Proceedings IEEE Workshop on OmniDirectional Vision, June, 2000, pp. 47-53

Acquiring and Rendering High-Resolution Spherical Mosaics

ADAM KROPP NEEL MASTER SETH TELLER
MIT Computer Graphics Group
{akropp,neel,seth}@graphics.lcs.mit.edu

Abstract

We describe an acquisition and viewing method for high-resolution, high-dynamic range spherical image mosaics. These mosaics consist of hundreds of high resolution, high-dynamic-range color images, each annotated with 3-DOF orientation about a common optical center. An automated acquisition platform captures a pre-specified hemispherical tiling of images. Each image is acquired with specified approximate camera rotation and bracketed exposures for high-dynamic range.

Relative rotations among images, as well as internal camera parameters, are automatically refined using a correlation-based global optimization. The improved rotation and internal parameter estimates are used in an interactive viewer for smooth inspection of the resulting large spherical mosaic. The renderer uses a modified form of adaptive, incremental, parallel raycasting to sample the original images at pixel centers, preserving detail from the raw imagery while maintaining interactive display rates. We show an example dataset that resolves millimeter-scale detail at a viewing distance of ten meters.

1 Introduction

Our group is developing automated capture techniques for extended urban environments [8]. The idea of the effort is automatic reconstruction of textured 3D geometric models of urban landscapes from large numbers of images annotated with approximate intrinsic and extrinsic camera parameters. Images are grouped into *nodes*, each a set of images acquired at a common optical center. These form the fundamental data object of the city scanning Project [2].

A single node contains the information necessary to recreate a view of the world from one point in space and time. By combining all of the images in a super-hemispherical tiling around the viewpoint, it is possible to trace the color and intensity of a sample of light rays arriving at the optical center, effectively allowing one to look around with an appropriate viewer (e.g., as in [1]). Information viewed from a single point need not have any notion of the surrounding geometry (i.e., depth).

This paper describes a method for acquiring and viewing a single high-resolution high-dynamic range node, with a camera rig and software viewing system, respectively. The novelty here is the sheer amount of data – we show a mosaic of a large interior space which combines several hundred high-resolution, high-dynamic range images – and the fact that it can be viewed smoothly on a modest graphics engine.

To facilitate the acquisition of high-resolution mosaics, a specially designed sensor automatically acquires a node based on a user-specified sphere tiling (collection of orientations). The mechanized system enables the acquisition of nodes with hundreds of images, which would difficult and tedious if done manually. The platform also automatically generates a high-dynamic-range radiance map for each image [5].

A modified ray-casting algorithm interactively generates a view from the original images in real time, preserving the resolution of the acquired node. Radiance data is used to expand the dynamic range of the generated imagery.

2 Acquisition

The acquisition platform employs a motorized pan-tilt head, which rotates the camera about its optical center. The camera used for acquisition is a Wintriss Opsis 1300 camera with resolution of 1524×1012 pixels, fit with a 28mm lens, for an effective field of view of about 18 degrees horizontally and 15 degrees vertically. The hemispherical tiling to be acquired is specified programmatically by the user. By choice of optics and tiling pattern, we can achieve nearly any conceivable sampling density on the sphere, at the cost of course of longer acquisition times for mechanical slew, shuttering. Figure 1 depicts the 258-image tiling used for the mosaics and I/O in this paper.

High dynamic range images are acquired using five bracketing exposures as in [5]. This is performed automatically at each tiling orientation through programmatic camera control. The acquisition platform (Figure 6) is also equipped with GPS receivers, wheel encoders, and inertial navigation sensors to provide an initial 6-DOF estimate of absolute pose (exterior camera orientation) for each node [6]. Only the refined 3-DOF orientation estimates are used in this paper. Refined 3-DOF translation estimates are used elsewhere in the system for 3D reconstruction and off-node viewing.

3 Mosaicing

The sensor actuation and inertial unit provide good, but not perfect, annotations of rotation for each image. These are typically good to about a degree or so, whereas with our optics and image resolution pixel-perfect registration requires orientation known to about a sixtieth (one arcminute) of a degree.

