tangent plane at the intersection point. In the limit, a differentially offset ray intersects the surface in the tangent plane of the intersection point.

3.1.2 Reflection

Given a ray that has been transferred onto a surface by (9), the equation for a reflection ray [7] is given by:

$$\begin{aligned}\mathbf{P}' &= \mathbf{P} \\ \mathbf{D}' &= \mathbf{D} - 2(\mathbf{D}\cdot\mathbf{N})\mathbf{N}\end{aligned} \tag{13}$$

For a ray differential, reflection is given by:

$$\begin{aligned}\frac{\partial \mathbf{P}'}{\partial x} &= \frac{\partial \mathbf{P}}{\partial x} \\ \frac{\partial \mathbf{D}'}{\partial x} &= \frac{\partial \mathbf{D}}{\partial x} - 2\left[(\mathbf{D}\cdot\mathbf{N})\frac{\partial \mathbf{N}}{\partial x} + \frac{\partial(\mathbf{D}\cdot\mathbf{N})}{\partial x}\mathbf{N}\right]\end{aligned} \tag{14}$$

where:

$$\frac{\partial(\mathbf{D}\cdot\mathbf{N})}{\partial x} = \frac{\partial \mathbf{D}}{\partial x}\cdot\mathbf{N} + \mathbf{D}\cdot\frac{\partial \mathbf{N}}{\partial x} \tag{15}$$

This equation requires the evaluation of the derivative of the normal at the point of intersection, a topic that will be addressed in Sections 3.2 and 3.3.

3.1.3 Refraction

Once a ray has been transferred onto a surface, the equation for a refracted ray [7] can be expressed as:

$$\begin{aligned}\mathbf{P}' &= \mathbf{P} \\ \mathbf{D}' &= \eta\mathbf{D} - \mu\mathbf{N}\end{aligned} \tag{16}$$

where we use the shorthand notation:

$$\begin{aligned}\mu &= \left[\eta(\mathbf{D}\cdot\mathbf{N}) - (\mathbf{D}'\cdot\mathbf{N})\right] \\ \mathbf{D}'\cdot\mathbf{N} &= -\sqrt{1-\eta^2\left[1-(\mathbf{D}\cdot\mathbf{N})^2\right]}\end{aligned} \tag{17}$$

and η is the ratio of the incident index of refraction to the transmitted index of refraction. Differentiating, we get:

$$\frac{\partial \mathbf{P}'}{\partial x} = \frac{\partial \mathbf{P}}{\partial x}$$
$$\frac{\partial \mathbf{D}'}{\partial x} = \eta \frac{\partial \mathbf{D}}{\partial x} - \left(\mu \frac{\partial \mathbf{N}}{\partial x} + \frac{\partial \mu}{\partial x} \mathbf{N} \right) \quad (18)$$

where (referring to (15) from Section 3.1.2):

$$\frac{\partial \mu}{\partial x} = \left[\eta - \frac{\eta^2 (\mathbf{D} \cdot \mathbf{N})}{(\mathbf{D}' \cdot \mathbf{N})} \right] \frac{\partial (\mathbf{D} \cdot \mathbf{N})}{\partial x} \quad (19)$$

3.2 Surface Normals

The formulae derived for reflection and refraction of ray differentials in Sections 3.1.2 and 3.1.3 depend on the derivative of the unit normal with respect to *x*. In differential geometry [17], the *shape operator* (S) is defined as the negative derivative of a unit normal with respect to a direction tangent to the surface. This operator completely describes a differentially small area on a surface. For our computation, the tangent direction of interest is given by the derivative of the ray's intersection point, and thus:

$$\frac{\partial \mathbf{N}}{\partial x} = -\mathrm{S}\left(\frac{\partial \mathbf{P}}{\partial x} \right) \quad (20)$$

For a planar surface, the shape operator is just zero. For a sphere, the shape operator given a unit tangent vector is the tangent vector scaled by the inverse of the sphere's radius. Formulae for the shape operator of both parametric and implicit surfaces may be found in texts on differential geometry (e.g., [17]) and other sources [13], and thus will not be covered here in further detail.

