CHAPTER 5

IMAGE SPACE SOLUTION METHODS

Traditionally, most graphics programs produce pictures by determining a color value for each pixel of a raster screen. This is done in two steps: finding out which surface is visible through that pixel, and finding the radiance coming from the surface toward the pixel. This class of methods operates by finding only those radiances that contribute to the image. In some sense this is really solving for I by using lazy evaluation of the Global Radiance Function; we find radiances at only those locations that are visible. In this chapter these *image based* methods will be described, and some extensions of previous techniques will be shown.

In Section 5.1, the concept of the image function, I, will be expanded, and weighted area averaging techniques of converting I to a discrete (raster) image will be discussed. That discussion also includes some guidelines for selecting filters (weighting functions) that disallow many standard filters such as the cone, pyramid, sinc, and nice. A new filter that does satisfy the guidelines, while maintaining some of the good characteristics of the standard filters, is also presented.

Section 5.2 outlines the use of direct lighting and the ambient term in traditional computer graphics. Whitted-style ray tracing is presented in Section 5.3. More modern stochastic ray tracing methods, including a careful review of the basic mathematics behind them, are presented in Section 5.4. A new method of shadow ray optimization, where only one shadow ray is sent for each viewing ray, is also described.

In Section 5.5, methods of static sampling (where the number of samples in a pixel is predetermined) are reviewed and compared. This discussion differs from that of previous authors in Computer Graphics because the machinery of Integration Theory, rather than Signal Processing, is used to predict sampling performance. In addition, a new static sampling method that has several advantages over even Poisson Disk sampling is presented. Adaptive sampling methods are reviewed in Section 5.6, and a new adaptive strategy is presented. That section also argues, against prevailing wisdom, that hierarchical sampling *cannot* be straightforwardly applied to Distributed Ray Tracing because of the peculiarities of the sampling space used when performing Uncorrelated Jittering.

In this and the next two chapters, issues of color will be ignored. The implications of adding wavelength dependencies will be addressed in Chapter 7.

5.1 The Image Function

As outlined in Chapter 1, we can create an image using a viewer model (Figure 1.1) or camera model (Figure 1.3). Ultimately, we will display the image on a device, or generate hardcopy using a film recorder or color printer. Almost all display devices we might use are digital and represent pictures with a rectangular lattice of *pixels* (short for picture elements). To set values for these pixels, most devices use one number (three or four values for color systems).

Assuming we want to produce a greyscale image for a digital display device, we have to create a digital image, specified by a number (usually one byte long) for each pixel. In other words, we need to specify all values $P(x_i, y_j)$, where i and j are row and column numbers on the device. Currently, a high-end RGB monitor will be 2048 by 1536 or 1280 by 1024. These are some of the few numbers *not* going up explosively in the computer industry. A color film recorder or color printer will often have greater resolution, with up to 200 points per inch for a thermal color printer, and up to 800 points per inch for a film recorder[36].

Assuming we have a pin-hole camera model specified by a pin-hole location and a film-plane, then we first wish to find the radiance at the pin-hole seen from each spot on the film. This radiance determines the film response at each point, and can be described by the image function $I(x, y)$, where x and y are coordinates on the plane. Usually the film is assumed to be perfectly linear in response, with infinite resolution. Assuming I is known, then we can set pixel values:

$$P(x_i, y_j) = f_{ij}(I(x, y))$$

Where f_{ij} is a function that operates on I. Usually one function f is used for all f_{ij}, and f typically is the integral of a weighting function w (centered at (x_i, y_i)) multiplied by I:

$$P(x_i, y_j) = \int w(x - x_i, y - y_i) I(x, y) dA$$

Here the of area integration is wherever the weighting function is nonzero. This region is called the *support* of w. Because we do not want the pixel value to change when an image is flipped horizontally or vertically about that pixel, w will usually be symmetric about both the x and y axes. By similar logic, w should be diagonally symmetric so that 90° rotations in I will give 90° rotations in the digital image. In practice, the support of w will be only a few pixels across, so that $P(x_i, y_i)$ depends on values of I nearby (x_i, y_i).

If we'd like the overall radiance of the digital image to be similar to the overall radiance seen by the film plane, w should have unit volume:

$$\int w(x, y) dA = 1$$

If radiance scaling is desired, then some other constant than one can be used. This idea can be extended by requiring that the average intensity of the digital image is the same as the original continuous image. In other words, a small feature moving in the continuous image should not cause 'twinkling' in the digital image[36]. This can be stated quantitatively by requiring that the total contribution of a impulse (delta function) is the same regardless of position:

$$\sum_i \sum_j w(x - x_i, y - y_i) = \text{constant}$$

This constraint ensures that the DC component of the original and discrete images will be the same. This is equivalent to the constraint imposed by Mitchell and Netravali[74]. The example of an impulse in I also implies that w should be strictly nonnegative to avoid the possibility of negative pixel colors. In summary, we want w to have several features:

1. w has unit area.
2. w is horizontally, vertically, and diagonally symmetric.
3. The support of w has limited width.
4. $w \geq 0$ for all x and y.
5. $\sum_i \sum_j w(x - x_i, y - y_i) = \text{constant}$.

In addition, w may have either or both of two additional simplifying features[35]:

A. w is separable: $w(x, y) = a(x)a(y)$
B. w is rotationally symmetric: $w(x, y) = a(x^2 + y^2)$

A commonly used w that has all of the required features and is separable is a positive constant on a square centered at the origin. The width of the square is usually set to be the distance between pixel centers, but can be wider. This function is usually called a *box filter*.

In most of the graphics literature, the preceding discussion is usually viewed using signal processing theory. The image function I is convolved with a filter g, and becomes a new image function I'. The pixel values are then set by letting:

$$P(x_i, y_j) = I'(x_i, y_i) - (f \circ I)(x_i, y_i)$$

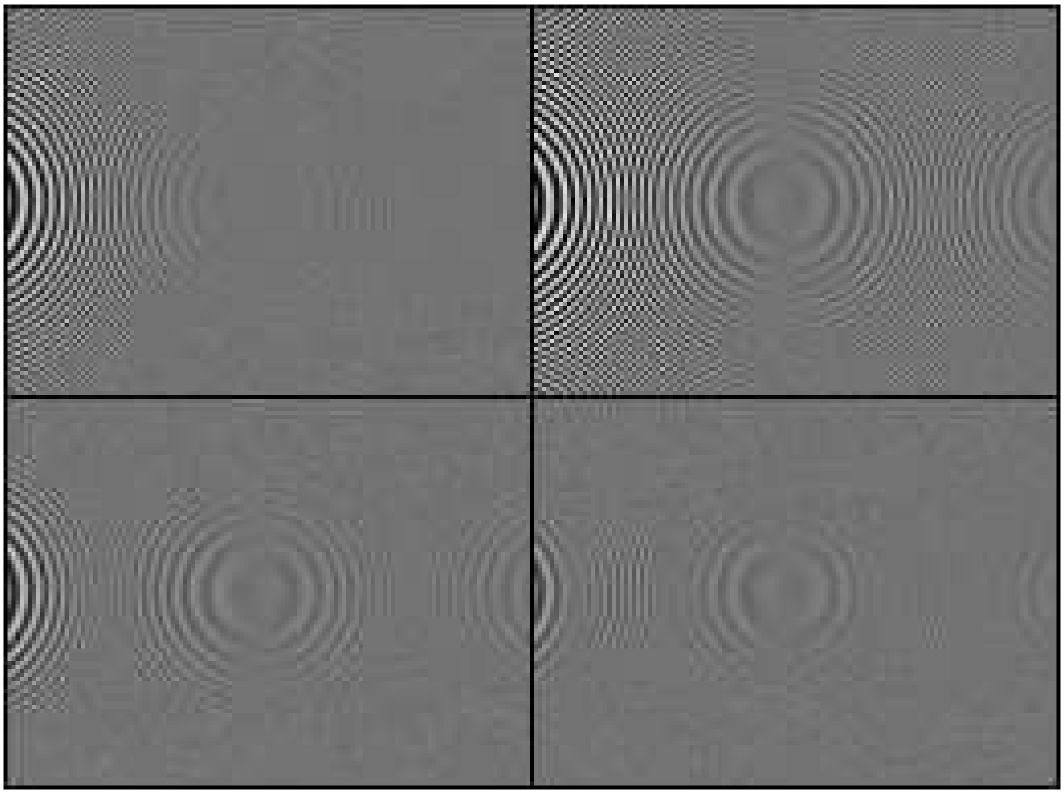

Figure 5.1: Images generated by (left to right, top to bottom) nonuniform filter, box filters with one pixel width, two pixel width, and three pixel width. The circular pattern on the left of each image is correct, while the circular pattern centered in the middle and on the right are caused by aliasing.

This approach is surveyed clearly by Blinn[13, 12]. Its limits and implications were deeply investigated by Kajiya and Ullner[59]. Because the shape of the intensity function of real pixels, and because our error metric is perceptual, signal processing theory does not yield an easy answer for what w is best[59]. There is, however, some consensus that signal processing theory implies that a nonuniform w with a maximum at the origin is preferable to a box filter[50, 51, 70, 36].

To develop an example of a nonuniform w, we can first assume a support that is restricted to at most a square of two pixel widths centered at the origin. We can further assume that w is separable ($w(x, y) = a(x)a(y)$) and that a is a cubic:

$$w(x, y) = \left(A|x|^3 + B|x|^2 + C|x| + D\right) \left(A|y|^3 + B|y|^2 + C|y| + D\right)$$

Note that this w is *not* circularly symmetric. Applying conditions 1-5 leaves yields four equations that imply $A = B = 0$, $C = -1$, and $D = 1$:

$$w(x, y) = (1 - |x|)(1 - |y|)$$

This w is similar to the bilinear filter shown in Figure 3 of [37]. In Figure 5.1, this nonuniform w and box filters of width one, two and three pixels is applied to the rather pathological image function $I(x, y) = \sin(x^2 + y^2)$. The origin is just to the left of each image. The concentric pattern on the left is 'real', and the others are artifacts caused by the regular grid of pixels and the character of w. The nonuniform w minimizes unwanted artifacts without excessive blurring of desired features.

It should be emphasized that the best w may be highly dependent on the display used. Amanatides and Mitchell have shown that NTSC displays in particular must be handled as a special case[4].

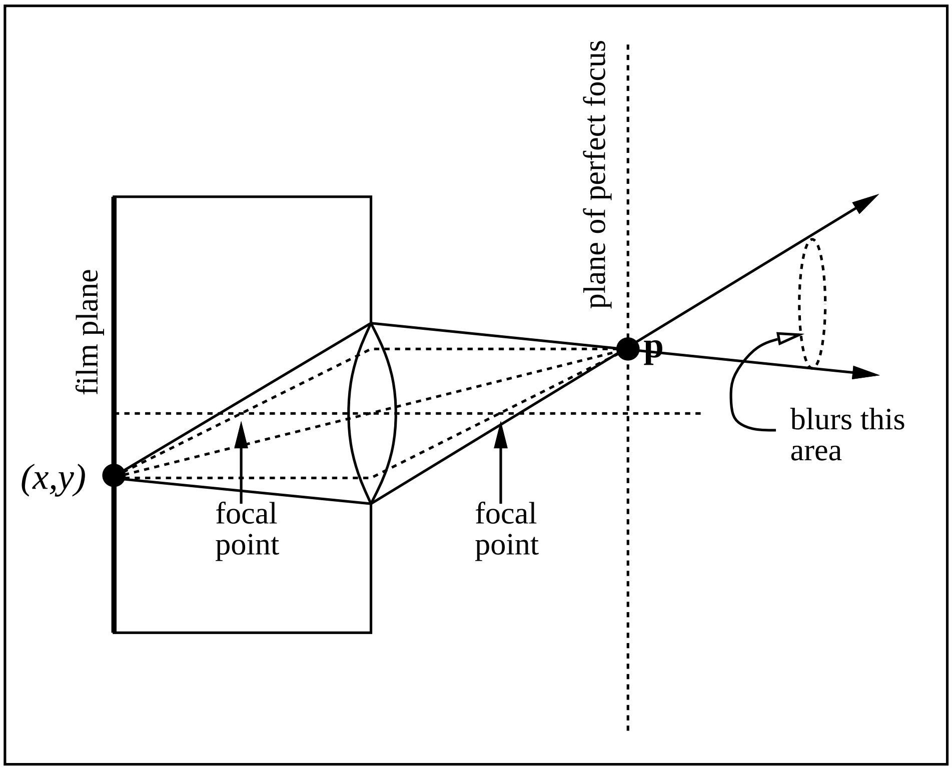

Figure 5.2: A thin lens camera

5.1.1 Finite Aperture Camera Model

A simple lens camera model can be substituted for a pin-hole camera model. This will make some objects appear to be blurred because of focusing effects. Such camera models have been used in both scanline[20, 86] and ray tracing[26] applications. In this model, the image function $I(x, y)$ is no longer the radiance seen through the pin-hole; instead it is the average radiance seen on the lens area from (x, y). The lens is assumed to be 'thin', so that it obeys certain rules illustrated in Figure 5.2: all light coming to point (x, y) passes through a point $\mathbf{p}$ on a plane of perfect focus; light traveling through the center of the lens will be undeflected; light passing through a focal point will be deflected by the lens along the axis of the lens. The second

Figure 5.3: Three Fujis on brushed steel. The middle Fuji is in the plane of perfect focus.

and third rules can be used to determine $\mathbf{p}$ for a particular (x, y)[1]. Figure 5.3 shows a image calculated using a thin lens camera model.

Averaging the radiance seen at the lens means the image function for a lens of area A is:

$$I(x, y) = \frac{1}{A} \int_{\mathbf{q} \text{ on lens}} L(\mathbf{q}, \mathbf{q} - \mathbf{p}) dA$$

This means the expression for pixel intensity becomes:

$$P(x_i, y_j) = \frac{1}{A} \int \int_{\mathbf{q} \text{ on lens}} w(x - x_i, y - y_i) I(x, y) L(\mathbf{q}, \mathbf{q} - \mathbf{p}) dA dA'$$

Thus, even if L is known, creating a digital image for a particular viewpoint is not trivial! One thing to note is that I does not drop off as the solid angle subtended by the lens decreases when (x, y) strays from the center of the film (as is also true for the pin-hole model).

