The Graph Camera

Abstract
A planar pinhole camera captures only a small fraction of a 3-D scene due to occlusions, which makes exploring complex scenes inefficient. Dynamic scenes pose the additional challenge of transient regions of interest that move or degrade by the time sequential pinhole navigation reveals them to the user.

We introduce the graph camera, a non-pinhole with rays that circumvent occluders to create a single layer image that shows simultaneously several regions of interest in a complex 3-D scene. The graph camera image exhibits good continuity and little redundancy. The graph camera is constructed from a planar pinhole camera through a series of view frustum bending, splitting, and merging operations. We have developed several constructors, from simple ones that enhance a pinhole with a few additional viewpoints, to complex ones which distribute rays throughout the scene and eliminate the need for navigation altogether. Although there are many frusta, the graph camera model provides a fast projection operation. This enables rendering at interactive rates in feed-forward fashion, making dynamic scenes tractable. Finally we explore the use of the graph camera for the integration of multiple video feeds, enabling monitoring complex spaces with a single image.

1 Introduction
Most 3-D computer graphics applications rely on the planar pinhole camera (PPC) model to compute images of a 3-D scene. One reason is that the PPC model is simple, enabling efficient software and hardware implementations that render complex scenes at interactive rates. Another reason is that the PPC model approximates the human eye well, producing images that are familiar to the user. However, the PPC model is limited. Whereas the field of view limitation has been addressed by innovations such as spherical, cylindrical, or cube map pinhole camera models, relatively little has been done to remove the requirement that all rays pass through a common point—the pinhole. This pinhole constraint limits images to scene regions to which there exists direct line of sight. In scenes with complex occlusions, a PPC image captures only a small fraction of the scene.

A conventional approach for alleviating the pinhole limitation is based on interactive navigation. The user guides a virtual PPC to gain direct line of sight to scene regions of potential interest, one at a time. Such sequential visualization has two important disadvantages. First, it is inefficient, especially for scenes where the density of regions of interest is low. The user spends time systematically ruling out regions as opposed to examining the regions of interest. Second, sequential navigation can be ineffective in dynamic scenes when the user misses regions of interest that move or decay by the time navigation reaches them.

A possible solution is to render the scene with several PPCs simultaneously. However, a large number of cameras are required to achieve satisfactory coverage, and the resulting images are poorly integrated. Discontinuities across the boundaries of
individual images require the user to examine the images one at a time in order to adapt to each one of the multitude of contexts. The palliative solution of building redundancy into the set of images not only fails to solve the problem but it is also expensive. We propose a novel 3-D graphics paradigm that advocates abandoning the traditional simplicity and rigidity of the camera model in favor of designing and continually fitting the camera model to the 3-D scene it renders. This camera model design paradigm is supported by the graph camera, a novel non-pinhole camera model with rays that circumvent occluders to sample simultaneously many or all regions of interest in a 3-D scene. The graph camera integrates many PPC images into a single image, which improves navigation efficiency. The graph camera image also exhibits good continuity and little redundancy, therefore avoiding the disadvantages of multiple PPC images.

In Figure 1 two graph cameras were constructed to see beyond the first road intersection, one in front the user (right half of graph camera image) and one behind the user (left half). The user is located in front of the green house seen in the ray visualization image. Each graph camera integrates several PPC images with good continuity and no redundancy. For example the forward graph camera samples all 4 road branches of the clock store intersection. The graph cameras share the same near clipping frame which is flush with the road to provide continuity between the two graph camera images. The final image is obtained by projecting the two graph camera images on a dynamic surface.

A graph camera is constructed starting from a regular PPC whose frustum undergoes a series of bending, splitting and merging operations. The resulting graph camera is literally a graph of many PPCs, each defining a frustum. In order to describe the graph camera rays we generalize the definition of a camera ray to the set of 3-D points that project at a given image location, which allows for rays that are not straight lines. With this, a graph camera ray is a set of connected segments. Each graph camera in Figure 1 was constructed with a split, split, merge sequence and contains 6 PPC frusta. A typical ray has 3 segments: to, through, and beyond the road intersection.

The literal malleability of the graph camera model allows for many strategies for defining the rays that best capture a given scene at a given moment in time. We have developed several graph camera constructors which span a continuum. At one end are constructors that produce simple graph cameras which enhance a PPC with a few additional viewpoints. At the other end are constructors that produce complex graph cameras which sample all regions of interest in a 3-D scene eliminating navigation altogether.

