Computing Visibility for Triangulated Panoramas

Chi-Wing Fu[†] cwfu@cse.cuhk.edu.hk
Tien-Tsin Wong[‡] ttwong@acm.org
Pheng-Ann Heng[†] pheng@cse.cuhk.edu.hk

[†]The Chinese University of Hong Kong
[‡]Hong Kong University of Science and Technology

Abstract. A visibility algorithm for triangulated panoramas is proposed. The algorithm can correctly resolve the visibility without making use of any depth information. It is especially useful when depth information is not available, such as in the case of real-world photographs. Based on the optical flow information and the image intensity, the panorama is subdivided into variable-sized triangles, image warping is then efficiently applied on these triangles using existing graphics hardware. The visibility problem is resolved by drawing the warped triangles in a specific order. This drawing order is derived from epipolar geometry. Using this partial drawing order, a graph can be built and topological sorting is applied on the graph to obtain the complete drawing order of all triangles. We will show that the time complexity of graph construction and topological sorting are both linear to the total number of triangles.

1 Introduction

In this paper, we focus on solving the visibility problem in warping a given reference panorama to generate the desired panorama from a new viewpoint. Given an image together with its depth information, image as viewed from another viewpoint can be synthesized by reprojecting each pixel [2, 10]. Since multiple pixels may be mapped to the same location in the new image, visibility has to be resolved. The most straightforward method is depth-buffering. However, in some cases, the depth information may not be available or not accurate. This is especially common for real-world photographs. In that case, only the correspondences or optical flow information can be determined.

McMillan [13, 12] proposed a clever solution to the visibility problem. Once the mapping of pixels from the reference image to the desired image is known (either by pixel reprojection [2, 10] or by finding the point correspondences [6] or optical flow [14, 5]), the image can be warped correctly. *No* depth-buffering is needed. The visibility is solved by mapping pixels in a specific order. Due to the nature of the drawing order, only small image entities, such as pixels, can be applied. However, warping images in a pixel-by-pixel manner (*pixel-based warping*) is time-consuming and cannot utilize existing graphics hardware. Moreover, gaps occur between adjacent pixels after they are warped. If the image is subdivided into larger image entities (triangles) and the mapping is then applied on them instead of pixels, we can make use of graphics hardware to accelerate the image warping. The gap problem can also be solved at the same time. We call this triangle-by-triangle image warping the *triangle-based warping*. Unfortunately, McMillan's drawing order cannot be applied to larger entities. We introduce a visibility sorting algorithm to find out the ordering of triangles for image warping. We will also show that the time complexity of the algorithm is linear to the number of triangles.

In this paper, we propose a triangle-based visibility algorithm for image types which satisfy the *mapping criterion* described in Section 5. Planar perspective images satisfy this mapping criterion. Hence cube-based panoramas formed by six planar perspective images can be warped correctly using the proposed algorithm. Although cylindrical panoramas do not satisfy this criterion, approximated results can be obtained by applying the algorithm. The approximated results do not exhibit any noticeable artifact.

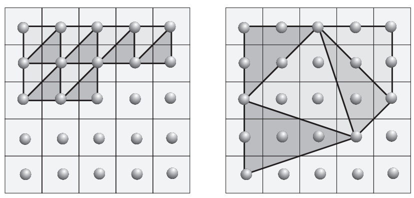
In Section 2, some related work are discussed and compared. We then have a brief overview of epipolar geometry in Section 3. In Section 4, details of the image triangulation are described. Based on the epipolar geometry, the drawing order of all triangles is derived in Section 5. Section 6 shows the results of our implementation. Finally, some conclusions and future directions are drawn in Section 7.

2 Related Work

Chen and Williams [2] warped images by reprojecting each pixel onto the new image. Depth-buffering is used to solve the visibility. Darsa *et al.* [3] subdivided the depth image into variable-sized triangles and performed reprojection on each of them. Again in their work, the visibility is solved by depth-buffering. Seitz and Dyer [16] introduced the view morphing which can correctly interpolate two different views based on image morphing [1]. Additional information such as the position of the camera and the correspondences of some feature points are required.

McMillan [11, 13, 12] first proposed a drawing order to solve the visibility without using depth-buffering. The problem is solved by drawing pixels in a specific order. The drawing order is derived from epipolar geometry. Mark and Bishop [7] studied the memory access pattern of McMillan's pixel drawing ordering. The difference between McMillan's and ours is that his drawing ordering is *only* valid for pixel-sized image entities. In our work, there is no restriction on the size of image entities. In this sense, our work can be regarded as an extension of McMillan's pixel drawing order.

If pixels are forwardly mapped to the new image, gaps will appear in between those pixels. Laveau and Faugeras [6] used a backward mapping, which maps pixels from the desired image back to the reference image, in order to prevent the appearance of gap. It is similar to the backward texture mapping. Mark *et al.* [9] solved the gap problem by two methods, namely splatting and modeling the image as a triangular mesh. To prevent gap using splatting, the footprint of pixel must be large enough. However large footprint may excessively blur the image. They also suggested to model the image as a triangular mesh to prevent the gap. Since McMillan's drawing order can only be applied to pixel-sized entities, they have to subdivide the image into pixel-sized triangles by connecting neighboring three pixel samples as in Figure 1(a). Hence a 512×512 image may be subdivided into more than five hundred thousand triangles. Even with the assistance of graphics hardware, the warping is still slow. Note that their triangular mesh approach is different from the one proposed in this paper. Their triangles are still in pixel size while our triangles can be in arbitrary size and shape. Figure 1 shows the differences. Shade *et al.* [17] further extended the usage of McMillan's drawing order to image with multiple layers of depth values.



(a) (b)

Fig. 1. Comparison of the (a) pixel-sized and (b) arbitrary-sized triangulation.