3.3 Discussion

One interesting consequence of casting ray differentials in the framework of elementary differential calculus is that we may forgo any understanding of surfaces and differential geometry, even in expressing the derivative of a unit normal. For that matter, we may forgo the geometric interpretation of any calculation, such as the interpretation made for equations (10) and (12). For example, if we know that the equation of a unit normal to a sphere with origin **O** and radius *r* at a point **P** on the sphere is given by:

$$\mathbf{N} = (\mathbf{P} - \mathbf{O}) / r \quad (21)$$

then we may "blindly" differentiate with respect to *x* to get:

$$\frac{\partial \mathbf{N}}{\partial x} = \frac{\partial \mathbf{P}}{\partial x} \Big/ r \quad (22)$$

This differentiation may be performed on any surface or for any phenomenon. If we know the formula for how a ray is affected by a phenomenon, then we can differentiate it to get a formula for how a ray differential is affected without any understanding of the phenomenon. For example, if we apply an affine transformation to a ray so that we may perform intersections in object space coordinates, then the ray differential is transformed according to the derivative of the affine transformation. If we apply an ad hoc non-linear warp to rays in order to simulate a fish-eye lens effect, then we can derive an expression for warping the ray differentials by differentiating the warping function. This is a large advantage of being based on elementary differential calculus rather than on a physically-based mathematical framework.

In graphics, we often use non-physical surfaces that separate the geometric normal from the shading normal, such as normal-interpolated triangles or bump mapped surfaces. For such surfaces, the use of ray differentials is straightforward. In the case of transfer to the point of intersection, the geometric normal is used for (12) because a neighboring ray would intersect the surface according to the shape defined by the geometric normal. For reflection and refraction, however, the shading normal is used because neighboring rays would be reflected and refracted according to the shading normal. Because of its common use, we derive an expression for the derivative of the shading normal for a normal-interpolated triangle in the box below.

The computational cost of tracing ray differentials is very small relative to the other costs of a ray tracer. Each of the

Normal-Interpolated Triangles

The position of a point **P** on the plane of a triangle may be expressed as a linear combination of the vertices of triangle:

$$\mathbf{P} = \alpha \mathbf{P}_\alpha + \beta \mathbf{P}_\beta + \gamma \mathbf{P}_\gamma$$

where the barycentric weights α, β, and γ are all positive when **P** is inside the triangle and add up to one when **P** is on the plane of the triangle. These values may be calculated as the dot product between the point **P** expressed in normalized homogeneous coordinates (i.e., *w*=1) and a set of planes $\mathbf{L}_\alpha$, $\mathbf{L}_\beta$, and $\mathbf{L}_\gamma$:

$$\alpha(\mathbf{P}) = \mathbf{L}_\alpha \cdot \mathbf{P}$$
$$\beta(\mathbf{P}) = \mathbf{L}_\beta \cdot \mathbf{P}$$
$$\gamma(\mathbf{P}) = \mathbf{L}_\gamma \cdot \mathbf{P}$$

$\mathbf{L}_\alpha$ can be any plane that contains $\mathbf{P}_\beta$ and $\mathbf{P}_\gamma$ (e.g., one that is perpendicular to the triangle), and its coefficients are normalized so that $\mathbf{L}_\alpha \bullet \mathbf{P}_\alpha = 1$; $\mathbf{L}_\beta$ and $\mathbf{L}_\gamma$ can be computed similarly. The normal at a point is then computed as a linear combination of the normals at the vertices:

$$\mathbf{n} = (\mathbf{L}_\alpha \cdot \mathbf{P})\ \mathbf{N}_\alpha + (\mathbf{L}_\beta \cdot \mathbf{P})\ \mathbf{N}_\beta + (\mathbf{L}_\gamma \cdot \mathbf{P})\ \mathbf{N}_\gamma$$
$$\mathbf{N} = \frac{\mathbf{n}}{(\mathbf{n} \cdot \mathbf{n})^{1/2}}$$