A portal-based constructor modifies a PPC such that it sees inside given rectangles (i.e. portals), see Figure 3. An occluder-based constructor modifies a PPC such that it sees inside given rectangles (i.e. portals), see Figure 3. An occluder-based constructor generates rays that reach around an object (i.e. occluder) to sample surfaces that are in the occlusion shadow of the object (Figure 4). Another type of constructor uses a 3-D maze as a scaffolding around which the graph camera is built recursively. The graph cameras in Figure 1 were built using a maze, and the recursion construction was stopped at level 1. The graph cameras in Figure 5 and Figure 6 were also constructed with a maze but with deeper recursion levels. A final type of constructor relies on the user to design the graph camera interactively. The graph camera used in Figure 7 was built interactively in minutes and produces an animated image that summarizes the cartoon town scene.

Despite its complexity, the graph camera model provides a fast projection operation that maps a given 3-D point directly to the output image, with consistent depth allowing visibility sorting. The projection operation enables rendering efficiently in a single
pass, without the need for compositing individual PPC images into the final graph camera image.

A graph camera can be implemented physically with a video camera for each frustum. Such a graph camera integrates the individual feeds and enables monitoring the entire space with a camera for each frustum. Such a graph camera integrates the into the final graph camera image.

The graph camera overcomes occlusions by moving scene regions that compete for the same PPC image location to disjoint graph camera image locations. In essence, the graph camera trades image visibility or depth layer resolution. For this approach to be effective the graph camera image has to be sufficiently large to accommodate the multiple regions of interest. The approach is supported by current high resolution displays. A single LCD display with a WQXGA resolution of 4 million pixels can show a graph camera image with 64 regions of interest with an average footprint of 256 by 256 pixels. A 16-million-pixel tiled display can show 256 such regions of interest. The user can change the screen real estate allocation dynamically, emphasizing one or a few regions, while the other regions continue to be rendered, providing context.

In summary, this paper’s contributions are:
- the graph camera, a versatile and efficient non-pinhole camera model that overcomes occlusions in complex 3-D scenes,
- portal, occluder, 3-D maze, and interactive graph camera constructors, and
- a method for the physical implementation of the graph camera.

2 Prior work

In computer graphics non-pinhole cameras have been developed in the context of image-based rendering, of artistic rendering, and of reflection rendering. We briefly review each of these contexts, discussing the suitability of the camera model developed for creating a non-redundant and continuous image of several regions of interest of a 3-D scene. Then we briefly review related work in scientific and information visualization.

Image-based rendering

Several non-pinhole camera models have been developed in the context of image based rendering for the purpose of scene modeling and rendering. The light field [Levoy 1996, Gortler 1996] is a 2D array of PPC images which amounts to a powerful camera model that captures a dense set of rays. However, light fields are ill-suited for the application at hand. First, under the diffuse surface reflectance model assumption, the light field is highly redundant and extracting a single copy of a given set of regions of interest is difficult. Second, (synthetic) light fields have lengthy rendering times: the scene has to be rendered for each of the many PPCs. Ulterior research has reduced the number of redundant rays using surface geometry information (e.g. surface [Wood 2000] and unstructured [Buhler 2001] light fields), but construction remains an offline process. Light fields have to be used as a set of pre-computed color samples rather than as a set of rays, which precludes dynamic scenes.

Layered depth images (LDIs) [Shade 1998] generalize the PPC image by allowing for more than one sample along a ray. Like the graph camera, the LDI camera is a PPC whose rays are broken into several segments, but in the case of the LDI the segments are collinear. The application of LDIs is 3D image warping [McMillan 1995] without the problem of disocclusion errors, which are artifacts due to missing samples for surfaces that are visible in the desired view but were not visible in the reference image. The LDI avoids the redundancy of light fields, but it remains difficult to combine many regions of interest in a single-layer output image. Moreover the number of samples stored at each LDI pixel varies widely; therefore it is impractical to construct the LDI by successive rendering passes and by peeling off the nearest layer. Adequate LDIs are built by combining a large number of PPC images rendered from views around the LDI reference view. Like in the case of light fields, LDIs are built offline and the LDI camera is too inefficient to accommodate dynamic scenes.

Occlusion cameras [Mei 2005] are a family of non-pinholes with rays that reach around occluders to gather samples that are barely occluded from the reference viewpoint and thus are likely to be needed to support viewpoint translation without disocclusion errors. Occlusion cameras produce single layer images that show more than what is visible from a single point, and they can be rendered efficiently with hardware support, but they do not offer the ray modeling flexibility required to reach distant regions of a complex 3-D scene.