3 Epipolar Geometry

Given a viewpoint (or center of projection), image synthesis is accomplished by firing bundle of rays from the viewpoint to the surrounding and sampling radiances received along the rays. If a rectangular plane is placed in front of the viewpoint, a planar perspective image can be formed by projecting the radiance values onto this plane. The rectangular plane is one kind of *projection manifold*, more specifically, a planar projection manifold. Similarly, other geometry can be used as the projection manifold, such as cylinder or sphere. Since the viewpoint is a point in space and the rays are fired from the viewpoint, a sphere is the most natural and general form of projection manifold which can record the radiance received along any ray. Radiances recorded by any other projection manifold can always be reprojected onto the spherical projection manifold.

From now on, we focus on the discussion of spherical projection manifold due to its generality. Consider a spherical image I_c captured with the center of projection at $\dot{c}$. We use the dot notation $\dot{a}$ to denote a 3D position and the arrow notation $\vec{a}$ to denote a 3D directional vector. A desired spherical image I_e is generated with a new center of projection at $\dot{e}$. Figure 2 shows the geometry in both 3D and 2D.

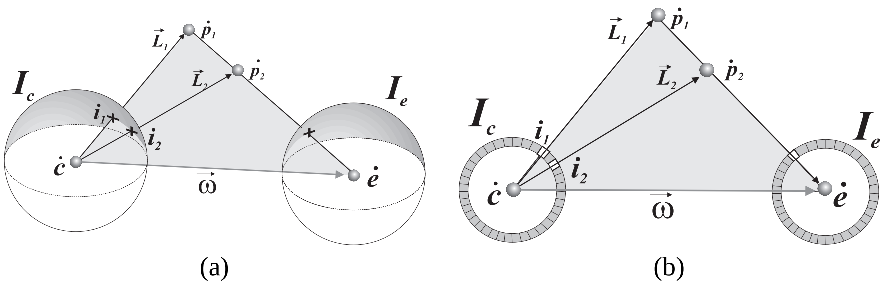


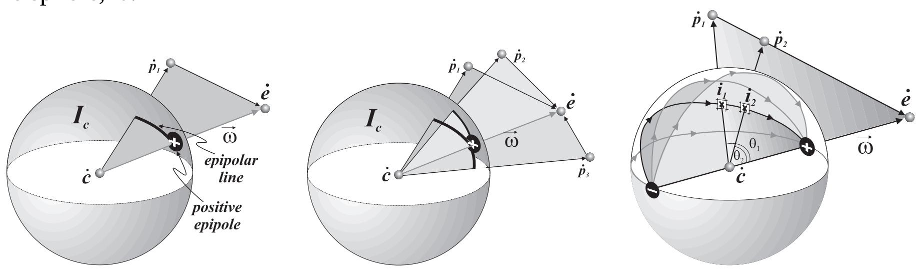
Fig. 2. The geometry of two cameras (a) in 3D and (b) in 2D.

Each pixel i in the image I_c stores the radiance along the ray $\vec{L}$ which is fired from $\dot{c}$ passing through the pixel window associated with i. Now, let's choose an arbitrary pixel i_1 from image I_c. A ray $\vec{L_1}$ is associated with it. The intersection point $\dot{p}_1$ associated with i_1 must lie somewhere on the ray $\vec{L_1}$. To generate a new view from $\dot{e}$, $\dot{p}_1$ has to be reprojected onto I_e. The plane constructed by $\dot{c}$, $\dot{e}$ and $\dot{p}_1$ is known as *epipolar plane* in computer vision literature. The vector, $\vec{\omega}$, originated from $\dot{c}$ pointing towards $\dot{e}$ is called *positive epipolar ray* while the vector, $-\vec{\omega}$, originated from $\dot{c}$ pointing to the opposite direction is called *negative epipolar ray*.

Now let's choose another pixel i_2 from image I_c. Occlusion happens only when $\dot{p}_1$ and $\dot{p}_2$ are reprojected onto the same location in I_e (see Figure 2). If $\dot{p}_2$ does not lie on the epipolar plane associated with $\dot{p}_1$, $\dot{p}_1$ and $\dot{p}_2$ will never occlude each other. Hence occlusion happens only when $\dot{c}$, $\dot{e}$, $\dot{p}_1$ and $\dot{p}_2$ all lie on the same plane. Moreover the necessary condition of $\dot{p}_2$ occluding $\dot{p}_1$ is $\dot{e}$, $\dot{p}_1$ and $\dot{p}_2$ are collinear and $\dot{p}_2$ is in between $\dot{p}_1$ and $\dot{e}$, as illustrated in Figure 2.

From Figure 2, we know that $\dot{p}_2$ will never be occluded by $\dot{p}_1$ as viewed from $\dot{e}$ no matter where the exact positions of $\dot{p}_1$ and $\dot{p}_2$ are. Therefore, if we always draw $\dot{p}_1$ before $\dot{p}_2$ during reprojection, the visibility problem is solved without comparing their depth values. And hence, if we can identify those pixels whose intersection points may occlude each other and derive the drawing order, the visibility problem can be solved without depth-buffering.

To identify the pixels which may occlude each other, we first intersect the epipolar plane with the spherical projection manifold (image I_c). The intersection curve on the sphere is called the *epipolar line*. Figure 3(a) illustrates the terminologies graphically. When the positive epipolar ray $\vec{\omega}$ intersects with the projection manifold I_c, the intersection point on the projection manifold is known as *positive epipole*. Figure 3 denotes it by a positive sign. On the other hand, the intersection point of the negative epipolar ray and the sphere is known as *negative epipole* and denoted by a negative sign. Note all epipolar lines terminated at two epipoles. Moreover, each epipolar line must be a half of the great circle (Figure 3(c)) as the epipolar plane passes through the center of the sphere, $\dot{c}$.



(a) (b) (c)

Fig. 3. (a) & (b): The epipolar line is the intersection of the projection manifold and the epipolar plane. (c): The drawing order between two pixels that lie on the same epipolar line.

All pixels in I_c that lie on the same epipolar line have a chance to occlude each other. Figure 3(c) shows two pixels, i_1 and i_2, lying on the same epipolar line. As their associated intersection points $\dot{p}_1$ and $\dot{p}_2$ are on the same plane with $\dot{c}$ and $\dot{e}$, they may occlude each other. Moreover, $\dot{p}_1$ will never occlude $\dot{p}_2$ as $\dot{p}_1$'s angle of derivation θ_1 is greater than, θ_2 of $\dot{p}_2$ (see Figure 3(c)). In other words, if i_2 is closer to the positive epipole on the epipolar line than i_1, i_1 will never occlude i_2. Hence we should always draw i_1 first. Arrows on the epipolar line in Figure 3(c) indicate the drawing order of pixels.

