# IVCam: Rendering with Real-Lens Optics

**David M. Sirkin, Alex C. Snoeren, and David C. Zhang**

**6.837: Introduction to Computer Graphics**
**Massachusetts Institute of Technology**
**{sirkin, snoeren, davidz}@mit.edu**

## Abstract:

This paper presents a set of Ivray extensions to render scenes using a physically-based camera model. By using actual lens prescriptions, the resulting renderer, Ivcam, is able to accurately reproduce lens effects including focus, depth of field, barrel distortion, aperture, and exposure. Ivcam replaces the pinhole camera used in Ivray with a film plane and lens system and simulates the propagation of individual light rays through the various lens elements and aperture stops.

The Ivcam UI contains extensions to support lens selection and focusing functionality. The Inventor camera parameters are continually updated to present an approximate preview, although Inventor is clearly unable to accurately reflect many of the effects. We show sample images rendered using a variety of lens systems, including fish-eye, telephoto, double-gauss, and wide-angle. In addition, images rendered with a wide-angle lens are compared to actual photographs of the scene using a similar lens.

## Introduction

Ivray ray-traces scenes as if they were viewed through a pinhole lens onto a pinhole focal plane. The result is that the focal length is fixed and the entire scene is clearly focused with no distortion. Real cameras have variable focal lengths, apertures and focal planes with non-trivial area and finite depth-of-field (DOF).

Composing a photograph can be considered a balancing of these four effects. Wide-angle lenses cover a larger viewing angle than normal or telephoto lenses from the same subject distance, but also increase the perceived depth between foreground and background. Moving the camera position can compensate for lens coverage, but with the result of a change in perspective. The effect can be used to expand or flatten apparent depth in a photograph.

Small apertures create larger DOF from the same subject distance with the same focal length, but also decrease the light intensity on the film plane. Opening up the aperture can compensate for exposure, but with the result of a change in DOF. The effect can be used to de-focus distracting backgrounds from an important foreground subject.

Lenses of various sort create their own unique image distortions. Lenses with large surfaces or many elements suffer from chromatic abberation, barrel or pincushion distortions. Lenses with small surface areas (or equivalently, small apertures) suffer from diffraction and loss of critical focus. In practice, photographers have to trade-off perspective, focal length and exposure with lens- and aperture-induced distortions. Our project incorporates the influence of such camera realities, allowing computer-generated scenes to be viewed alongside film or video more convincingly.

The remainder of this paper is organized as follows. Section 2 itemizes the specific goals of this project. The following section describes the large subset of these goals that were achieved, and details many of the technical issues difficulties encountered along the way. The contribution of each team member is listed in Section 4, and the paper concludes by discussing some of the many lessons learned during the course of the project.

## Goals

Our goal was to extend Ivray to capture the various effects of lens imaging by developing a new application, Ivcam, which wraps around the basic ray-tracing functionality of Ivray. Ivcam's user interface would allow the user to adjust various aspects of the camera model, including lens structure, focus, and aperture.

Specifically, we hoped to account for the following four effects:

1. Perspective variations due to distance between the subject and lens.
2. Depth-of-field (DOF) of sharp focus, which depends on factors such as image size and distance, lens focal length, and aperture.
3. Exposure variations due to the radius and geometry of lens elements and aperture, including vignetting.
4. Reduced image quality due to distortion (diffraction and abberation) introduced by the lens design and aperture.

**Figure 1:** Simulated Wide-Angle

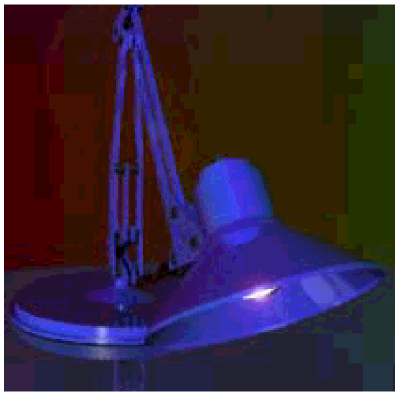
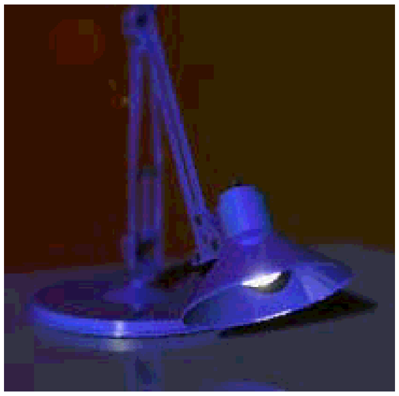


**Figure 2:** Simulated Telephoto



Figures 1 and 2 demonstrate the type of effects we set out to capture. These images were not rendered, but rather manually doctored to simulate the effects of photo-realistic rendering. Both images are focused on the plane of the light bulb, and have equally-sized bases.

Figure 1 simulates the results of using a wide-angle lens. The DOF covers the entire scene, perspective is very strong, and there is significant barrel distortion (note the curvature in the armature). For comparison, Figure 2 shows the same image through a telephoto lens. Here, DOF is very shallow (note the out-of-focus armature and front shade edge), there is little sense of perspective, and distortion is minimal.