Differentiating, we get:

$$\frac{\partial \mathbf{n}}{\partial x} = \left(\mathbf{L}_\alpha \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{N}_\alpha + \left(\mathbf{L}_\beta \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{N}_\beta + \left(\mathbf{L}_\gamma \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{N}_\gamma$$
$$\frac{\partial \mathbf{N}}{\partial x} = \frac{(\mathbf{n} \cdot \mathbf{n})\ \frac{\partial \mathbf{n}}{\partial x} - \left(\mathbf{n} \cdot \frac{\partial \mathbf{n}}{\partial x}\right)\ \mathbf{n}}{(\mathbf{n} \cdot \mathbf{n})^{3/2}}$$

where a direction (e.g., the derivative of **P**) is expressed in homogeneous coordinates (i.e., *w*=0). The sum of the three barycentric weights for the derivative of **P** add up to zero when the direction is in the plane of the triangle.

Similarly, a texture coordinate can be expressed as a linear combination of the texture coordinates at the vertices:

$$\mathbf{T} = (\mathbf{L}_\alpha \cdot \mathbf{P})\ \mathbf{T}_\alpha + (\mathbf{L}_\beta \cdot \mathbf{P})\ \mathbf{T}_\beta + (\mathbf{L}_\gamma \cdot \mathbf{P})\ \mathbf{T}_\gamma$$

and its derivative is given by:

$$\frac{\partial \mathbf{T}}{\partial x} = \left(\mathbf{L}_\alpha \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{T}_\alpha + \left(\mathbf{L}_\beta \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{T}_\beta + \left(\mathbf{L}_\gamma \cdot \frac{\partial \mathbf{P}}{\partial x}\right)\ \mathbf{T}_\gamma$$

interactions in Section 3.1 requires a few dozen floating-point operations. On a very rough scale, this is approximately the same cost as a single ray-triangle intersection, a simple lighting calculation, or a step through a hierarchical acceleration structure, thus making the incremental cost insignificant in all but the simplest of scenes.

4 TEXTURE FILTERING

One practical application of ray differentials is texture filtering. If we can approximate the difference between the texture coordinates corresponding to a ray and its neighboring rays, then we can find the size and shape of a filtering kernel in texture space. The texture coordinates of a surface depend on the texture parameterization of the surface. Such a parameterization is straightforward for parametric surfaces, and algorithms exist to parameterize implicit surfaces [14]. In general, the texture coordinates of a surface may be expressed as a function of the intersection point:

$$\mathbf{T} = \mathrm{f}(\mathbf{P}) \tag{23}$$

We can differentiate with respect to x to get a function that is dependent on the intersection point and its derivative:

$$\frac{\partial \mathbf{T}}{\partial x} = \frac{\partial[\mathrm{f}(\mathbf{P})]}{\partial x} = \mathrm{g}\left(\mathbf{P}, \frac{\partial \mathbf{P}}{\partial x}\right) \tag{24}$$

We also derive the expression for the derivative of texture coordinates for a triangle in the box on the previous page.

Applying a first-order Taylor approximation, we get an expression for the extent of a pixel's footprint in texture space based on the pixel-to-pixel spacing:

$$\begin{aligned} \Delta\mathbf{T}_x &= [\mathbf{T}(x+\Delta x, y) - \mathbf{T}(x,y)] \approx \Delta x \frac{\partial \mathbf{T}}{\partial x} \\ \Delta\mathbf{T}_y &= [\mathbf{T}(x, y+\Delta y) - \mathbf{T}(x,y)] \approx \Delta y \frac{\partial \mathbf{T}}{\partial y} \end{aligned} \tag{25}$$

4.1 Filtering Algorithms

Assuming texture coordinates are two-dimensional, (25) defines a parallelogram over which we filter the texture. This is illustrated by Figure 2. Given this parallelogram, one of several texture filtering methods can be used. The most common method, mip mapping [19], is based on storing a pyramid of pre-filtered images, each at power of two resolutions. Then, when a filtered sample is required during rendering, a bilinearly interpolated sample is taken from the level of the image pyramid that most closely matches the filtering footprint. There are many ways of selecting this level-of-detail, and a popular algorithm [10] is based on the texel-to-pixel ratio defined by the length of the larger axis of the parallelogram of Figure 2:

$$lod = \log_2\left[\max\left(\left(\Delta\mathbf{T}_x \cdot \Delta\mathbf{T}_x\right)^{1/2}, \left(\Delta\mathbf{T}_y \cdot \Delta\mathbf{T}_y\right)^{1/2}\right)\right] \tag{26}$$

Because the computed level-of-detail can fall in between image pyramid levels, one must round this value to pick a single level. In order to make the transition between pyramid levels smooth, systems will often interpolate between the two adjacent levels, resulting in trilinearly interpolated mip mapping.

