remote sensing

MDPI

Article

GPVC: Graphics Pipeline-Based Visibility Classification for Texture Reconstruction

Xiangxiang Huang, Quansheng Zhu and Wanshou Jiang *

State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China; huangxiangxiang@whu.edu.cn (X.H.); zhuqs@whu.edu.cn (Q.Z.)

* Correspondence: jws@whu.edu.cn; Tel.: +86-27-6877-8092 (ext. 8321)

Received: 6 September 2018; Accepted: 30 October 2018; Published: 1 November 2018

Abstract: The shadow-mapping and ray-tracing algorithms are the two popular approaches used in visibility handling for multi-view based texture reconstruction. Visibility testing based on the two algorithms needs a user-defined bias to reduce computation error. However, a constant bias does not work for every part of a geometry. Therefore, the accuracy of the two algorithms is limited. In this paper, we propose a high-precision graphics pipeline-based visibility classification (GPVC) method without introducing a bias. The method consists of two stages. In the first stage, a shader-based rendering is designed in the fixed graphics pipeline to generate initial visibility maps (IVMs). In the second stage, two algorithms, namely, lazy-projection coverage correction (LPCC) and hierarchical iterative vertex-edge-region sampling (HIVERS), are proposed to classify visible primitives into fully visible or partially visible primitives. The proposed method can be easily implemented in the graphics pipeline to achieve parallel acceleration. With respect to efficiency, the proposed method outperforms the bias-based methods. With respect to accuracy, the proposed method can theoretically reach a value of 100%. Compared with available libraries and software, the textured model based on our method is smoother with less distortion and dislocation.

Keywords: Visibility checking; occlusion detection; multiview-based texture reconstruction; oblique photogrammetry; computer graphics pipeline

1. Introduction

Occlusion or visibility testing is nonnegligible in the fields of computer vision (CV), virtual reality (VR), computer graphics (CG), etc., in both 2D and 3D space. In 2D space, occlusion can be ascribed to static intersection or contact between polygons [1,2]. Tracking a deformable or moving object in an intricate and crowded surrounding [3–5] requires consideration of occlusion. In 3D space, occlusion issues become more complicated and diversified. Situations in which 3D visibility occurs include stereo matching [6], object reconstruction [7,8] and motion trajectory optimization [9–11]. With the development of multiview-based reconstruction [12], there has been an increasing focus on texture generation of 3D surfaces [13–17]. Visibility checking is highly important for the smoothness and verisimilitude of textured models. However, it appears that the visibility issue in this field has not attracted as much attention as texture generation.

Frueh et al. [18] ignored the disturbance of occluded pixels in textured models and did not consider the occlusion problem during texture reconstruction. This approach is acceptable only when the occluded area is sufficiently small; if the occluded area is large, then texture dislocations and seams occur (as shown in Figure 1a).

(**a**) (**b**)

Figure 1. The importance of visibility tests in texture reconstruction. (**a**) The textured model without visibility tests. (**b**) The textured model based on GPVC.

Xu et al. [19] introduced the hidden point removal (HPR) algorithm [20] to determine a face's visibility. If the inversion of a face's center lies on the convex hull of a mesh's vertices and a given viewpoint, then the face is fully visible from that viewpoint. However, HPR was initially proposed to compute a point's visibility in a point cloud; applying the algorithm to determine a face's visibility in a 3D mesh is insufficient. Even if the center C of a face is visible, the face can still be occluded (as shown in Figure 2a).