Our initial research produced two separate approaches, which we hoped to implement and compare, both to each other and to actual photographs. The obvious approach involved interposing actual lens elements in the scene between the eye and world-space objects. Assuming the ray-tracing engine properly modeled the optical effects, this should produce images similar to those captured with a camera. The second, more rigorous method involved modifying the camera model to include a lens system, and accurately simulating the radiometry inside the camera itself. By accounting for the lens effects inside the camera itself, we hoped not only to see a speed increase, but more accurate results as well.

## Lens Rendering

The first approach is straightforward. One could use a standard rendering engine (which correctly handles diffraction calculations for multiple adjacent or wholly-contained objects-*note the current implementation of Ivray does not do this*), and then render through lenses in world space. This involves modeling lens objects in Inventor.

While we believe many aspects of a real lens system can be modeled in this fashion, the additional computations required to traverse a lens system for each ray cast adds significant calculation to the rendering process. Furthermore, without intelligent super-sampling methods, various subtle characteristics of real lenses, such as variability in exposure and aperture may not be captured.

## Camera Model

A second, more sophisticated approach involves including the camera model in the rendering engine itself. This technique was used in previous work by Kolb, Mitchell, and Hanrahan [#!kolb!#]. Basically, we would model the camera as a lens system and a backplane, and accurately calculates the irradiance on the film from the incoming radiance of a rendered scene. This requires simulating the geometry of rays passing through the lens system, and appropriately sampling the the system to reflect the radiometry.

## Assumptions

Note that while we believe our project produced quite dramatic results, there are several aspects of lens photography that we did not, and never intended to capture. These included:

- *Chromatic abberation*: Physical lens systems often distort light rays in varying amounts dependent on the wavelength. This is often characterized by the V-number of a lens, which describes the change in the index of refraction with wavelength.

  While not terribly difficult to incorporate, it would have required us mapping RGB values to wavelengths, which is problematic given the varying gamma values of different video card and monitor combinations. Instead, we chose to refract all light based on the index of refraction at the sodium *d* line (587.6 nm), which is the canonical index of refraction in optical texts.

- *Parabolic Lenses*: While many optical effects can be captured by spherical and planar lens systems, some sophisticated imaging equipment utilizes parabolic and other non-spherical lens elements. Our camera model is fully capable of handling such components, but the required calculations would have considerably increased the level of complexity of the computations. On a more pragmatic level, we did not have any references describing non-spherical lenses, and hence would have been lacking for lens data.

- *Film Characteristics*: Image formation is greatly affected by the various attributes of the photographic film in use. While an interesting area for exploration, we decided to assume a film plate with infinite fidelity with regard to our display device. Similarly, we assume saturation varies linearly with exposure time, and is independent of wavelength. Standard film is 36x24mm, so we chose to render our default images as 360x240 (8 bit color, 10 pixels per mm).

## Projected Schedule

The project milestones were defined by the following checkpoints; each checkpoint is labeled with several tasks to be completed by the associated checkpoint.

1. **October 29:** Project proposal with time-line and division of labor.

2. **November 5:** Develop camera model and derive optical equations to accurately model a lens system, including diffraction, focal length, and depth-of-field. Establish coding conventions, begin Ivcam development by extending Ivray with skeleton GUI components.

3. **November 12:** Add diffraction focal length, and depth-of-field functionality. The result should be a fully-functional Ivcam tool. Develop a tool to generate accurate object models of a lens system.

Use the generated lenses to render images with the standard Ivray program.

4. **November 19:** Find or build an accurate Inventor model of a real scene. Obtain photographs of scene to compare with Ivcam and Ivray output. Add additional effects such as aperture and its effects on exposure, vignetting to Ivcam.

5. **December 3:** Render high-quality scenes for presentation and final report to demonstrate features of ivcam. Extend the ivcam camera model to support temporal effects, such as multiple exposure and motion blur. Final report submission and project presentation.

## Revisions

After receiving feedback on our initial project proposal and beginning work on the project, it became readily apparent that our first proposed approach, modeling lenses to be interposed between the eye and the scene, was both inadvisable and unworkable.

Inventor does not (to our knowledge) support rendering objects that consist of sphere sections. Using any polygon-based approximation is clearly unworkable, as the calculation of surface normal is only accurate to some arbitrary degree of precision, which may be insufficient for the resolution of the desired image. Furthermore, any fairly precise approximation would consist of such a high number of polygons that rendering time would be intractable.

Hence it appeared impossible to model spherical lens elements in Inventor without extending the modeling language. While certainly possible, given the benefits of the alternate approach in terms of calculation efficiency and accuracy, it was decided to abandon the first approach.

Additionally, given our film-plate assumptions with regard to linear saturation, any multiple exposure and motion blur effects can be equally well simulated by post-processing multiple rendered images, hence there is no additional benefit for incorporating the functionality into the camera model itself.