Mip mapping is an isotropic texture filtering technique that does not take into account the orientation of a pixel's footprint. When using (26), textures are blurred excessively in one direction

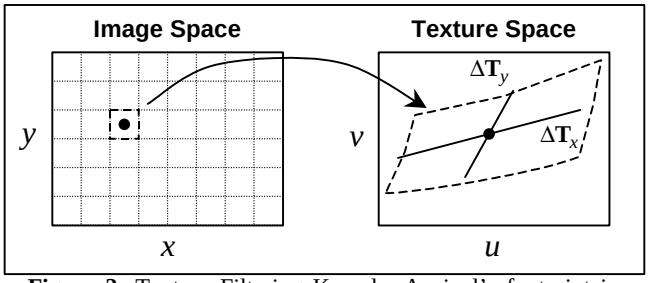

Figure 2: Texture Filtering Kernel. A pixel's footprint in image space can map to an arbitrary region in texture space. This region can be estimated by a parallelogram formed by a first-order differential approximation of the ratios between rate of change in texture space and image space coordinates.

if the parallelogram defined by (25) is asymmetric. Anisotropic filtering techniques take into account both the orientation and the amount of anisotropy in the footprint. A typical method [3, 15] is to define a rotated rectangle based on the longer of the two axes of the parallelogram, use the rectangle's minor axis to choose a mip map level, and average bilinear samples taken along the major axis. Again, one may interpolate between mip map levels.

4.2 Results

Figure 3 and Figure 4 demonstrate a scene rendered with four approaches towards texture filtering, all generated with a single eye ray per pixel at a resolution of 1000 by 666. In this scene, which consists entirely of texture mapped triangles, we are at a desk that is in a room with a wallpaper texture map for its walls and a picture of a zebra in the woods on the left. We are viewing a soccer ball paper weight on top of a sheet of text, and we are examining the text with a magnifying glass.

In Figure 3a, we perform texture filtering by doing a simple bilinear filtering on the texture map. The text in this image is noticeably aliased in the three places where minification is occurring: on the paper, in the reflection off of the ball, and around the edges of the lens. Additionally, the reflection of the walls and the zebra in the soccer ball is very noisy. This aliasing is also apparent on the frame of the magnifying glass, especially on the left edge as the zebra is minified down to only a few pixels on the x direction. Even at sixteen rays per pixel (not shown), this artifact is visible.

In Figure 3b, mip mapping is performed for texture lookups, and the level-of-detail value is calculated based on the distance a ray has traveled and projection onto the surface. The textures of the image are properly filtered for eye rays, but reflected and refracted rays use the wrong level-of-detail. For the reflections, the divergence of the rays increases because of the surface curvature, and thus the level-of-detail based on ray distance is too low. This results in aliasing off of the ball and the frame. For refraction, the rays converge, making the ray distance-based algorithm cause blurring.

In Figure 4a, we perform mip mapping with the level-of-detail being computed by ray differentials. The limitations of this isotropic filtering are most apparent in the text. In order to address the blurring of the text on the paper, in the reflection off the ball, and around the edges of the lens, we demonstrate anisotropic texture filtering in Figure 4b. This image still has visible aliasing due to silhouette edges and shadowing discontinuities, and Figure 5 demonstrates that a simple super-sampling of 4 rays per pixel combined with anisotropic texture filtering produces a relatively alias-free image.

4.3 Discussion

Given the use of ray differentials for texture filtering, two interesting problems arise on how to combine the technique with adaptive edge filtering and with illumination sampling algorithms. First, because the algorithm filters only texture data, some sort of filtering is still necessary for edge discontinuities. The brute-force algorithm used in Figure 5 solves this problem, but an adaptive algorithm would certainly be more efficient. An open question is what kind of adaptive algorithms would work most effectively. It would seem that adaptively super-sampling pixels whose ray trees differ from their neighbors' ray trees would work well, but implementing such an algorithm is challenging.