[1] This will break down for $(x, y) = (0, 0)$. Instead, the distance to the plane of perfect focus can be calculated, and the ray from (x, y) through the center of the lens will intersect the plane at $\mathbf{p}$.

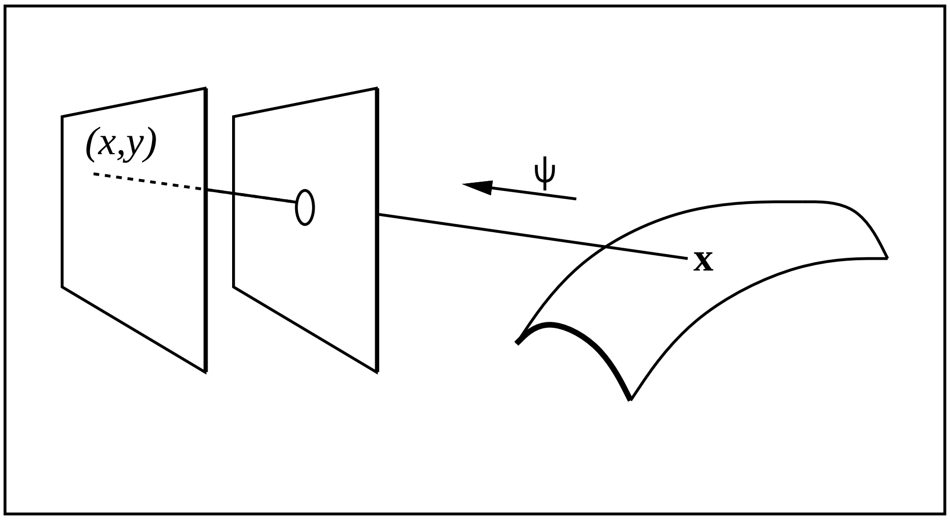

Figure 5.4: The image function at (x, y) is the radiance at $\mathbf{x}$ traveling toward the pin-hole.

5.2 Direct Lighting

As was seen in the last section, a pixel color can be determined by integrating the radiances seen in all directions. From the Ray Law, these radiance are the radiances L_{out} coming from the surfaces seen in those directions (see Section 3.2). For the pin-hole camera model, this means the image function I at a point (x, y) is:

$$P(x_i, y_j) = L_{out}(\mathbf{x}, \psi)$$

where $\mathbf{x}$ is the point on the surface seen through the pin-hole, and ψ is the direction from $\mathbf{x}$ to the pin-hole, as shown in Figure 5.4. To accurately find $L_{out}(\mathbf{x}, \psi)$, we would need to solve the rendering equation (Equation 3.4) at x. As an approximation, we can calculate the *direct light* reflected at x. Multiple reflections of light are not considered. Kajiya calls such methods *Utah Models* because of the pioneering work in this type of algorithm done at the University of Utah[61]. The benefit of a Utah Model is that the lighting calculation at $\mathbf{x}$ is entirely local.

Usually Utah Models assume that the light sources are point light sources infinitely far away. This allows Equation 3.4 to be evaluated for only one ψ_{in}.

5.2.1 Ambient Light

One problem with assuming only direct lighting is that the approximation is guaranteed to be too small. As a first approximation to fixing this problem, an arbitrary constant, the *ambient lighting*, is added to $L(\mathbf{x}, \psi_{in})$. The ambient term is supposed to approximate the *indirect* lighting at $\mathbf{x}$. Because indirect lighting is not constant, the ambient term will be in error for most $\mathbf{x}$. One technique for lowering the ambient error used in some graphics packages is to allow ambient terms to be specified for each object. Unfortunately, making good use of such a feature is more art than science.

One way to think of ambient lighting is to assume all radiance values visible from a point are some constant L_0. This L_0 is the appropriate value for the ambient component.

Researchers at Cornell have devised a method to intelligently guess a 'good' global ambient term for diffuse environments[22]. To do this they first calculate the average reflectance R and total surface area A in the environment. They then find the total power Φ emitted by all light sources. The fraction of Φ reflected immediately after being emitted by the sources is approximately $R\Phi$. Extending this idea to subsequent bounces estimates that the indirect power coming to a surface is approximately:

$$\Phi_{\text{indirect}} \approx \Phi(R + R^2 + R^3 + \cdots) = \Phi\frac{R}{1 - R}$$

Using Equation 4.1, this implies the ambient light reflected at $\mathbf{x}$, $L_a(\mathbf{x})$, is:

$$L_a(\mathbf{x}) = R(\mathbf{x})\frac{\Phi_{\text{indirect}}}{\pi A}$$

5.3 Whitted-Style Ray Tracing

The Utah Models perform best for primarily diffuse scenes. Kay used Snell's Law and ray tracing to include refractive effects[64]. Whitted used slightly more general ray tracing techniques to extend Utah Models to include perfect specular effects and shadows[121]. His technique is usually called *ray tracing*, but because that term has become so overloaded, I will refer to it as *Whitted-style* ray tracing.

In Whitted-style ray tracing, the image function $I(x, y)$ is calculated by sending a ray from (x, y) through the pin-hole, and determining the first point $\mathbf{p}$ hit by the ray. If $\mathbf{p}$ lies on a non-specular surface, then a Utah model is applied. However, the contribution of a particular light source is only counted if $\mathbf{p}$ is not in shadow relative to that source. Whether a point is in shadow is determined by sending a ray toward the light source and seeing if it hits any objects before the light.

If $\mathbf{p}$ is on a specular surface, then the radiance is calculated by attenuating the radiance seen in the direction of reflection. If the surface is not opaque, the attenuated color in the transmitted direction is added. Figure 5.5 shows how Whitted's method would process several rays.

We can implement Whitted's approach as a recursive function that evaluates the Global Radiance Function for a particular viewpoint. Suppose we have such a function, $L_p(x, y)$ defined for the film plane. This function could be written in terms of the Global Radiance Function $L(\mathbf{p}, \psi)$:

radiance function L_p(real x, real y)
    L_p returns the radiance value seen at (x, y) on film plane
    coming from direction of pin-hole.
begin
    direction ψ
    point $\mathbf{o}$