# Achievements

**Table:** A wide-angle lens prescription [#!smith!#, figure 18.5]

| radius | thick | $n_d$ | ap |
|---|---|---|---|
| 163.579 | 5.529 | 1.540 | 107.8 |
| 53.169 | 45.435 | 1.000 | 81.8 |
| 59.487 | 23.301 | 1.772 | 56.2 |
| -102.877 | 8.042 | 1.617 | 44.6 |
| 322.991 | 3.720 | 1.000 | 41.6 |
|  | 10.353 | 1.000 | 39.8 |
| -51.312 | 0.523 | 1.000 | 41.6 |
| -758.075 | 14.073 | 1.713 | 48.4 |
| -34.505 | 6.031 | 1.805 | 52.0 |
| -76.210 | 18.094 | 1.000 | 55.8 |
| -35.013 | 5.529 | 1.617 | 61.0 |
| -54.424 | 64.930 | 1.000 | 81.8 |

Using the Ivray program (the solution to 6.837 Assignment 6) as our code base, we extended its functionality and user interface to more closely match that of a real, interchangeable-lens camera system. To evaluate the output of Ivcam, we reproduced a real-world scene (David Sirkin's living room) in great detail as an Inventor environment. Then the real scene and virtual model were photographed and rendered, respectively, with their respective camera systems. The resulting photographs (Figures 7, 8, and 9) and images (Figures 10, 11, and 12) allow qualitative comparisons of field-of-view, focus and exposure can be qualitatively compared for several different lens/view combinations.

The bulk of our development was done on Athena SGI workstations, using standard Institute resources. The intensive computational requirements of the rendering process, however, led us to also port the Ivcam code to Linux so additional computational resources at LCS could be employed. Our basic contributions can be enumerated as follows:

- As part of Ivcam, created data structures to represent multiple-element lenses and developed computational modules to:

    1. Calculate lens characteristics (focal length, exit pupil, etc)
    2. Adjust focus (zoom, positioning)
    3. Render images with accurate focal effects (exposure, super-sampling)
    4. Extend the existing UI (camera menu, focus plane, focus wheel and command-line options)

- Developed a visualization tool to plot lens diagrams from lens prescriptions

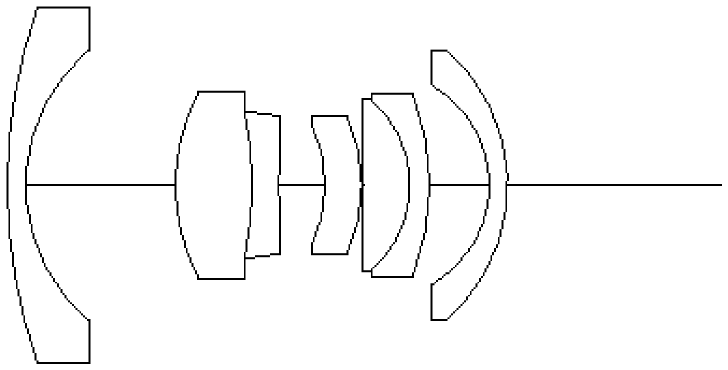
- Constructed a *detailed* and *accurate* Inventor model of Dave Sirkin's living room

- Carefully photographed and rendered the scene and model from an identical position using similar[1] lens systems.

## Lens System

We use optical lens prescriptions [#!smith!#] to describe the camera system through which we cast rays into scenes. A prescription is a table describing the geometry of that lens' component elements. Table 3 shows the prescription for the wide-angle lens depicted in figure 3.

**Figure 3:** The wide-angle lens in table 3



Each row represents a lens surface and each column represents that surface's radius, the axial distance to the next surface, the index of refraction of that distance (with respect to air), and the aperture. Our code reads through these prescriptions in both object-to-image and image-to-object directions, depending on whether we are rendering a scene or setting the focal distance. An advantage of using prescriptions to represent lenses is that we can then take advantage of the sequencing of the lens surfaces in our ray tracing, rather than using standard object intersection techniques.

Each prescription includes an aperture stop, which is a planar surface that limits the extent to which rays entering one side of the lens will exit on the other. Note that using prescriptions of this form limits us to using lenses comprised only of spherical or planar elements. Many modern lenses, and especially those with wider fields-of-view, use aspherical elements to correct for particular exposure or focusing problems.

### Simulation

In a fashion similar to the way optical designers develop lenses, we cast rays of light through our lens system in some direction and determine the direction those rays follow on their exit. To render a scene, we approximate the following process:

```
For all points, x, on the film plane,
                 For all points, y, in the hemisphere facing away
                 from the film plane, cast ray,
$R(x \to y)$:
        For each lens element $L_i$,
            If $R$, $L_i$ intersection outside $L_i$'s a
                $R$ never exits lens,
            Else
                Determine new $R$ usi
        Cast $R$ into the scene, computing $E$
    Integrate $E$ over $y$ to compute exposure at $x$
```

The derivation of this process, as well as our simulation methods and various approximations are discussed in the remainder of this section.