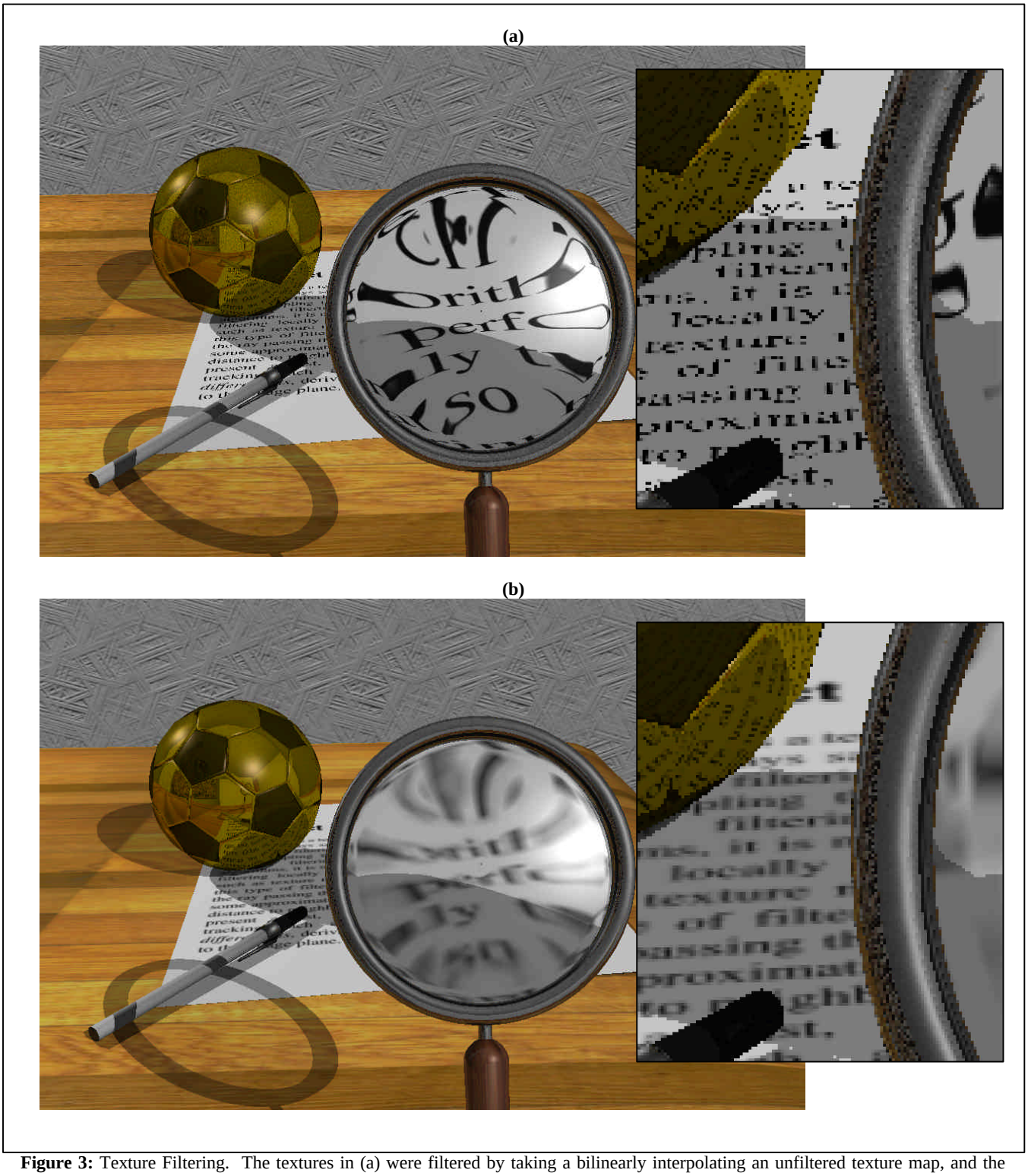

Figure 3: Texture Filtering. The textures in (a) were filtered by taking a bilinearly interpolating an unfiltered texture map, and the textures in (b) were filtered by trilinearly interpolating a mip map at a level-of-detail based on ray distance.