Multiple center of projection (MCOP) images [Rademacher 1998] collect samples with a vertical slit camera that slides along a user defined path. Possible goals in path selection are good scene coverage or artistic value of resulting image. The great flexibility in defining the rays of the MCOP camera and the resulting single-layer image makes it attractive in the context of our problem of simultaneously capturing several regions of a 3-D scene.
scene. An MCOP camera could be used to create an image like the graph camera image shown in Figure 2. However, MCOP cameras are inefficient: images have to be rendered by ray tracing or by rendering the scene in feed-forward fashion for each center of projection along the camera path. Non-pinhole cameras have also been employed to facilitate the creation of panoramas for cel animation [Wood 1997]. The rays of interest are defined by the desired scene shots. The non-pinhole renders a multiperspective panorama which simulates camera motion in a 3D scene when it is viewed through a rectangular frame sliding on a predetermined path. Like for MCOPs, the panorama is rendered by finely discretizing the 3D camera path and by rendering an image for each position along the path, which is slow. 

Artistic rendering 

Another application of non-pinhole camera models is in the context of artistic rendering where they are employed to render multiperspective images, similar to the ones produced by the graph camera. In one system, individual PPCs are attached to scene objects, and the resulting sprites are composited in a multi-projection image [Agrawala 2000]. For a small number of objects, the multi-projection image can be updated interactively. The approach of attaching a pinhole to each object has the disadvantages of not scaling with scene complexity, of difficult—sometimes impossible—visibility ordering, and of not supporting multiple perspectives per object. Another multiperspective rendering system [Yu 2004b] partitions an image plane into general linear camera (GLC) triangular images. A GLC is constructed from three given rays [Yu 2004a] so it offers some flexibility for modeling rays such that they reach the desired regions of a 3-D scene. Moreover, GLCs also have the advantage of fast projection. However, combining several GLCs is non-trivial. The solution adopted by Yu et al. was to blend the rays of neighboring GLCs to provide a continuous ray space which generates an image with smoothly varying perspective. The resulting compound non-pinhole camera model does not provide fast projection and rendering is performed offline by ray tracing. Multiperspective images of real-world scenes can be constructed by re-sampling a video cube—a stack of images gathered by moving a video camera along a continuous path [Seitz 2003]. The video cube has been used for impressionism, cubism, and abstract aesthetic video effects [Klein 2002].

Reflection rendering 

The need for non-pinhole cameras also arises in the context of reflection rendering. Curved reflective surfaces perturb the rays "leaving" the pinhole modeling the desired view, thus second and higher order reflected rays are not concurrent and amount to a non-pinhole camera. Indeed, reflections have been modeled with the help of the general linear camera multiperspective rendering framework [Yu 2005]. The sample-based camera reflection rendering method [Popescu 2006] leverages the coherence of small contiguous sets of reflected rays and replaces them with PPCs. The PPCs are stored at the leaves of a binary space partitioning tree which defines a sample-based camera. The graph camera and the sample-based camera are similar in the sense that both are collections of PPCs. However, the sample-based camera constructor focuses on tightly approximating a given set of non-concurrent reflected rays, whereas the graph camera constructor aims to provide flexibility for specifying rays such that they elude occluders and reach many or all regions of interest in a 3-D scene.

3 Graph camera model

The graph camera is a graph of planar pinhole cameras constructed from an initial planar pinhole camera $PPC_0$ by applying a sequence of one of three primitive construction operations: bending, splitting, and merging. $PPC_0$ defines the first segment of the piecewise linear rays of the graph camera and collects the graph camera image.

Bending Given a planar pinhole camera $PPC_i$ with center of projection $C_i$ a plane $p$ that intersects all the rays of $PPC_i$ and a point $C_{i+1}$, the frustum of $PPC_i$ is bent with the following steps:
- the far plane of $PPC_i$ is set to $p$
- a new planar pinhole camera $PPC_{i+1}$ is constructed with near plane $p$ and center of projection $C_{i+1}$
- the far plane of $PPC_{i+1}$ is set to some default yon distance
In Figure 9, left, the frustum of the camera before bending is \( P_iQ_iR_iS_i \). After bending the two frusta are \( P_iQ_iR_iS_iP_{i+1}Q_{i+1}R_{i+1}S_{i+1} \). The rays of the resulting graph camera change direction at \( p \) (see red segments).

**Splitting**

Given a planar pinhole camera \( PPC_i \), a set of planar polygons and a set of points, the frustum of \( PPC_i \) is split as follows:

- find the subset of polygons visible by \( PPC_i \)
- for each visible polygon \( PLY_{i+1}^k \) define a new planar pinhole camera \( PPC_{i+1}^k \) with center of projection \( C_{i+1}^k \), with near plane \( PLY_{i+1}^k \), and with clipping planes defined by the edges of \( PLY_{i+1}^k \)
- a ray of \( PPC_i \) that encounters a visible polygon \( PLY_{i+1}^k \) is clipped by the plane of \( PLY_{i+1}^k \).