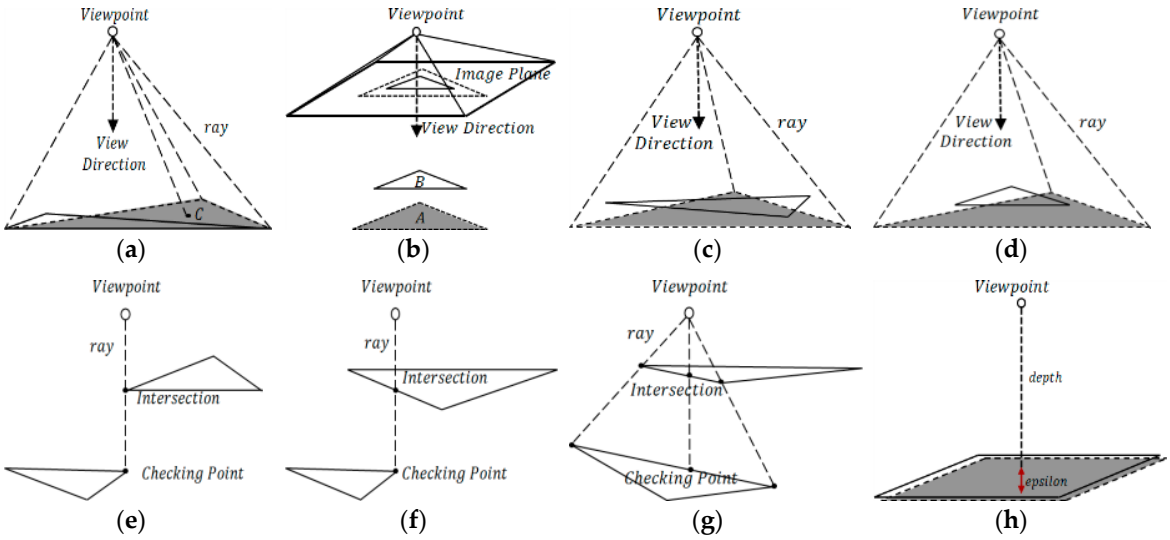


Figure 2. Visibility issues that available methods fail to handle. The visible faces are drawn with solid lines, and the occluded faces are drawn with dotted lines and filled with shadow. Occlusion detection based on HPR is unable to identify the occluded face in (**a**). The projection-based approach incorrectly marks the occluded face in (**b**) as visible. Without dense tracing, the ray-tracing method cannot identify the occluded faces in (**a**,**c**,**d**). In addition, ray tracing will incorrectly determine the visible faces in (**e**–**g**) as occluded. Without an appropriate bias, it is difficult to compute the correct visibility between two close faces in (**h**) through ray-tracing and shadow-mapping approaches.

Pages et al. [13] used a projection-based approach to handle the occlusion problem. If one vertex V of face A is projected inside the projection area of face B and B lies between V and a given viewpoint, then A is occluded by B. However, this technique only works in the occlusion condition, as depicted in Figure 3. If none of A's vertices are projected inside B (as shown in Figure 2b), then A will be incorrectly determined as visible.

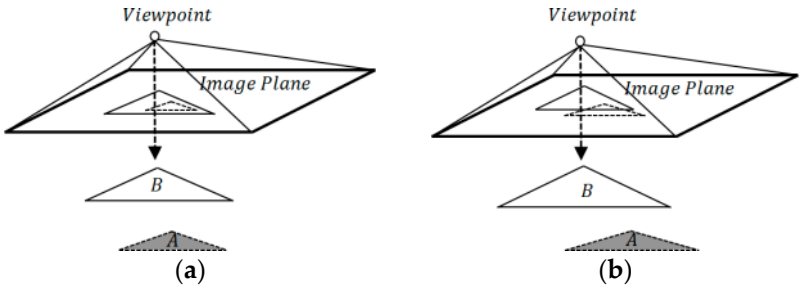


Figure 3. Occlusion conditions in which the projection-based approach can work. (**a**) Face *A* is entirely occluded by face *B*. (**b**) Face *A* is partially occluded by face *B*.

According to the coherence of perspective transformation between 3D rendering and pinhole camera imagery, visibility checking in texture reconstruction can be regarded as a point-based visibility test [21,22]. Therefore, many related studies have applied ray-tracing [23–26] and distance-based comparison [14,16,27–33] of CG to solve the visibility issue in texture reconstruction.

