Optical Splitting Trees for High-Precision Monocular Imaging

Anonymous CVPR submission

Paper ID 651

Abstract

Beam splitting is widely used to create multiple geometrically similar but radiometrically controlled views of a scene. However, acquiring a large number of such views is known to be a hard problem. We introduce the notion of an optical splitting tree that can recursively split a monocular view of a scene a large number of times. In this tree, the internal nodes are optical elements like beam splitters and filters, and the leaves are video sensors. Varying the optical elements allows us to capture at each virtual pixel multiple samples that vary not only in wavelength but also in other sampling parameters like focus, aperture, polarization, exposure, sub-pixel position, and frame time.

We present a framework to design and evaluate splitting trees, a configurable hardware system that captures up to eight views, and an algorithm for automatically discovering good designs that meet a specification. The algorithm employs an optimizer that considers light efficiency, accuracy, and cost. We demonstrate several examples of both user- and automatically-designed trees and demonstrate their utility for high dynamic range, multi-focus, high-speed, and hybrid high-speed/multispectral video.

1. Introduction

In the context of computer vision, sets of simultaneous images that are geometrically similar but radiometrically controlled are useful for many applications. These include high dynamic range (HDR), high speed, super resolution, focus/defocus analysis, and multispectral video. Beam splitting is widely used to create multiple reduced-amplitude copies of the plenoptic field for these applications. Multiple sensors within the same camera image the copies, creating multiple monocular views that share an optical axis but have varying capture parameters. These can be combined to capture more information (bits) than a single view, thus allowing for high precision measurements. We consider optical systems that form optical splitting trees, where the physical layout of the components and the optical path topology take the shape of a tree.

Designing optical splitting trees is challenging when many views with specific spectral properties are required. We introduce an automated method for designing efficient splitting-tree cameras; it can aid or replace manual design. It takes as input a specification for each view and a set of weights describing the user’s relative affinity for efficiency, accuracy, and cost. The weights determine the objective function of an optimizer. The output is a splitting tree that implements the input specification, and an analysis of the efficiency of each root-to-leaf path. Automatically designed trees appear comparable to those designed by hand; we even show some cases where they are superior.

Capture of multiple monocular views has previously been demonstrated in various contexts that we review, but rarely have more than three simultaneous views been cap-
tured. We have built the configurable optical splitting tree system shown in Figure[1] which captures up to eight views, and implemented several popular applications: HDR, multiview, high-speed, and hybrid high-speed multispectral video, assisted by our optimizer.

2. Related Work

2.1. Beam Splitters

Both prisms and half-mirrors are popular beam splitting mechanisms. They can be constructed to split light into two or more paths, and the ratio of intensities directed to each path at each wavelength can be adjusted. The most common and economical element is a half-silvered plate mirror. A drawback of plate mirrors is that their orientation must be calibrated relative to the optical path. In contrast, sensors placed immediately against the sides of a splitting prism are automatically registered up to a 2D translation. In the case of 3-CCD cameras, a dichroic prism that separates light by wavelength is often used to capture three copies of the image, each with a different spectral band. Prisms have also been used for HDR imaging [6].

Our implementation places beam splitters between the lens and the scene, which enables us to use separate lens parameters for each sensor. An alternative is to split the light in between the lens and the sensor [12]. That alternative shares a single lens over all sensors, which simplifies lens calibration and reduces lens cost, but makes calibration and filter changes more difficult.

McGuire et al. [7] use a beam splitter camera with three views for matting. They report that it can be extended to exhaustively sample the cross product of various sampling parameters. We extend these ideas with a detailed discussion and implementation of a full, calibrated eight-view system, analysis methods, and sparse parameter sampling that allows us to create views with completely independent characteristics. We demonstrate more efficient and economical camera designs for their application in Section 6.3.

2.2. Pyramid Mirrors

Another interesting way to create multiple copies uses a pyramid mirror placed behind the lens [13]. (Placing a mirror pyramid in front of the lens, e.g. as in [13], creates panoramic field of view that is unrelated to our work.) This arrangement creates a very compact optical path, but has some drawbacks. It requires a large aperture, which leads to a narrow depth of field and limits the situations to which it may be applied. It is also non-trivial to divide light intensity unevenly between the image copies, as might be desirable for HDR. Furthermore, the edges between the individual mirrors cause radiometric falloffs as discussed in [1]. Even after calibration these fall-offs reduce the effective dynamic range of each view. The defocus point spread function from such a camera is a differently oriented triangle in each view, instead of a disk as in a beam splitter camera. This makes it difficult to fuse or compare images in which objects are at different depths; objects outside the depth of field appear not only defocused but also shifted away from their true positions.