Based on the pattern of epipolar lines on the sphere, McMillan [11] derived a drawing order for pixel-sized image entities. Since the epipolar line can only tell the ordering of small image entities that lie on it, the drawing order only works for pixel-sized image entities. The drawing order cannot be easily extended to larger entities such as triangles which occupy a bundle of epipolar lines.

4 Image Triangulation

4.1 Optical Flow

Before the image is warped, the 2D image has to be first triangulated. To do so, we make use of the optical flow. Consider two images $I_c(x,y)$ and $I_e(x,y)$ of the same scene but with different viewpoints $\dot{c}$ and $\dot{e}$, one can determine the image correspondences between two images using existing computer vision techniques. These correspondences allow us to calculate the two dimensional *optical flow vector* [14, 5] for each pixel, *i.e.* the 2D pixel movement from I_c to I_e. The optical flow of a pixel i is defined as

$$\vec{f} = \begin{pmatrix} x_e - x_c \\ y_e - y_c \end{pmatrix}, \tag{1}$$

where (x_c, y_c) is the image coordinate of i in I_c,
(x_e, y_e) is the image coordinate of i in I_e.

If the optical flow vectors of two neighboring pixels are close, it is very likely that the intersection points associated with these pixels are on the same object. On the other hand, if the optical flow vectors are not close, the intersection points are more likely to be on different objects with different distances from the viewpoint. During image warping, they are very likely to be moved away from each other. Hence, one criterion of triangulating the image is to separate those pixels with large difference in optical flows. Otherwise, serious distortion will result during image warping. Hence we want to locate the regions with large variation in optical flow between neighboring pixels. To do so, we calculate a function F for each pixel,

$$F(x,y) = \max\left(\left|\frac{\partial^2 \vec{f}}{\partial x^2}\right|, \left|\frac{\partial^2 \vec{f}}{\partial y^2}\right|\right). \tag{2}$$

Function F is defined as the maximum of the magnitudes of two second order partial derivatives of the optical flow vectors along dimensions x and y.

4.2 Image Gradient

Besides the optical flow, image intensity is also used during triangulation. Regions with edges (sharp intensity change) are perceptually more noticeable to humans. A distortion in the edge region is more noticeable than the same distortion in the non-edge region. In other words, the tolerance of distortion in the region with different image content should be different. To quantify this factor, we first convert the color image to grayscale by transforming RGB to Y_tIQ color space. The Y_t is then used as the grayscale version of the image. The standard grayscale conversion [4] is

$$I'_c = 0.299 I^R_c + 0.587 I^G_c + 0.114 I^B_c, \tag{3}$$

where I^R_c, I^G_c, and I^B_c are the R, G and B values from image I_c.

To locate edge region, we simply apply the standard Laplacian operator, ∇^2, to the grayscale image I'_c and obtain function G,

$$G(x,y) = \nabla^2 I'_c(x,y) = \frac{\partial^2 I'_c}{\partial x^2} + \frac{\partial^2 I'_c}{\partial y^2}. \tag{4}$$

Both optical flow and image gradient are important criteria for triangulating the image. They are combined in the following potential function $P(x,y)$, which tells us where should we place the vertices of triangles.

$$P(x,y) = \alpha F^*(x,y) + (1-\alpha) G^*(x,y), \tag{5}$$

where $\alpha \in [0,1]$ is a weight, $\alpha = 0.7$ is a good choice,
F^* and G^* are the normalized F and G, they are normalized to range [0,1].

The larger the potential value of the pixel, the larger the potential that the pixel is on the silhouette of an object. Figures 10(a) and (b) show the reference panorama and the corresponding potential map calculated by Equation 5.

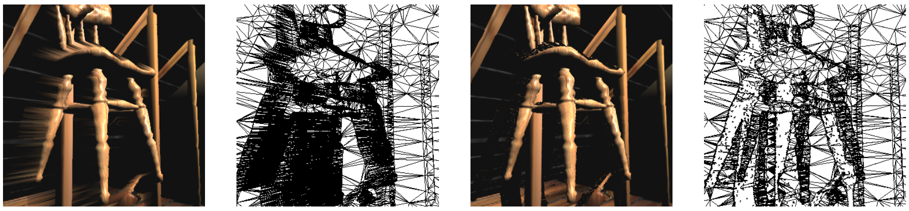
4.3 Triangulation

To triangulate the image, we stochastically distribute the vertices of triangles onto the potential map. The pixel with larger potential value has a larger chance of receiving a vertex. Then Delaunay triangulation [15] is applied to obtain the initial triangular mesh.

The initial triangular mesh is further refined to reduce the visual artifact during the actual image warping. A triangle is split into two smaller triangles if its *sum of potential* exceeds a pre-defined threshold τ_p. The sum of potential, ρ, of a triangle t is defined as the total sum of potential value of pixels within this triangle t.

$$\rho(t) = \sum_i P(x_{p_i}, y_{p_i}) \qquad \forall p_i \in t, \tag{6}$$

where p_i is a pixel inside the triangle t. The subdivision continues until all triangles satisfy the sum of potential requirement. Figure 10(c) shows the final triangular mesh.



(a) (b) (c) (d)

Fig. 4. Excessive stretching near the object silhouette during the image warping due to the triangle connectivity.

Note that all triangles are now connected. Therefore no gap can be found during image warping. However, another artifact appears if all triangles are connected together. Figures 4(a) and (b) show that triangles near the silhouette of the object are excessively stretched. During image warping, some of the occluded regions become visible. Since no information is available, these areas should be left blank instead of filling them with the excessively stretched triangles. This kind of artifact usually appears near the silhouette of an object. Hence it can be reduced by disconnecting the triangles at object silhouette. Again, the disconnection can be done with the guidance of function F as the map indicates object silhouette. If the common edge shared by two neighboring triangles is located at a region with average potential value above a user-defined threshold τ_c, these neighboring triangles shall be disconnected. Figure 4(c) and (d) show the result. Mark and Bishop [8] proposed an efficient reconstruction technique for filling this kind of holes. The process of triangulation can be done off-line. Once it has been done, the triangular mesh will not be changed during the actual image warping.