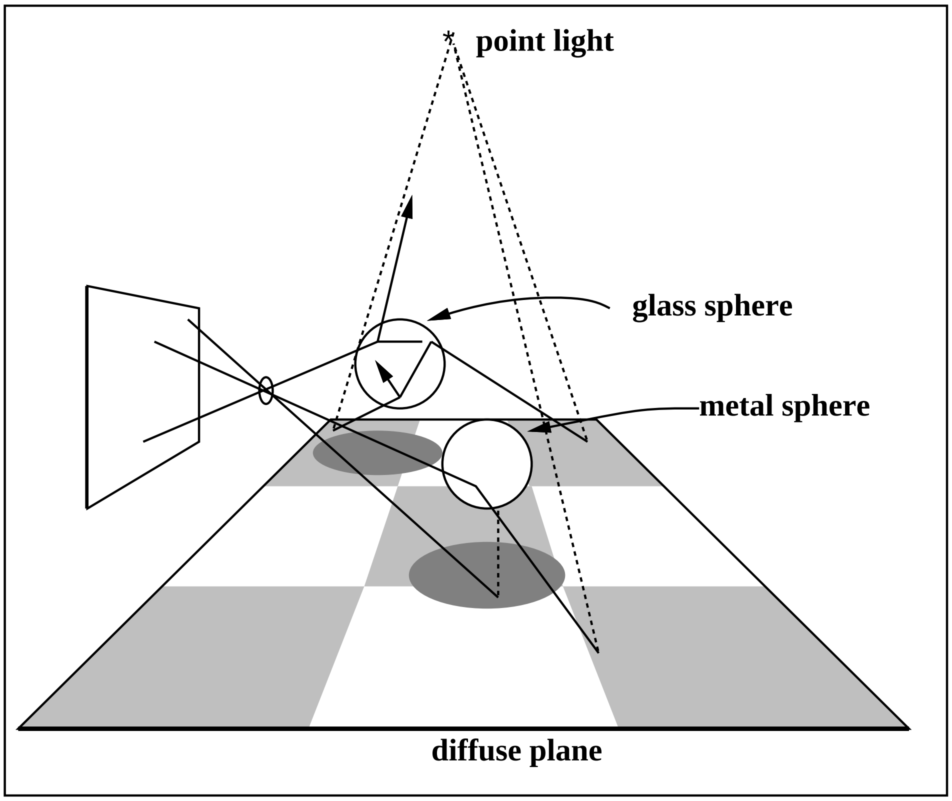

Figure 5.5: Several rays traced from the film plane. The solid lines are viewing rays, and the dashed lines are shadow rays.

```
        point ph
        o = position of (x, y) in object space
        ph = position of pin-hole in object space
        ψ = o − ph
        return L(o, ψ)
    end (L_p)
```

The Global Radiance Function also returns a radiance:

```
    radiance function L(point o, ψ)
        L returns the radiance value seen at o coming from direction ψ
    begin
        point p
        if ray o − tψ misses everything then
            return background radiance
        else
            find intersection point p of first object hit by ray
            if (object is opaque specular)
                find incoming reflected direction, ψ_r, by Equation 2.2
                return k_s L_in(p, ψ_r)
            else if object is transparent specular
                find incoming reflected direction, ψ_r,by Equation 2.2
                find incoming transmitted direction, ψ_t, by Equation 2.3.
                return k_s L_in(p, ψ_r) + k_t L_in(p, ψ_t)
            else apply Utah Model
                ψ_s = (l − p)
                if ray p − tψ_l hits something close than l then
                    return R(p)L_ambient
                else
                    return R(p)L_ambient+ direct lighting from l.
    end (L_p)
```

If a series of specular objects is hit we will have infinite recursion. Whitted avoids this by returning zero radiance after a certain number of reflections. Hall and Greenberg suggest stopping the recursion adaptively based on accumulated attenuation[44]. The adaptive technique is especially good when clear objects are present, and the internal reflections cause branching.

In some sense, Whitted-style ray tracing simply provides Utah-shaded objects, and reflections of Utah-shaded objects. This is coupled with the shadow ray technique that allows objects to shadow one another.

5.4 Stochastic Ray Tracing

Among the problems with Whitted-style ray tracing, and most other techniques that preceded ray tracing, is that they do not account for a finite aperture camera, non-point light sources, area sampling of I, or non-specular indirect lighting. Cook et al. attacked all of these problems at once by realizing that the intensity level for a pixel can be written as a multidimensional integral, and that classic Monte Carlo integration techniques can be used to solve that integral[26]. In this section Cook et al.'s technique is presented, followed by the other stochastic techniques of Kajiya and Ward et al.

5.4.1 Cook et al.'s Distributed Ray Tracing

The fundamental idea of Cook et al. is to perform a numerical integration for every pixel[26, 25, 14, 41]. Rather than using conventional regular quadrature techniques, they use stochastically distributed sample points. Using random sample points does not necessarily have a smaller error than regular sampling[2], but the random method's error will be less visually objectionable because there will not be coherence in the error between pixels.

For some insight into how the numerical integration is applied, consider the expression for the intensity of a pixel:

$$P(x_i, y_j) = \int\int w(x - x_i, y - y_i) I(x, y) dx dy \tag{5.1}$$

In Appendix B, it is shown that we can approximate an integral with a *primary unbiased estimator*:

$$\int_{a' \in \Omega} h(a') d\mu(a') \approx \frac{h(a)}{f(a)}$$

[2]Traditionally, Monte Carlo integration has better asymptotic error behavior if the dimension of the integral is sufficiently large[109], as it often is in graphics applications.

where a is a random variable with probability density function f. Saying that $h(a)/f(a)$ is an unbiased estimator for the integral simply means that the expected value is the value of the integral. We can come up with a 'better' unbiased estimator for the integral by averaging many of the primary estimators to form a *secondary estimator*:

$$\int_{a' \in \Omega} h(a') d\mu(a') \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h(a_i)}{f(a_i)}$$

The secondary estimator is better simply because it has a lower variance. The Law of Large Numbers tells us that the secondary estimate will converge to the value of the integral with probability one as N goes to infinity.

Assuming we know how to evaluate I, we can easily write down a primary estimator for the integral of Equation 5.1. First, assume that the pixel area is one (the distance between pixel centers is one), and that w is zero outside the pixel area. Using a constant probability density function $f = 1$ inside the pixel, and $f = 0$ outside the pixel will generate random points a on the pixel area. Thus the primary estimator will be:

$$P(x_i, y_j) = \int \int w(x - x_i, y - y_i) I(x, y) dx dy \approx I(a_x, a_y) \tag{5.2}$$

As discussed earlier, the secondary estimator is found by averaging a series of the primary estimators. Stratified sampling can be employed by subdividing the domain of the integral in Equation 5.2 and summing the primary estimator of each of these integrals.

Suppose instead that we use the nonuniform $w = (1 - |x|)(1 - |y|)$ with a width of two. Without loss of generality, assume that $x_i = y_j = 0$ (a change of coordinates):

$$P(x_i, y_j) = \int_{-1}^{1} \int_{-1}^{1} (1 - |x|)(1 - |y|) I(x, y) dx dy \tag{5.3}$$

A naive primary estimator can again be found with a uniform density $f(a_x, a_y) = 0.25$ on the support of w:

$$\int_{-1}^{1}\int_{-1}^{1}(1-|x|)(1-|y|)I(x,y)dxdy \approx 4(1-|a_x|)(1-|a_y|)I(a_x,a_y) \tag{5.4}$$

We can instead use a nonuniform f for choosing sample points with density f. If our choice of f is wise (i. e. reduces variance of the primary estimator), then we are using *importance sampling.* A natural choice is $f = w$ because the expressions are simplified:

$$\int_{-1}^{1}\int_{-1}^{1}(1-|x|)(1-|y|)I(x,y)dxdy \approx I(a_x,a_y) \tag{5.5}$$

Using a nonuniform f will require generating nonuniform random numbers. As shown in Appendix B, a series of one dimensional independent identically distributed according to f random variables $(\alpha_1, \alpha_2, \alpha_3, \ldots)$ (abbreviated $\alpha_i \sim f$) can be generated by suitably transforming a series of *canonical* random numbers $(\xi_1, \xi_2, \xi_3, \ldots)$. Canonical random numbers are simply uniformly distributed random numbers between zero and one. The actual transformation for a given f is:

$$\alpha_i = F^{-1}(\xi_i) \tag{5.6}$$

where F^{-1} is the inverse of the probability distribution F associated with the probability density f:

$$F(x) = \int_{-\infty}^{x} f(x')dx' \tag{5.7}$$

Generating multidimensional random variables is more difficult, but can usually be done in a generalization of the inverse distribution procedure if f is sufficiently well behaved (see Appendix B). For separable densities, $f(x,y) = g(x)h(y)$, we can choose (α, β) pairs with density f by choosing α according to g and β according to h.