## Exit Pupil

When looking through the lens system from any point on the film plane, only those rays passing completely through each aperture stop in the lens system are visible; the rest are blocked, and assumed to have zero intensity. The image of the aperture stop as viewed from the film plane is defined to be the *exit pupil*. Clearly we need only cast rays at the exit pupil-any cast elsewhere will not impact the exposure at the film plane.

Calculating the actual exit pupil is non-trivial, however, as it requires imaging the stop through the portion of the lens system between the stop and the film plane. Even using a thick-lens approximation, this is still an expensive operation. As noted in [#!kolb!#], a thick lens approximation is only valid if the lens system distortion is negligible, such that a circular disk is a reasonable approximation from an off-axis point on the film plane, which is not necessarily the case for wide-angle lenses.

Rather than empirically compute an exit pupil through some sort of approximation method (we experimented with a Newtonian-flavor bisection approach), we note that we can safely assume the exit pupil to be the rear-most lens element. Clearly it cannot be any larger, so we err on the conservative side. The only concern, then, is we may cast rays through that portion of our exit pupil approximation that don’t actually make it through the actual exit pupil. We examine this case in the following section.

## Exposure

Combining our initial assumption that film saturation varies linearly with exposure time with an assumption that irradiance is constant over the duration of the film exposure, the exposure at any differential film element is simply the irradiance over that element times some constant factor.

Assuming instantaneous shutter action, the total irradiance at a differential element can be computed by integrating over the hemisphere of rays originating at that element. Note that only the solid angle subtended by the exit pupil has a non-zero contribution, so we can avoid casting any rays outside of the exit pupil. The partial integral of the exit-pupil rays must then be adjusted by the ratio of the area subtended by the exit pupil to the entire hemisphere. Luckily, since we assume the same size exit pupil at every point on the film plane, this is just a constant factor. We aggregate this constant factor into the exposure duration, which, given our film assumptions, is completely arbitrary.

With these simplifications, the exposure, *E*, at position *x* on the film plane is given by the integral below

[#!kolb!#], where *L*(*x*,*x'*) is the irradiation of a ray cast from position *x* to *x'* on the exit pupil, *D* is the exit pupil, $\theta$ is the angle of between the ray and the film plane normal, and $\theta'$ is the corresponding angle from between the ray and the exit pupil normal:

$$E(x) \approx \int_{x' \in D} L(x,x') \frac{\cos\theta \ \cos\theta'}{||x'-x||^2} dA \qquad (1)$$

Since the exit pupil is parallel to the film plane, this can be rewritten as:

$$E(x) \approx \int_{x' \in D} L(x,x') \cos^4\theta \ dA \qquad (2)$$

We now return to the case described above, when a cast rays fails to pass through the lens system due to a miss-approximation of the exit pupil. In this case, we consider that ray to have a zero contribution to the total exposure, but still count the ray in the super-sampling total. Given a suitably large number of super-sampling rays, this gives a reasonable approximation of the varying size of the exit pupil over the film plane. Additionally, it also captures *vignetting*, the blocking of light that occurs when rays are blocked by lens elements other than the aperture stop in some lens systems when entering the system at a wide angle.

## Sampling

In order to calculate exposure at each differential film element, equation 2 integrates over the exit pupil. Computing the exposure for an individual pixel requires yet another integration over the pixel area. This double integral could be estimated by sampling over a four-dimensional hypercube.

When the number of samples is small, however, sampling uniformly over the hypercube leads to a high degree of noise; stratified sampling improves the accuracy for low numbers of samples. Additionally, we sample separately over two separate two-dimensional domains using jittered sampling [#!shirley!#], taking several samples of the exit pupil for each sample point on the pixel element. This reduces the noise introduced by a particularly poor exit pupil sample.

This works well for accurately sampling across the square pixel element. The exit pupil, however, is a disk. As noted in [#!shirley!#], mapping a uniformly sampled two-dimensional point onto a disk is not as straightforward as it first appears. To be strictly correct, the mapping from unit square to disk should have a constant Jacobian, so that the points are equally dispersed about the disk. Due to the small number of sampling rays cast in our images (typically 16 per pixel), however, the obvious mapping,

$$r = \sqrt{u}, \ \theta = 2\pi v \qquad (3)$$

does not produce any objectionable effects for the images we've rendered. [#!kolb!#] suggests the transformation developed by Shirley [#!shirley!#], mapping concentric squares to concentric circles, provides even better results. Unfortunately, due to time constraints, we have not yet fully implemented Shirley's mapping.

## Focusing

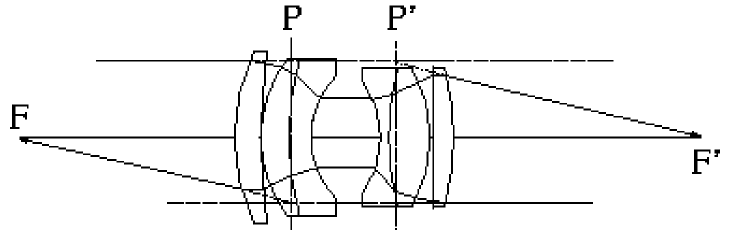
When focusing the camera on a fixed focal plane, there are two options: move the lens system, or move the film plane. Physical cameras generally move the lens system, as the camera (and the contained film plane) remain stationary during focusing. In our model, however, the front of the lens system is positioned at the Inventor eye location, so moving the lens system would correspond to moving the virtual camera. Instead, we chose to move the film plane.