In Figure 9, middle, a PPC with center of projection \( C_i \) is split with a set of 2 polygons \( P_iA \) and \( AQ_i \) and 2 points \( C_{L,i} \) and \( C_{R,i} \). After the split there are 3 PPCs with centers of projection \( C_{L,i} \), \( C_{L,i+1} \), and \( C_{R,i+1} \). The resulting rays diverge after the splitting polygons, as a result of incorporating the two new viewpoints into the camera model.

**Merging**

Merging is the dual of splitting. After merging, rays become concurrent. Given a set of planar pinhole cameras \( PPC_i^k \), a plane \( p \) that intersects all the rays of all cameras, and a point \( C_{i+1}^k \), the

Figure 10 Portal-based graph camera image (top) and corresponding PPC image for comparison (bottom). The graph camera samples inside 4 portals to reveal alleys and side streets.

Figure 11 Portal graph camera (left) and PPC (right) image.

Figure 12 Occluder construction.

The graph camera model and the generic primitive construction operations defined above offer plenty of flexibility for designing rays that reach in the 3-D scene where desired. Graph camera design goals are comprehensiveness (sample many regions of interest), continuity (integrate the frustum images smoothly), non-redundancy (minimize repeated samples), and rendering efficiency (keep the number of frusta as low as possible).

4 Graph camera construction

The input to the portal-based graph camera construction algorithm is a set of rectangular portals and the current PPC \( PPC_0 \) used to explore the scene. Portals are defined in the scene by the user in a pre-processing stage. In addition to the portal quad, the user also specifies a field of view for the PPC that samples it. In Figure 10 a narrow-field of view portal reveals the narrow alley near-left. The resulting graph camera has depth 1, with one leaf per portal.

The center of projection \( C \) for the PPC of a portal is determined based on the desired field of view and on the center of projection \( C_0 \) of \( PPC_0 \). \( C \) is placed at the same height as \( C_0 \) to align the horizons of the main and leaf cameras. In order to allow the user to walk through a portal without visual discontinuity, the portal camera needs to retract as the view approaches the portal. This is simply achieved by interpolating \( C \) into \( C_0 \) as the distance to the portal decreases. When the two centers of projection are identical, the portal split becomes a no-op. In order to allow a user to walk parallel to a portal while still benefiting from the view inside the portal, the interpolation factor is also modulated based on the angle between the view direction of PPC and the portal normal. Large angles make the portal view persist longer.

As \( PPC_0 \) navigates through a portal \( P \), additional portals, previously hidden by \( P \), can become visible. To avoid an abrupt change in the graph camera images, views of newly visible portals are gradually deployed over several frames. Portal-based graph cameras take advantage of naturally occurring scene exploration hotspots (Figure 11). The resulting graph camera typically has only a few frusta and is thus very efficient.

4.2 Occluder-based construction

Large objects can hide a considerable part of the scene to
a pinhole, requiring the user to navigate around such occluders to examine the hidden parts. We have developed a graph camera constructor that takes as parameters an occluder bounding box \(BB\), a desired shadow length \(l\), a desired outgoing field of view \(fov\), and the current PPC \(PPC_0\) and constructs a graph camera of depth \(d\) through a splitting followed by merging (Figure 12).

\(PPC_0\) is split into two PPCs with centers of projection \(C_L\) and \(C_R\). The disocclusion capability of the graph camera stems from the converging view directions of the two cameras. \(C_0\) is found as follows (similar for \(C_R\):

- compute axis aligned bounding box \(BB_a\) of \(BB\) in \(PPC_0\) eye space
- splitting depth \(z_s\) is given by the closest \(BB_a\) corner
- splitting point \(S\) is given by the projection of the center of \(BB\) onto the \(z=z_s\) plane
- merging depth \(z_m\) is computed as \(z_s+l\)
- point \(S_0\) is the \(PPC_0\) projection of \(S\) onto plane \(z=z_m\)
- line \(SC_L\) is defined by the far left corner of \(BB_a\)
- point \(L_s\) is found as the \(PPC_0\) projection of \(L_s\) onto plane \(z=z_m\), where \(L_s\) is the extent of \(PPC_0\)’s frustum at \(z=z_m\)
- point \(C_0\) is finalized on \(SCL\), to obtain a field of view \(LCLS\) equal to the field of view of \(PPC_0\)
- the center of projection \(E\) of the outgoing frustum is found using points \(L_s\), \(R_s\) and the input parameter \(fov\)

Since rays \(C_0S_L\) and \(C_0S_R\) are constructed to clear \(BB_a\) the rays left and right of them, respectively, are guaranteed to clear the occluder. The shadow is effectively shrunk to \(l\) (Figure 13).