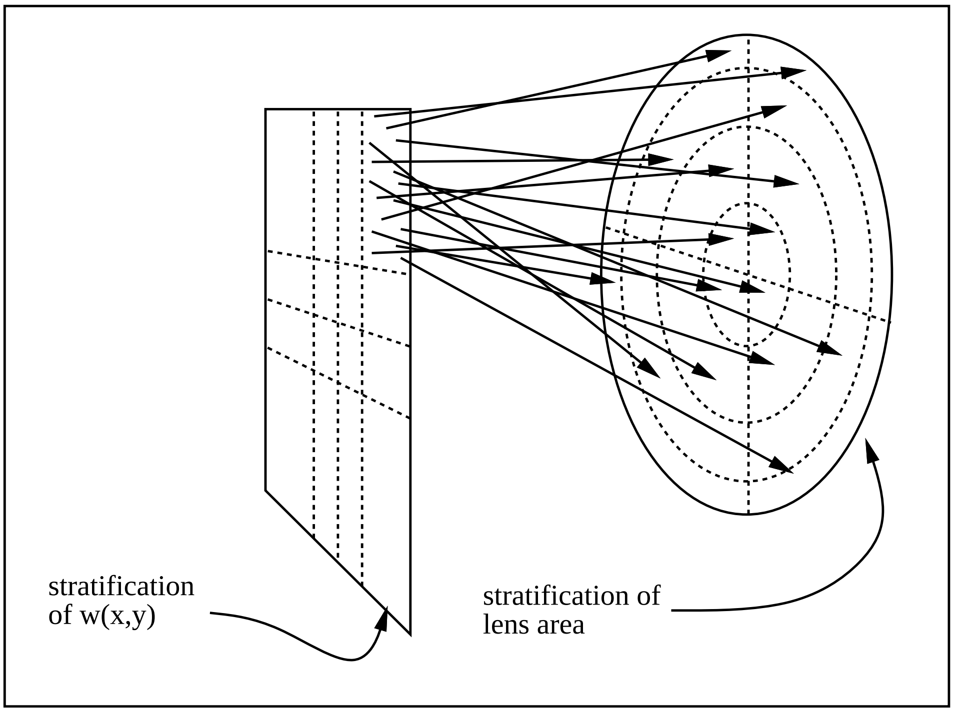

Figure 5.6: 16 rays fired from one stratum of pixel toward 16 strata on lens

.

Getting back to our pixel sampling, the weighting function $f(x,y) = (1 - |x|)(1 - |y|)$ is separable, and $g = h$, so we can generate (α_x, α_y) pairs according to $f(x) = (1 - |x|)$. The distribution function for this f is:

$$F(x) = \int_{-1}^{x} (1 - |x'|)dx' = \frac{1}{2} + x - \frac{1}{2}x|x| \tag{5.8}$$

and thus the inverse of F is:

$$F^{-1}(x) = \begin{cases} 1 - \sqrt{2(1-x)} & \text{if } x \geq 0.5 \\ -1 + \sqrt{2x} & \text{if } x < 0.5 \end{cases} \tag{5.9}$$

Thus, using $(F^{-1}(\xi_i), F^{-1}(\xi_j))$ from Equation 5.9 will generate pairs (α_i, α_j) with density $f(x,y) = (1 - |x|)(1 - |y|)$.

An immediate thing to wonder is whether we can mix importance sampling and stratified sampling. This actually can be done in a very simple manner: pick a set of stratified canonical

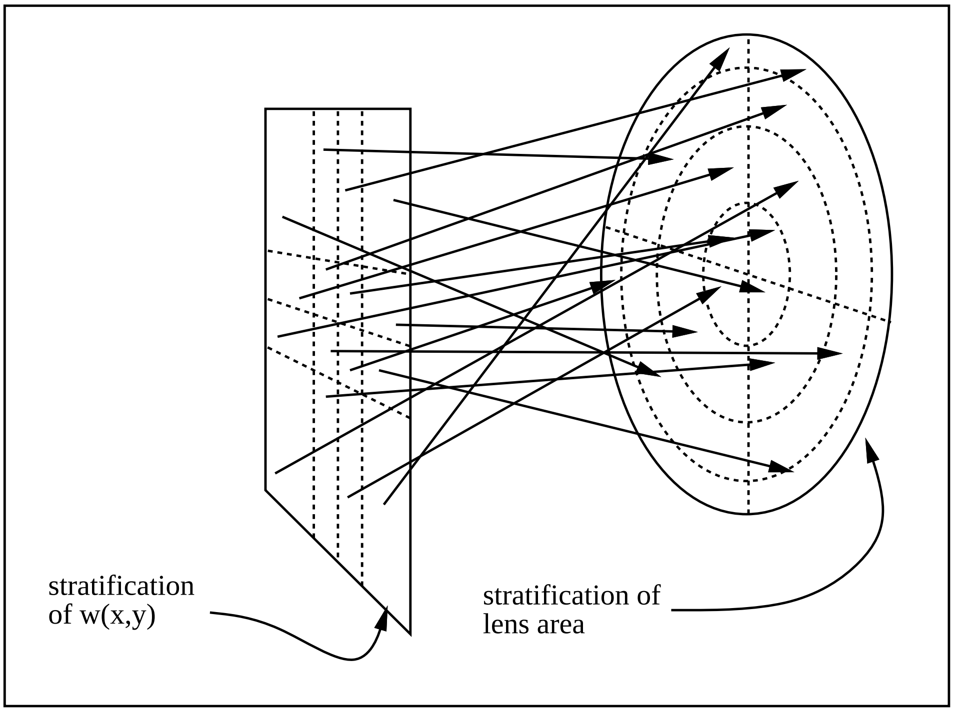

Figure 5.7: 16 rays fired from all strata of pixel toward 16 strata on lens

.