5 Image-based Visibility Sorting

5.1 Ordering of Two Triangles

Epipolar geometry provides sufficient information for us to resolve the visibility problem when warping triangles. Our algorithm resolves the visibility problem of triangles on the surface of the sphere. Hence, we need to project a triangle in the image to a spherical triangle on the surface of the sphere which encloses viewpoint $\dot{c}$.

Each 2D image must be related to an implicit projection manifold. For examples, a planar perspective image is related to a planar projection manifold. A cylindrical panoramic image is related to a cylindrical projection manifold. As the image triangulation algorithm discussed in the previous section triangulates the image in 2D, the validity of the proposed algorithm relies on one criterion of mapping (from 2D to sphere): *any straight line on the 2D surface must be mapped to one geodesic curve on the sphere and vice versa* (Figure 5). A geodesic curve connecting two points on the sphere is the shortest path on sphere from one point to another. It must also be a segment of the great circle of sphere. Two dimensional images resulted from planar projection manifold satisfy the above mapping criterion. Hence the proposed algorithm is applicable to cube-based panorama. On the other hand, a straight line on the 2D unfolded cylindrical or spherical panoramic image may not be mapped to a geodesic curve on the sphere.

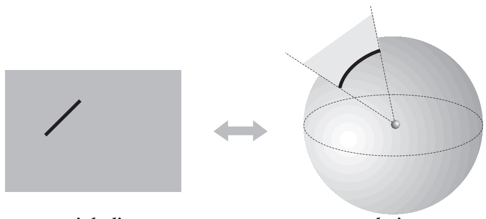


Fig. 5. Mapping criterion: a straight line in 2D can be mapped to a geodesic curve on the sphere.

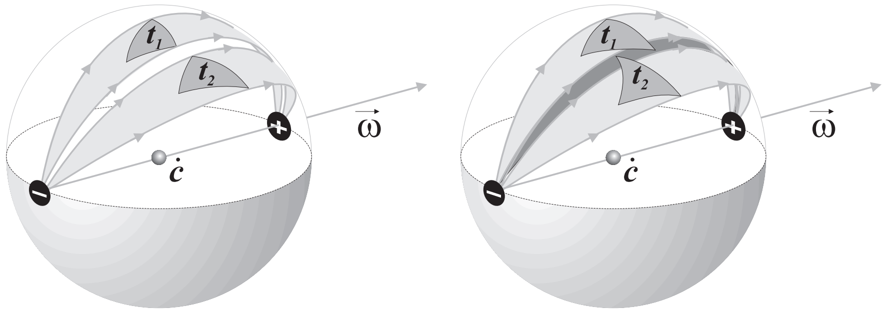
If an image can be projected onto the sphere and the mapping satisfies the above criterion, a triangle in the image can always be mapped to a spherical triangle on the sphere. Each edge of the spherical triangle must be a geodesic curve. Consider two arbitrary spherical triangles, t_1 and t_2, obtained by triangulating the 2D image and being mapped onto the sphere. We can determine whether they may occlude each other, by checking the bundles of epipolar lines they occupy. Let's call the bundle of epipolar lines occupied by a triangle the *epipolar band*. Figure 6(a) shows the epipolar bands occupied by two spherical triangles in light gray. If their epipolar bands do not overlap (Figure 6(a)), no occlusion will occur between them. There is no element (pixel) in these two spherical triangles sharing any common epipolar line. Hence the order of drawing these two spherical triangles is irrelevant. On the other hand, if the two epipolar bands overlap (Figure 6(b)), some elements from these spherical triangles lie on the same epipolar line. Hence occlusion may happen after image warping. Therefore, the ordering of these triangle does matter. In the specific example of Figure 6(b), t_1 may occlude t_2 as t_1 is closer to the positive epipole than t_2 in the overlapping region.
We now define two ordering relations of spherical triangles:

Definition 1 *If all elements in a spherical triangle t_1 must be drawn before any element in another spherical triangle t_2 in order to preserve the correct visibility, we say t_1 must be drawn before t_2 and denote this ordering relation as $t_1 \to t_2$.*

Definition 2 *If the drawing order between the elements in spherical triangle t_1 and the elements in another spherical triangle t_2 is irrelevant, the drawing order between these two spherical triangles are also irrelevant. We denote this ordering relation as $t_1 \leftrightarrow t_2$.*

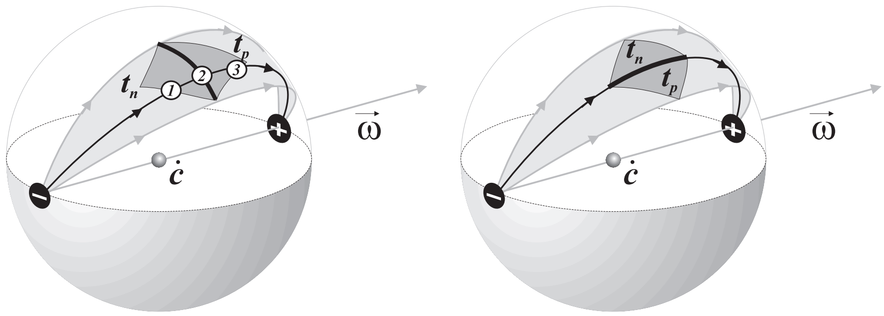
Since all the spherical triangles are the result of triangulating an image as viewed from the viewpoint $\dot{c}$, they all must be visible, non-overlapping and connected as viewed from $\dot{c}$. Instead of considering the ordering of any two arbitrary spherical triangles, we first consider the ordering between each pair of neighboring spherical triangles which share a common edge as in Figure 7. Now we will show that the ordering of any two neighboring spherical triangles can be determined by the position and orientation of spherical triangles.