The construction algorithm described is not guaranteed to sample all surfaces that \(PPC_0\) did. For example the graph camera does not sample inside \(R_sR_sR_s\). This could be remedied by using \(R_s\) instead of \(R_s\) to define the right end of the frustum of \(C_0\). A conservative constructor would use the yon plane of \(PPC_0\) and make sure that the entire original frustum is sampled. This implies enlarging the resolution of the output image beyond that of \(PPC_0\), an expense not justified since the user is most interested in the surfaces emerging from underneath the occluder and not in the surfaces sampled at the periphery of the \(PPC_0\) frustum. The user has the option of lengthening the shadow of the occluder interactively to bring back the periphery surfaces when so desired. As the occluder or the graph camera move, the graph camera is reconstructed effectively tracking the occluder.

\[\text{Figure 14 Maze used to construct graph cameras used in Figure 1.}\]

\[\text{Figure 15 Illustration of graph cameras created by D1 maze-based constructor.}\]

\[\text{Figure 16 Steps in creating the final D1 image (Figure 1), shown for illustration purposes.}\]

\[\text{4. 3 Maze-based construction}\]

Portals and occluder construction works less well when portals or occluders are seen from afar and their image projection is small. The image of an entire adjacent room or street collapsed in a tiny area of the screen does not allow the user to examine the space revealed without first getting closer.

To further increase the disocclusion power of the graph camera in complex scenes, we have developed a constructor that uses a 3-D maze as a virtual scaffolding to support the graph camera frusta. Figure 14 shows such a 3-D maze. The nodes are modeled with boxes of various sizes. The corridors connect corresponding node faces. Rings can be placed on corridors to modulate their cross-section without incurring the cost of additional frusta.

\[\text{D1 construction}\]

One maze-based construction variant allows the user to navigate along the central line of the corridors of the maze. The position of the user is defined by a rectangular frame sliding along the corridor, with a size varying according to the corridor rings. Two graph cameras are constructed sampling in front and behind the user. The graph cameras go beyond the first intersection (hence the name D1); in other words, each samples the 1 incoming and 3 outgoing branches at an intersection. Each graph camera is constructed as illustrated on the maze in Figure 15. The split is achieved in two steps. First the main frustum is split in 2, and then each of the two frusta is split again in 2. Finally the two forward going leaf frusta are merged.

In Figure 16, the left image shows the two graph camera images, forward at the top and backward at the bottom. In order to improve the readability of the image, the bottom image is rotated to the left of the forward image (Figure 16, right). Since the geometry of the buildings lining the road is considerably different from that of the maze corridor, sometimes 3-D split lines remain partially visible in the graph camera image introducing a discontinuity. To further improve the readability of the image, the image is cut along the visible part of the 3-D split lines that form the 3 leaf frusta. Finally the leaf frustum images are spread apart using the cut segments. In the final image (Figure 1) the leaf frustum images are connected by a central core, and are fused up to where the split lines are hidden by geometry. The depth of the
cuts changes from frame to frame depending on the geometry that occludes the vertical split 3-D lines. To prevent a jittery update of the display surface, the cut depth is averaged over a sequence of frames. The display is a triangle mesh texture mapped with the graph camera images.

The construction algorithm described so far applies to the case when the user is in a corridor, sufficiently far away from an intersection. As the user approaches the intersection, the backward leaves collapse gradually in favor of the central leaf. Then all but one forward leaf collapse in favor of the leaf showing the desired outgoing direction. At this time both graph cameras have become PPCs. The two PPCs facing in opposite directions and attached through the near frame turn as desired by the user. Once the intersection is cleared, all leaf frusta are deployed once again.

Comprehensive construction

Whereas portal and occluder constructors were operating on a freely moving PPC, D1 construction reduces navigation to one degree of freedom, progress along corridor center lines and freely moving PPC, D1 construction reduces navigation to one degree of freedom, progress along corridor center lines and attached through the near frame turn as desired by the user. Once the intersection is cleared, all leaf frusta are deployed once again.

Interactive construction

The interactive constructor exposes the controls of the primitive bending, splitting, and merging operations to the user via a graphical user interface. Efficient rendering enables real time graph camera updates. The graph camera used in Figure 19 connects two floors by collapsing the stairs.

