Triangle-based View Interpolation Without Depth-Buffering

Chi-Wing Fu[†] Tien-Tsin Wong[‡] Pheng-Ann Heng[†]

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

Abstract

In this paper, we propose a triangle-based view interpolation algorithm which can correctly resolve the visibility problem without depth-buffering. The algorithm is especially useful when depth information is not available, such as in the case of real-world photographs. By subdividing the reference image into variable-sized triangles, view interpolation can be done efficiently using existing graphics hardware. We derive the drawing order between each pair of neighboring triangles from the epipolar geometry. Using this drawing order, a graph can be built and topological sorting is applied on the graph to obtain the complete drawing order of all triangles in linear time.

1 Introduction

Traditional geometry-based computer graphics requires significant amount of time to render complex scenery due to the dependency of the rendering time on the scene complexity. Even with the state-of-the-art graphics accelerator, interactive rendering is still far from satisfactory. Image-based computer graphics provides an alternative to render complex scene within a short period of time. Several image-based approaches have been proposed during the last few years. In this paper, we focus on solving the visibility problem when warping a given image (reference image) to generate an image (desired image) from a new viewpoint.

Given an image with depth, image as viewed from another viewpoint can be synthesized by reprojecting each pixel. 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 are determined. We cannot resolve the visibility by depth-buffering.

McMillan [14, 13] 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 [4] or by finding the point correspondences [10] or optical flow [16]), the image can be warped correctly. *No* depth-buffering is needed. The visibility is solved by mapping pixels in a specific order. From now on, whenever we use the term *McMillan's drawing order*, we actually refer to this pixel drawing order.

Due to the nature of the drawing order, only small image entities, such as pixels, can be applied. However, warping image in a pixel-by-pixel manner (*pixel-based warping*) is time-consuming and cannot utilize existing graphics hardware. Moreover, gap occurs between adjacent pixels after they are warped. If the image is subdivided into larger image entities (a group of neighboring pixels like triangles) and the mapping is then applied on them instead of pixels, we can make use of the graphics hardware to accelerate the image warping. The gap problem can also be solved at the same time. We call the triangle-by-triangle image warping the *triangle-based warping*. Unfortunately, McMillan's drawing order cannot be applied to larger entities. We

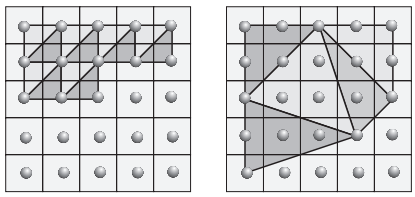
introduce a visibility sorting algorithm to find out the ordering of triangles for image warping. We will also show that the time complexity of the algorithm is linear to the number of triangles. Since the image is now subdivided into triangles, warping can be done efficiently with the assistance of graphics hardware.

2 Related Work

Chen and Williams [4] warped images by reprojecting each pixel onto the new image. Depth-buffering is used to solve the visibility as multiple pixels may be mapped to the same location in the new image. Darsa *et al.* [5] subdivided the depth image into triangles and performed reprojection on each of them. Again the visibility is solved by depth-buffering. Seitz and Dyer [19] introduced the view morphing which can correctly interpolate two different views based on image morphing [2]. The positions of the cameras and the correspondence of some feature points are required.

McMillan [12, 14, 13] first proposed a drawing order to solve the visibility without using depth-buffering. The optical flow (*i.e.* the 2D mapping from the reference image to the desired image) has to be determined either by pixel reprojection just as in view interpolation [4] or correspondence determination [7] in computer vision.

If the pixels are forward mapped to the new image, gaps will appear in between those pixels. Laveau and Faugeras [10] used a backward mapping, which maps pixels from the desired image back to the reference images, in order to prevent the existence of gap. It is similar to the backward texture mapping in computer graphics. Mark *et al.* [11] solved the gap problem by two methods, namely splatting and modeling the image as a triangular mesh. To prevent the gap using splatting, the footprint of the pixel must be large enough. However large footprint may 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.* [20] further extended the usage of McMillan's drawing order to image with multiple layers of depth values.

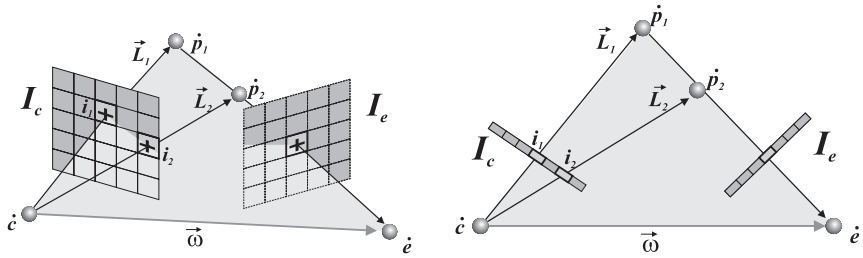


(a) (b)

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

3 Epipolar Geometry

Before describing the proposed algorithm, we first describe some basics of epipolar geometry. Consider a planar perspective image I_c captured with the center of projection at $\dot{c}$. We use the dot notation $\dot{a}$ to denote a 3D point and the arrow notation $\vec{a}$ to denote a 3D directional vector. A desired image I_e is generated with a new center of projection at $\dot{e}$. Figure 2 shows the geometry in both 3D and 2D.



(a) (b)

Figure 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 2D location in I_e. If $\dot{p_2}$ does not lie on the epipolar plane associated with $\dot{p_1}$, then $\dot{p_1}$ and $\dot{p_2}$ will never occlude each other (see Figure 3). Hence occlusion happens only when $\dot{c}$, $\dot{e}$, $\dot{p_1}$ and $\dot{p_2}$ all lies 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 Figures 2 and 5.