In optics, a simple lens is often described by its focal points and principal planes. A focal point is where light rays entering one side of the lens intersect the axis on the other side. Where lenses are well-formed, rays entering one side that emanate from a single point will converge on the opposite side to a single point. We denote the focal point on the image-side of the lens as *F*, and on the object side as *F'*.

While the rays actually refract and change direction on both entering and exiting the lens, an *ideal thin lens* approximation is often used for optical calculations. In this case, the refraction is assumed to have occurred at the plane defined by the intersection between the paths of the incoming and exiting rays. The effective focal length of the lens is then considered the distance between this plane and the image-side focal point *F* (the front focal length is the distance to the object-side focal point *F'*).

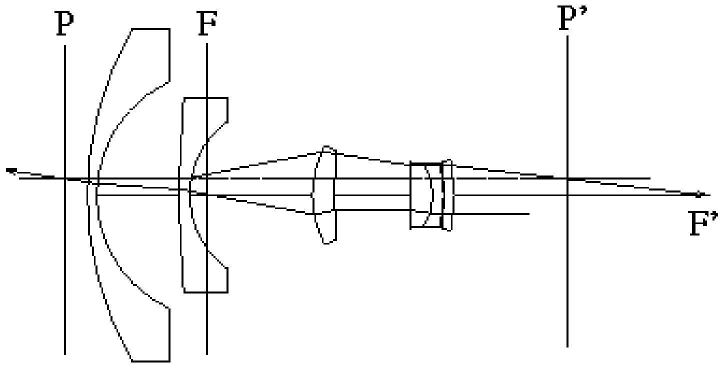
However, a complex lens composed of many elements will have many focal points and principal planes. We therefore use a use a *thick-lens approximation* for our exit pupil, exposure and focus calculations. In this case, the focal points of the system as a whole are used and the refraction is assumed to have occurred along two planes, one on the image-side denoted *P* and one on the object-side *P'*. Planes may not coincide and in fact may be on either side of each other, but the distance between them is always the thickness of the lens approximation. The focal lengths on image- and object-sides (*f* and *f'*) are then equal but of opposite sign.

**Figure 4:** Calculating principal planes



We find these points and planes by tracing axis-parallel rays through the lens system in both directions and determining the appropriate intersections, as shown in Figure 4. Unfortunately, it wasn't until about half-way through the project time-line that we learned the process is not the same for all lenses. Since principal points and planes may lie both within and without the lens system and may be in either order (that is, the principal plane may be *outside* the focal point), the testing for intersection must be generalized for either case. Figure 5 shows an example of this effect in a fish-eye lens.

**Figure 5:** Finding focal length in fish-eye lenses



Given the focal length and principal planes of a lens system, we can now focus the system at a particular point in the scene. We developed two methods for adjusting the film plane: one computes a distance to move the *lens system* relative to it's position when focused at infinity (this distance, known as the Back Focal Length, or BFL, is given in the lens prescription), the other positions the *film plane* absolutely based on first principals of the lens system.

In the absolute method, given a point in object space a distance *z* along the axis from plane *P*, the following fundamental lens equation holds [#!smith:optics!#]:

$$1/z' - 1/z = 1/f' \qquad (4)$$

where *z'* is the image space distance along the axis from plane *P'*. Since *z* and *z'* are measured from different planes, we introduce another variable *t* as the difference between them.

We can use *t* in relative positioning, which translates the lens (not the film plane!) from an initial position focused at infinity in order to pull-focus onto a desired point in object space. To find the distance *T* we have to translate from the film plane, we use the conjugate equation to derive [#!kolb!#]:

$$T^2 + T(2f' + t - z) + f'^2 = 0 \qquad (5)$$

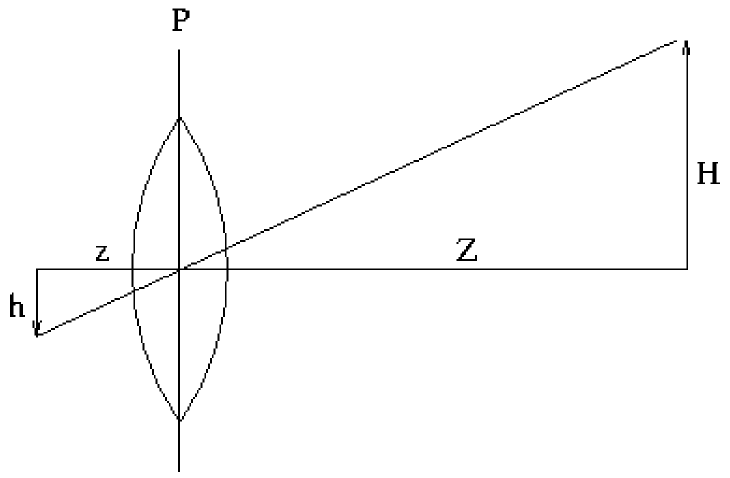
The two roots correspond to the lens being closer to the image or the object. We assume that the former will always be the case and so we select the smaller of the two solutions.

