Dynamically Reparameterized Light Fields

Aaron Isaksen
MIT LCS Computer Graphics Group
aisaksen@graphics.lcs.mit.edu
http://graphics.lcs.mit.edu/~aisaksen

Leonard McMillan
MIT LCS Computer Graphics Group
mcmillan@graphics.lcs.mit.edu
http://graphics.lcs.mit.edu/~mcmillan

Steven J. Gortler
Harvard University
sjg@cs.harvard.edu
http://www.cs.harvard.edu/~sjg

Abstract

An exciting new area in computer graphics is the synthesis of novel images with photographic effect from an initial database of reference images. This is the primary theme of image-based rendering algorithms. This research extends the light field and lumigraph image-based rendering methods and greatly extends their utility, especially in scenes with much depth variation. First, we have added the ability to vary the apparent focus within a light field using intuitive camera-like controls such as a variable aperture and focus ring. As with lumigraphs, we allow for more general and flexible focal surfaces than a typical focal plane. However, this parameterization works independently of scene geometry; we do not need to recover actual or approximate geometry of the scene for focusing. In addition, we present a method for using multiple focal surfaces in a single image rendering process.

Introduction

The light field [Levoy96] and lumigraph [Gortler96] rendering methods use similar four-dimensional data structures for representing a half-space of rays through a volume of space. We will refer to this data structure as a *ray database*. Novel images are synthesized from this database by querying it for each ray needed to construct a desired view. The set of viewpoints that can be generated are restricted to those within an empty region of space lying outside of the convex hull of objects in the scene that are composed entirely of rays from the selected half-space. Several ray databases can be used to represent a scene, and desired images may combine rays queried from different databases. A pair of planes is typically used to parameterize a ray database, although other parameterizations have been suggested [Camahort96].

A continuous representation of a ray database would be sufficient for generating any desired viewpoint under the previously described viewing restrictions. However, continuous databases are impractical or unattainable for all but the most trivial cases. In practice, we must work with a discretely sampled ray databases. As with any sampled representation the issues of choosing an appropriate initial sampling density as well as defining methods for reconstructing continuous representations from the given sample set are crucial factors in representing the underlying model. A previous motivation for selecting a new parameterization for the ray database was to facilitate better or more uniform sampling. Improving the uniformity of sampling density helps find an adequate sample rate to avoid aliasing artifacts. These artifacts due to an initial undersampling cannot be removed though a subsequent process unless additional information or constraints are provided.

The choice of a ray database parameterization also affects the choice of reconstruction methods that can be used in synthesizing desired views. Thus, even when supplied with an adequately sampled dataset, it is frequently the case that a non-ideal reconstruction filter will introduce artifacts into the result, whereas a better reconstruction filter on the same dataset might have generated a more correct result. This process of introducing artifacts in the reconstruction process is often called *postaliasing* [Mitchell88]. Postaliasing artifacts include excessive high-frequency leakage, sometimes called ringing, and excessive pass-band attenuation, or blurring. The standard planar parameterizations of ray databases have a substantial impact on the choice of reconstruction filters. Here, we present an alternative parameterization that allows for a more flexible choice of reconstruction filters.

To date, most light fields are constructed for object-centered, or outside-looking-in, environments rather than viewer-centered, or inside-looking-out environments. This is not entirely coincidental: the original two-plane light field can best represent points that are located near the exit plane of the ray database. Objects located a small distance from this exit plane will appear out of focus (either blurred or ghosted, depending on the extent of aperture filtering). Thus, an object-centered model is better suited, as the distance the object lies from the exit plane is well represented by a plane.

We would like to represent inside-looking-out light fields with a wide variations in depth. This requires a more flexible parameterization of the ray database.

In the sections that follow, we present an extension of the light field parameterization that introduces the notion of a focal surface. Then, we discuss how the treatment of a light field as a discrete synthetic aperture camera will provide dynamic variations of depth of field. Next, we explain how moving and orienting the focal surface will affect the images created using this ray database. We then present the idea of using multiple focal planes, how to create them, and how to use them when rendering. Finally, we present ideas on how one would optimally make these multiple focal planes.

Focal-Plane Abstraction

Overview

Our parameterization of ray databases is analogous to a two-dimensional array of pinhole cameras treated as a single optical system with a synthetic aperture. Each constituent pinhole camera captures an image in clear focus, and this camera array acts as a discrete aperture in the image formation process. Because we have a discrete, finite aperture, some amount of depth of field defocusing will be present in our renderings. However, by using an arbitrary plane of focus, we can establish correspondences between the rays from different pinhole cameras. That is, we can control which items we want to be in focus. This is essentially the approach used to simulate depth-of-field effects in ray-traced images [Cook84].

Mathematical formulation

In the standard two-plane ray database parameterization there is an entrance plane, with parameters *(s, t)* and an exit plane with parameters *(u, v)*, where each ray is uniquely determined by the 4-tuple *(s, t, u, v)*, as illustrated in Figure 1. It is often instructive to consider and/or interpret subspaces of such a ray database [Gu96]. A two dimensional subspace given by fixed *s* and *t* values resembles an image, whereas fixed *u* and *v* values give a hypothetical radiance function. Fixing *t* and *v* gives rise to an epipolar plane image, or EPI [Bolles87].