To refine rotation estimates within a node, an automatic spherical mosaicing algorithm is used, based on maximizing a dense inter-image correlation function [3]. The algorithm assumes approximate initial rotation estimates and sufficient overlap between adjacent images to drive the optimization to a global minimum.

A coarse-to-fine approach is employed in which pose estimates from mosaicing low resolution images are used to bootstrap the optimization of higher resolution images. The algorithm represents image rotations as quaternions, which are subsequently used by the viewing algorithm to efficiently perform a projection of each image point to 3D. The output of the mosaicing stage is a rotation for each input image which relates that image to base image for each node.

The image dataset presented in these experiments was approximately 900MB. This large amount of data presents a challenge in performing a nonlinear optimization based on dense correlation of pixels. This is compounded by mosaicing method's use of luminance and derivative information for each pixel. To mosaic this dataset, the optimization was modified to improve both its memory footprint as well as the scheduling of computation and I/O. Since adjacencies between images are known beforehand, the algorithm computes correlation information on the current pair of images while pre-fetching the next set of required images into memory. A dedicated memory manager backs out image derivative and luminance information to disk when memory is at full capacity. This prevents having to recompute image information each time an image is swapped out to disk. This also requires a large amount of swap space, about four times the size of the input dataset. If swap space runs out, the memory manager will remove the image information on disk, and recompute the derivatives and luminance the next time the image is used.

Total time to mosaic the 900MB dataset on a four-processor 200Mhz Pentium-Pro with 1GB of RAM is approximately 17 CPU hours.

4 Viewing

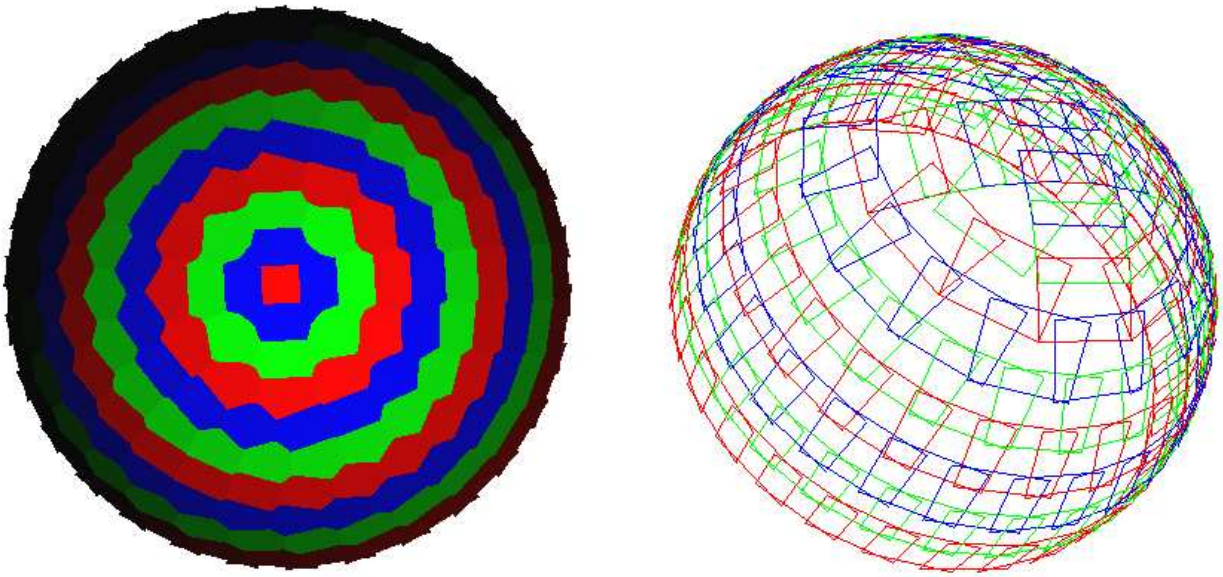
We designed and implemented a viewer to inspect high-resolution mosaic samples pixels from the original images at interactive rates with a modified form of raycasting. Although the viewer cannot view full- resolution mosaics at full frame rate, it can trade off smoothly between quality and update rate, a significant advantage for interactivity.