Theorem 1 *Given two neighboring spherical triangles which share a common edge, a plane that passes through the center of projection $\dot{c}$ and the shared edge (which is a geodesic curve) can be constructed (see Figure 8). It divides the sphere into two*



(a) (b)

Fig. 6. Epipolar bands on the sphere.



(a) (b)

Fig. 7. a) The epipolar line starting from the negative epipole must always enter t_n before t_p if it cuts both t_n and t_p. b) If the epipoles and the shared edge lie on the same plane, the ordering of t_n and t_p is irrelevant.

equal halves and separates two spherical triangles. The spherical triangle with the positive epipole on its side should be drawn later during warping. On the other hand, the triangle with the negative epipole on its side should be drawn first during warping. If the epipoles (positive and negative) lie exactly on the constructed plane, the ordering of these two triangles is irrelevant.

Proof: Due to the mapping criterion, all edges of spherical triangle must be geodesic curves. These geodesic curves lie in the planes that pass through the center of the sphere. Therefore one can always separate two neighboring spherical triangles by constructing a plane that contains the shared common edge and the sphere center. This is also why we need the mapping criterion. Let's denote the spherical triangle with the positive epipole on its side as t_p and the other as t_n as shown in Figure 7(a). Now let's draw a geodesic curve from the negative epipole to the positive epipole. Since the negative epipole is on the same side as t_n, whenever this geodesic curve passes through both t_n and t_p, it should first pass through t_n, then the shared edge and finally t_p (Figure 7(a)). Therefore whenever there are elements in t_n which are sharing a common epipolar line (geodesic curve) with some elements in t_p, elements in t_n should be closer to the negative epipole than those elements in t_p. Hence, no element in triangle t_n will occlude any element in t_p and we must draw t_n before t_p during warping, *i.e.* $t_n \rightarrow t_p$.

When both epipoles and the shared edge lie on the same plane, epipolar bands of t_n and t_p have no intersection (Figure 7(b)). One can always separate the epipolar bands of these two spherical triangles by a plane that contains the shared edge, the center of sphere, and the two epipoles. In other words, no element in t_n and t_p shares any common epipolar line. Therefore, their ordering is irrelevant and we say $t_n \leftrightarrow t_p$.

Hence, the drawing order of two spherical triangles t_1 and t_2 can be determined by checking on which side of the constructed plane the negative epipole resides. This can be done with some simple vector mathematics. Figure 8 illustrates the mathematical symbols used in the following equations. Vectors $\vec{e_a}$ and $\vec{e_b}$ are the vectors from the center of projection, $\dot{c}$, to the two endpoints of the shared common edge. Vectors $\vec{v_1}$ and $\vec{v_2}$ are the vectors from $\dot{c}$ to the unshared vertices of spherical triangles t_1 and t_2 respectively.

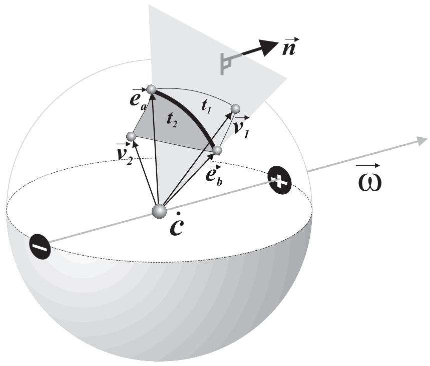


Fig. 8. Mathematical symbols used for the calculation of drawing order.

$$\vec{n} = \vec{e_a} \times \vec{e_b} \qquad \beta = \vec{n} \cdot \vec{\omega} \qquad \gamma = \vec{n} \cdot \vec{v_1}$$

Based on the values of β and γ, the drawing order is determined as follows:

1. If $\beta = 0$, $\vec{\omega}$ is orthogonal to $\vec{n}$. In other words, the epipoles and the shared edge must be on the same plane. Hence, the drawing order of t_1 and t_2 is irrelevant, *i.e.* $t_1 \leftrightarrow t_2$.
2. If β and γ have the same sign (either both are positive or negative), t_2 is on the same side as the negative epipole. Hence $t_2 \rightarrow t_1$.
3. If β and γ have the different signs, t_1 is on the same side as the negative epipole. Hence $t_1 \rightarrow t_2$.

5.2 Graph Construction

Using the method described, one can always derive the drawing order of two neighboring triangles. This ordering can be further extended to cover any two arbitrary triangles from mesh by constructing a drawing order graph. By representing each triangle as a node and the relation $\rightarrow$ as a directed edge in the graph, we can construct a graph of drawing order. No edge is needed to represent the relation $\leftrightarrow$ as the ordering is irrelevant. Note the constructed graph may contain disjointed subgraphs. Figure 9(a) shows seven connected triangles. The drawing order of each pair of neighboring triangles are shown as arrows crossing the shared edges between neighboring triangles. The constructed graph is shown in Figure 9(b). Figure 9(c) shows two valid drawing orders derived from the example graph. Note there is no unique ordering for the same graph.

In the actual implementation, there is no need to construct the graph explicitly. The graph can be implicitly represented as a set of ordering relation between each pair of neighboring triangles. Hence, for each shared edge, we determine the drawing order between neighboring triangles using Theorem 1. The time complexity of graph construction is obviously $O(E)$ where E is the number of shared edges. As each triangle

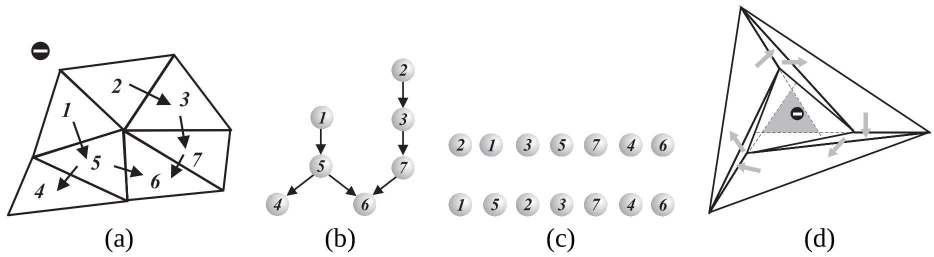


Fig. 9. (a), (b) & (c):Construction of drawing order graph. (d): Cycle may exist in the graph.

has three edges, E is at most $3N$ where N is total number of triangles. Hence, the time complexity should be linear to the total number of triangles.