5 Rendering

Good rendering performance is essential for the graph camera to support dynamic scenes and to provide immediate feedback during interactive construction. A scene modeled with triangles is rendered with a graph camera one PPC frustum at the time. For each frustum, the relevant triangles are first found using a conventional hierarchical space subdivision scheme. We use an octree. A triangle t is rendered with a frustum PPC with the following steps:

1. Clip t with the faces of PPC to triangles t_i
2. For each {t_i}
   2.1. For each vertex V_j
      2.1.1. Compute distorted vertex V'_j
   2.2. Render distorted triangle (V'_0, V'_1, V'_2) with PPC0

PPC0 is the initial planar pinhole camera that collects the graph camera image. The rendering algorithm essentially computes a distorted scene (Figure 8) which when rendered with PPC0 produces the same result as when rendering the original scene with the graph camera. Given a vertex V contained by a frustum PPCk the distorted vertex V’ is computed as follows.

First, the projection (u_k, v_k) of V on the image plane of PPC0 is found with the equation:

\[
V = C_k + M_k \begin{bmatrix} u_k \\ v_k \\ 1 \end{bmatrix}
\]

\[
\begin{bmatrix} u_k \\ v_k \\ 1 \end{bmatrix} = M_k^{-1} (V - C_k) \frac{1}{w_k}
\]

where C_k and M_k are the COP and 3x3 camera matrix of PPCk. The projection is then mapped directly to the output image plane of PPC0 at coordinates (u_k, v_k) by multiplication with a 3x3 matrix Q_k. Q_k concatenates the mappings from the image plane of PPC_i to that of PPC_k, from the image plane of PPC_k to that of PPC0, and so on all the way to PPC0 (see Appendix A). The matrix Q_k is pre-computed for each PPC_k during graph camera model construction.
of streams is a long-term challenging computer vision problem. Providing high-resolution per-pixel depth for the video Section 5) is implemented by the physical video camera, pixel depth. Since the first step of the projection (equation 1 in can be combined into a graph camera without the need of per-
the rendering algorithm can be amended such that video streams need to be overcome.

order to implement the graph camera physically, several problems comprehensive single-image visualization to real-world scenes. In
Creating a physical graph came ra brings the benefits of
6  Physical implementation of the graph camera

Applying the matrix $Q_k$ we obtain:

$$
\begin{bmatrix}
  u_0 \\
  v_0 \\
  w_0 \\
  1
\end{bmatrix}
= \begin{bmatrix}
  u_k \\
  v_k \\
  1 \\
\end{bmatrix}
= Q_k M_k^{-1} (V - C_k) \frac{1}{w_k}
$$

The equation above is sufficient to calculate the projection $(u_0, v_0)$ of $V$ with $PPC_0$. However, using an arbitrary depth to establish the position of $V'$ along $PPC_0$ leads to an incorrect interpolation of rasterization parameters. What is needed is a depth of $V'$ which ensures that a linear variation in the undistorted space of $V$ matches to a linear variation in the distorted space of $V'$, such that conventional model space interpolation of the distorted triangle produces correct results. In order to find a correct depth of $V'$ we first write the projection of $V'$ with $PPC_0$ as:

$$
V' = C_0 + M_0 \begin{bmatrix}
  u_0 \\
  v_0 \\
  1
\end{bmatrix} w'
$$

where $w'$ indicates the unknown depth of $V'$. It follows that:

$$
\begin{bmatrix}
  u_0 \\
  v_0 \\
  1
\end{bmatrix} w' = M_0^{-1} (V' - C_0)
$$

Examining equations (1) and (4) it follows that a possible value of $w'$ is the product $w_0 w_0$. Consequently:

$$
\begin{align*}
  w' &= w_0 w_k \\
  M_k^{-1} (V' - C_0) &= Q_k M_k^{-1} (V - C_k) \\
  V' &= C_0 + M_0 Q_k M_k^{-1} (V - C_k) \\
  V' &= (C_0 - M_0 Q_k M_k^{-1} C_k) + (M_0 Q_k M_k^{-1}) V \\
  V' &= T + RV
\end{align*}
$$

In conclusion the distorted vertices are computed by rotating and translating the undistorted vertex $V$, which can be done with one 4x4 matrix multiplication, supported by simple fixed pipeline graphics hardware. The clip planes of the frustum of $PPC$ are also supported by the fixed pipeline. When the number of clip planes of a frustum exceeds the number of clip planes supported, the frustum is split and replaced with frusta with fewer faces. We use OpenGL which allows for 6 clip planes.