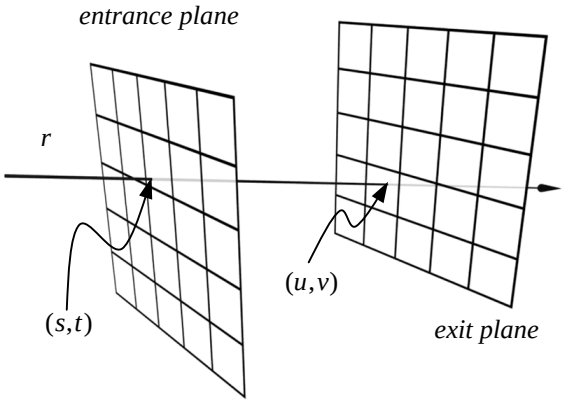

Figure 1: In the standard light field parameterization, a ray is referenced by its intersections with an entrance plane and an exit plane.

Our new parameterization is best described in terms of three 2-D surfaces, which are shown in Figure 2 below. Our *camera surface*, described in terms of two parameters *s* and *t*, is identical in function to the entrance plane of the standard parameterization. Our *image surfaces* describe a discrete set of rays from a given point, (*s*,*t*), on the camera surface and has the form ($u_{s,t}$, $v_{s,t}$). The elements of the ray database are accessed via a four-tuple *(s, t,* $u_{s,t}$, $v_{s,t}$*)*. Our focal surface is described in terms of two parameters, (u_F, v_F), that are independent of all others. Our parameterization also requires a mapping $M_F : (s,t,u_F,v_F) \rightarrow (u_{s,t},v_{s,t})$; this maps from focal surface parameters to image surface parameters given a specific camera surface coordinate. That is, this mapping tells us which ray *(s,t,*$u_{s,t}$,$v_{s,t}$*)* in the ray database is the same as the ray (s,t,u_F,v_F).

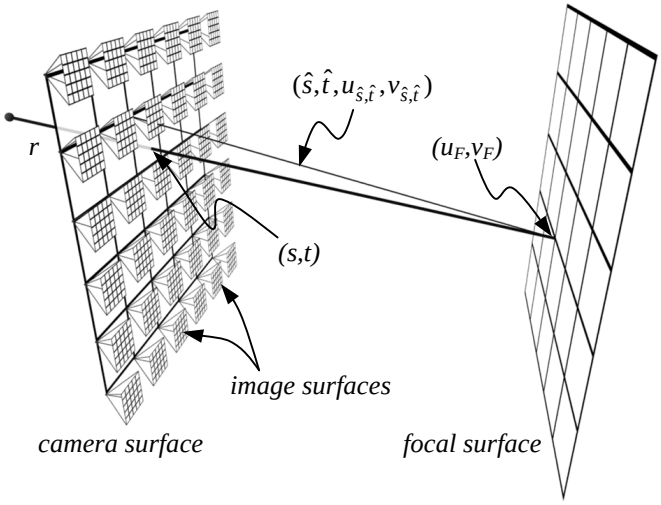

Figure 2: Our new parameterization has added a focal surface.

When querying a discrete ray database, we wish to find the ray $\hat{r}$, a ray that has been recorded in the ray database, that best approximates a ray *r*. Given *r* and a focal surface *F*, one calculates the ray's intersections with the camera surface and focal surface to get (s,t,u_F,v_F). Then, (s,t,u_F,v_F) is quantized to $(\hat{s},\hat{t},u_F,v_F)$, because the camera surface is sampled discretly, not continuously. This 4-tuple is passed throught the mapping $M_F : (\hat{s},\hat{t},u_F,v_F) \rightarrow (u_{\hat{s},\hat{t}},v_{\hat{s},\hat{t}})$ to obtain the nearest ray $\hat{r}$ in the ray database which passes through $(\hat{s},\hat{t})$. By varying the focal surface and the focal surface mapping, different rays $\hat{r}$ will be returned for a given ray *r*.

The key difference between our ray-database query formulation and that used by the light field and the lumigraph is the identification of independent image and focal surfaces. In the case of a standard two-plane parameterization the mapping function from *(*u_F,v_F*)* to *(*$u_{s,t}$,$v_{s,t}$*)* is an identity. Therefore, every point on the camera surface shares a common image surface, and the image surface is coincident with the focal surface.

However, we have separated these surfaces, and the relationship between them is determined by the focal-plane-to-image-plane mapping function. This mapping can be modified dynamically, and these alterations do not effect the organization of the underlying ray database. Thus, we defer the selection of the focal surface until image synthesis time, and make the specification of this surface available to the user.

The addition of a focal surface abstraction has only a minor impact on the image synthesis process. Assume that the center-of-projection of the desired image lies at the origin. In the case where the camera, image, and focal surfaces are defined as planes the mapping from a desired ray to a database query can be structured as a pair of projective mappings.

$$\begin{bmatrix} rs \\ rt \\ r \end{bmatrix} = \mathbf{C}\bar{d}$$

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \mathbf{I}_{\hat{s},\hat{t}}\mathbf{F}\bar{d}$$

In these equations, $\bar{d}$ is the direction of the desired ray. The 3 by 3 matrix **C** gives the intersection of the ray with the camera plane in terms of *s* and *t*. Likewise, the matrix **F** gives the (u_F,v_F) intersection of the ray with the focal plane. The 3 by 3 matrix $\mathbf{I}_{s,t}$ maps focal-plane coordinates into pixel coordinates of the specified camera, $(\hat{s},\hat{t})$. These mappings are computed for each image synthesized. In light fields and lumigraphs, all points on the camera plane share a common image plane that is also the focal plane. Therefore, only two mapping functions are required, one for the entrance plane and one for the exit plane. In our parameterization the focal surface is defined dynamically. Thus, **F** is determined by the user. The composite map, $\mathbf{I}_{\hat{s},\hat{t}}\mathbf{F}$, must be determined for every discrete position on the camera plane where an image plane is specified, for example at 256 points. This quantity can be computed lazily, but in any case it is only a small overhead compared to the ray queries.