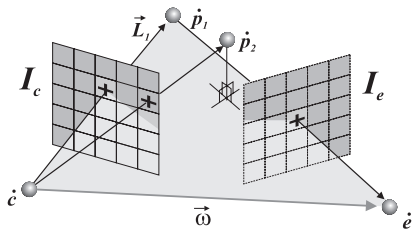


Figure 3: Since $\dot{p_2}$ does not lie on the epipolar plane associated with $\dot{p_1}$, $\dot{p_2}$ and $\dot{p_1}$ will never occlude each other.

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 $\dot{i_1}$ before $\dot{i_2}$ during reprojection, the visibility problem is solved without knowing or 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 planar projection manifold (image I_c). The intersection line is called the *epipolar line*. Figure 4(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 4 denotes it by a positive sign. On the other hand, if the negative epipolar ray intersects with the projection manifold, the intersection point is known as *negative epipole* and denoted by a negative sign. Note all epipolar lines pass through the

epipole (either positive or negative). When the epipolar rays parallel to the planar projection manifold, no intersection point is found on the plane. All epipolar lines are in parallel.

All pixels in I_c that lie on the same epipolar line have a chance to occlude each other. Figure 5 shows two pixels, i_1 and i_2, lying on the same epipolar line. Their associated intersection points $\dot{p_1}$ and $\dot{p_2}$ are coplanar and may occlude each other. And $\dot{p_1}$ will never occlude $\dot{p_2}$ as $\dot{p_1}$'s angle of derivation θ_1 is greater than, θ_2 of $\dot{p_2}$. 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. The arrow on the epipolar line in Figure 5 indicates the drawing order of pixels. On the other hand, if i_2 is closer to the negative epipole on the epipolar line than i_1, i_2 will never occlude i_1. By intersecting all of the epipolar planes with the image I_c (Figure 4(b)), we obtain pictures of the drawing order (Figure 6). Note that once $\dot{c}$ and $\dot{e}$ are known, the picture of drawing order is already determined. It is not necessary to define the epipolar planes explicitly. Hence no depth information is required in constructing the drawing order.

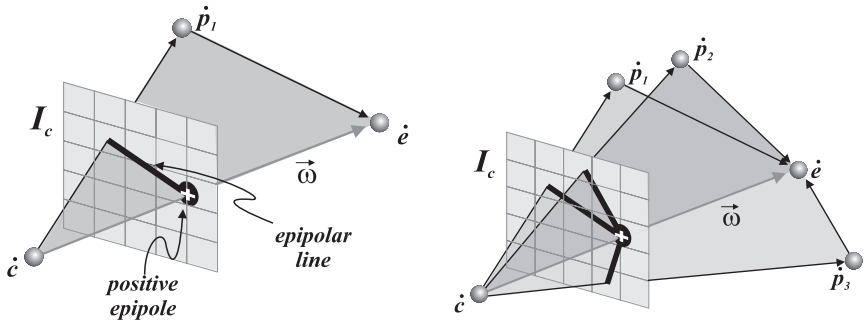


Figure 4: The epipolar line is the intersection of the projection manifold and the epipolar plane.

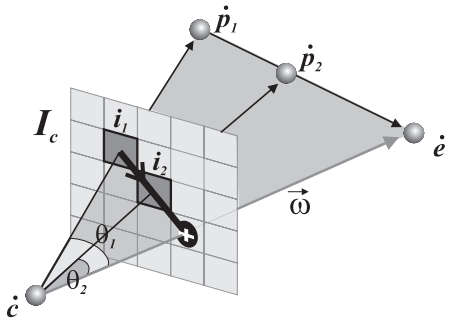


Figure 5: The drawing order between two pixels that lie on the same epipolar line.

Only three main categories of drawing order exist. If the positive epipolar ray intersects with the projection manifold, a converging pattern (Figure 6(a)) will be obtained. On the other hand, if the negative epipolar ray intersects, a diverging pattern is resulted (Figure 6(b)). If the epipolar rays parallel to the projection manifold, the epipoles can be regarded as located infinitely far away and the epipolar lines will be all in parallel (Figure 6(c)).

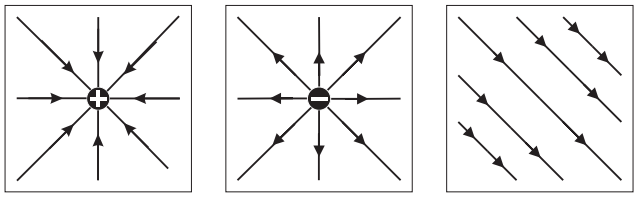


Figure 6: The drawing patterns

4 Drawing Order for Pixel-Sized Entities

McMillan derived two drawing orders of pixels from the patterns in Figure 6. They are shown in Figure 7. Following these drawing orders, the visibility can be correctly resolved. No depth-buffering is required. Pixels on different epipolar lines can be drawn in arbitrary order. However, the epipolar lines only tell us the ordering of pixel-sized entities that lie on the same line. If we group pixels to form larger entities (such as triangles) which overlap with multiple epipolar lines (Figure 8), the ordering of them is not clear. McMillan's drawing order in Figure 7 is no longer applicable for larger entities.

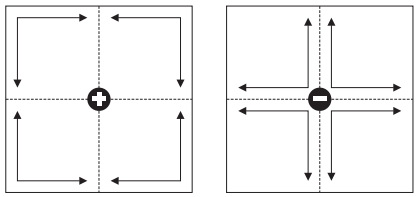


Figure 7: Two drawing orders derived from the patterns of epipolar lines.

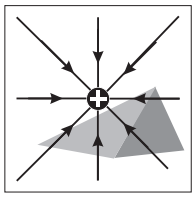


Figure 8: Larger image entities overlap with multiple epipolar lines.