4.1 Overview

Every sample in the view originates from one or several source images. This guarantees that the node may be sampled to the same resolution as the original images provided, allowing very fine detail to be observed. This avoids limitations that would otherwise be imposed by underlying hardware. For example, our graphics engine allows a maximum texture map size of 2048x1024 pixels. Even a node acquired with our ordinary optics contains 46 images, each with 1524×1012 pixels. Thus with a direct sampling approach we have access to 34 times the data that hardware texture mapping would allow. In a high-resolution node this multiplier exceeds 200. This becomes apparent in Figures 3 and 5 where a large portion of an indoor lobby is shown, and then an object inside the scene is magnified. Details as small as millimeters at a distance of ten meters can be resolved in this node. The other main advantage of our raycasting approach is that per-pixel operations may be applied on the fly, in contrast to graphics hardware which applies operations per vertex, then interpolates to interior pixels. This allows for such effects as blending between multiple images to obtain a final pixel value; seamlessly integrating low-resolution and high-resolution images to use extra memory only in areas that need it; and adjusting the viewer's visual response function for viewing high dynamic range images interactively (i.e., without image preprocessing).

4.2 A Raycasting Method

The viewer, in its simplest form, takes a user-controlled view direction and field of view and rasterizes it onto the screen. For each viewport pixel, a ray is calculated from the optical center of the node to the corresponding point on the viewport, then extended out into space. The raycaster then determines the radiance along that pixel based on its direction. This involves determining which image or images that ray intersects, calculating the point on each image in image coordinates where the ray intersects it, and finally performing a weighted average of the color from each image based on the distance from the ray to the center of the image, as described in [7], [1]. This entire calculation is



(a) (b)

Figure 1. Visualization of 258 image frustum: (a) the FOV for each image; (b) wireframe of overlapping frusta.

enclosed in a nested set of `for` loops that scans through each line and each pixel within that line, and can be easily incrementalized and parallelized.

To speed the calculation of the ray, an incremental approach is used. First, the ray to the lower-left corner of the viewport is determined (corresponding to pixel (0,0) of the viewport). A ray `start` is constructed from the eye to the point. Next two delta vectors, `dx` and `dy`, are constructed to increment the starting vector in the screen `x` and `y` directions. Finally, at the start of each line, the view vector is set to `start + dy*j` (where `j` is the raster index). It is then incremented after each pixel calculation by adding `dx`. Next, the image or images that the ray intersects must be determined. The naive approach to this problem is to simply project the ray onto the plane in which each image is located, then check that the point of intersection is within the borders of the image. This is efficient for a small number of images, but as the number of images in a node increases it becomes wasteful. Since this operation must be performed once per pixel, it must be as fast as possible to allow interactive inspection. To speed this search, a preprocessing stage subdivides space in a tree structure, and keeps pointers to the images that intersect each region. The structure is built by adaptive refinement of the triangular faces of an initial (full-sphere) octahedron (Figure 2), not unlike quad-tree or kd-tree data structures [4].

4.3 Ray Location Data Structure

During the preprocessing stage, a subdividable ImageBranch record is instantiated for each triangle. This record keeps track of all images that fall within its associated triangular region. The preprocessor passes all input images to each of the eight ImageBranches. If a branch contains the image, it is added to the branch's internal list. Once the number of images reaches a constant threshold, the ImageBranch subdivides into four smaller equally-sized triangles, passing its stored images to its children. The resulting structure can be searched with $O(\lg n)$ region comparisons rather than $O(n)$, giving a large reduction in computation and increase in viewing speed (or quality for fixed speed).

Many of the values needed for the ray-triangle intersections used in this search can be pre-computed, giving an additional time savings. The structure is built such that each ray is passed to the top level, which determines which of eight branches to search. A ray can be placed into one of these regions using three comparisons (one for each directional component) with zero. The ray is passed to that branch, which determines if it contains the ray. If it does, the branch searches its (short) list of images for intersections, or passes the ray to its four children. The process is repeated recursively until image intersections are found, or none are found. The query returns a list of images intersected by the ray, if any.