Discrete Synthetic Aperture Camera

When quantizing $(s,t) \rightarrow (\hat{s},\hat{t})$, the nearest ray $\hat{r}$ is constrained to pass through $(\hat{s},\hat{t})$. Clearly, except in the case where $(s,t) = (\hat{s},\hat{t})$, the ray $\hat{r}$ is not the same as *r*, and errors in the output image are apparent. Evaluating M_F only once for each ray *r* leads to noticeable discontinuities in the output image, as the quantization $(s,t) \rightarrow (\hat{s},\hat{t})$ suddenly jumps to a new location on the camera surface, even when (s,t) may only change a small amount (see figure 3 below).

To reduce these discontinuities, one can ask for the four rays from the four nearest $(\hat{s},\hat{t})$ samples. Then one can interpolate between these rays to find a better approximation than any one of these rays alone. In the original light-field paper, this was referred to as *st-interpolation*. However, a more general approach would be to take a linear combination of a set of nearby rays. This is analogous to a camera system's point-spread function. Unfortunately, this trades discontinuities for focusing problems, as we have now added a finite aperture and therefore a limited depth of field (related to the distances between the samples on the camera surface). Nevertheless, we are used to dealing with real world camera systems which exhibit these depth of field problems: we accept them, and even derive artistic value from them. However, we are not accepting of discontinuities in an image, and the tradeoff is a useful one.

Figure 3: Although the image is clearly focused, using only a single nearest neighbor ray creates noticeable discontinuities.

Variable apertures

Depth of Field

One can render depth of field effects by blending a larger set of approximate rays. If M_F is evaluated for all cameras $(\hat{s},\hat{t})$ within a given radial distance from (s,t), the aperture radius of the synthetic camera is increased. In Figure 4, 7 different cameras will be used for a single input ray. Whereas one can only *decrease* the aperture radius by sampling the camera surface more densely, one can *increase* the aperture at run-time by averaging more rays together (i.e. changing the radius of the gray circle in Figure 4).

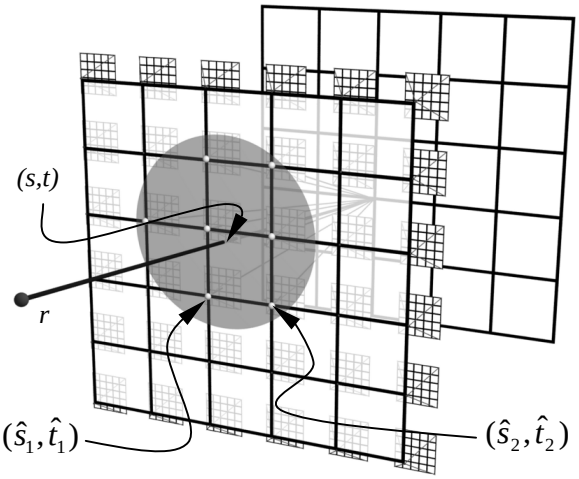

Figure 4: By including rays from all cameras within a radius of the actual camera surface intersection, we can increase the aperture of our synthetic camera.

By combining rays from cameras farther away from (s,t), only objects that are near the focal surface will be in focus. By definition, the set of nearest rays obtained through M_F for a

given (u_F, v_F) will intersect at (u_F, v_F), regardless of $(\hat{s}, \hat{t})$. That is, these rays are 'looking' at the point (u_F, v_F) on the focal surface *F*. If there was an object at that location when the ray database was captured, the rays will agree on the color of that surface (up to view dependent variations). In the left side of Figure 5, r_1, r_2, r_3, and r_4 agree. However, as the actual object gets farther from the focal surface, the agreement of the rays diverge, as in the right side of Figure 5. By increasing the aperture radius, more rays will be averaged, and these colors will diverge faster. Thus, we have a control over depth of field that is intuitive to photographers: the f-stop on a camera is inversely proportional to aperture radius.

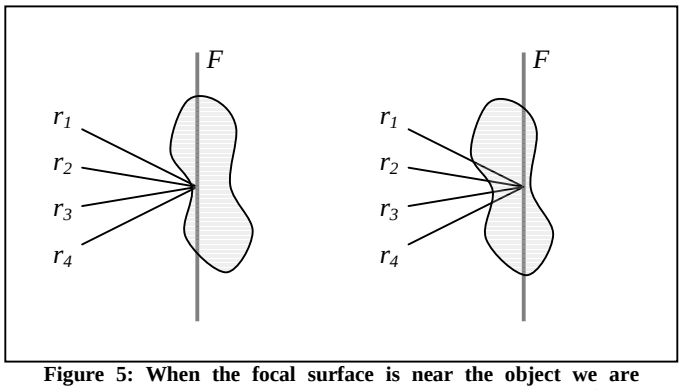

Figure 5: When the focal surface is near the object we are looking at, the rays agree and the object appears to be in focus (left). If the object is further from the focal surface (right), then the rays do not agree, and the object will appear out of focus.