The standard ray-tracing algorithm traces a virtual ray from a viewpoint to a point p on a 3D surface; if an intersection between the viewpoint and the surface is found before reaching p, then p is occluded [34]. This algorithm was adopted in previous studies [15,35] to check visibility. In the implementation reported in Reference [14], the authors designed sparse tracing (three rays are cast to a face's vertices) based on an open-source library [36]. Clearly, the occluded faces in Figure 2a,c,d will be inappropriately identified as visible by this type of implementation. In addition, when intersections occur on edges, as shown in Figure 2e,f, the visible faces will be identified as occluded. More importantly, all calculations involving ray tracing will introduce a bias [37], which requires an epsilon to reduce computation error. If the epsilon is small, then self-intersections may occur, resulting in visible regions becoming occluded; if the epsilon is large, then missed intersections may occur, resulting in occluded regions becoming visible. Thus, without an appropriate epsilon, it is very difficult to compute the correct visibility between two close faces, as shown in Figure 2h. In practice, a uniform grid data [23] or a hierarchical tree structure [24–26] is built to avoid needless intersection checking. However, when axis-aligned primitives exist, a greater number of intersection tests must be conducted [37].

In the distance-based comparison, given a viewpoint or a projection plane, among all points on a 3D surface that can be projected into the same pixel, the closest point to the viewpoint or the projection plane is visible. However, projecting all faces of a complex mesh and comparing the distance at each pixel [28] is a brute-force choice that requires hundreds of samplings. In previous studies [14,27–29], the projection plane of each view was tessellated with large grid cells to reduce samplings. Clearly, the visibility results based on this method are sensitive to the size of a grid cell. In previous studies [14,16,31–33], the authors applied the shadow-mapping algorithm [38–40] with the aid of the well-known Z-buffer. The distance of a given test point to the viewpoint is computed first, and then a comparison is made between the computed distance and the corresponding distance stored in the shadow map; if the computed distance is smaller, then the point is visible; otherwise, the point is occluded in shadow. In Reference [32], an empirical epsilon was added to the shadow map to reduce numerical error. However, the regular shadow mapping [38] suffers from the shadow acne artifact, which refers to erroneous self-shadowing [39]. Sampling the Z-buffer through the standard depth bias approach yields insufficient precision [41] because a constant depth bias does not work for every part of a geometry [40]. A small bias produces self-occlusion, and a large bias means missing occlusion. Thus, this Z-buffer-based approach is still limited by an uncontrollable bias. Without an appropriate bias, the correct visibility in Figure 2h is also in danger.

In this study, we propose a high-precision graphics pipeline-based visibility classification (GPVC) method to solve the visibility problem in texture reconstruction. The best advantage offered by GPVC is that visibility can be computed without introducing an improper bias. Our method consists of two

stages. First, we design a shader-based rendering to generate each view's initial visibility map (IVM) and record visible primitives in a shader storage buffer object. Three spatial properties of IVMs are excavated to form our visibility classification mechanism. Second, we propose two algorithms, namely, lazy-projection coverage correction (LPCC) and hierarchical iterative vertex-edge-region sampling (HIVERS), to classify the visible primitives into fully visible or partially visible primitives. Compared with per-primitive checking, the advantage of LPCC is that LPCC only handles the visible primitives that are stored. Another breakthrough provided by LPCC is that we take the polygon rasterization principle and rasterization error into account, which has not been considered in other fragment-based visibility tests. HIVERS is proposed to identify a primitive's visibility on an arbitrary surface (manifold or nonmanifold) with fewer samplings.

The remainder of this paper is organized as follows. Section 2 describes the proposed method. Section 3 presents the implementation details in the graphics pipeline. Experimental results are presented in Section 4. Improvement of the proposed method is discussed with related studies in Section 5. Finally, Section 6 presents conclusions.

2. Graphics Pipeline-Based Visibility Classification

In this section, we provide the details of GPVC. First, IVMs are generated in the fixed graphics pipeline. Second, visibility classification is executed in the parallel computation stage. The framework of GPVC is depicted in Figure 4.