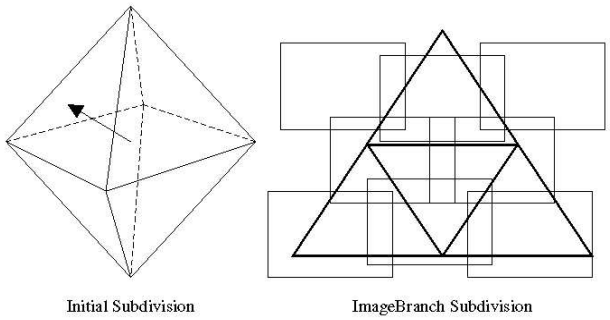


Figure 2. The data structure for rapid ray location in the spherical tiling.

Once the images intersected by the ray are known, pixels can be sampled in each image. Each image has an associated transformation matrix specifying its projection from the node origin (optical center). The ray is projected by this matrix, resulting in a point (x, y, z) that can be used to index into the actual image data as $(\frac{x}{z}, \frac{y}{z})$. The radiance value from each image is then weighted inversely by its distance from the center of the image, and the values are added and divided by the total weight. Vignetting effects which occurred during image formation are corrected during the aquistion process.

4.4 Optimizations

Three modifications were made to optimize performance.

4.4.1 Precomputing weights

First, the pixel weights depend only on x, y index, not on the image itself, so they can be pre-computed and stored in an array. The appropriate weight can be accessed by the same index used to read the radiance sample from the image. This approach increases speed at the cost of additional memory usage, an acceptable tradeoff when responsiveness is of primary concern. If all images are guaranteed to have the same size, a single set of weights can be calculated at the beginning of time for use with all images; otherwise a per-image array could be used.

4.4.2 Exploiting continuity

The second improvement comes from noting that, in general, adjacent pixels come from the same image. Additionally, the transformation to image space is expensive, so an incremental method of finding the next pixel is desirable. Both of these are addressed during the sampling stage. When the ray transformation is performed, an additional ray through the next pixel is also transformed. Since this is effectively a perspective transformation, distances are not preserved, so a simple delta between pixels will not yield proper results. However, the z value is recorded as well, and by storing the delta for x, y, and z, the viewer can increment all of them across the image. Using this incremental value in each image, a count is determined for the number of samples left before an edge of the image is reached. The top-level loop can then simply increment each pixel location and sample each image until the count reaches zero. At that point, it must again find the image set to be sampled, by appeal to the ImageBranch data structure. This reduces the number of ray-image intersection searches from once to pixel to once per image edge intersected per scan line. Since each image typically spans between an eighth and a fourth of the viewing area (or more when the field of view is quite wide), a full search must be performed only a few times per scan line.

4.4.3 Parallelization

The final optimization is to split the raycasting task among more than one processor on a multiprocessor machine using `pthreads`. Two main structures facilitate this. First, a RaycastInfo record is created for each scan line of the viewport, encapsulating all of the information needed to calculate one line of the image, including the ray pointing to the beginning of the line, the `dx` vector, a pointer to the line of the output image, and various other accounting information. Next, the record is added to a thread-safe queue. Worker threads wait on a semaphore until a line is available for processing. Once a line is available, the acquiring thread removes it from the queue and processes the line. When done, it delivers the rendered raster and again waits on the queue. This produces an improvement in frame rate which is nearly linear in the number of available processors.

4.5 High Dynamic Range

The raycaster described above can use an arbitrary number of images of arbitrary spatial resolution, but is limited in its reproduction of a node due to physical limitations of the camera – digital cameras have a very limited dynamic range (typically seven bits or so) – and of the computer graphics display. The location of this range can be adjusted by changing the exposure time on the camera, but a single image cannot capture detail in both shadows and very bright areas. A trial node was captured at multiple exposure settings, and the results were combined to produce a radiance map that reflects the true brightness (radiance) of every pixel up to a single absolute scale [5]. This map was then compressed using a logarithmic scale into 24-bit

(a) (b)

Figure 3. Figure (a) wide angle view (b) narrow fi eld of view, zoomed. The dotted lines in (b) denote image boundaries

pixels, but retaining the full dynamic range. For each radiance image minimum and maximum radiance values are stored, allowing conversion to true radiance values.