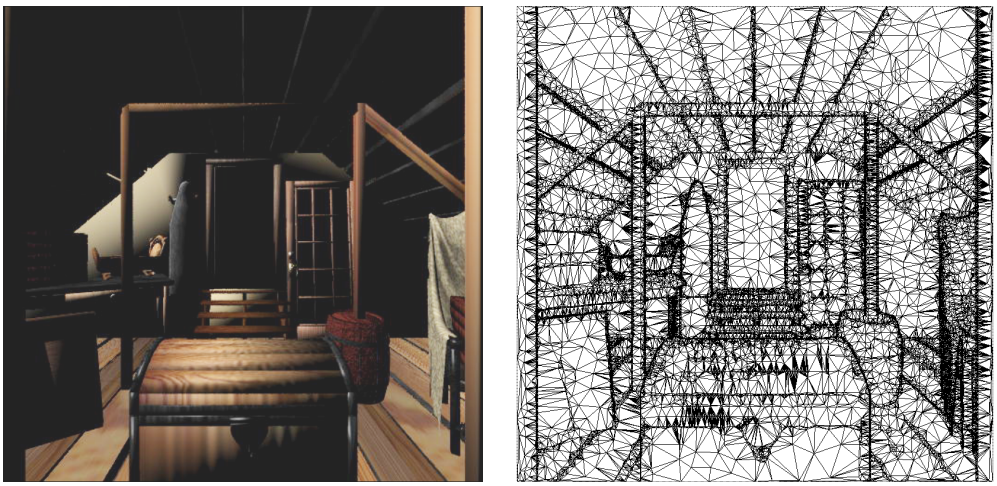
5 Triangle-based Image Warping

5.1 Triangulation

To warp the image efficiently with triangles, we first have to triangulate the reference image into a set of triangles based on the associated depth map or the map of optical flow. In our implementation, we use depth map. But the basic idea of triangulation is applicable to optical flow map as well.

The depth map is a two-dimensional array of depth values. Each 2D integral coordinate in the depth map is associated with a scalar depth value. Therefore, the depth map can be regarded as a special kind of heightfield. The most obvious way to triangulate the heightfield is to form a grid first. Then each rectangle in the grid is triangulated into two triangles, as depicted in Figure 1(a). However, the number of triangles is about twice the number of samples in the depth map which is very large. Several adaptive methods [17, 6, 15] have been proposed to directly triangulate the heightfield and range data into a mesh of variable-sized triangles. We can also start with a fine triangular mesh and decimate it using various post-processing mesh reduction algorithms [8, 18, 21]. Since the triangulation methods can be found in several literatures, we will not discuss them in detail. Note that we only need a 2D triangulation on an image, not 3D. The output of the above algorithms are 3D. We can reproject those triangles onto the image to obtain the required 2D triangular mesh.

One major criterion of choosing the triangulation method is that all elements in the same triangle should have similar depth or optical flow values. Otherwise, the triangle will be excessively distorted after warping and introduces visual artifacts. The triangle should be subdivided into smaller triangles if the elements inside have depth or optical flow values with significant difference. A threshold can be used to guide the subdivision. Hence it is a tuning problem. Figure 9 shows the result of triangulating an attic scene.



(a) (b)

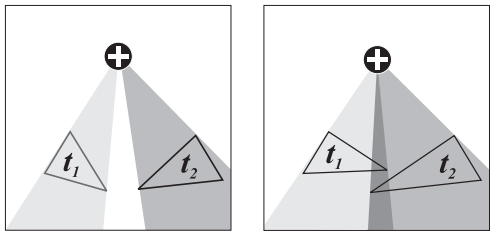
Figure 9: Triangulation of the attic scene.

Note the triangulation is done only once before any image warping. Hence the triangulation can be done off-line without affecting the frame rate of image warping. The resultant triangular mesh is stored in a winged-edge data structure [1, 22] as the connectivity information will facilitate the visibility sorting of triangles in the next phase. A more compact alternative to store the triangular mesh with connectivity is directed-edge data structure proposed by Campagna *et al.* [3].

5.2 Image-based Visibility Sorting

Given two arbitrary triangles, t_1 and t_2, obtained by triangulating the image. We can check whether these two triangles may occlude each other, by checking the range of epipolar lines these triangles occupy. Let's call the range of epipolar lines occupied by a triangle the *epipolar band*. Figure 10(a) shows the epipolar band occupied by triangle t_1 in light gray while that of triangle t_2 in dark gray. If the epipolar bands occupied by these two triangles have no intersection (Figure 10(a)), no occlusion will occur between these two triangles.

Because there are no element (pixel) in the two triangles sharing any common epipolar line. Hence the order of drawing these two triangles is irrelevant. On the other hand, if two epipolar bands intersect (Figure 10(b)), some elements from the two triangles are lying on the same epipolar line. Hence the occlusion may occur after warping. Therefore, the ordering of these two triangles does matter. In the specific example of Figure 10(b), t_1 may occlude t_2 as t_1 is closer to the positive epipole than t_2 in the intersection region.



(a) (b)

Figure 10: Epipolar band.

Now, let's define two ordering relations $\rightarrow$ and $\leftrightarrow$.