While either focusing method would work, we initially chose to position the lens system at a fixed point in space-the Inventor eye space. Hence moving the lens during focusing proved problematic. Instead, we move the film plane using the absolute positioning method discussed previously.

As the distance between the film plane and back lens varies with focus, the image disk subtended by the cone of light that is cast onto the film plane changes in size accordingly. Our system accurately represents this effect as an apparent change in field-of-view.

## Zoom

**Figure 6:** Computing the height angle



In order to present an accurate preview of the rendered image in Ivcam, it is necessary to appropriately adjust the height angle of the camera. Computing the height angle of a thin lens approximation is straightforward. As can be seen in Figure 6, the distance to the film plane, *z* and the focal plane, *Z*, as measured from the principal plane, *P*, the height of the film plane, *h*, and the height of image of the film plane at the focal plane, *H*, are related by similar triangles [#!smith:optics!#]:

$$h/z = H/Z \tag{6}$$

The height angle, $\theta$, is then given by:

$$\theta = 2 * \arctan(h/z) \tag{7}$$

In Ivcam *h* is fixed at 12mm (since our film plane is 36x24mm), and *z* is computed as described previously in order to focus the system (with appropriate adjustments for our thick lens). Hence $\theta$ can be computed with minimal effort. Unfortunately, the resulting angle accurately reflects the formed image when only projected from the principal plane of the lens system. The Inventor camera is physically located at the front of the lens system. This introduces a slight error in the preview, which becomes increasingly noticeable as the camera is moved very close to objects.

This disparity could be corrected by placing the Inventor camera at the principal plane of the lens system rather than the front of the lens. Associated modifications to the clipping planes would also be required. Due to the time constraints of the project and the minimal error introduced by this adjustment, it is not being accounted for at this time.

## UI Extensions

We modified the Ivray's scene viewer interface mainly in three ways. First, we added a camera menu, from which several lens choices are available. Secondly, we extended the render area to show the focal distance as a plane in world space. Lastly, we added a thumb wheel used to control the focal distance.

### Camera Menu

Adding the camera selection menu is straight forward. When a selection is made, Ivcam immediately updates the Inventor preview area to reflect the correct preview for the selected lens. This is done by updating the height angle for the Inventor perspective camera node using the calculations discussed in section 3.2.1. This is a useful feature for comparing the views from different lenses.

### Focus Plane

In a physical camera such as a single-lens reflex (SLR), focusing is done by looking through the view-finder and adjusting the lens until the desired object is in focus. This is not possible with Ivcam since Inventor's hardware-supported rendering does not include focus support, and ray-tracing an image with focal effects is too costly to compute interactively. We realized early on that focusing in Ivcam would be difficult without seeing the focus effect immediately.

Our solution is to insert a semi-transparent ‘‘plane’’ into the scene, representing the location of the focal plane, which Inventor can render and animate interactively. While the focal plane is actually a spherical manifold centered about the camera, it was decided that the plane provided a sufficient level of accuracy for a preview mode.

In an attempt to depict the distance to the focal plane, we shrink the plane as it moves away from the camera. This provides feedback regarding the distance to the plane. Unfortunately, it presented a more practical problem, since the plane may disappear behind objects. To alleviate this, David Sirkin suggested using a both an infinite plane and co-planar scaled region for intuitive distance cuing.

### Focus Wheel

The focus plane was initially implemented using a `SoTranslate1Dragger` node, which required the user to drag the plane inside the render area. Since the user drags in the X-Y plane, while the plane moved only in the Z direction, focusing became quite counter-intuitive. Instead, we now provide a thumb wheel dedicated to moving the focal plane.

We then further discovered moving the focal plane with constant speed makes focusing rather tedious, especially in a large scene. This was corrected by increasing the speed of the plane proportional to the speed of thumb wheel rotation.

When focus changes, it is necessary to update the preview area to reflect the new viewing angle.

### Command Line Options

While the enhanced UI makes it very easy to interactively compose an image to be rendered, rendering large numbers of images in this fashion can be a tedious process. Especially when computing frames in an animation, it is desirable to support command line methods of controlling the lens configurations.

To this end, we added the following command line switches to support automatic image generation:

- **-f** ***distance*** sets the focal plane to the specified *distance* away from the camera in meters.
- **-z** ***factor*** scales the selected lens by the specified zoom factor. By linearly scaling the lens system, it can be forced to have arbitrary EFL. Our supplied lenses all have an approximate EFL of 100mm.
- **-o** ***file*** forces Ivcam to save the generated image to the specified *file*, rather than an automatically generated file name.
- **-headlight** toggles the headlight attached to the camera. This simulates flash photography. In contrast to Ivray, the default is *off*, simulating a camera without flash.
- **-batch** starts Ivcam without invoking the GUI and automatically begins ray tracing. This is especially useful for background rendering.