Seeing through Objects

Other algorithms could be used to create effects not available to photographers. In Figure 6, we used an aperture that included every camera in our data set. If that were taken with a single real camera, the aperture would be about the size of a 3-story building! Because our depth of field is so narrow, we can "look through" objects.

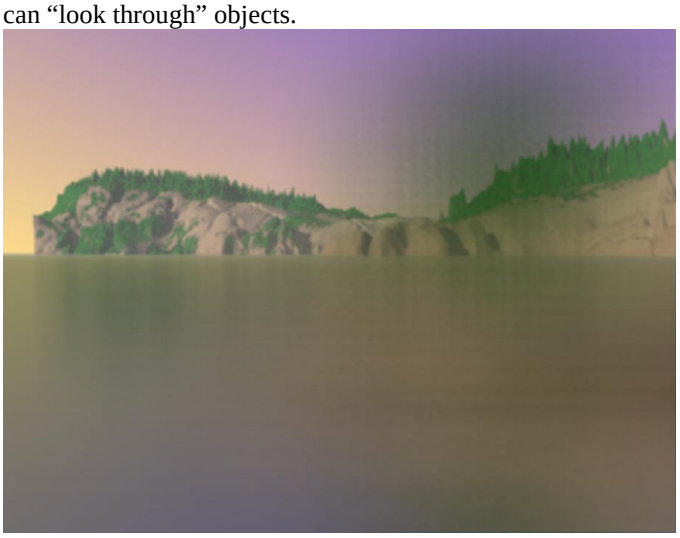

Figure 6: By making the aperture very large, we are able to look through objects. In this case, there is a tree and island occluding the hills where the slight haze appears. Figures 8,9, and 10 are the same scene with a smaller aperture; the tree is visible.

Varying focus

Though we have shown a way to change the size of the aperture, this could have been done with the standard light field parameterization. We will now show what can be accomplished with a parameterization that allows one to *dynamically control* what is in focus.

Moving the Focal Surface

A photographer using a camera can not only change the depth of field, but he can change what is in focus. Using our parameterization, one changes the focal surface in order to change what appears in focus. As before, a ray *r*, a camera surface, and a focal surface *F* intersect at (s,t,u_F,v_F). This 4-tuple is then quantized and passed through a mapping $M_F : (\hat{s},\hat{t},u_F,v_F) \rightarrow (u_{\hat{s},\hat{t}}, v_{\hat{s},\hat{t}})$ to obtain the nearest ray $\hat{r}$ in the ray database.

When the focal surface is changed to *F'*, the same ray *r* now intersects the camera and focal surfaces at the new coordinates $(\hat{s},\hat{t},u_{F'},v_{F'})$. Thus, by dynamically changing the focal surface, we are dynamically changing which ray $\hat{r}$ in the ray database is 'nearest' to the ray *r*. When we change the focal surface from F to F', we are changing from $\hat{r}$, a ray that passes through (u_F, v_F), to $\hat{r}'$, a ray that passes through $(u_{F'}, v_{F'})$ (see Figure 7). Since objects nearest to the focal surface intersection will be in focus when using a finite aperture, we have added a variable focus into the light field parameterization.

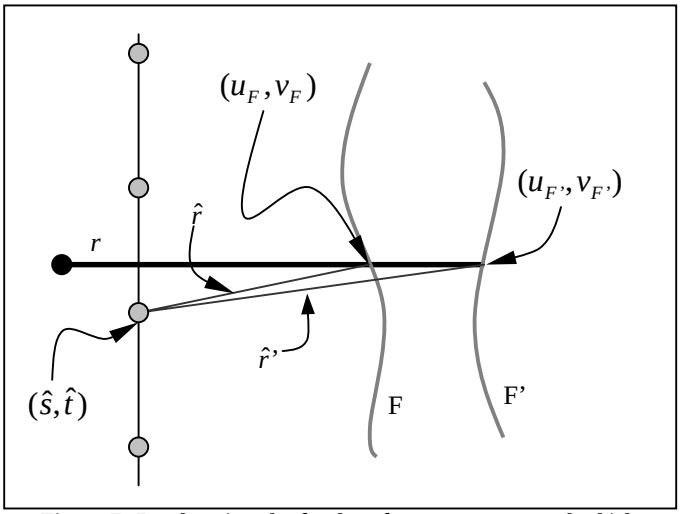

Figure 7: By changing the focal surface, we can control which ray in the ray database best approximates a given ray *r*.

Without the focal surface mapping, the light field always returned the same ray $\hat{r}$ for an input ray *r*. Since this deficency is tied into the storage of the ray database, light fields and lumigraphs have a fixed focus. Whereas the standard light fields implementations could render either Figure 8 *or* Figure 9, it could not do both without rerendering the database, clearly not a dynamic operation. Depth-

corrected lumigraphs would allow a dynamic focal surface, but only a single one.

Freely oriented focal surfaces

Since the focal surface determines which regions of space in the light field will be in focus, moving the focal surface allows the user to focus on different parts of the scene. For example, when we use a plane parallel to the image plane as a focal surface, it makes it easy to see what lies at a given depth in the scene. In figure 8, we have chosen to make the tree in focus, while figure 9 focuses on the hills behind the island. Moving the focal plane only changes 1) the mapping function M_F and 2) where the ray r intersects with F at (u_F, v_F). This is a simple change that does not affect the storage of the ray database.