5.3 Topological Sorting

The final step to find out the ordering of all triangles is to perform a topological sort on the drawing order graph. Basically it is a two-pass algorithm. Before describing the algorithm in detail, let's define the terminology. A triangle (node) is called degree zero if there is no triangle needed to be drawn before it, *i.e.* it is not on the right hand side of any $\rightarrow$ relation. It is called degree 1 if one of its three neighbors has to be drawn before it. A triangle is at most of degree three.

In the first pass of algorithm, it looks for all zero-degree triangles and put them into a pool. In the second pass, one triangle t_1 in the pool is picked, output and removed from the pool. Then for each neighboring triangle t_2 of t_1, such that $t_1 \rightarrow t_2$, decrease the degree of t_2 by one. If the degree of t_2 drops to zero after deduction, put it into the pool. The process of picking and drawing continues until no more triangle is left. The algorithm is shown on the right. The example graph in Figure 9(b) is sorted using this algorithm.

```
// First pass
Pool0 = ∅
For each triangle t of degree zero
   Pool0 = Pool0 ∪ {t}
For each triangle t of degree one
   Pool1 = Pool1 ∪ {t}

// Second pass
While there exists triangle not output
   If Pool0 ≠ ∅
      Pick a triangle t1 from Pool0 and output
   else there is cycle
      Randomly pick a triangle t1 from Pool1
      and output it
   For each neighbor triangle of t2 s.t.  t1 → t2
      Decrease the degree of t2 by one.
      If the degree of t2 is zero
         Pool0 = Pool0 ∪ {t2}
      If the degree of t2 is one
         Pool1 = Pool1 ∪ {t2}
```

It seems that the graph will be a directed acyclic graph. However cycles do exist in extremely rare cases. Figure 9(d) shows one special example of triangulation such that cycle exists. If the projected epipole (projected onto reference image) locates inside the gray region, cycle will occur. In practice, cycles seldom occur and no cycle was found in all our experiments. However, the above algorithm does handle the case when cycle is found. It randomly picks a triangle of degree one, draws it on the screen and hence breaks the cycle. A pool of degree-one triangle is setup for this purpose. This approach may result in visual artifact.

The time complexity of the first pass of algorithm is $O(N)$. Since each triangle will be put into the pool and picked out from the pool at most once, the time complexity of the second pass is also linear. Hence the topological sorting is linear. The time complexity of the whole visibility sorting algorithm is also linear to the number of triangles.

6 Results

In our implementation, we use a cylindrical panorama to record the environment. Although we have mentioned before that cylindrical panorama does not satisfy the mapping criterion, approximated results without noticeable artifacts can be obtained. To apply the proposed algorithm to cylindrical panorama, the edges of triangles should not be too long, especially horizontal edges near the top and the bottom of the unfolded panorama.

Determining accurate optical flow is a well-known hard problem. Since our work mainly concentrates on solving visibility, we obtain the accurate optical flow map by reprojecting pixels using depth values. But no depth value is used in the following visibility determination.

As the image is being warped, originally occluded regions become visible and these areas are left blank as there is no information. To minimize the unfilled region, multiple reference panoramas are warped to the same position and then blended together. Figures 13(a) and (c) show the warped results of two reference panoramas. They are warped to the same position. The green regions highlight the blank areas. Figure 13(e) shows the result of blending Figures 13(a) and (c). Figures 13(b) and (d) show the corresponding warped triangulation together with the drawing order. To distinguish one triangle from another, three distinct colors are used to color the neighboring triangles. The intensity of the triangle indicates the drawing order. The darker the color, the earlier is the triangle in the drawing order.

Table 1 shows the timing for visibility sorting and rendering. The desired image is a planar perspective image while the reference image is a cylindrical panorama. All the timings are recorded on an SGI Octane with CPU MIPS R10000/250MHz and MXE graphics accelerator. As expected, major computation is spent on visibility sorting (graph construction + topological sorting). Rendering only occupies a minor portion of the time as it is assisted with graphics accelerator. Nevertheless, the overall image warping can still be done at interactive speed. Note that the visibility sorting is not necessary if the user does not change the walking direction $\vec{\omega}$.

Figure 11 shows the result of warping and blending panoramas of an attic scene. Note how the pillars and the chair correctly occlude the background objects. The correctness of resultant visibility demonstrates the validity of the proposed algorithm. The times below the images indicate how far the image has been warped. When the time equals to zero, no warping is done. The image is simply the first reference panorama. When the time equals to 0.5, both the first and the second reference panoramas are warped to the middle position. When the time equals to 1.0, the image is simply the second reference panorama. Another set of warping panoramas of a city scene is shown in Figure 12. Note how forwardly moving buildings occlude neighboring buildings.

Data set	Reference pano. image resolution	Desired pers. image resolution	Number of triangles	Build graph (sec.)	Topology sort (sec.)	Average Rendering time (sec.)
attic	1024×256	512×512	54611	0.1190	0.0879	0.0577
city	1024×256	512×512	53550	0.1160	0.0853	0.0557

Table 1. Timing of triangle-based image warping.

7 Conclusions and Future Directions

In this paper, we propose a triangle-based visibility algorithm without using depth information. Grouping pixels to form triangles allows image warping to be done in an efficient manner. Hardware graphics board can further accelerate the rendering of warped image. Moreover, the gap problem due to pixel-based image warping is also removed simultaneously. Both graph construction and topological sorting in the visibility algorithm have a linear time complexity and only need to be performed whenever the user changes $\vec{\omega}$.

We have only described how image warping can be done correctly for triangles. The merging of two warped images usually exhibits visual discontinuity due to the surface properties of the objects and illumination configuration during image capturing. Therefore, how to merge two warped images seamlessly is another important issue. In some cases (especially complex scene), holes (unfilled pixels) will still exist even multiple images are warped and merged. The sampling scheme (placement of the panorama nodes) requires further investigation.