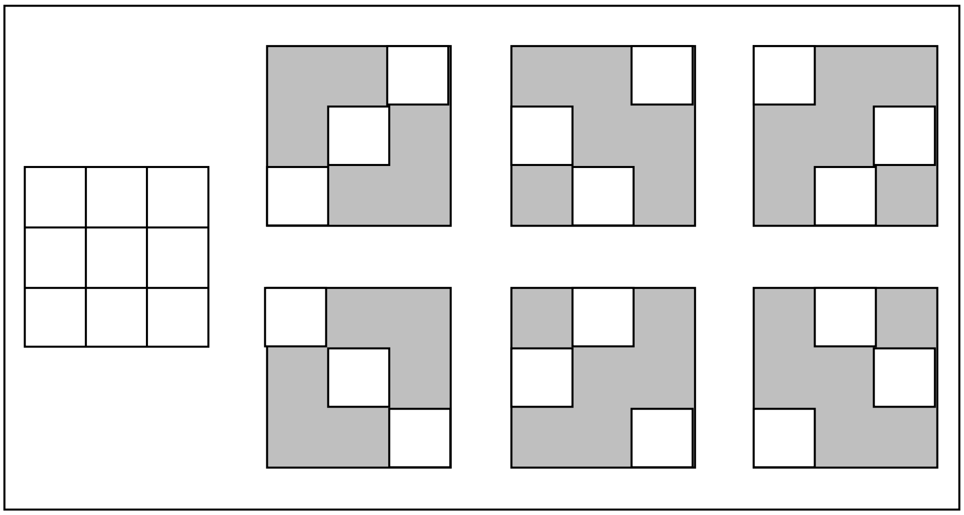

Figure 5.8: All valid permutations of three uncorrelated strata

.

samples (ξ_i, ξ_j) from the unit square, and transform them using the inverse distribution function. We are actually sampling according to a different probability function in each stratum, but we can add the estimators as if they were identically distributed. This idea *greatly* simplifies implementing a stochastic ray tracing code.

The pixel area is not the only space that needs to be integrated over. If we add a camera lens model, then we have a four dimensional integral. If we put a polar coordinate system on the camera lens we have (for w with support of width 2):

$$P(x_i, y_j) = \frac{1}{A_{lens}} \int_{x=-1}^{1} \int_{y=-1}^{1} \int_{\theta=0}^{2\pi} \int_{r=0}^{R_{lens}} w(x,y) L(x,y,r,\theta) r \sin\theta \, dr \, d\theta \, dx \, dy$$

To straightforwardly apply Monte Carlo integration we would generate four dimensional random variables to generate primary estimators. Figure 5.6 shows a pixel sampling function w and lens area each divided into 16 strata. This makes $16^2 = 256$ strata in total, so 256 rays will be fired in all. In the figure, the sixteen rays that would come from one of the pixel stratum is shown. All of the other strata on the pixel would also send this bundle of rays. As the dimension of the integral grows larger (as it will once we add shadows and reflection and motion blur), the explosion of rays will increase, so the number of strata for each dimension must be cut to keep the number of rays at a reasonable level. This problem was avoided by Cook using what he called uncorrelated jittering[25]. In this method, we associate each stratum on the pixel with a stratum on the lens, and make sure no stratum has more than one association. In this way, one ray is fired through each stratum, as shown in Figure 5.7.

Cook uses the term *uncorrelated* because any consistent mapping between particular strata will cause artifacts in the image. Instead we should use a different mapping for each pixel. Uncorrelated jittering is especially helpful when some of the dimensions of the integral are constant. For example, if the surface seen from the pixel is in perfect focus (the same point is seen regardless of the point chosen on the lens), then we still get a full stratification of the pixel space.

One problem with uncorrelated jittering is that the mathematics behind it has never been investigated in the Computer Graphics literature. This lack of foundation can cause confusion when trying to extend the technique. The basis for uncorrelated jittering can best be seen by looking at a two dimensional example. Suppose we have the unit square divided into 9 equal squares. As discussed earlier, we can get a primary estimator for an integral over the square by evaluating an expression at a random point within the square. Another way to get a primary estimator would be to choose one of the 9 squares at random and then choose a random point within the square. We could extend this idea by selecting more than one square and taking one sample from each square. This will still be an unbiased estimator as long as each square is equally likely to be chosen in the long run. In two dimensional uncorrelated jittering, we would choose 3 of the 9 squares, making sure that each square is the sole occupant of each row and column. This will allow full stratification in both dimensions. All such sets of three squares are shown in Figure 5.8. If we choose any one of these allowed sets of three squares at random, the estimator will be unbiased because each square is a member of the same number of allowed sets. One way to generate the sets is to permute the sequence $(1, 2, 3)$ and use this as row numbers and the unpermuted $(1, 2, 3)$ as column numbers.

This basic idea of uncorrelated sampling is relatively unknown in the Monte Carlo literature. It is briefly mentioned as an untried possibility in the book by Kalos[63], but Computer Graphics seems to be the only field in which it has been applied. The specifics of uncorrelated jittering add some complexity to the basic idea; the two pixel dimensions (x and y) and the two lens dimensions are each linked as a pair. Valid mappings between pixel strata $(p_1, p_2, \cdots, p_N)$

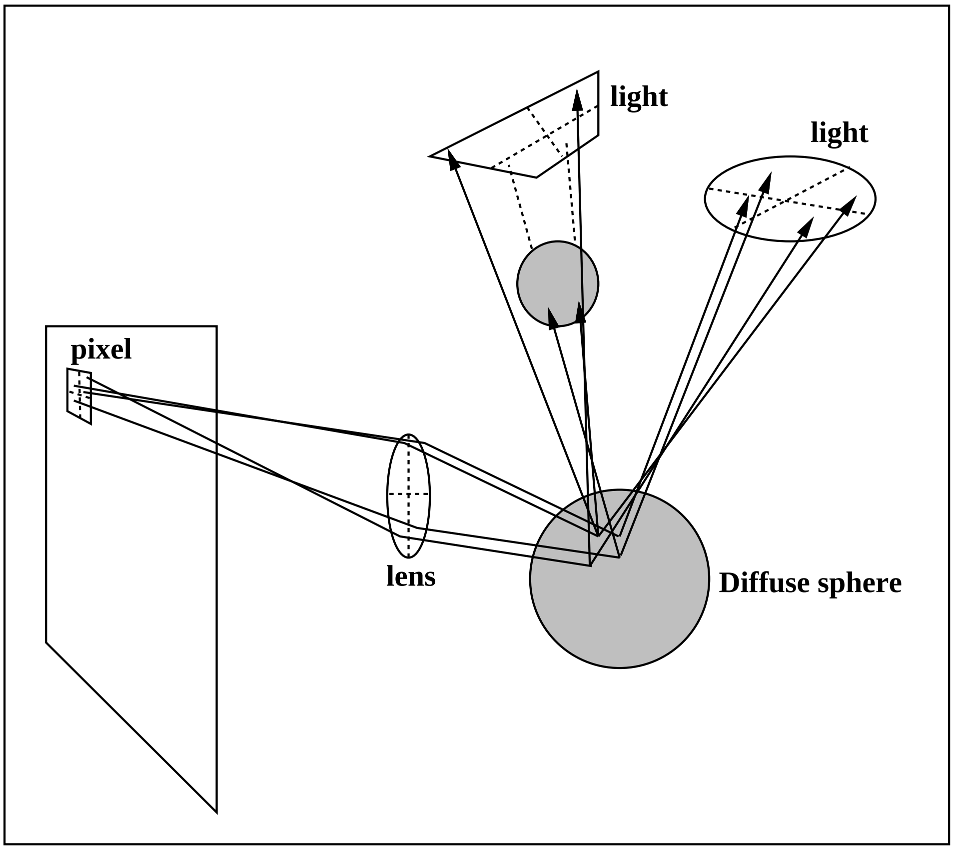

Figure 5.9: Rays traced in distributed ray tracing with four rays per pixel.

and lens strata $(l_1, l_2, \cdots, l_N)$ can still be found by permuting one of the sequences. Cook actually used a more restrictive mapping method that avoided mappings that allowed too much correlation[3]. This would avoid sample sets that had many rays in the same area of the pixel going to the same area of the lens. This is no longer a strictly Monte Carlo procedure, but can be justified if every stratum gets 'equal opportunity'. This technique of restricting the acceptable random sets relates to traditional 'quasi-random' methods.