In adding radiance map support to our viewer, one goal was to adjust the displayed range and visual response function on the fly. This precludes mapping each image with the appropriate visual response function each time the user adjusts gamma. Also, because of the use of logarithms in both the decoding and visual response functions, it would be prohibitively slow to convert each pixel as it is sampled. The solution was to keep the images in their logarithmically compressed form, and generate a 256-entry lookup table for each image which converts each channel (r, g, b) to the appropriate value. These lookup tables must be regenerated each time the user changes the response function or dynamic range, but they contain only 256 entries, instead of over a million pixels. In the decode stage, a simple table lookup is performed per channel per pixel, causing a minimal performance hit. As can be seen in Figure 4, details in the shadows of MIT's Lobby 7 are visible, while with a different gamma the clouds outside the windows are visible, though it is a bright sunny day. The image can be viewed with a compressed dynamic range so that all values are visible at once, or the range can be expanded to bring out detail in one particular segment of the range (at the expense of saturating regions that are brighter or darker than the target region).

5 Conclusion

We described an actuated camera rig and control system for capturing spherical mosaics with high spatial resolution and high dynamic range. The data is viewed with an optimized, adaptive, parallelized software viewer that requires only an ordinary CPU and framebuffer, but no specialized graphics hardware. The viewer smoothly trades off between image quality and frame rate. Overall these elements provide a novel capability for capturing and inspecting high fidelity spherical imagery.

Future work includes improvement of the modeling and use of the visual response function. Currently, users interactively specify an gamma to modify the contrast in the image. One approach would be to try and adaptively change the gamma as the viewer navigates the mosaic. Another avenue for future work is to better handle high contrast images, as described in [9]. This poses a challenge due to the requirement of a real-time response for a large amount of data. Furthermore, as the user varies the field of view, the displayed dynamic range for the current view should change continuously.

6 Acknowledgments

We are grateful to the anonymous reviewers for their comments and suggestions.

(a) (b)

Figure 4. Figures (a) long exposure and (b) short exposure

References

[1] S. E. Chen. Quicktime VR – an image-based approach to virtual environment navigation. In *SIGGRAPH '95 Conference Proceedings*, pages 29–38, Aug. 1995.

[2] S. Coorg, N. Master, and S. Teller. Acquisition of a large pose-mosaic dataset. In *CVPR '98*, pages 872–878, 1998.

[3] S. Coorg and S. Teller. Automatic spherical mosaics with quaternions and dense correlation. In *IJCV*, 2000. To appear.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 1997.

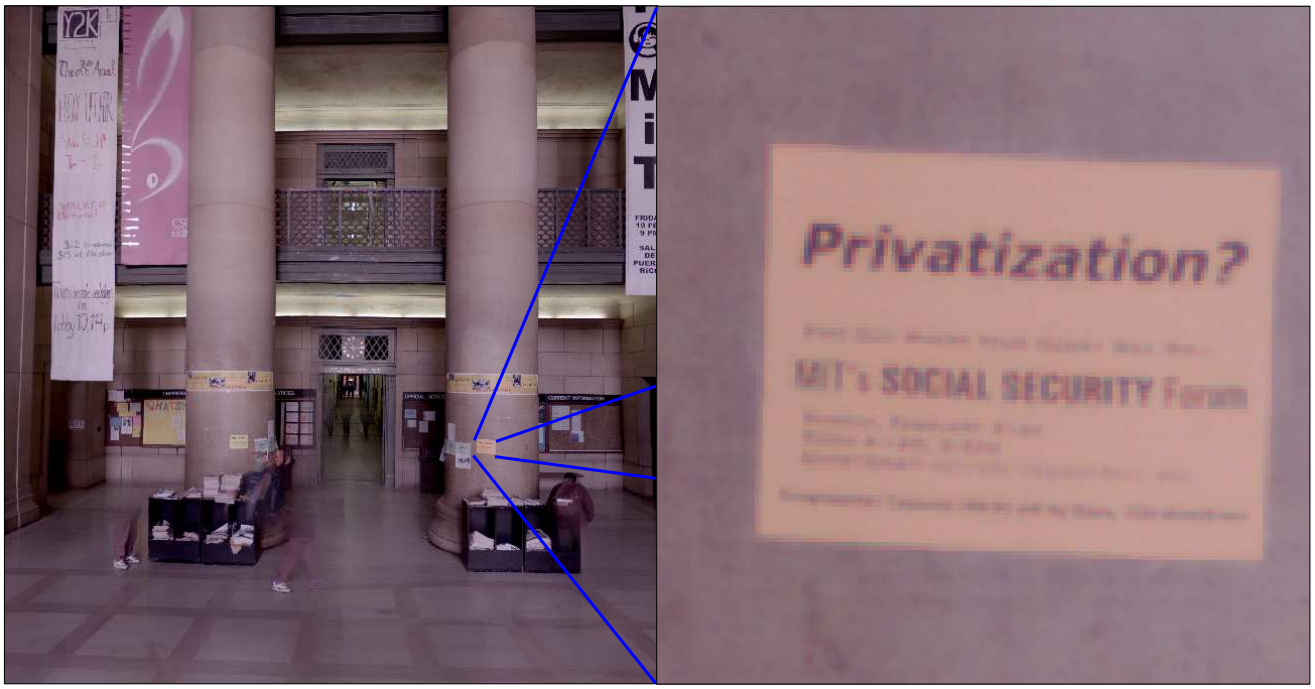
[5] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97 Conference Proceedings*, Aug. 1997.