Another open issue with using ray differentials for texture filtering involves surface interactions. Although many surfaces can be described using reflection, refraction, and a localized shading model, one of the large advantages of ray tracing is its ability to implement all sorts of shading and illumination algorithms. For example, a general bi-directional reflectance function (BRDF) is necessary for describing physically correct reflectance models. BRDFs are usually sampled, and an open question is how to combine a ray's footprint with a sampled reflectance model. The idea of dull reflections (as presented for cone tracing [2]) suggests that a large amount of efficiency can be gained by factoring out texture filtering from BRDF sampling.

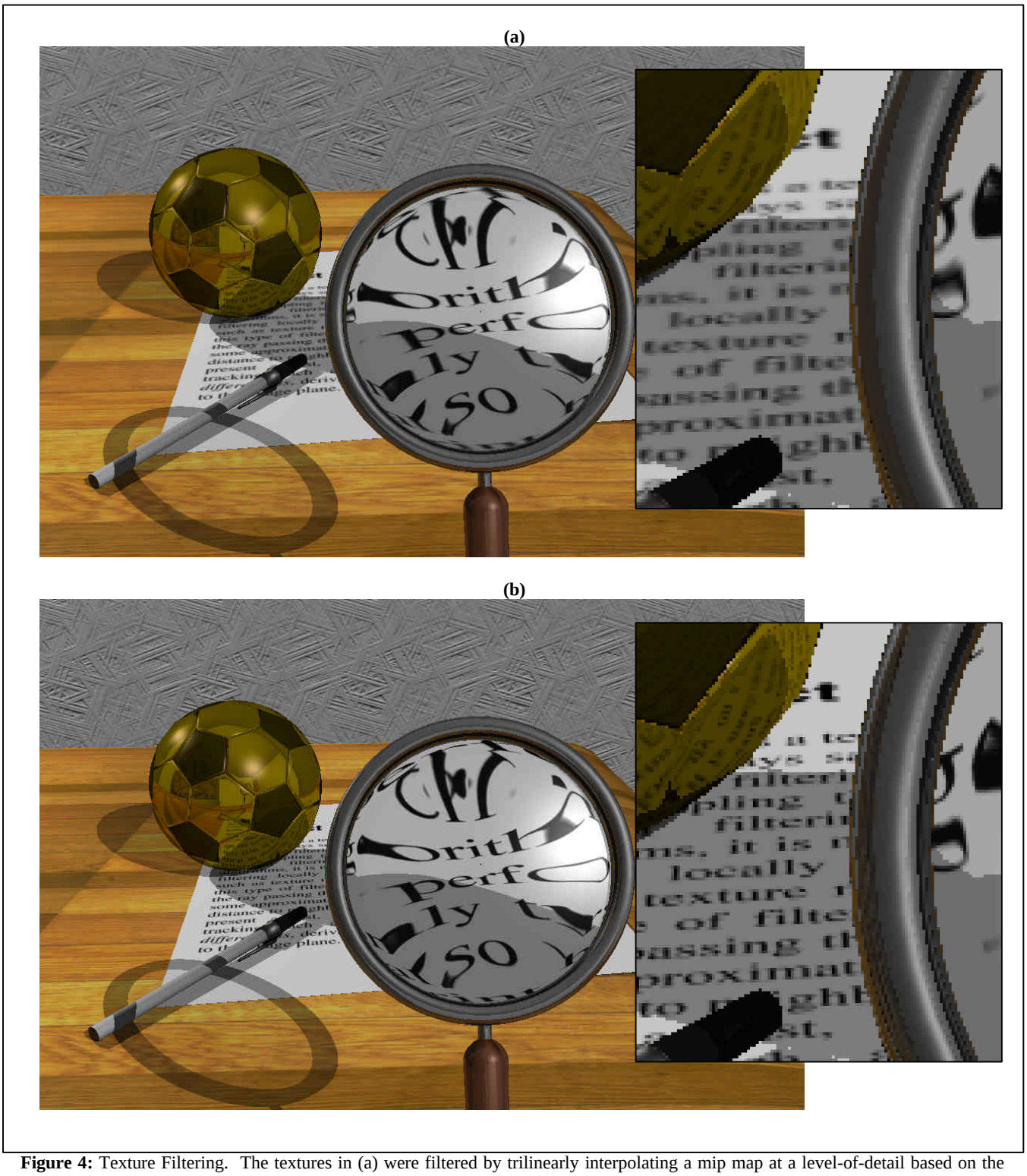

Figure 4: Texture Filtering. The textures in (a) were filtered by trilinearly interpolating a mip map at a level-of-detail based on the ray differential approximation of this paper. To reduce blurring, (b) performs anisotropic texture filtering based on ray differentials.

5 CONCLUSION

In this paper, we have presented a novel algorithm for tracking an approximation to a ray's footprint based on ray differentials, the derivatives of a ray function with respect to the image plane. This technique can robustly handle anisotropic pixel spacing and anisotropic texture filtering. Because our algorithm is based on elementary differential calculus, the application of ray differentials to a variety of physical and non-physical graphics algorithms is straightforward. Furthermore, the incremental cost of tracking ray differentials is very small compared to other costs of a ray tracer. Finally, we have demonstrated the use of ray differentials to efficiently perform texture antialiasing without super-sampling the image plane.

Acknowledgements

We would like to thank Pat Hanrahan, Matthew Eldridge, Matt Pharr, Tamara Munzner, and the reviewers for their assistance with this paper. Financial support was provided by Intel and DARPA contract DABT63-95-C-0085-P00006.