Figure 8: A focal plane has been placed through the tree.

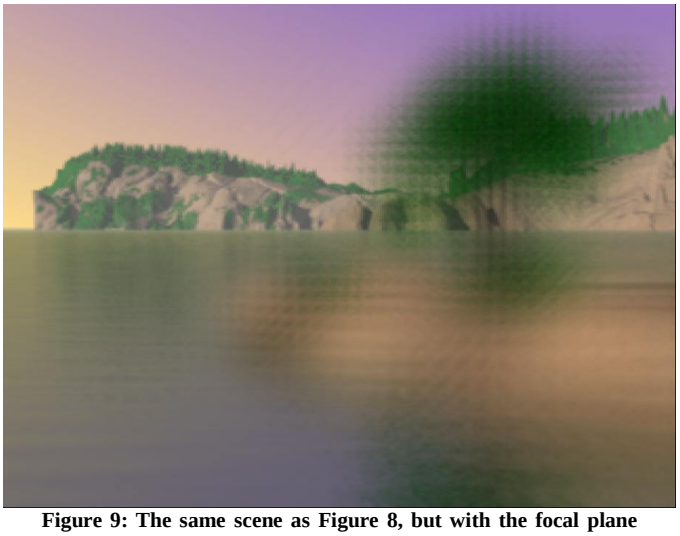

Figure 9: The same scene as Figure 8, but with the focal plane passing through the hills behind the island.

We do not need to keep the focal surface parallel to the image plane. If we orient the plane such that it passes through various objects in the scene, we can constrain these objects to be in focus. In Figure 10, we pass the focal surface through the front rock, part of the tree, and the rock at the left edge of the island. This non-parallel focal plane is available to photographers that use a bellows on their camera, but bellows are not common equipment and can be difficult to align with the optical system. And, of course the focus cannot be dynamically changed after film has been exposed. Since rotating the focal plane again simply causes a change of M_F and a different (u_F, v_F) for a ray r, it no more difficult to arbitrarily orient a focal plane than it is to move one.

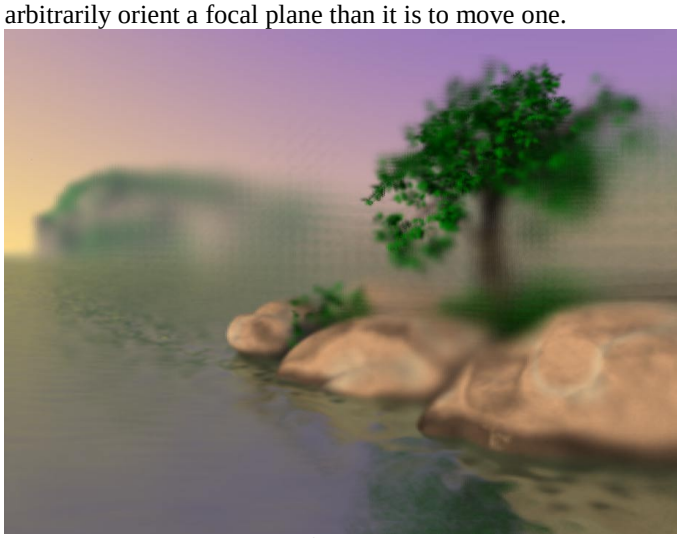

Figure 10: We have placed a focal plane that is not parallel to the image plane. In this case, the plane passes through part of the tree, the front rock, and the leftmost rock on the island. The plane of focus can be seen intersecting with the water in a line.

Non-planar focal surfaces

Clearly, these example scenes can not be entirely focused with a single plane. Of course, the focal surfaces do not have to be planar. One could create a focal surface out of a parameterized surface patch that passes through key points in a scene. Or, one could even use a depth map of the scene as a focal surface, insuring that all visible surfaces were in focus. This would analogous to depth-corrected lumigraphs, where a proxy surface helps focus the representation But, in reality, these depth maps would be hard and/or expensive to obtain with simple hardware, and would likely only be applicable to synthetic ray databases.

Multiple Focal Surfaces

In general, we would like to have more than just the points near a single surface in clear focus. One solution is to use multiple focal surface, something not available to real cameras. In a real lens system, only one continuous region is in focus at one time. However, since we are not confined by physical optics, we can have two or more distinct regions that are in focus. For example, in Figure 11, the red bull in front and the monitors in back are in focus, yet the objects in between, such as the yellow fish and the blue animal, are out of focus. Using a real camera, this can be done by first taking a set of pictures with different planes of focus, and then taking the best parts of each image and compositing them together [Haeberli94].

Figure 11: Using two focal surfaces allows us to make the front and back objects in focus, while those in the middle are blurry.

Since a ray *r* will intersect each focal surface, some scoring scheme is needed to pick which focal surface will be used. We would like to pick the focal surface which will make the picture look most focused, which means we need to pick the focal surface which is closest to the actual object being looked at. We can augment each focal surface with some scoring $\sigma(u_F, v_F)$, which is the likelihood a visible object is near (u_F, v_F). Then, we calculate σ for each focal surface, and we can pick the focal point with the best score σ. In Figure 12, we would like σ_2 to have the best score, for it is closest to the object. Note that an individual score $\sigma(u_F, v_F)$ is independent of the view direction; however, the set of scores compared for a particular ray *r* is dependent on the view direction. Therefore, although our scoring data can be developed with out view dependence, we can still extract view dependent information from it.