6 Physical implementation of the graph camera

Creating a physical graph camera brings the benefits of comprehensive single-image visualization to real-world scenes. In order to implement the graph camera physically, several problems need to be overcome.

First, the graph camera rendering algorithm needs scene geometry. Providing high-resolution per-pixel depth for the video streams is a long-term challenging computer vision problem which does not have a robust real-time solution yet. Fortunately the rendering algorithm can be amended such that video streams can be combined into a graph camera without the need of per-pixel depth. Since the first step of the projection (equation 1 in Section 5) is implemented by the physical video camera, projection can begin with the second step (equation 2). In other words, given a video camera, the rendering algorithm projects the plane that connects the video camera to its parent video camera, bypassing the need of knowing the scene geometry.

The second problem is extrinsic calibration of the video cameras. The 3 rotations and 3 translations defining a video camera’s position and orientation are found by registration to a geometric scene model (i.e. proxy). For the scene in Figure 2 we used a model of the hallways. Although using 3 points on the virtual plane that connects two video cameras is sufficient to register the two cameras together, registering to the proxy provides flexibility for choosing the connection plane, after registration.

The third problem is finding a placement of the construction pinhole cameras that is physically realizable. For the maze graph camera in the construction cameras have a narrow field of view of a few degrees and are placed outside the maze. The narrow field of view is desirable in order to keep the perspective foreshortening along the maze corridors under control. For the physical graph camera in Figure 2 camera A was placed as far back as possible, against the opposite wall. The field of view is still significantly larger than in the case of the maze, resulting in “wasting” too many pixels for the near part of the central hallway. A possible solution to this problem would be to move the camera back in software, which requires scene geometry. The geometry proxy would be adequate for the corridor walls but estimating the depth for a person in the corridor is more challenging.

The fourth problem is implementing the clipping planes which

![Figure 19 Graph camera constructed interactively.](image1.png)

![Figure 20 The top image shows that the person is located in the left branch of the corridor yet the person also appears in the right branch (top). The correct near clipping plane of the video camera sampling the right branch of the corridor is implemented by background subtraction, replacing the incorrect pixels with background pixels (bottom).](image2.png)
delimit the frusta of the PPCs that model the segments of the graph camera. To minimize the interference of objects that are closer to a video camera than the desired near clipping plane the video camera should be placed as close as possible to the clipping plane. This implies a large field of view which contradicts the previous requirement for perspective. Instead we simulate the near clipping planes using background subtraction, as illustrated in Figure 20. Since the pixels erased with the background are not live and could hide an actual object, some applications could prefer that they are highlighted in red to mark the uncertainty. Non-leaf video cameras also need far clipping planes which are defined by the connection planes. Non-leaf video cameras also need far clipping planes which are defined by the connection planes. Background subtraction is used again to display a person in the central corridor over the leaf camera contributions.

7 Results and discussion

The graph camera images presented in the figures of this paper as well as in the supplemental and additional supplemental videos produce quality images of complex, dynamic 3-D scenes. The graph camera renders in feed forward fashion using graphics hardware, so there are no low-level quality issues. The clipping planes provide perfect continuity from frustum to frustum.

High-level quality is also good—the images show more than what is visible in a conventional rendering, yet the images remain readable. The graph camera is a good tool for summarizing 3-D scenes. We asked skilled computer graphics students to replicate the graph camera image from Figure 7. The image shown in Figure 21 was assembled in 8 ½ hours from 14 rendered shots, using 29 Photoshop layers, and the trees are conspicuously absent. The graph camera was constructed in less than 5 minutes. Moreover, the graph camera renders an animated image, whereas most of the segmentation work incorporated into the collage becomes obsolete once animation is turned on.

The cartoon town (Figure 1), city (Figure 10), and house (Figure 19) models comprise 0.97, 1.1, and 1.5 million triangles, respectively. The portal graph cameras used in Figure 3, Figure 10, and Figure 11 render with at 11/22, 8/22, and 10/22Hz minimum/maximum frame rate. The number of portals visible is the dominant performance factor. The occluder graph camera (Figure 4) renders at 15Hz. The D1 graph camera image shown in Figure 1 renders at 1.3Hz. The graph cameras in Figure 5 and Figure 6 render at 1.9, and 2.6Hz, and have 48, and 35 frusta. The interactively constructed graph camera (Figure 7) renders at 12Hz. Timing info was collected on a 4 Core Xeon X5460 @ 3.16 GHz, 4 GB Ram, nVidia 280 GTX workstation. The 2 real world graph cameras shown in the video render at 9 and 9.5Hz, respectively, matching the performance of our 6 wireless surveillance video camera system. Background subtraction uses CUDA 1.2.