References

[1] K. Akeley. RealityEngine Graphics. *Computer Graphics* (SIGGRAPH 93 Proceedings), **27**, 109-116, 1993.

[2] J. Amanatides. Ray Tracing with Cones. *Computer Graphics* (SIGGRAPH 84 Proceedings), **18**, 129-135, 1984.

[3] A. Barkans. High-Quality Rendering Using the Talisman Architecture. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 79-88, 1997.

[4] M. Born and E. Wolf. *Principles of Optics*. Pergamon Press, New York, 190-196, 1959.

[5] S. Collins. Adaptive Splatting for Specular to Diffuse Light Transport. *Fifth Eurographics Workshop on Rendering*, 119-135, 1994.

[6] R. Cook, T. Porter, and L. Carpenter. Distributed Ray Tracing. *Computer Graphics* (SIGGRAPH 84 Proceedings), **18**, 137-145, 1984.

[7] A. Glassner, ed. *An Introduction to Ray Tracing*. Academic Press, San Diego, 288-293, 1989.

[8] L. Gritz and J. Hahn. BMRT: A Global Illumination Implementation of the RenderMan Standard. *Journal of Graphics Tools*, **1**(3), 1996.

[9] A. Gullstrand. Die reelle optische Abbildun g. *Sv. Vetensk. Handl.*, **41**, 1-119, 1906.

[10] P. Heckbert. Texture Mapping Polygons in Perspective. *NYIT Computer Graphics Lab Technical Memo #13*, 1983.

[11] J. Loos, P. Slusallek, and H. Seidel. Using Wavefront Tracing for the Visualization and Optimization of Progressive Lenses. *Computer Graphics Forum* (Eurographics 98 Proceedings), **17**(3), 1998.

[12] D. Mitchell. Generating Antialiased Images at Low Sampling Densities. *Computer Graphics* (SIGGRAPH 87 Proceedings), **21**, 65-72, 1987.

[13] D. Mitchell and P. Hanrahan. Illumination from Curved Reflectors. *Computer Graphics* (SIGGRAPH 92 Proceedings), **26**, 283-291, 1992.

[14] H. Pederson. Decorating Implicit Surfaces. *Computer Graphics* (SIGGRAPH 95 Proceedings), **29**, 291-300, 1995.

[15] A. Schilling, G. Knittel, and W. Strasser. Texram: A Smart Memory for Texturing. *IEEE Computer Graphics and Applications*, **16**(3), 32-41, 1996.