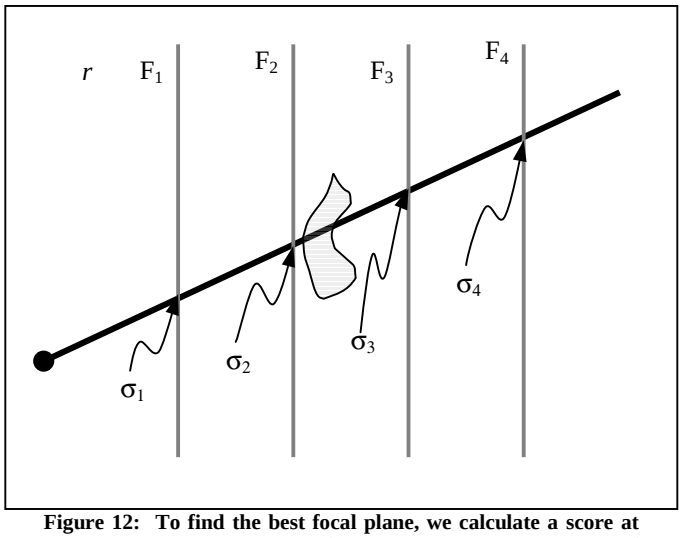

Figure 12: To find the best focal plane, we calculate a score at each intersection and take the focal plane with the best score. If the scoring system is good, the best score will be the one nearest the surface we are looking at.

If we are given the light-fields but no knowledge about the geometry of the scene, we must create these scores from information in the images alone. We would like to avoid computer vision techniques that involve deriving correspondences to discover the actual geometry of the scene, as vision algorithms may deal with ambiguities that are not relevant to generating synthetic images. For example, flat regions without texture can be troublesome to a vision system, and can make it hard to find an exact depth. However, when making images from a light field, picking the wrong depth near the same flat region would not affect us, because the region would still look in focus. Therefore, we would like a scoring system that allows us to take advantage of this extra freedom. And, because we only have the original images as input, we need a scoring system that can be easily created from these input images.

So, we have chosen to look for locations on the focal surfaces that approximate radiance functions. Whereas we have usually thought of the light-fields as looking in at objects, we can also use them to generate radiance functions. The collection of rays in the ray database that intersect at (u_F, v_F) is the discretized radiance function of the point (u_F, v_F). If the point lies on an object and is not occluded by other objects, the radiance function will be smooth, as in the left side of Figure 13 below. However, if the point is not near an actual object, then the radiance function will not be smooth, as in the right side of Figure 13.

To measure smoothness, we look for a lack of high frequencies in the radiance function. High frequencies in a radiance function identify 1) a point on a extremely specular surface, 2) an occlusions in the space between a point and a camera, or 3) a point in empty space. Thus, if we identify the regions where the there are no high frequencies in the radiance function, we know the point must be near a surface. Because of their high frequency content, we may miss areas that are actually on a surface but have an occluder in the way.

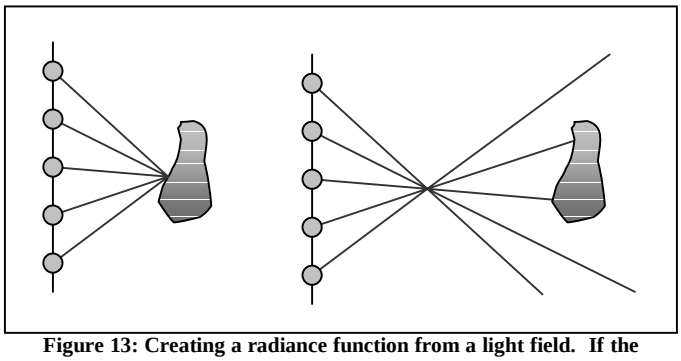

Figure 13: Creating a radiance function from a light field. If the radiance function is near an object, then the function will be smooth. If the radiance function is not near an object, it will vary greatly.

Because calculating the radiance function is a slow process, we create the scores for discrete points on each focal surface as a preprocessing step. This allows us to use expensive algorithms to setup the scores on our focal surfaces. Then, when we are rendering, we only need to recall the

prerecorded scores for each focal surface intersection and compare them.

To find the best focal surface for a ray, an algorithm must first obtain intersections and scores for each focal surface. Since this is linear in the number of focal surfaces, we would like to keep the number of focal surfaces small. However, the radiance functions are highly local, and small changes in the focal surface position can give large changes in the score. Nevertheless, the accuracy needed in placing the focal surfaces is not very high. That is, we do not need to find the exact surface that makes the object we are looking at in perfect focus; we just need to find a surface that is close to the object. Therefore, we can first calculate the scores by sampling the radiance functions for a large number of planes, and then 'squash' the scores down into a smaller set of planes using some function $\lambda(\sigma_i,...,\sigma_{i+m})$. For example, in Figure 14, we first calculate the radiance scores for 16 planes. Then, using some combining function λ, we combine the scores from these 16 planes down to new scores on four planes. These four planes and their scores will be used as focal surfaces at run time. We chose to use the maximizing function, that is, $\lambda(\sigma_i,...,\sigma_{i+m}) = \max(\sigma_i,...,\sigma_{i+m})$. Other non-linear or linear weighting function might provide better results. Figure 22 was created using 8 focal planes combined down to 4.