One nice thing about uncorrelated jittering is that extra dimensions can be easily added. For example, if a diffuse reflector is seen from a particular pixel, we'll want to do a lighting calculation. This will be in the form of an integral across each light source area. Using

[3]It is not clear from Cook's presentation that his strategy yields an unbiased estimator.

Equation 3.5 yields:

$$P(x_i, y_j) = \frac{1}{A_{lens}} \int_{x=-1}^{1} \int_{y=-1}^{1} \int_{\theta=0}^{2\pi} \int_{r=0}^{R_{lens}} \int_{x' \text{ on light}} w(x,y) g(\mathbf{x}, \mathbf{x}') R(\mathbf{x}\lambda) I(\mathbf{x}')\, dr\, d\theta\, dx\, dy$$

Here $\mathbf{x}$ is the point seen by a particular ray from the lens and $g(\mathbf{x}, \mathbf{x}')$ is evaluated by sending a shadow ray toward the light. $I(\mathbf{x}')$ represents the lighting expression from Equation 3.5. The rays traced for this situation are shown in Figure 5.9. If the object hit is not diffuse then instead of testing for shadow and shading, a reflected ray is sent.

5.4.2 Kajiya's Path Tracing

Kajiya extended distributed ray tracing by phrasing the problem as a Monte Carlo solution to an integral equation. Recall that the radiance at a point was written down in Equation 3.5, which can be written without wavelength dependency as:

$$L_{out}(\mathbf{x}, \psi) = L_e(\mathbf{x}, \psi) + \int_{\text{all } \mathbf{x}'} g(\mathbf{x}, \mathbf{x}') \rho(\mathbf{x}, \psi, \psi') L_{out}(\mathbf{x}', \psi') \cos\theta \frac{dA' cos\theta'}{\|x' - x\|^2} \tag{5.10}$$

In Appendix B it is shown that for an equation of the form:

$$a(x) = b(x) + \int_{x' \in \Omega} k(x, x') a(x') d\mu(x') \tag{5.11}$$

We can write down an unbiased primary estimator:

$$\begin{aligned} a(x) \quad = \quad & b(x) + \\ & \frac{k(x,x^1)b(x^1)}{f_1(x^1)} + \\ & \frac{k(x,x^1)k(x^1x^2)b(x^2)}{f_2(x^1,x^2)} + \\ & \frac{k(x,x^1)k(x^1,x^2)k(x^2,x^3)b(x^3)}{f_3(x^1,x^2,x^3)} + \\ & \vdots \\ & \frac{k(x,x^1)\cdots k(x^{n-1},x^n)b(x^n)}{f_n(x^1,\cdots,x^n)} + \end{aligned}$$

$$\vdots \tag{5.12}$$

where $f_n(x^1, \cdots, x^n)$ is the probability density function for a sequence $(x^1, \cdots, x^n)$. The series can be terminated by waiting until we get a k with value zero (or accepting truncation error), or by *russian roulette*, where it is terminated probabilistically[6]. Russian roulette eliminates the bias of truncation.

To directly apply this series method to the rendering equation, we can view the space Ω as $\mathbf{x}^\star$, the set of all points on all surfaces. Once we have chosen a point on the pixel and lens, then we want to know $L(\mathbf{x}_0, \psi)$, where $\mathbf{x}_0$ is a point on the lens, and ψ is an incoming direction determined by the thin lens rules. We trace a ray to find the first surface $\mathbf{x}_1$ seen in direction $-\psi$. By the Ray Law $(\mathbf{x}_0, \psi) = L_{out}(\mathbf{x}, \psi)$. Equation 5.10 gives an expression for $L_{out}(\mathbf{x}, \psi)$ in the form of Equation 5.11. The function L_{out} maps to a, L_e maps to b, and the complicated expression in the integrand (with L divided out) maps to k.

To get an estimator we just need to choose a series of points according to some distributions f_n, and evaluate the series. For each series of n terms, we need to do $(n-1)$ evaluations of the visibility term g for adjacent points. This is accomplished by tracing a ray between the points and seeing if they are visible to each other. In practice, we should choose the points carefully. Kajiya suggests eliminating zero terms by only using series that have visible adjacent pairs. If we carry this idea farther by setting the probability functions to be proportional to the k (automatically setting zero probability for zero terms) and allow truncation we get:

$$\begin{aligned} a(x) \quad \approx \quad & b(x) + \\ & K(x, x^1)b(x^1) + \\ & K(x, x^1)K(x^1x^2)b(x^2) + \end{aligned}$$

$$
\begin{aligned}
&K(x,x^1)K(x^1,x^2)K(x^2,x^3)b(x^3) + \\
&\vdots \\
&K(x,x^1)\cdots K(x^{n-1},x^n)b(x^n) \qquad (5.13)
\end{aligned}
$$

where $K(x^{n-1}, x^n)$ is the volume of $k(x^{n-1}, x^n)$. Applying this to the rendering equation gives:

$$
\begin{aligned}
L_{out}(\mathbf{x},\psi) \approx\ & L_e(\mathbf{x},\psi) + \\
& R(\mathbf{x},\psi)L_e(\mathbf{x}',\psi') + \\
& R(\mathbf{x},\psi)R(\mathbf{x}',\psi')L_e(\mathbf{x}'',\psi'') + \\
& \vdots \qquad (5.14)
\end{aligned}
$$

Where x^n is chosen by sending a ray from $x^{(n-1)}$ in direction $-\psi^{(n-1)}$, and $\psi^{(n-1)}$ is chosen with a density given by the **SPF** at $\mathbf{x}^{n-1}$ given incoming direction $\psi^{(n-2)}$. This *sounds* very complicated, but in practice it's very simple. Starting on the lens, send a ray and find the point hit. If the point emits any light, accumulate it. Reflect the ray as if it were a real light ray traveling 'forward' (this is allowed because of the Helmholtz reciprocity condition), and find the new surface hit. If this second point emits light accumulate its value times the reflectivity of the first surface. Send a new ray according the the **SPF** of the second surface and find the third surface hit. If the third surface emits light, accumulate its value times the product of the reflectivities of the first two surfaces. The third surface sends a reflected ray, and so on. The process stops when the product of the reflectivities falls below a certain value, or at an arbitrary number of reflections. Since the method traces light paths (in reverse) through the room, Kajiya called it *path tracing*. An example of this process is shown in Figure 5.10, where a room with uniform diffuse reflectivity 0.5, except for a glass ball and a light source with radiance 8 and zero reflectivity. The series is terminated when the accumulated reflectivity falls