Acknowledgements

This work is supported by Hong Kong Government RGC CRCs Scheme Grant No. CRC4/98. We would like to thank all anonymous paper reviewers for their valuable comments and pointing out the error.

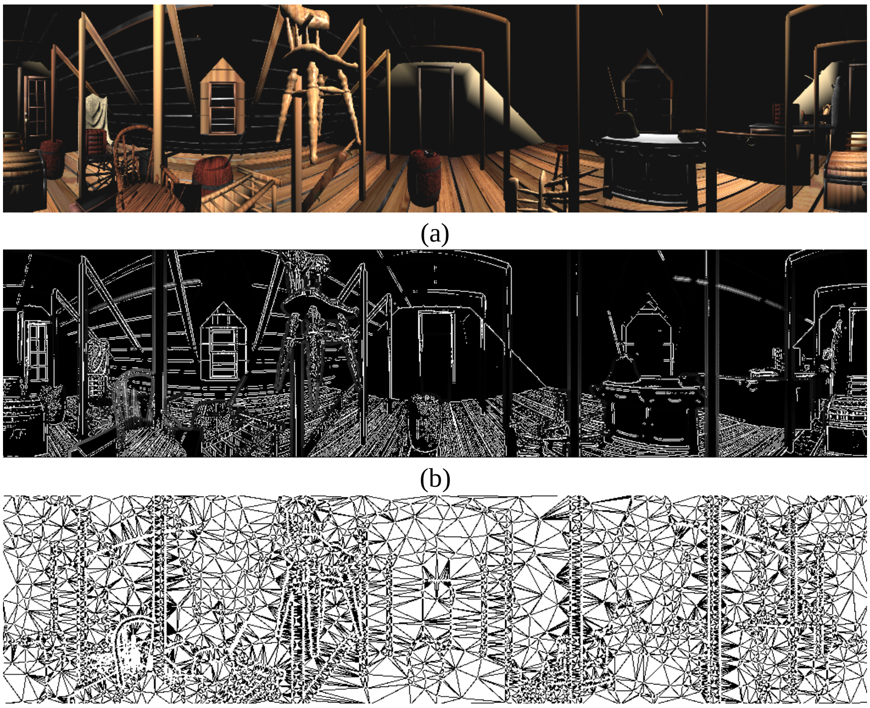
Web Availability

The warped images and movies of tested scenes can be found at the following web pages: `http://www.cs.ust.hk/~ttwong/papers/panowalk/panowalk.html` and `http://www.cse.cuhk.edu.hk/~cwfu/papers/panowalk/panowalk.html`

References

1. T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 35–42, July 1992.
2. S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 279–288, 1993.
3. L. Darsa, B. C. Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 25–34, April 1997.
4. R. Hall. *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, 1988.
5. B. Horn. *Robot Vision*. MIT Press, 1986.
6. S. Laveau and O. Faugeras. 3-D scene representation as a collection of images. In *Proceedings of the Twelfth International Conference on Pattern Recognition (ICPR '94)*, pages 689–691, October 1994.
7. W. R. Mark and G. Bishop. Memory access patterns of occlusion-compatible 3d image warping. In *Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware*, pages 35–44, August 1997.
8. W. R. Mark and G. Bishop. Efficient reconstruction techniques for post-rendering 3d image warping. Technical report, University of Northern Carolina at Chapel Hill, March 1998. UNC CS #TR98-011.
9. W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.
10. N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.
11. L. McMillan. Computing visibility without depth. Technical report, University of North Carolina, October 1995. UNC Computer Science TR95-047.
12. L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1997.

13. L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 39–46, August 1995.
14. K. Prazdny. On the information in optical flows. *Computer Vision, Graphics and Image Processing*, 22(9):239–259, 1983.
15. F. P. Preparata and M. I. Shamos. *Computational Geometry, An Introduction*. Springer-Verlag, 1985.
16. S. M. Seitz and C. R. Dyer. View morphing. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 21–30, 1996.
17. J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 231–242, July 1998.

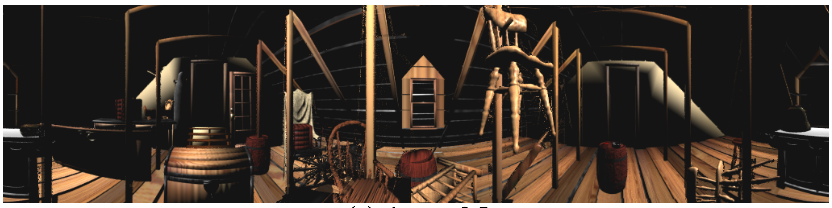


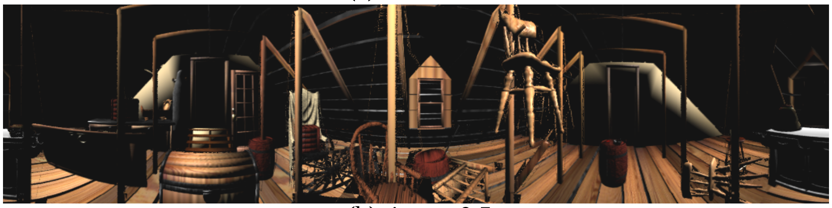
(a)

(b)

(c)

Fig. 10. Triangulation of the panorama. a) The reference panorama, b) the potential map, and c) the triangulated mesh.