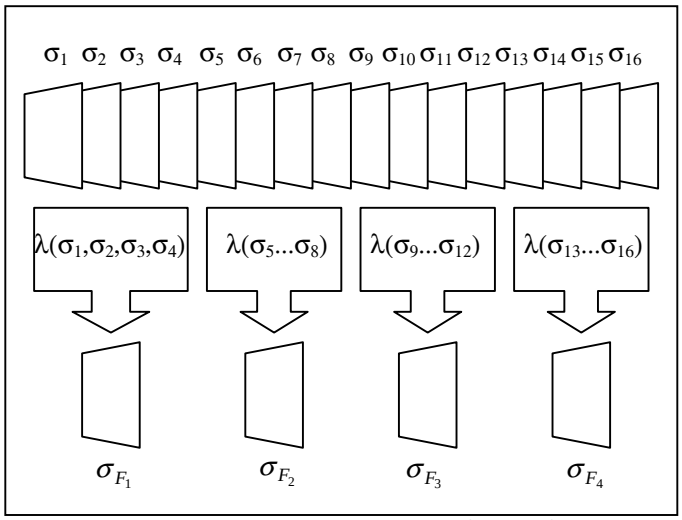

Figure 14: We can compute scores on many focal surfaces, and then combine them to a smaller set of focal surfaces, so the run-time algorithm will have less scores to compare.

Figure 15: Here is a visualization of the scores used on the front focal plane for the picture in Figures 11 and 20. The closer to white, the better the score σ, which means objects are likely to be located near that plane.

Figure 16: Here is a visualization of the scores on the back focal plane, analogous to Figure 15.

Selecting a Focal Surface through Auto-focus Techniques

We often select focal planes by hand, allowing a user to select the subject of the image. However, it is possible to determine these focal planes algorithmically, much an auto-focus camera. It is possible that by adapting these algorithms, we could identify a minimal set of focal planes that would put the most items in focus.

For a single plane, this would be analogous to using an auto-focus camera [Pentland87]. To do auto-focusing, one can create a series of images with an extremely narrow depth of field, where each image would put the focal plane at a different depth. This narrow depth of field can be implemented by increasing the aperture radius so that a ray from every camera is averaged to produce a single output ray. The resulting images will have out-of-focus and in-focus regions. The out-of-focus regions will have little high-

frequency energy, where as the regions in focus will. Since only structure very near the focal planes will be in focus, we know that the in-focus regions identify regions where there is structure. Thus, if we pass a high-pass filter over these narrow depth of field images and then identify the regions with high-frequency energy, we will find the regions in space where structure exists. Then, we can use the plane (or set of planes) which gives rise to the image with the most high-frequency energy: this plane is our auto-focus plane. Likewise, we could take the *n*-best planes for a multiple focal plane rendering.

Objects with little high-frequencies, even when they are in focus, such as flat regions with little texture, will not be detected by this process. However, objects with little high-frequency will look as good if they are out of focus as if they are in focus. Whereas using computer vision techniques to find depth from a set of images would have to further analyze these ambiguous regions, we do not have to delve further since several values will be good enough: we can simply take the best one that we find.

Results

The two light field data sets shown in this picture were created as follows. The tree data set, with 256 input images, was rendered in Povray 3.1, each image at 320x240. The stuffed animal data set was more complex to create. We attached an Electrim EDC1000E CCD camera (654x496) with an 16mm variable aperture lens to an X-Y motion platform from Arrick Robotics (30”x30” displacement). Then, we took 256 pictures on a (approximately) 16”x16” grid, which took approximately 30 minutes. To calibrate the camera, we first used a Faro Arm, a submillimeter accurate contact digitizer, to measure the 3-D spatial coordinates (x,y,z) of the centers of 24 large circular calibration pattern on two perpendicular planes filling the camera’s field of view. Then, using a picture of the calibration pattern, we found the centroids (i,j) of these dots in the images with MATLAB. We then fed the 24 5-tuples (x,y,z,i,j) into the Tsai-Lenz camera calibration algorithm [Tsai87] which reported focal length, CCD sensor element aspect ratio, principle point, and extrinstic rotational orientation. We ignored radial lens distortion, which was reported as less than 1 pixel per 1000 pixels. Finally, we resampled the raw 256 654x496 images down to 327x248 before using them as input to the renderer.

Internally, our light fields used no compression techniques as presented in earlier papers. Our particular high frequency filter looked for the mean energy in a sampled radiance function that had been processed by a gradient magnitude Sobel edge detection filter [Lim90]. The scores on the focal surface were rendered at 920x690 and not interpolated. We use bilinear interpolation between samples on the image surfaces. When rendering with the variable apertures, we used cone weighting to combine the rays from each camera.

Using our renderer, we can typically render images with the four nearest cameras in about one second. In our code, we have maintained a general interface that allows for flexibilty and fast development. In the near future we plan to implement a real time renderer that will optimize for speed.

Conclusions

Previous implementations have tried to solve focusing problems by 1) using scenes that were roughly planar, 2) using aperture filtering to blur the input data, or 3) using approximate geometry. Unfortunately, most scenes can not be confined to a single plane, aperture filtering can not be undone or controlled at run time, and proxy surfaces can be hard to obtain. We have presented a new parameterization that allows run-time control of what should be in focus. In addition to describing focus control through aperture size and a moving focal surface, we have presented a strategy for using multiple focal planes and methods to create them. This new parameterization allows light fields to capture data sets with depth, and helps bring us closer to truly photorealistic virtual reality.