2.3. Alternatives

For flat scenes and scenes far (> 10m) from the camera, there are neither parallax nor view-dependent effects. In those cases the calibration problem is comparatively easy since the optical centers of the sensors need not be aligned. Dense arrays of side-by-side sensors, e.g., [14], have captured multiple approximately-monocular views for such cases. Arrays capture much more light than a beam splitter system. However, a beam splitter system can capture nearby and deep scenes, and offers the possibility of sharing optical elements like filters over multiple sensors.

One can use a mosaic of filtered CCD pixels to sample multiple parameters in a single image. The Bayer mosaic tiles per-pixel band-pass filters, sampling three wavelengths with a single monochrome sensor. Recently, filter mosaics have been proposed for sampling other parameters with high precision (see [9] [10]). This approach can be implemented compactly and requires no calibration (once manufactured), making it ideal for many applications. The drawback is that it trades spatial resolution for resolution along other imaging dimensions. Such a system also makes it difficult to experiment with aperture and timing effects, which are explored in this paper.

Previous automatic optical design systems like SYNOP-SYS, ADOS, and ZEMAX emphasize the tracing of rays through lenses. To the best of our knowledge, none significantly address the issues of sampling the plenoptic function through splitting and spectral response that we discuss.

3. Optical Splitting Trees

Optical splitting trees are schematic representations of a tree-topology filter systems. The edges are light paths and the nodes are optical elements. Nodes with a single child represent filters and lenses. Nodes with multiple children are beam splitters. Leaf nodes are sensors. The plenoptic field enters the system at the root. The physical path length to each sensor’s optical center is identical. However, the tree-depth of a sensor (the number of internal nodes between it and the root) may differ.

Figure 2 shows a schematic of a full binary splitting tree, viewed from above, where light enters on the upper right. This layout packs components into a small form factor with no optical path occluded. The thick black lines are light baffles preventing the reflective child of each splitter from imaging stray light. Further abstracting the structure and using the symbols in Figure 3 we can emphasize the most
which is band-pass filtered immediately before the lens. Note that both designs are useful; the splitting tree concept makes it easy to switch between them if necessary.

Many applications require a balanced binary tree, in which each sensor has the same tree depth and the beam splitters divide incident light evenly between their children. In others (like HDR) it is useful to unbalance the tree. We may do so either by using beam splitters with uneven division ratios, or by creating a structurally unbalanced tree where the sensors’ tree-depths vary.

4. Automated Design

For a complex camera manual design becomes challenging, in part because elements like beam splitters and ND filters have non-ideal responses (see Figure 5). Manually accounting for this spectral distortion is difficult, so people are likely to design sub-optimal cameras, especially in the experimental stage. Another complicating factor is that multiple trees may satisfy a specification but differ in other ways, e.g. Figures 7 and 4 demonstrate cost and efficiency tradeoffs.

We now describe our method for automated design of a splitting tree from a specification. It addresses the complexity of designing large trees and can balance non-ideal components against each other. It also takes into account efficiency, cost, and accuracy as defined later. The design process contains two phases. The first deterministically constructs an inefficient, expensive tree that is close to the specification and then applies a deterministic series of simplifications to reduce the cost. The second phase is an optimizer that performs short random walks through the space of all possible trees, searching for similar but better trees. It continues until explicitly terminated by the user. Our implementation is in Matlab, using trees from the Data Structures toolbox. All results were computed in under an hour, although usually the first minute brought the objective function values within 10% of the peak value and the remaining time was spent addressing minor spectral deviations.

4.1. Algorithm Input

Our algorithm takes a set of design specifications and objective function weights as input and selects components from a real optics catalog. For each view $v$ the user specifies a desired wavelength response curve as a vector $X_v[\lambda]$; a scalar importance $I_v$; and a vector $Y_v$ that packs together the scalar parameters of a sensor: aperture area, exposure time, relative sensitivity to S- and P-polarized light, time shift, horizontal and vertical sub-pixel shift, focal length, and focus depth. The hat notation distinguishes specification variables from their counterparts that are adjusted dur-

\footnote{Edmund Optics Catalog 2005}
ing optimization. Note that no layout hints are provided with this input, which is a simple text file.

**Efficiency** is a core concept in our approach. The specification dictates how the tree distributes light, but the scale is necessarily relative because at input time the light loss inherent in the design is unknown. Likewise, each component in the catalog must be annotated with a curve describing its transmission (for a filter), split ratio (for a beam splitter), or digital output sensitivity (for a sensor) at several wavelengths. Figure 5 shows examples of these curves.