(a) time = 0.2


(b) time = 0.5

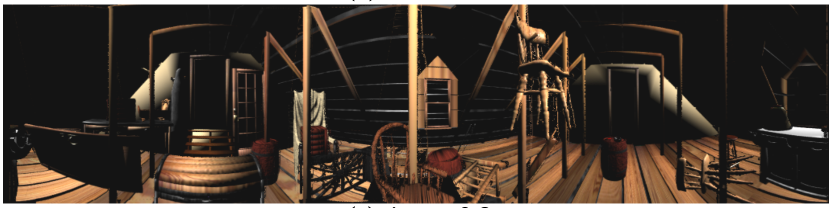
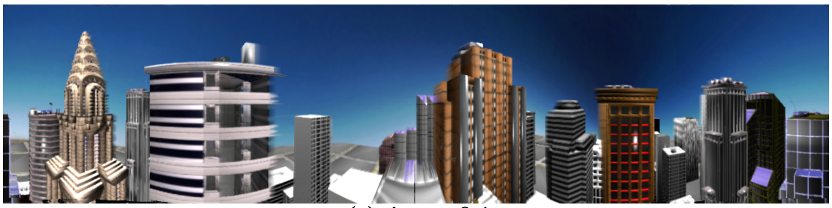
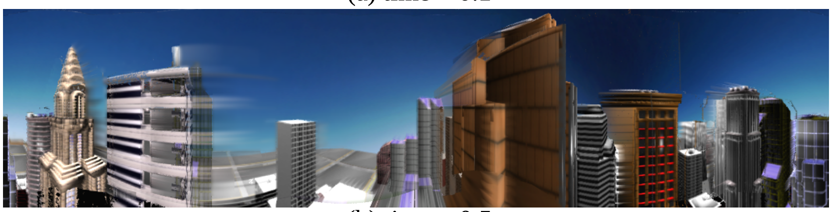

(c) time = 0.8

Fig. 11. Warping the attic scene.


(a) time = 0.1


(b) time = 0.5

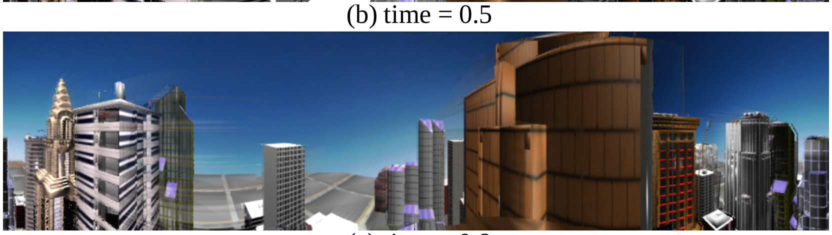
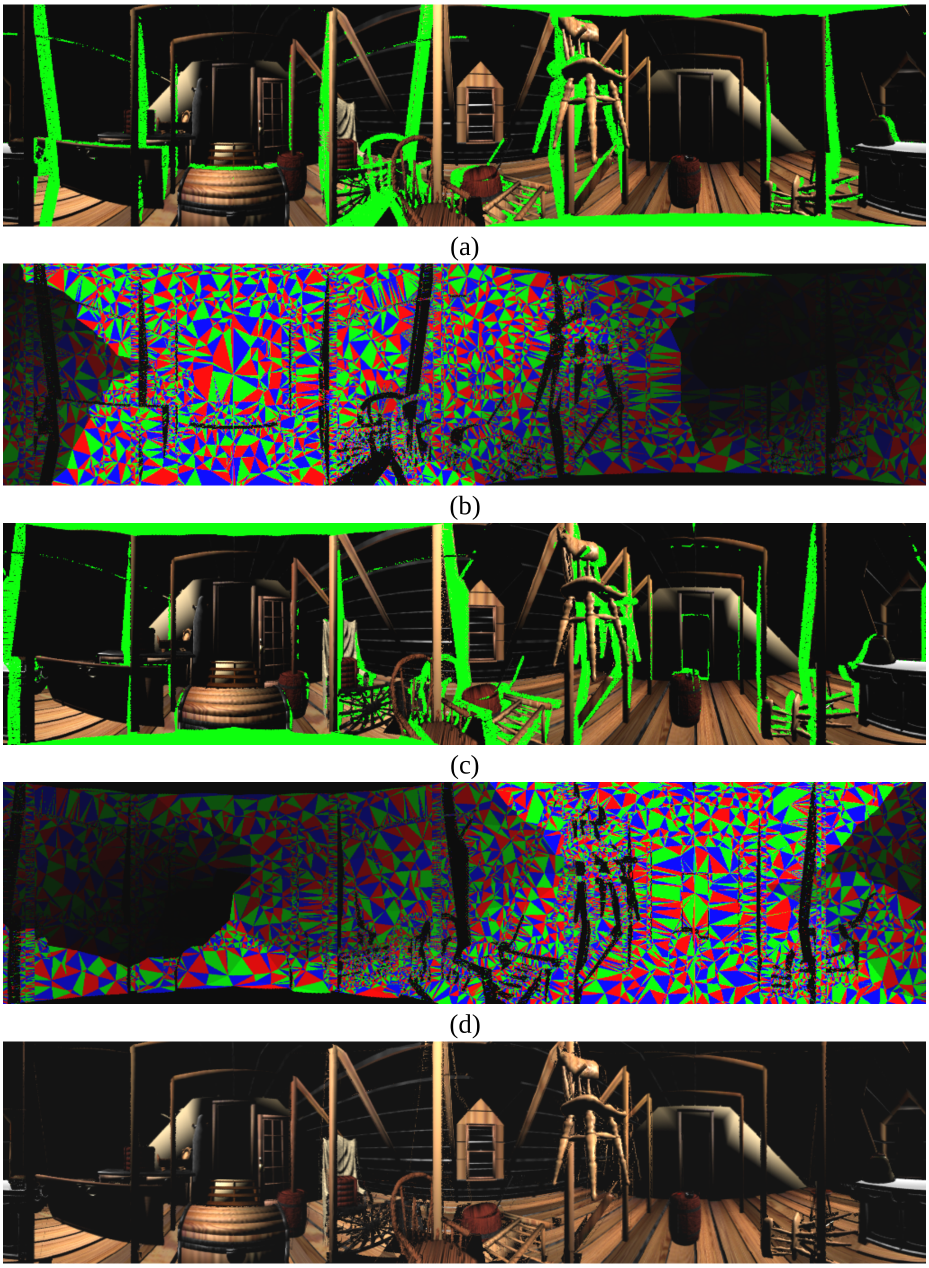

(c) time = 0.8

Fig. 12. Warping the city scene.



(a)

(b)

(c)

(d)

(e)

Fig. 13. Blending of two warped panoramas.