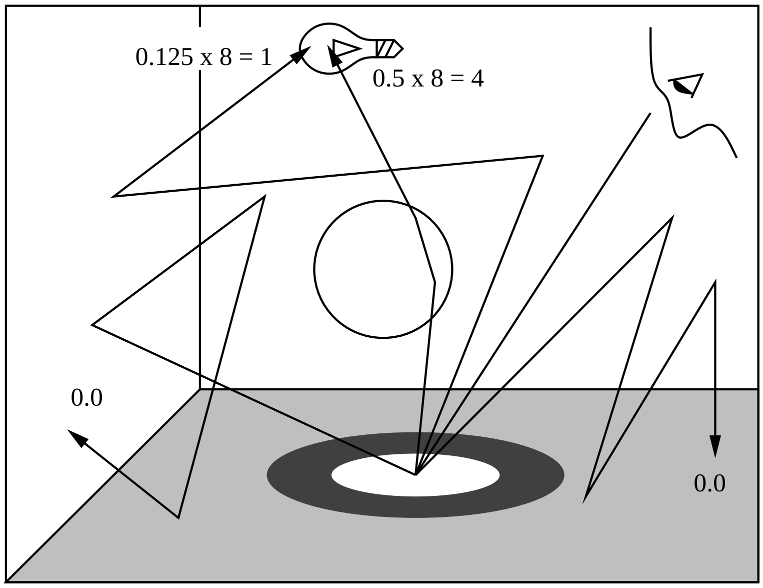

Figure 5.10: Path tracing in a room with a glass ball, walls with reflectivity 0.5, and a light with radiance 8.

below 0.1 (4 bounces). In the four rays shown, two take four reflections and contribute nothing. Two others contribute 1 and 4. The estimate for radiance is the average of these, or 1.25.

The problem with this technique is that the variance will be very high unless the emitted light is divided over a large area. Kajiya tried to lessen this problem by calculating the direct lighting at each selected point. This is best thought of as recursively applying distributed ray tracing. When a diffuse surface is seen, a reflection ray is sent in addition to the shadow ray. If the reflection ray *directly* hits a light source, the direct contribution is not included. If it hits the light source after reflection from a specular surface, then the contribution is counted. These indirect contributions allow for effects like the bright spot under the glass ball in Figure 5.10. Because these indirect terms are handled in the same way as those of crude path tracing, the indirect lighting may have very high variance.

Kajiya's path tracing can be thought of as lazy evaluation of the global radiance function at the lens; only those radiances at points contributing to the image are calculated. Unfortunately, many surfaces that are not seen will contribute their radiances indirectly, and the radiances of these surfaces might be recalculated many times. Ward et al. suggested that once a radiance value is calculated it should be saved in a geometric table for later use[119]. This type of storage was implemented for diffuse reflectors, where the table needed no directional information. Ward et al.'s technique is especially effective for diffuse interreflection. Like path tracing, it has fairly high variance for effects like bright spots under glass balls.

5.4.3 Shadow Ray Optimization

One problem with traditional ray tracing methods is that shadow rays are sent toward every light source[121, 25]. An example of why this is a problem would occur when ray tracing an image of a street lit by one hundred streetlights. At any particular spot on the street, we will send one hundred shadow rays in total, even though most of the lights make negligible contributions. Kajiya noted that shadow rays could be sent in various numbers in a probabilistic way, but did not propose a specific strategy. Shirley implemented a method where one shadow ray is sent toward all light sources, and the contribution is either all or nothing depending on whether the ray is obstructed[100].

The difficult part of sending the shadow ray is constructing the probability space that determines where the ray is sent. The first step is to choose the target light based on its total contribution. This way more attention is paid to the most important lights (the nearby streetlights in the earlier example). Then a point on the light source is chosen to send the shadow ray toward. This should all be done using Cook's uncorrelated jittering, otherwise all of the rays through a pixel could go to the same area on the same light. Given a set of canonical random number pairs $(\xi_1, \xi_2)_i$ chosen in a stratified manner from the unit square, we choose

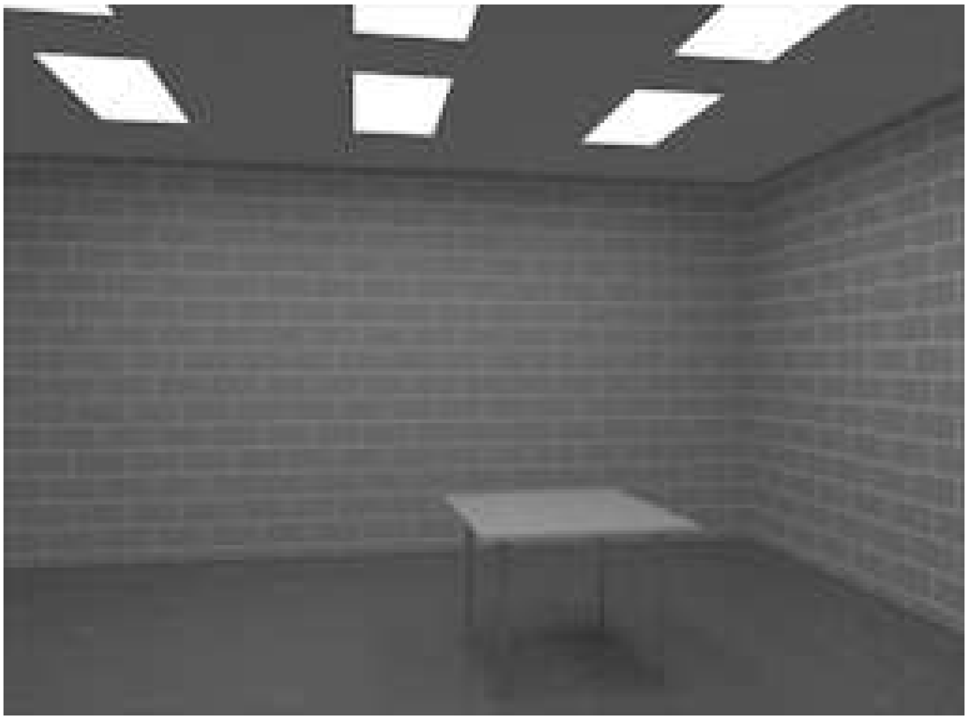

Figure 5.11: Room with one shadow ray per viewing ray.

the target light by using just ξ_i. Suppose we have two lights that contribute radiances of 1 and 9 at the target point. If $\xi_1 < 0.1$, then the first light will receive the shadow ray. Otherwise the second will get it. If the first light receives the ray ($\xi_1 < 0.1$), then we know that $(10\xi_1, \xi_2)$ are a pair of canonical random numbers, and this pair is used to choose a spot on the target light. This idea can be generalized to N lights by setting up N intervals for ξ_1 and dividing by the width of the chosen interval.

Figure 5.11 shows a room lit with nine lights. Each of the 16 viewing rays produced only one shadow ray. Figure 5.12 shows an art gallery lit by 5 spotlights with one shadow ray for each of the sixteen viewing rays per pixel. Since the spotlights are very directional, most locations send almost all shadow rays to one light. Both figures have indirect lighting calculated by the zonal techniques discussed in the next chapter.