Definition 1 *If all elements in a triangle t_1 must be drawn before any element in another 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 $t_1 \rightarrow t_2$.*

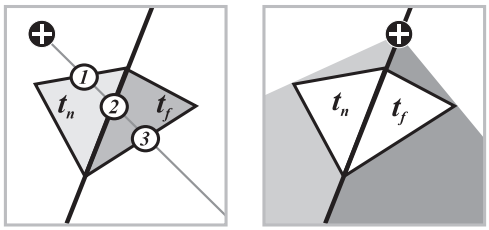
Definition 2 *If the drawing order between the elements in triangle t_1 and elements in another triangle t_2 is irrelevant, the drawing order between these two triangles are also irrelevant. We denote this ordering relation as $t_1 \leftrightarrow t_2$.*

Since all the triangles are obtained by triangulating the image with center of projection at $\dot{c}$, they must be visible, non-overlapping and connected as viewed from $\dot{c}$. Instead of considering the order of any two arbitrary triangles from the mesh, we first consider the ordering between each pair of neighboring triangles which are sharing a common edge as in Figure 11(a). Now we will show that the ordering of any two neighboring triangles can be determined by the position of the positive or negative epipole.

Theorem 1 *Given two neighboring triangles which are sharing a common edge. The planar projection manifold can be divided into two halves by extending the shared edge. The 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 epipole (either positive or negative) lies exactly on the shared edge, the ordering of these two triangles is irrelevant.*

Proof: Let's denote the triangle with the positive (negative) epipole on its side as t_n and the other as t_f. All epipolar lines on the planar projection manifold must be straight lines. And they must all pass through the positive (negative) epipole. Now we can draw a straight epipolar line starting from the positive (negative) epipole and passing through both t_n and t_f. Since the positive (negative) epipole is on the same side as t_n, whenever the straight epipolar line passes through both t_n and t_f, it should first pass through t_n, then the shared edge and finally t_f (Figure 11(a)). Therefore whenever there are elements in t_n which are sharing a common epipolar line with some elements in t_f, the elements in t_n should be closer to the epipole than those in t_f. If the epipole is positive, all elements in t_n are closer to the positive epipole than any element in t_f. Hence, no element in triangle t_f will occlude any element in t_n and we must draw t_f before t_n during warping, *i.e.* $t_f \rightarrow t_n$. Figure 12(a) shows one such example ($t_n = t_1$ and $t_f = t_2$). On the other hand, if

the epipole is negative, no element in t_n will occlude any element in t_f as all elements in t_f are farther away from the negative epipole. So we must draw t_n before t_f during warping, *i.e.* $t_n \rightarrow t_f$. Figure 12(d) shows a specific example ($t_n = t_1$ and $t_f = t_2$).



(a) (b)

Figure 11: (a) The straight line starting from the epipole always enter t_n before t_f whenever the line cuts both t_n and t_f. (b) If the epipole lies on the shared edge, the epipolar bands of these two neighboring triangles have no intersection.

When the epipole lies on the shared edge, the epipolar bands of t_n and t_f have no intersection (Figure 11(b)). Because one can always separate the epipolar bands of the two triangles by drawing a line which coincides with the shared edge. In other words, no element in t_n and t_f shares a common epipolar line. Therefore their ordering is irrelevant and we say $t_n \leftrightarrow t_f$. Figures 12(b) & 12(e) show two such cases.

If the epipolar ray does not intersect with the planar projection manifold, the epipoles can be regarded as located infinitely far away. All epipolar lines are in parallel and pointing from the negative epipole to the positive epipole. We can still determine which triangle is on the same side with the infinite epipole by determining the direction of the epipolar line. Figure 12 shows all the possible cases and their corresponding drawing order between two neighboring triangles. From the diagrams (g) to (l), the epipoles are drawn outside the projection image to indicate that they are located infinitely far away.

5.3 Topological Sorting

Using the simple 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 the 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 13(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 13(b). Figure 13(c) shows two valid drawing orders derived from the example graph. Note there is no unique ordering for the same graph.

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 the neighboring triangles using Theorem 1. The time complexity of the graph construction is obviously $O(E)$ where E is the number of shared edges. As each triangle has three edges, E must be smaller than $3N$ where N is total number of triangles. Hence, the time complexity should be linear to the total number of triangles.

The final step to find out the ordering of all triangles is to perform a topological sort on the drawing order graph. The details of topological sort can be found in various introductory algorithm literatures [9, 23]. The basic idea of topological sort is to find out and output a triangle t_1 such that no other triangle is needed to be