There is much future work to be done. We would like to improve our scoring system for our multiple focal planes: we need a method to differentiate between high frequencies in the radiance function caused by occlusion and those caused by empty space. In Figure 20, the errors surrounding the red bull identify how these errors affect the final images. Also, we would like an algorithm for optimally picking the *n* best focal planes, perhaps using the presented auto-focus techniques. The camera calibration step is somewhat tedious, and we would like to self-calibrate using the light fields. This would give us the optimal camera model for each light field, as opposed to assuming the light field to works with a prior camera calibration. Finally, we are working on methods to speed up the renderer so that we can view these light fields in a head-mount-display.

We would like to thank Hughes Research Labs, Intel Corporation, and Microsoft Corporation for monetary, equipment, and software donations. Also, thanks to Neil Alexander for his *"Alexander Bay"* tree model, and to Charles Lee for help in making our pictures and animations.

Figure 17: Using a single focal plane, only the red bull is in focus, while the yellow fish and the monitor are out of focus.

Figure 18: By moving the focal plane back in the scene, we can make the fish in focus, while the red bull and the monitor are out of focus.

Figure 19: Finally, the focal plane is at the back of the room, making the monitor in focus, while the fish and red bull are blurry.

Figure 20: By using two focal planes, we can make the bull and the monitor in focus, while the regions in between are still out of focus.

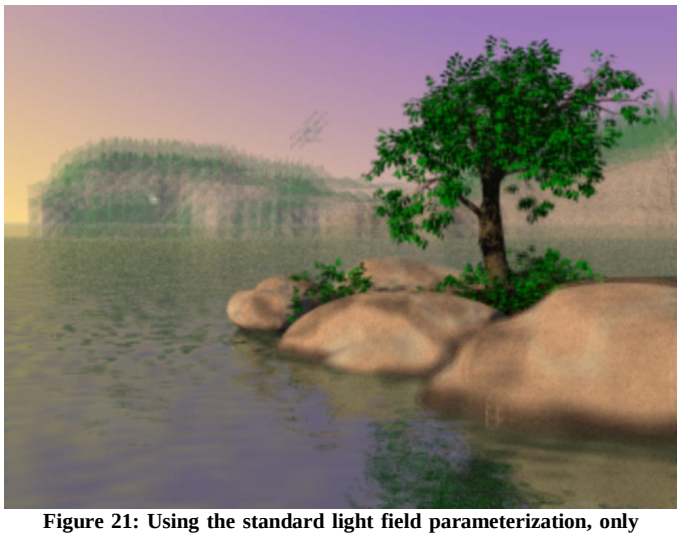

Figure 21: Using the standard light field parameterization, only one fixed plane can be in focus. Using the smallest aperture available, this would be the best picture we could create.

Figure 22: By using 4 focal planes (originally 8 focal planes with scores compressed down to 4), we can clearly do better than the image in Figure 21. Especially note the hills in the background and the rock in the foreground.

References

[Bolles87] Bolles, R. C., H. H. Baker, and D. H. Marimont, "*Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion*," **International Journal of Computer Vision**, Vol. 1, 1987.

[Camahort98] Camahort, E., A. Lerios, and D. Fussell, "*Uniformly Sampled Light Fields*," **Proceedings of the 9th EUROGRAPHICS Workshop on Rendering**, Vienna, Austria, June/July 1998

[Cook84] Cook, R.L., T. Porter, and L. Carpenter, "*Distributed Ray Tracing*," **Computer Graphics** (SIGGRAPH'84 Conference Proceedings), July 1984, pp. 137-145.

[Gortler96] Gortler, S.J., R. Grzeszczuk, R. Szeliski, and M.F. Cohen, "*The Lumigraph*," **Computer Graphics** (SIGGRAPH'96 Conference Proceedings), August 1996, pp. 43-54.

[Gu96] Gu, X., S.J. Gortler, M.F. Cohen, "*Polyhedral Geometry and the Two-Plane Parameterization*," **7th Eurographics Workshop on Rendering**, 1996. (also, http://hillbilly.deas.harvard.edu/~sjg/papers/tpp.ps)

[Haeberli94] Haeberli, Paul, "*A Multifocus Method for Controlling Depth of Field*," http://www.sgi.com/grafica/depth/index.html, October 1994.

[Levoy96] Levoy, M. and P. Hanrahan, "*Light Field Rendering*," **Computer Graphics** (SIGGRAPH'96 Conference Proceedings), August 1996, pp. 31-42.

[Lim90] Lim, J.S., **Two-dimensional Signal and Image Processing**, Prentice Hall P T R, New Jersey, 1990, pp 476 –483.

[Mitchell88] Mitchell, D.P. and A.N. Netravali, "*Reconstruction Filters in Computer Graphics*," **Computer Graphics** (SIGGRAPH '88 Conference Proceedings), August 1988, pp. 221-228

[Pentland87] Pentland, A.P., "*A New Sense for Depth of Field*," **IEEE Transactions on Pattern Analysis and Machine Intelligence**, vol. 9, no. 4, July 1987, pp. 523-531.

[Tsai87] Tsai, R. Y., "*A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses*," **IEEE Journal of Robotics and Automation**, Vol. RA-3, No. 4, August 1987.