[16] M. Shinya and T. Takahashi. Principles and Applications of Pencil Tracing. *Computer Graphics* (SIGGRAPH 87 Proceedings), **21**, 45-54, 1987.

[17] D. Struik. *Lectures on Classical Differential Geometry*, Second Edition. Dover Publications, New York, 1961.

[18] T. Whitted. An Improved Illumination Model for Shaded Displays. *Communications of the ACM*, **23**(6), 343-349, 1980.

[19] L. Williams. Pyramidal Parametrics. *Computer Graphics* (SIGGRAPH 83 Proceedings), **17**, 1-11, 1983.

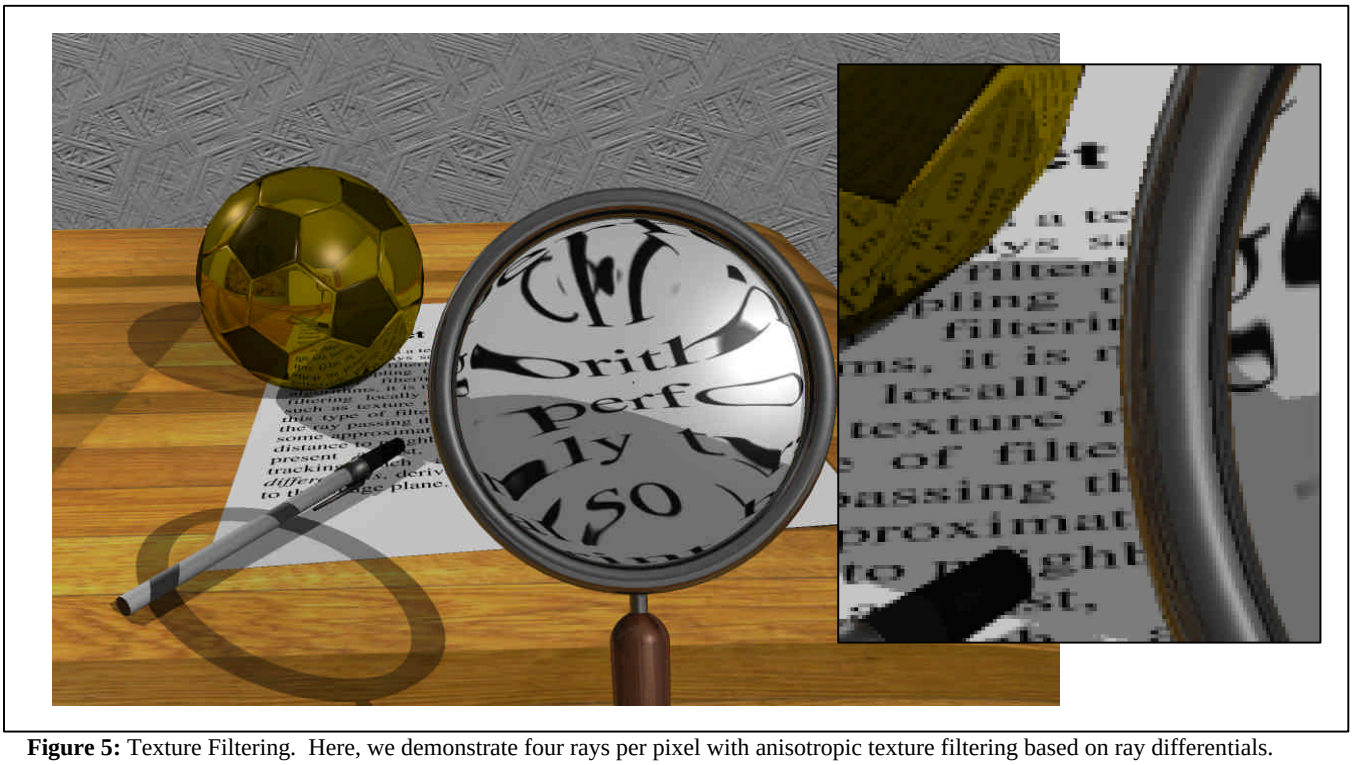

Figure 5: Texture Filtering. Here, we demonstrate four rays per pixel with anisotropic texture filtering based on ray differentials.