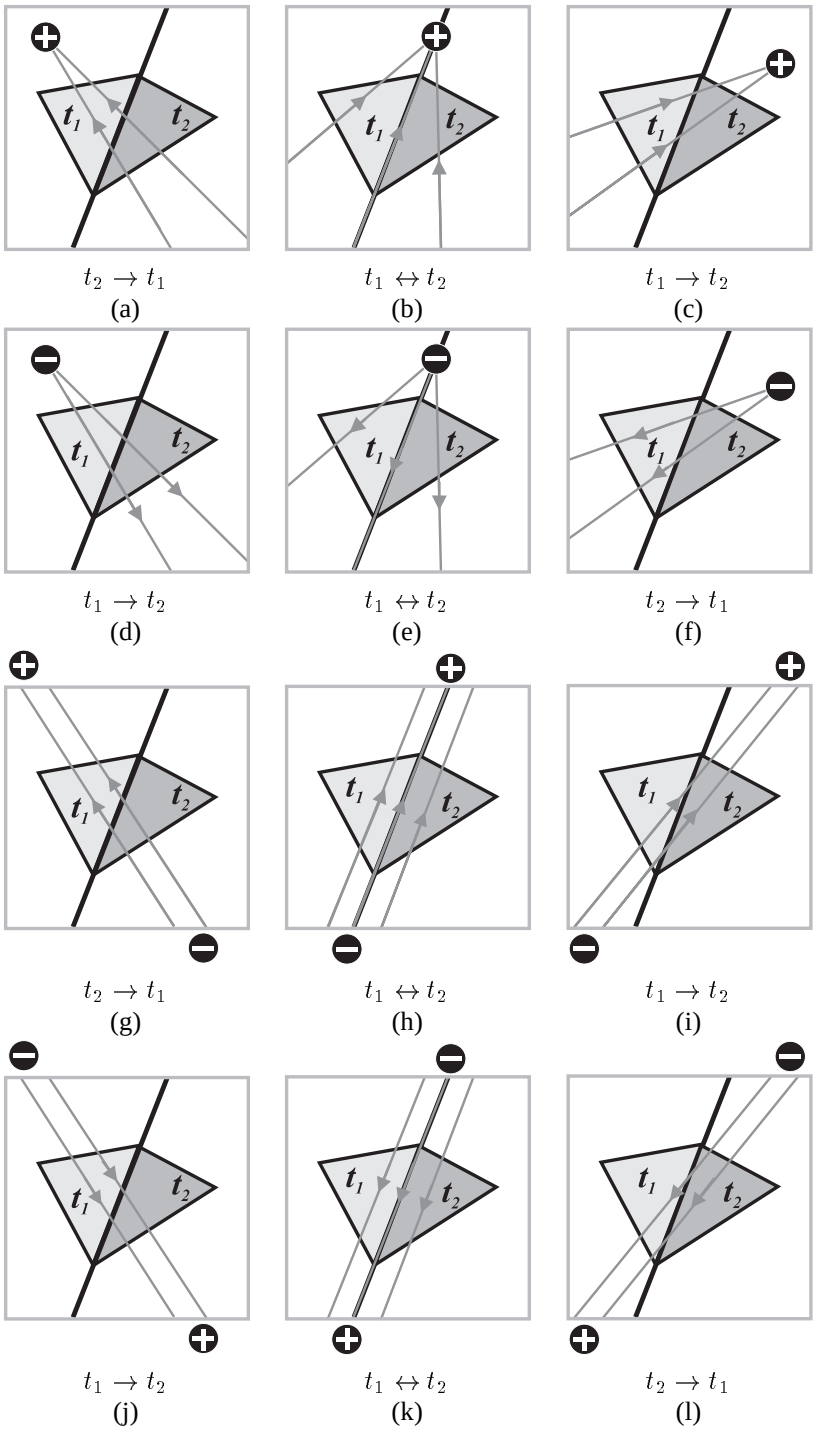


Figure 12: The drawing orders of all cases.

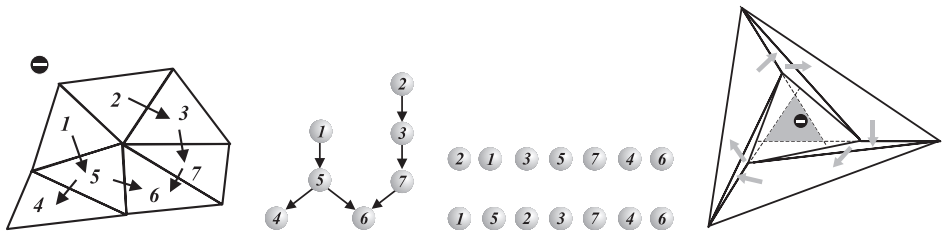


Figure 13: (a)-(c): Construction of drawing order graph. (d) Cycle may exist in the graph.

drawn before t_1, *i.e.* t_1 is not on the right hand side of any $\rightarrow$ relation.

Topological sort can always generate an ordering if the drawing order graph does not consist of any cycle. It seems that the graph will be a directed acyclic graph. However cycle does exist in extremely rare cases. Figure 13(d) shows one special example of triangulation such that cycle exists. If the epipole locates inside the gray region, cycle will occur. If a relation satisfies the criteria of *partial ordering* [9] (transitivity, antisymmetry and reflexivity), the associated directed graph can be guaranteed cycle-free. However the relation $\rightarrow$ we have defined is not transitive. In order words, if $t_1 \rightarrow t_2$ and $t_2 \rightarrow t_3$, it does not imply $t_1 \rightarrow t_3$. Because we say $t_1 \rightarrow t_2$ if t_1 *must* be drawn before t_2 *in order to preserve the correctness of the visibility*. If $t_1 \rightarrow t_2$ and $t_2 \rightarrow t_3$, it is very common that $t_1 \leftrightarrow t_3$, *i.e.* the drawing of t_1 and t_3 is no longer relevant. This can be demonstrated by Figure 13(d).

The original topological sort must be modified to handle the case when a cycle is found. This can be easily achieved by a linear-time cycle detector, randomly picking a triangle in the cycle and drawing it to the screen, hence breaking the cycle. This approach may result in visual artifact (occlusion among different objects may be incorrect). Another approach is to fall back to use pixel-based warping for triangles in the cycle.

In practice, cycle seldom occurs and no cycle has been found in all our experiments. Physically, if a cycle exists, it implies there exists some objects a, b, and c such that a is in front of b, b is in front of c and c is in front of a (like the objects in M. C. Escher's drawing). Obviously, it is not possible in real world. We believe the source of cycle is due to improper triangulation. If the triangulation algorithm fails to separate objects with different depth values, the cycle problem may appear.

The time complexity of the topological sort is $O(E + N)$ where E is the number of relations (edges in the graph) and N is the number of triangles. Since E is at most $3N$, the time complexity is actually linear to the number of triangles.

6 Results

Figure 14 shows three frames from an animation sequence of warping an image of Beethoven statue. The images on the first row show the result if the image is forward warped in a pixel-by-pixel manner. Since no splatting is performed, gap exists in between the pixels. Images on the second row are the final images of running our algorithm. The third row shows their corresponding warped triangulation together with the drawing order. To distinguish one triangle from another, we use three distinct colors to color neighboring triangles. The intensity of the triangle indicates the drawing order. The darker the color, the former is the triangle in the drawing order. In Figures 14(h) & 14(i), it shows how the visibility is resolved by the drawing order. Note all triangles in the front (those closer to the viewpoint) are lighter in color than those in the back. The coloring in Figure 14(g), 15(d) and 16(d) look random because the drawing order is irrelevant. In this case, all triangles are visible and no occlusion occurs as viewed from the original viewpoint c.