[6] D. DeCouto. Instrumentation for rapidly acquiring pose imagery. Master's thesis, Dept. of Electrical Engg. and Computer Science, MIT, 1998.

[7] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, Mar. 1996.

[8] S. Teller. Automated urban model acquisition: Project rationale and status. In *Proceedings of the Image Understanding Workshop*, 1998.

[9] J. Tumblin and G. Turk. Lcis: A boundary hierarchy for detail-preserving contrast reduction. In *SIGGRAPH '99 Conference Proceedings*, pages 83–90, Aug. 1999.



(a) (b)

Figure 5. Figure (a) wide angle view (b) narrow fi eld of view, zoomed

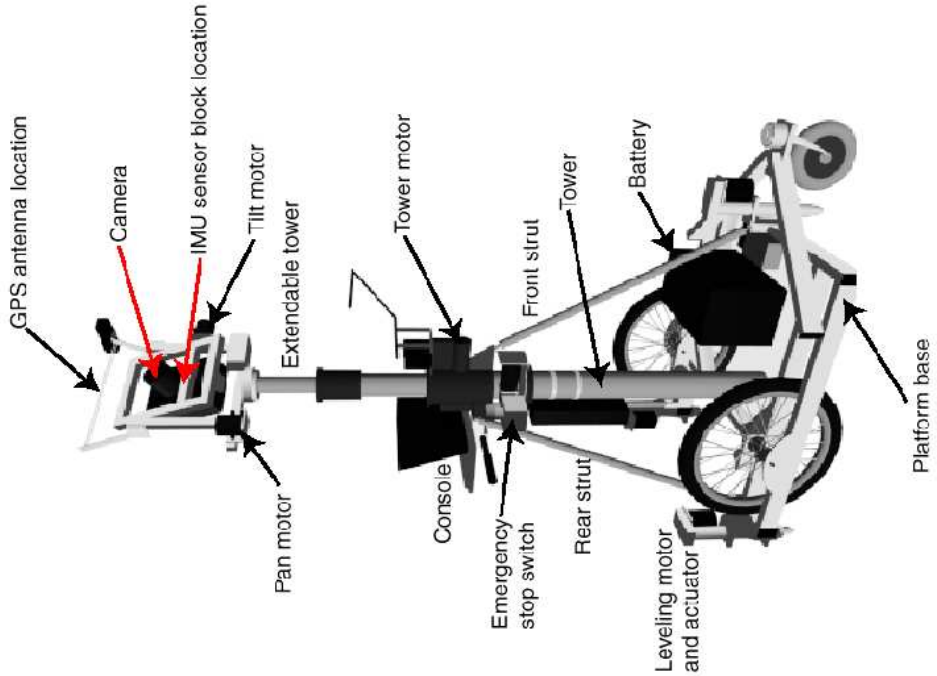


Figure 6. Diagram of the acquisition platform