We call this curve the quantum efficiency of a component. We represent it with a vector \( \hat{q} \) describing the ratio of output to input light at a node, sampled at twelve wavelengths between 400nm and 950nm. Because our components are passive, \( q[\lambda] \leq 1 \). For a view \( v \), let \( q_v = \prod \hat{q}_i \) over every component \( i \) on the path from \( v \) to the root. Let scalar \( \bar{q}_v \) be the mean efficiency of that view with respect to \( \lambda \), and let \( \bar{q}_{tree} = \sum_{views} \bar{q}_v \), denote the efficiency of the entire camera. With these definitions, \( \hat{X}_v = q_v/\bar{q}_v \) specifies the shape of the desired response and \( \hat{I}_v = \bar{q}_v/\bar{q}_{tree} \) is the fraction of measured light captured by \( v \).

### 4.2. Objective Function

Our optimizer seeks camera designs that maximize the goals of efficiency, accuracy, and cost. The objective function is the weighted sum of expressions representing each of these goals,

\[
\text{obj}(\text{tree}) = \alpha \bar{q}_{tree} - \sum_{\text{views}} \left( \beta \left| X_v - \hat{X}_v \right|^2 + \gamma \left| I_v - \hat{I}_v \right|^2 \right)
\]

These expressions drive the optimizer as follows:

**Efficiency** (1): maximize the total light measured;

**Spectral accuracy** (2): minimize the difference from the color specification;

**Importance accuracy** (3): respect the specified splitting ratios;

**Parameter accuracy** (4): minimize difference from the specified pixel shift, temporal shift, polarization, motion blur, and defocus; and

**Economy** (5): minimize the dollar cost.

Each expression is quadratic to create stable maxima. Recall that \( X \) and \( Y \) are vectors, so \( \cdot \) is a dot product, and that the hats denote specification variables.

Greek-letter variables are user-tunable weights, which may vary from zero (which ignores a term) to arbitrarily large positive numbers (which demand that the optimizer exactly meet the specification). They are all scalars except for \( \bar{c} \), which is a vector so that each parameter may be weighted independently. We choose initial weights so that each of the five expressions has approximately the same magnitude. (Note that this depends on the units selected for \( Y \)).

Quality is most sensitive to \( \alpha \) because preserving specification introduces filters that absorb light. Other weights can vary within a factor of two without affecting the output, since cost and accuracy are less contentious when many filter choices are available.

### 4.3. Deterministic Phase

The goal of this phase is to construct a tree that accurately meets the view specifications. To simplify the process, economy and efficiency are left unconstrained. The system first groups the views into separate binary trees of similar \( X \) to increase the later likelihood of shared filters and then links those trees into a single large tree.

All splitters are “50R/50T” half-mirrors at this stage. To satisfy the \( X \) values, the system evaluates the actual \( X \) at each leaf and introduces band pass filters immediately before the sensors to optimize the spectral accuracy term. It then sets all parameters as dictated by \( Y \). Finally, it evaluates \( I \) at each leaf and inserts ND filters until the importance accuracy term is optimized.

### 4.4. Search Phase

From the deterministically computed tree we search for steps in the design space that increase \( \text{obj} \) using the uphill simplex method. Each step begins with a randomly chosen transformation. Because many transformations require parameter changes to be beneficial, the \( Y \) vectors are adjusted to increase spectral and importance accuracy before \( \text{obj} \) is evaluated for the altered tree.

Several transformations preserve \( X \) and \( I \) while potentially reducing the cost of the tree or increasing \( \bar{q}_{tree} \). These “tree identities” are:

1. No transformation (allows \( Y \) change without tree change).
2. If the same filter appears on both children of a beam splitter, move it to the parent of the splitter.
3. Replace a chain of filters with a single, equivalent filter.
4. Reorder the filters in a chain (tends to encourage #2 and #3).
5. Experimental System

To test our design framework we built a physical configurable splitting tree system with eight computer vision sensors that uses 100 × 100mm half-mirror and hot-mirror beam splitters (see Figure 1) the black baffles in the top view were removed for bottom view).

The sensors are 640 × 480 Bayer-filter A601fc color cameras and monochrome A601f cameras by Basler. Each sensor is equipped with a Pentax 50mm objective. The tree exactly fits on a 2 × 2 ft² optical breadboard with 1/2" in hole spacing. The sensors are connected to a single 3 GHz P4 PC using the FireWire interface. We wired the hardware shutter trigger pins to each of the eight data pins of the PC parallel port, which can be precisely controlled by writing bit masks to that port through a special Win32 driver.