## Porting

While not part of the original proposal, Ivcam was ported to Linux. Rendering a standard 360x240 frame of the complexity seen in this report with full recursive shading and super-sampling takes almost 2 hours on an Athena SGI O2. While optimized for graphics applications, the MIPS processors found in O2s are not known for their floating point speed. Floating point isn't exactly Intel's forte, either, but empirical evidence shows that a 450Mhz Pentium II renders almost 50% faster.

The speed increase, plus the ability to work from the comforts of his office rather than an Athena cluster, spurred Alex Snoeren to port Ivcam to Linux. Using Mesa [#!mesa!#], a publicly available OpenGL library, LessTif [#!less!#], a free Motif clone, and a commercial Linux implementation of OpenInventor [#!tgs!#], he succeeded in building a Linux version of Ivcam that was used to render a large fraction of the images in our final presentation and report.

## Scene Modeling

We selected our test scene in order to demonstrate the effects Ivcam creates and its convenience for modeling and photographing. We settled on an antique telephone and textbook (Smith's *Modern Optical Engineering* [#!smith!#]) sitting on opposite ends of a tiled coffee table. Real-world and Inventor scenes were photographed using optics with focal lengths of approximately 20 (Figures 7 and 10), 60 (Figures 8 and 11), and 180 millimeters (Figures 9 and 12).

Unfortunately, camera manufacturers guard their lens prescriptions as trade secrets so we were unable to get the specifications for the particular lenses used. The result is that there should be minimal difference between the scenes in focus and depth-of-field (since the focal lengths are similar) but the perspectives may be somewhat different (as the number and position of individual lens elements will vary).

Note how in both photographed (Figures 7,8, and 9) and rendered images (Figures 10, 11, and 12) focused on the foreground telephone, as the focal length increases:

- Perspective flattens
- Depth-of-focus decreases, and
- Vignetting decreases

In the photographed and rendered images, when pulling focus between foreground telephone and background book:

- Image blur shifts and
- Field-of-view changes

# Individual Contributions

Our project proposal presented an initial division of work which, not surprisingly, turned out to be completely irrelevant. The following is our attempt to more accurately describe our individual contributions to the project. Absent from the individual sections, however, is the large amount of time we all spent helping each other understand the physics that underlie each of our individual tasks.

## David M. Sirkin

David was charged with the core focusing functionality (Section 3.2). He wrote the code that finds the intersections that comprise the principal points and planes for the general case (telephoto, normal and fisheye lenses) and returns the absolute or relative lens positioning. This involved considerable code degugging (to find the cause that required a generalized solution), library research, and some creativity (to find the eventual solution itself).

Since the module interacts directly with Alex's raytracing code and David Zhang's object-to-filmplane caculation code, David worked closely with the other teammates during the latter stages of the project. Meanwhile, he spent many hours researching lenses and digging through old library stacks to turn up the various prescriptions used in our implementation (Section 3.1).

He painstakingly measured, built, modeled, and photographed our beautiful Inventor demonstration scene, a life-sized mock-up of his living room as descried in Section 3.5. As the photographer in the group, David was also called upon to explain where possible the real-world implications of what the optical engineering underlying Ivcam implied.

David also wrote much of the original project proposal and put together the proposal web page, including doctoring Ivray-generated images to simulate the eventual Ivcam output.

## Alex C. Snoeren

Alex's main responsibility was the core lens-tracing functionality (Section 3.1.1). He developed the code that traces individual rays from the film, through the lens system, and into object space. This included developing the super-sampling and exposure techniques (Sections 3.1.3 and 3.1.4). Using the computed focal distance and lens database as input, the lens tracing routine generates a standard 36x24mm film image of the scene as viewed from the specified camera position.

Due to the complex interaction between the main lens tracing code and the other components of the system, Alex was also extensively involved in the initial design and debugging of the focusing subsystem, although the coding and extensions to support fish-eye lenses were David Sirkin's work.

In addition, Alex initially did some limited UI hacking, researching where in the Ivray code our extensions would need to be made. While the vast majority of UI modifications were done by David Zhang, Alex made the first modifications to the menuing system to include Camera choices.

In order to support the batched rendering process required to automatically generate the frames used in our pull-focus animation, Alex added the additional command line switches described in Section 3.3.4. This includes programmatic support for linearly scaling the defined lenses to any EFL.

Alex authored the bulk of the final report, although each member wrote the sections describing their own work, and everyone contributed to the final editing and project presentation.

Finally, in what started as a side project, Alex was solely responsible for the Linux port (Section 3.4) which turned out to be a tremendous time saver. The Linux version running on a Pentium II substantially outperformed any of the SGI machines available to us, which allowed us to considerably decrease the rendering time.