8 Conclusions and future work

The graph camera model is essentially a set of instructions for combining PPC construction images. The camera model is dynamic; entering through portals, tracking occluders, approaching intersections, emphasizing one region or another all modify the model qualitatively, from frame to frame.

The graph camera image is not a map, global spatial relationships are not respected. Quite to the contrary—the graph camera is a powerful tool that allows escaping the rules of line of sight and proportional distance. The graph camera juxtaposes regions of a scene that cannot be imaged simultaneously by any pinhole, facilitating the detection of distant relationships.

Camera models are important infrastructure. In computer graphics images are not only used as output but also as intermediate representations for acceleration purposes. For example, the graph camera could prove useful in tackling interactive rendering of global illumination effects (e.g. reflection, ambient occlusion), providing comprehensive yet efficient sampling of complex scenes. In scientific and information visualization the visual presentation of complex spatial and non-spatial data remains an important unsolved problem which the flexibility and efficiency of the graph camera can help tackle. As the ubiquity of digital cameras increases, image processing and computer vision applications will have the opportunity and the task to process large collections of images, which a graph camera could distill to one or a few comprehensive images. Note that the entire computational apparatus developed for pinholes can be easily translated to graph cameras which are piecewise pinholes. The potential gains are efficiency—no redundancy, and robustness—fewer discontinuities, global consistency. Humans are not accustomed to non-pinhole images because these rarely occur in nature. Highly specular & smooth reflectors do produce rays that are not concurrent, but the size of the reflector is usually small compared to the viewing distance so the rays are close to concurrent. Therefore work is also needed in human vision and perceptual psychology to learn how to make non-pinhole images that have the highest communication bandwidth possible.

References


We derive the mapping \( Q_{k+1} \) of a point on the image plane of planar pinhole camera \( k+1 \) (PPC\(_{k+1}\)) to PPC\(_k\) by first establishing the mapping \( R_{k+1} \) between PPC\(_{k+1}\) and PPC\(_k\) as shown in Figure 22. Then we show by induction that \( Q_{k+1} = R_k R_{k+2} ... R_{k+1} \). The base case is verified as follows:

\[
\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = R_k \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} u_2 \\ w_1 \end{bmatrix} = \begin{bmatrix} u_3 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_4 \\ w_2 \end{bmatrix} = \begin{bmatrix} u_5 \\ v_3 \end{bmatrix}
\]

(A.1)

By the induction hypothesis:

\[
Q_{k} = R_k R_{k+2} ... R_{k+1}
\]

(A.2)

Using the equations in Figure 22 we obtain:

\[
\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = R_k R_{k+1} \begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} u_{k+2} \\ w_{k+1} \end{bmatrix} = \begin{bmatrix} u_{k+3} \\ v_{k+2} \end{bmatrix} = \begin{bmatrix} u_{k+4} \\ w_{k+2} \end{bmatrix} = \begin{bmatrix} u_{k+5} \\ v_{k+3} \end{bmatrix}
\]

(A.3)

Combining equations A.2 and A.3 terminates the proof:

\[
\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = R_k R_{k+1} ... R_{k+1} \begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} u_{k+2} \\ w_{k+1} \end{bmatrix} = \begin{bmatrix} u_{k+3} \\ v_{k+2} \end{bmatrix} = \begin{bmatrix} u_{k+4} \\ w_{k+2} \end{bmatrix} = \begin{bmatrix} u_{k+5} \\ v_{k+3} \end{bmatrix}
\]

(A.4)

\[
\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = R_k R_{k+1} ... R_{k+1} \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_{k+2} \\ y_{k+2} \end{bmatrix} = \begin{bmatrix} x_{k+3} \\ y_{k+3} \end{bmatrix} = \begin{bmatrix} x_{k+4} \\ y_{k+4} \end{bmatrix} = \begin{bmatrix} x_{k+5} \\ y_{k+5} \end{bmatrix}
\]

The diagram shows the derivation of mapping PPC\(_{k+1}\) to PPC\(_k\) with COP's \( C_{o1} \) and \( C_{o2} \). Vectors \( x \) and \( y \) give the row and column direction and are one pixel width and one pixel length long, respectively. Vectors \( \alpha \) point from the COP to the top left corner of the image. Point \( P_{o1} \) on the image plane of PPC \( k+1 \) is mapped to point \( P_{\alpha} \) on the image plane of PPC \( k \) through matrix \( R_{\alpha+1} \).