5.1. Calibration

The difficulty of calibration increases with the number of elements. Sensors that share an optical center are also more difficult to calibrate than array systems where the views are not expected to align perfectly.

We first orient the half-mirror beam splitters at 45° to the optical axis. To orient these, we place a lens cap over each camera and shine a laser along the optical axis to illuminate a single point near the center of each lens cap. Working through the splitting tree from the root to the leaves, we rotate the beam splitters until each dot appears exactly in the center of the lens cap.

Second, we construct a scene containing a nearby target pattern of five bulls eyes on transparent plastic and a distant, enlarged pattern on opaque poster board so that the two targets exactly overlap in view 1. We then translate all other sensors until the target patterns also overlap in their views. This ensures that the optical centers are aligned.

Third, we compute a software homography matrix to correct any remaining registration error. We find corresponding points in 3D by filming the free movement of a small LED light throughout the scene. Let $C_s$ be the $N \times 3$ matrix whose rows are homogeneous 2D positions, i.e. $[x \ y \ 1]$, of the light centroid at subsequent frames in view number $s$. The transformation mapping pixels in view 1 to those in view $s$ is

$$H_s = \arg \min \left( |H_s C_s - C_1| \right) = C_1 C_s^\dagger, \quad \text{(6)}$$

where $\dagger$ denotes a pseudo-inverse. We solve this system by singular value decomposition because it is frequently ill-conditioned. For color calibration, we solve the corresponding system in color space using pixel values sampled from a Gretag Macbeth color chart instead of 2D positions.

6. Applications and Results

We implemented various video capture applications using a mixture of data acquisition trees that were hand-designed using the framework and ones produced automatically by the optimizer. We were able to reconfigure and calibrate the system for each of these in a couple of hours, even when outside the laboratory. In each example, the result figures and videos demonstrate accurate color, temporal, and spatial image capture and registration across many more monocular views than in previous work.

The deterministic phase always produces a viable design, so optimization will technically never fail. However, in about two out of ten cases, it takes longer to adjust the weights than it would to simply adjust the output of the deterministic phase manually. In its current form, the optimizer is therefore most useful when applied to designs with many sensors, significantly non-uniform components, or tricky spectral constraints.

---

3http://www.logix4u.net/inpout32.htm
the depth of field varies and where the aperture varies. Now motion blur is correct but weight and decreasing the aperture weight in and low that approach are inconsistent motion blur between cameras within the dynamic range of the display.

four different sensors to keep all scene elements visible and city in the distance are bright. On the right are result- sequence number one (top), the actor is brightly lit and the sky shows an artificially wide depth of field. Many systems (e.g., ) capture three monocular views, each with focal length \( f = 50 \text{mm} \) and each at equal importance: a pinhole camera with a \( f/12 \) aperture, and two \( f/1.6 \) views focussed at different depths. Their design uses two “50R/50T” beam splitters and two ND filters.

Our optimizer created the alternative tree in Figure 10. Compared to the original, this design achieves higher efficiency through a “70R/30T” beam splitter; a single, weaker

6.1. High Dynamic Range

Figure 7 shows several splitting trees for a simple high dynamic range camera where the relative intensities observed by the views are powers of two. We designed these by hand based on previous work and verified that with appropriate weights the optimizer rediscovered equivalent structures. In (a), a large economy weight \( \gamma \) gives the inexpensive variable exposure solution 4. The drawbacks of that approach are inconsistent motion blur between cameras and low \( q_{\text{tree}} = 15/32 \) efficiency. Increasing the exposure weight and decreasing the aperture weight in \( \bar{r} \), where the aperture varies. Now motion blur is correct but the depth of field varies and \( q_{\text{tree}} \) is still low.

An alternative is to use ND filters as in (b), somewhat similar to Mitsunaga et al. 8. This corrects both motion blur and depth of field but not efficiency. Our optimizer has not discovered this approach—instead it finds a better tree when the efficiency weight \( \alpha \) is large. Tree (d) has \( q_{\text{tree}} = 30/32 \). Instead of blocking light with the iris, shutter, or filters, it redirects excess light at a sensor to other sensors that can measure it. Aggarwal and Ahuja 11 mention this design, but it produces asymmetric point spread functions on their pyramid camera and was never implemented.