An indoor scene is shown in Figure 15 while an outdoor scene is shown in Figure 16. The first row shows the final warped images while the second row shows the warped triangles together with the drawing order. Note how the visibility is correctly resolved even no depth-buffering is used. The shaft in the attic scene and the building in the city scene correctly overlap the background without comparing the depth values. Since the image is now triangulated as a set of connected triangles, no gap exists in between them. The holes in between the triangles in our result are intentionally introduced to prevent excessive elongation after warping two neighboring triangles with discontinuous depth or optical flow values.

The best condition for executing the proposed algorithm is when the image resolution is high and the scenery contains objects with gradually changing depth values (distances from the viewpoint). In this case, the image can be triangulated into few large triangles. Both topological sorting and image warping can be performed rapidly. On the other hand, if the image resolution is low and the scenery contains objects with abruptly changing depth values, the image may be triangulated into many small triangles. The computational time of topological sorting may dominant the overall execution time and hence overwhelm the advantage of drawing triangles.

7 Conclusions

In this paper, we proposed a triangle-based image warping algorithm which solves visibility without using depth-buffering. By grouping pixels to form triangles, the image warping can be done more efficiently with the utilization of graphics hardware. Moreover, the gap problem of pixel-based approach is also removed at the same time.

Deriving from the epipolar geometry, we introduce the drawing order between two neighboring triangles. The ordering relation allows us to construct a graph of drawing order. Applying the topological sorting on this graph gives us the drawing order of all triangles. No depth-buffering is needed if the triangles are drawn in this order. Both the graph construction and topological sorting have a linear time complexity. Moreover the graph construction and topological sorting are required only when the epipolar ray changes. If the user only translates the viewpoint along the epipolar ray or rotates about the original viewpoint, the same ordering can be used without rebuilding the graph and sorting the relations.

Web Information

Demonstrative movies and sample images are available on the web at
http://www.acm.org/jgt/papers/FuWongHeng99/
http://www.cse.cuhk.edu.hk/∼cwfu/papers/triOrder/triOrder.html
http://www.cs.ust.hk/∼ttwong/papers/triorder/triorder.html

Chi-Wing Fu, Department of Computer Science & Engineering, The Chinese University of Hong Kong (CUHK), Shatin, Hong Kong (cwfu@cse.cuhk.edu.hk) (http://www.cse.cuhk.edu.hk/∼cwfu/)

Tien-Tsin Wong, Department of Computer Science, The Hong Kong University of Science & Technology (HKUST), Clear Water Bay, Kowloon, Hong Kong (ttwong@acm.org) (http://www.cs.ust.hk/∼ttwong/)

Pheng-Ann Heng, Department of Computer Science & Engineering, The Chinese University of Hong Kong (CUHK), Shatin, Hong Kong (pheng@cse.cuhk.edu.hk) (http://www.cse.cuhk.edu.hk/∼pheng/)

References

[1] Bruce G. Baumgart. Winged edge polyhedron representation. Technical report, Stanford Artificial Intelligence Laboratory, Stanford University, October 1972. Technical Report CS-320.

[2] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 35–42, July 1992.

[3] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges – a scalable representation for triangle meshes. *Journal of Graphics Tools*. to appear.

[4] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '93)*, pages 279–288, 1993.

[5] Lucia Darsa, Bruno Costa Silva, and Amitabh 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.

[6] Michael J. Dehaemer and Michael J. Zyda. Simplification of objects rendered by polygonal approximations. *Computer & Graphics*, 15(2):175–184, 1991.

[7] Oliver Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, MA, 1993.

[8] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 19–26, August 1993.

[9] Donald Knuth. *The Art of Computer Programming; Volume 1: Fundamental Algorithms*. Addision-Wesley, 1968.

[10] Stephane Laveau and Olivier 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, Jerusalem, Israel, October 1994.

[11] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.

[12] Leonard McMillan. Computing visbility without depth. Technical report, University of North Carolina, October 1995. UNC Computer Science Technical Report TR95-047.

[13] Leonard McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 1997.

[14] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '95)*, pages 39–46, August 1995.

[15] Tim Poston, Tien-Tsin Wong, and Pheng-Ann Heng. Multiresolution isosurface extraction with adaptive skeleton climbing. *Computer Graphics Forum*, 17(3):137–148, September 1998. Eurographics'98 issue.

[16] K. Prazdny. On the information in optical flows. *Computer Vision, Graphics and Image Processing*, 22(9):239–259, 1983.

[17] Lori Scarlatos and Theo Pavlidis. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[18] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