## David C. Zhang

David's main responsibility was to study Inventor and Motif widgets in order to modify the user interface. David read the Inventor Mentor [#!invm!#] from front to back trying out different combinations of sensors and engines, finally implementing those used them in the UI.

David was also responsible many small hacks in Ivcam to correct clipping planes, change the default headlight behavior, integrate the calcuated zoom factor with the preview mode, and change default window size. His discovery of the meter as Inventor's internal unit of measurement was fundamental in the correct scaling of lens sizes.

David also created the initial lens database, and wrote a visualization tool that plots lens diagrams (see

Figures 13-16) in xfig automatically from the our lens database.

## Lessons Learned

We all learned a great deal about optics, both theoretically and how cameras as physical mechanisms operate-as Alex puts it, ‘‘this is the best optics class [we’ve] ever taken.’’ On a related note, we also learned about photography, both as an aesthetic and mechanical process. In preparing for our pull-focus animation demonstrated during our project presentation, we considered several alternative functions for the rate at which focus moves through the scene. We also generated a number of test scenes to use as a storyboard for determining the eventual choice. The process is just what animators or cinematographers use to design their shots, so we got some unexpected experience in that regard as well.

We believe that our rendered Inventor scenes compare quite favorably with the photographed scenes. We expected the perspective to be somewhat inconsistent due to the differences in lens prescription, but otherwise the focus, exposure and vignetting are represented quite accurately. David Sirkin was especially surprised to see his living room come to such accurate representation in the virtual world.

We were somewhat disappointed that we couldn’t get access to prescriptions for our particular lenses, hence we could not make quantitative comparisons between scenes. As pointed out by professor Teller, a future project might be finding optics that we can match to prescriptions so a more accurate comparison could be made. Such a rigorous comparison would surely turn up more effects that are currently unaccounted for.

In terms of pragmatics, nothing can quite prepare you for what it means to need 8 *un-interrupted* hours of CPU time on Athena until you try it. It’s quite obvious that a practical implementation of our approach for larger images would require a distributed implementation, possibly based on MPI. We should also note that much pain could have been saved by heeding Professor Teller’s advice and developing a more robust lens ray visualization tool, which might have allowed us to avoid many hours of debugging, manually comparing ray origins and directions and finding intersections with lens diagrams. In or haste to limit the scope of our project, which already was way too large, we struggled through with a less robust visualization tool.

In conclusion, we are quite excited by our results. While many 6.837 projects produce flashy simulations or rendered animations, the end result is exactly that: a single artifact. What we set out to do, and accomplished quite successfully, was to develop a fundamental mechanism for rendering graphics with certain effects. Our project had a *significant* modeling and animation component as well, to be sure-we spent over 300 CPU hours rendering our pull-focus video. But the tool we used to create it, Ivcam, can also be used to render *anyone’s* models.

## Acknowledgements

The authors would like to thank our TA, Charles Lee, for his guidance during the course of this project, and Damian Isla for making the full sources to Ivray available. We are also indebted to the kind folks at Template Graphics Software for providing us with a free 30-day license for their Linux OpenInventor

library for educational use.

2

# Appendix

Ivcam is available from a CVS repository in our 6.837 team locker, `/mit/6.837/F99/team15/CVS`. Checking out the module `ivcam` will create the following directories:

- `ivcam/` contains the source code to Ivcam and the following subdirectories:
- `ivcam/camera` contains the camera datafiles used to generate the images corresponding to the photos in this report. Invoke Ivcam with the `-camread` option to utilize them.
- `ivcam/docs` Contains the LaTeX source for this report and all of the figures included therein. `docs/figs` also contains the source code for David Zhang's lens visualization utility.
- `ivcam/images` Contains the rendered images used in this report. They were created by invoking Ivcam in the following fashion: `ivcam -batch -j 16 -s r -f # -z # -camread camera/cam-#.dat scene/livroom.iv`
- `ivcam/photos` contains David Sirkin's actual photographs, both original and re-touched.
- `ivcam/presentation` holds the HTML code for our presentation
- `ivcam/scene` stores the `.iv` files for our sample scene.
- `ivcam/ivfiles` is left over from our debugging; it contains the `.iv` files distributed with 6.837 Assignment 6.
- `ivcam/ivwidgets` and `ivcam/vwraplib` are the supporting code for the Ivcam UI

Additionally, our team locker contains the directory `/mit/6.837/F99/team15/images` which holds both the pull-focus video shown during our presentation and the 200-odd still frames that compose it. The `csh` script `ivcam/pull-focus.sh` will generate the frames (it can be run on multiple machines at the same time), which then are aggregated together to create the final QuickTime movie.

# About this document ...

**IVCam: Rendering with Real-Lens Optics**

The command line arguments were:
**latex2html** `-no_navigation -split 0 paper.tex`.

The translation was initiated by Alex C. Snoeren on 1999-12-03

---

**Footnotes**

... similar[1]
: We were unable to obtain prescriptions for any of the lenses in our possession. Hence the rendered images use lenses of the same general class (with similar EFLs), but not exactly the same as those used in the photographs

---

*Alex C. Snoeren*
*1999-12-03*