Figure 8 shows results from two HDR experiments. In sequence number one (top), the actor is brightly lit and the city lights are dim in the background. In sequence number two (bottom), the actor is inside a dark office and the sky and city in the distance are bright. On the right are resulting tone-mapped HDR images. These combine frames from four different sensors to keep all scene elements visible and within the dynamic range of the display.

6.2. Multiple Focus and Defocus

Images focused at multiple depths can be used to recover depth information 12, 13, 14 and to form images with an infinite depth of field. Many systems (e.g., ) split the view behind the lens. Splitting in front of the lens allows us to vary not only the location but also the depth of the field by changing the aperture.

To capture images with varying depths of field like those in Figure 9 we use a full binary tree with eight cameras. Each sensor is focused at a different depth, ranging from 20cm from the optical center (about 4cm from the first beam splitter) to 20m (effectively infinity). We use wide \( f/1.4 \) apertures for a narrow depth of field on each sensor.

Figure 9 shows an artificially wide depth of field achieved by a weighted sum of each view. The weight at each pixel is proportional to the local contrast (luminance variance) in the view, squared.

6.3. Matting and Other Designs

Two matting algorithms from the computer graphics literature use custom cameras that can be expressed within our framework. We compare the originally published designs against new ones created by our optimizer from their specifications.

McGuire et al. 11 capture three monocular views, each with focal length \( f = 50\text{mm} \) and each at equal importance: a pinhole camera with a \( f/12 \) aperture, and two \( f/1.6 \) views focussed at different depths. Their design uses two “50R/50T” beam splitters and two ND filters.

Our optimizer created the alternative tree in Figure 10. Compared to the original, this design achieves higher efficiency through a “70R/30T” beam splitter; a single, weaker
Figure 10. Optimized matting camera based on McGuire et al. [7].

Figure 11. 240 fps video of a soda can opening. Each of the eight sequential frames shown was captured by a different sensor. The tree is on the right; note the 1/120s overlapping exposures, longer than is possible for a single-sensor high speed camera.

The matting cameras are simple. Figure 12 shows the tree computed for an arbitrary complex specification. The optimizer correctly created an efficient HDR (c,d,e) subtree. However, it failed to place a hot mirror between (a) and (b), probably because we weighed accuracy much higher than efficiency for this test. The six plots show how well it did match the desired accuracy; it even chose between different brands of ND filters based on their $q$-curves.

6.4. High Speed

Figure 11 shows eight frames from a high speed sequence of a soda can opening and the capture tree used. Each frame has an exposure time of 1/120s and the entire sequence is captured at 240fps, so the views overlap temporally. This gives $q_{tree} = 1/4$, which is lower than the $q_{tree} = 1$ for an array like the one by Wilburn et al. [14]. The advantage of our approach is that the sensors share an optical center for accurate capture of scenes with depth variation and view-dependent effects.

6.5. Multimodal High Speed

HDR, high speed, etc. are sampling strategies. They are useful for building high-level applications, like surveillance, in an HDR environment. It is natural to build hybrid sampling strategies, which are easy to express and experiment with within our splitting tree framework.

We use the configuration from Figure 13 designed by the optimizer to capture hybrid high-speed visible/IR video. A hot-mirror directs IR down the right sub-tree and visible light down the left sub-tree. Each subtree has four cameras with temporal phase offsets, so the entire system yields 120fps video with four spectral samples. Figure 14 shows a frames from a sequence in which a person catches a tumbling IR remote control and then transmits at the camera. Because the configuration captures four spectral samples at 120fps, the high-frequency IR pattern transmitted by the remote is accurately recorded, as is the fast tumbling motion at the beginning of the sequence.
7. Conclusions and Future Work

We presented a framework useful for manual camera design and an algorithm for automatic design. We also implemented a configurable system that samples multiple parameters per frame and demonstrated its utility. Using our framework and this system, high-precision imaging applications become easier to develop and produce results of comparable or better quality than alternative solutions. Manual designs are easily understood but rely excessively on the notion of ideal components. Automatically designed trees contain surprising and complex element choices. These micro-balance the true response curves of components and are frequently more efficient.

As future work, we are investigating multispectral HDR for surveillance, alternative multi-focus approaches for matting, and multispectral high-speed for material testing in the context of splitting trees. Another area of future work is addressing the limitations of the current automatic design approach to find more efficient and general designs. For example, element cost should be a function of tree depth, since filters closer to the root must be larger to fill the field of view. Genetic algorithms are naturally suited to tree combination and should be more effective than our uphill simplex.

The optical elements form a filter system that terminates at digital sensor. In an application, this is always followed by a digital filter system. The next step is to simultaneously design both the optical and software filter systems.

References