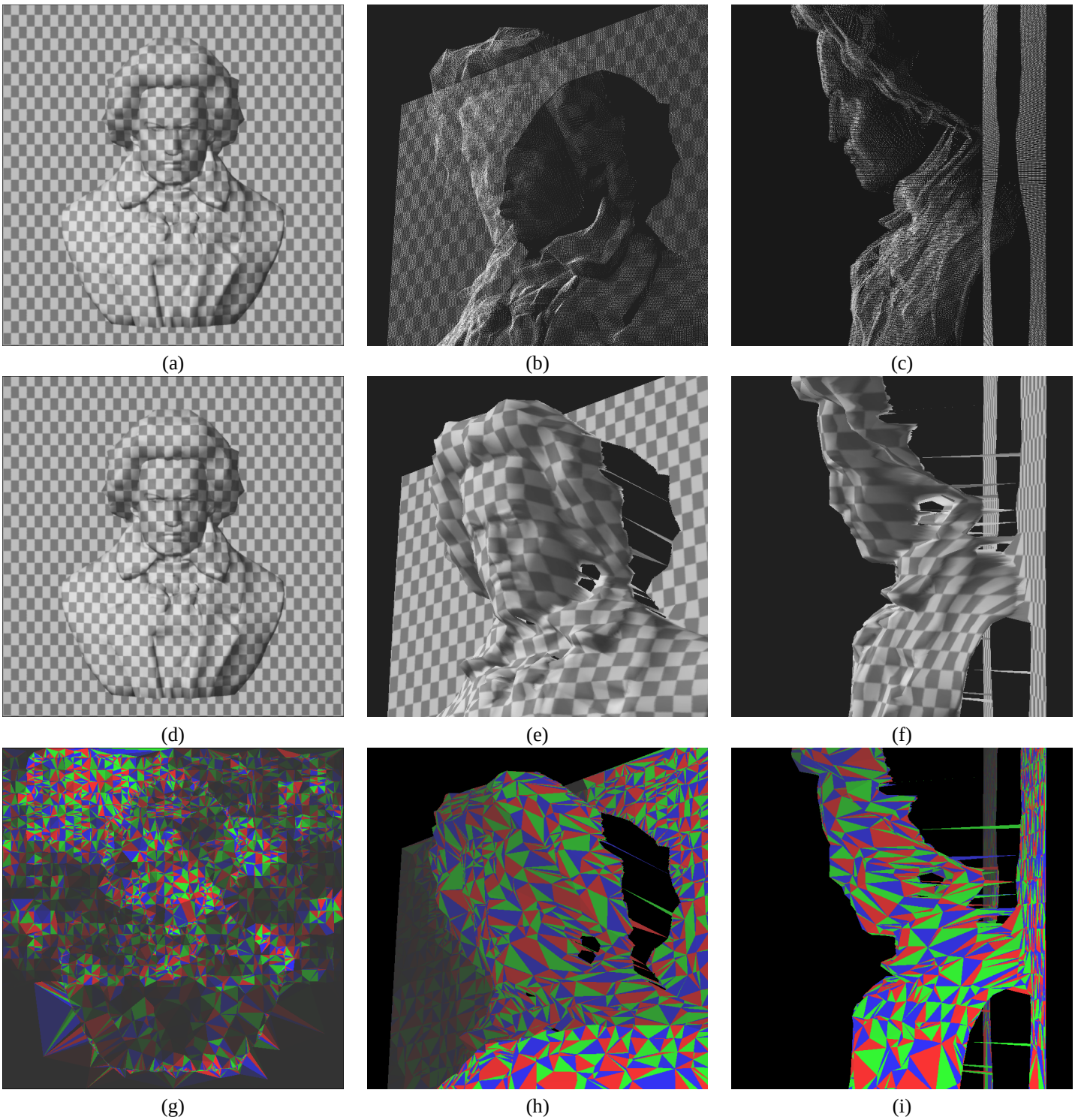
[19] S. M. Seitz and C. R. Dyer. View morphing. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '96)*, pages 21–30, 1996.

[20] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '98)*, pages 231–242, July 1998.

[21] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[22] K. Weiler. Polygon comparison using a graph representation. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 10–18, July 1980.

[23] Mark Allen Weiss. *Data structures and algorithm analysis*. Benjamin/Cummings Pub. Co, 1992.

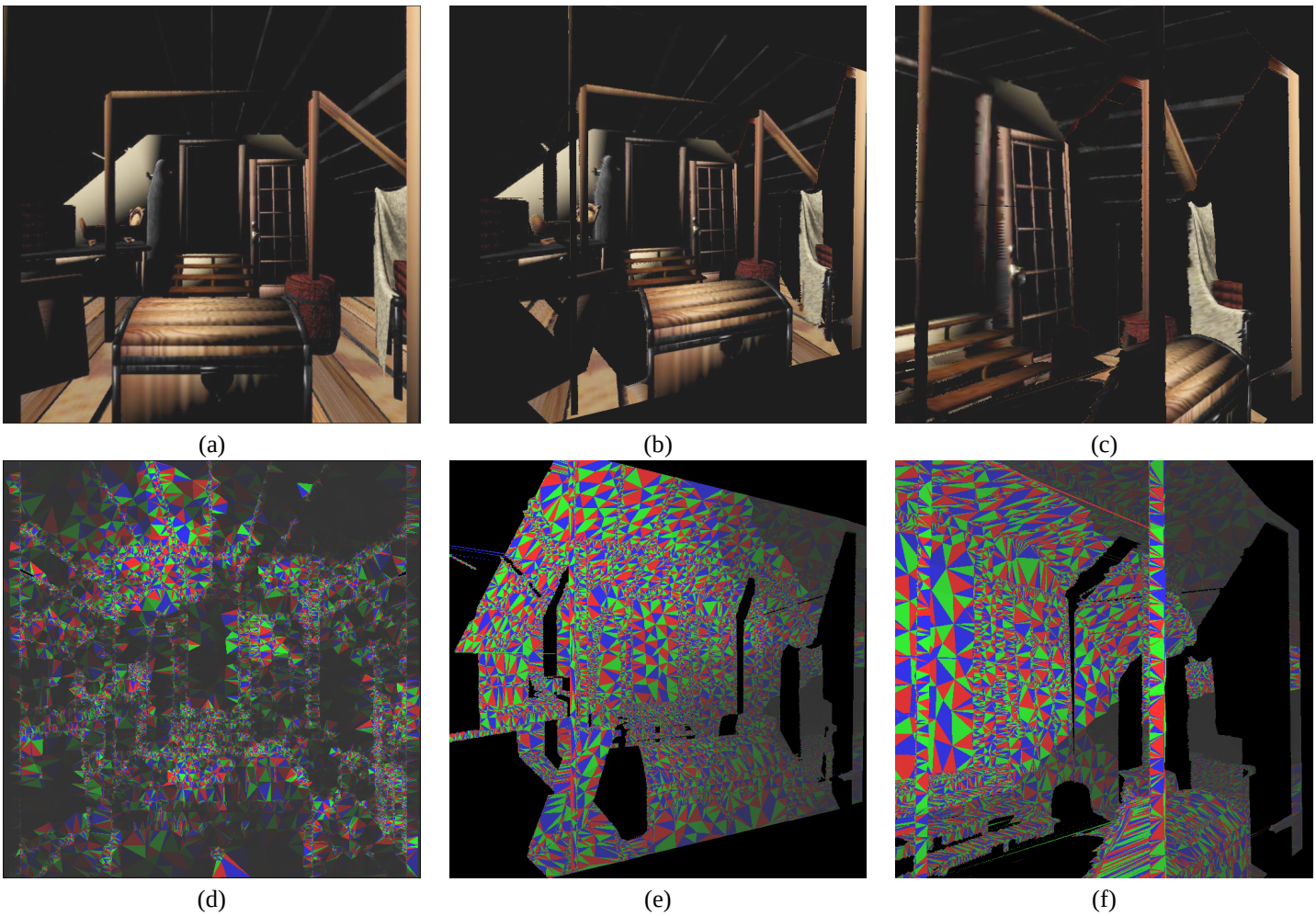


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

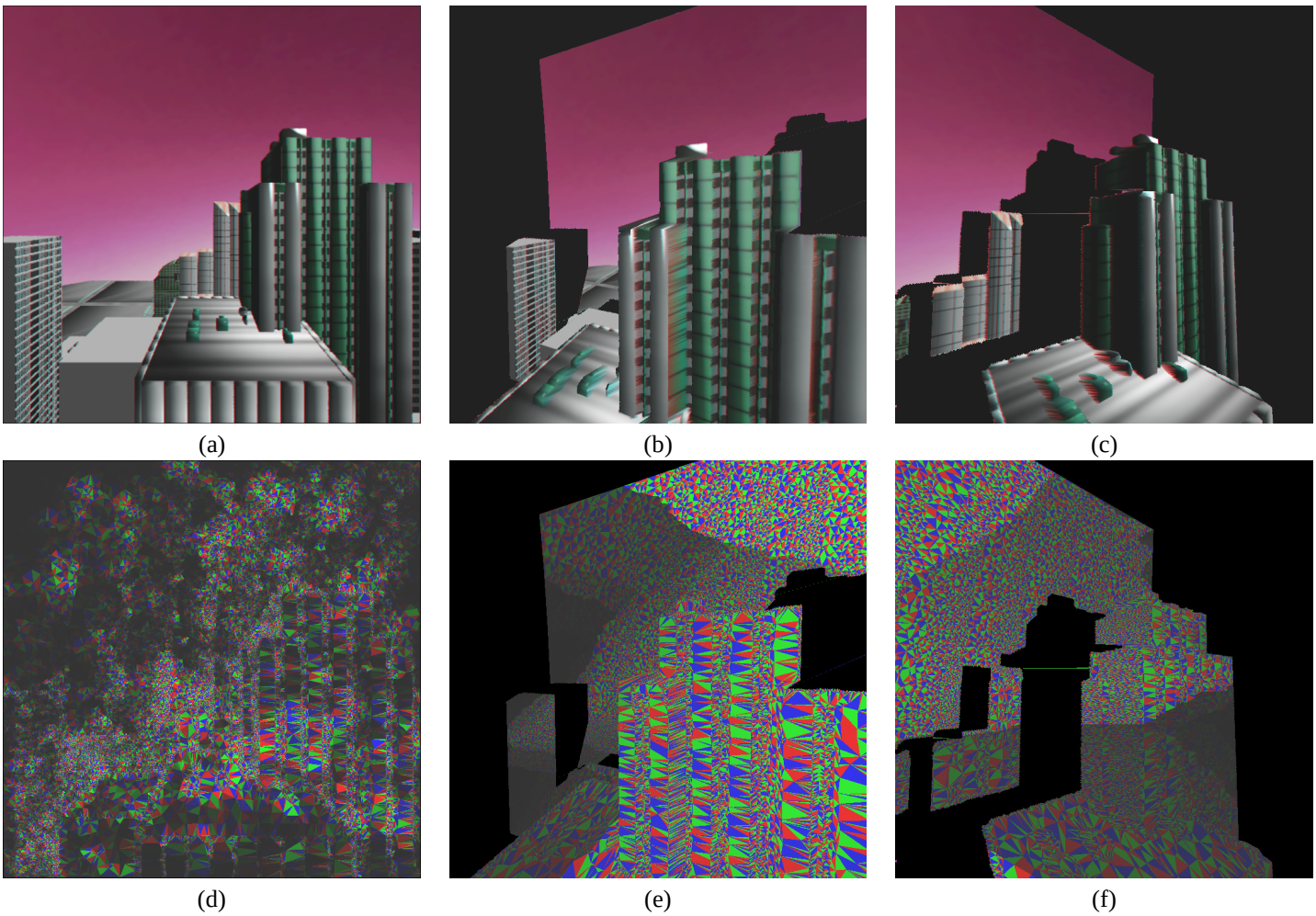
Figure 14: Beethoven.



(a) (b) (c)

(d) (e) (f)

Figure 15: Attic.



(a) (b) (c)

(d) (e) (f)

Figure 16: City.