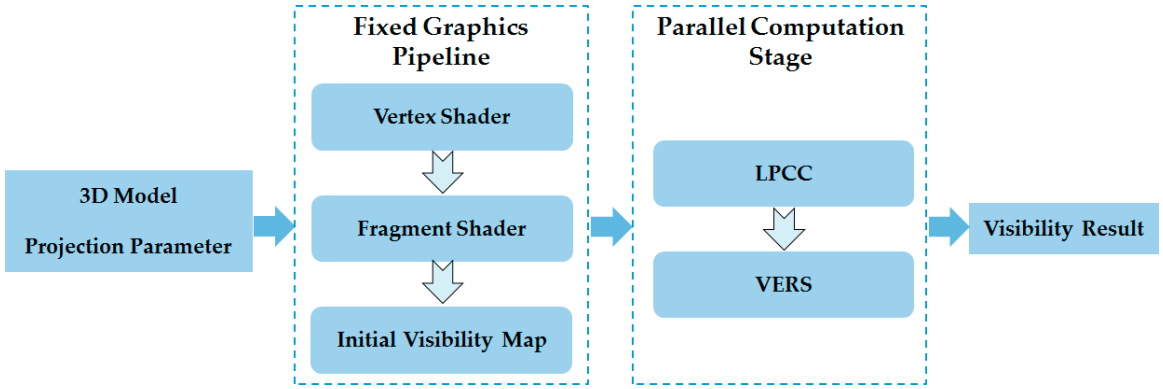


Figure 4. The framework of GPVC.

2.1. IVM Generation

Unlike the Z-buffer-based methods used in [14,16,31–33], in which the shadow maps store the closest depth values, IVMs of GPVC store visible faces' indexes as pixels. GPVC draws inspiration from digital camera photography, and we use two techniques to ensure that IVMs record the visible pixels coincident with digital images. First, in the XY plane, we design a shader-based rendering to achieve the locational coherence of corresponding pixels. The implementation mechanism of the shader-based rendering is described in Section 3. Second, in the Z-direction, we use the reversed projection [41] to avoid Z-fighting [42,43] and improve the precision of the Z-buffer. We explain the generation of IVMs according to the rasterization principle and present three spatial properties of IVMs.

2.1.1. Z-Buffer Precision Improvement

The default perspective projection matrix P in OpenGL is symmetric, which is defined as

$$P = \begin{bmatrix} p_x & 0 & 0 & 0 \\ 0 & p_y & 0 & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -\frac{2NF}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (1)$$

Thus, the nonlinear mapping relationship between the z-component of the eye coordinate z_e and the $depth$ stored in the Z-buffer can be expressed as

$$\begin{cases} z_n = \frac{F+N}{F-N} - \frac{2FN}{(F-N)(-z_e)} \\ depth = s * z_n + b \end{cases} \tag{2}$$

In Formula (2), $s = \frac{d_f - d_n}{2}$ and $b = \frac{d_f + d_n}{2}$ when z_n is within $[-1,\ 1]$, or $s = d_f - d_n$ and $b = d_n$ when z_n is within $[0,\ 1]$, where d_f is the depth value of the far plane F and d_n is the depth value of the near plane N.

According to Formula (2), the z-component of the normalized device coordinate z_n will be distributed in the symmetrical interval $[-1,\ 1]$; thus, $depth$ has decreasing precision as points move away from a viewpoint (as shown in Figure 5a). To avoid this drawback, we use the asymmetric projection matrix P_r in DirectX instead of the default configuration in OpenGL.

$$P_r = \begin{bmatrix} p_x & 0 & 0 & 0 \\ 0 & p_y & 0 & 0 \\ 0 & 0 & -\frac{F}{N-F} - 1 & -\frac{NF}{N-F} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3}$$

Therefore, we obtain

$$z_n = \frac{N}{N-F} + \frac{FN}{(F-N)(-eye_z)} \tag{4}$$

In this manner, z_n is distributed over the asymmetric interval $[0,\ 1]$, in which the near plane N is mapped to 1, and the far plane F is mapped to 0. To distribute the effective depths in the range with higher precision, i.e., close to 0 (as shown in Figure 5b), we design a "zero near plane" configuration; we compute N and F according to a 3D surface's bounding box and a view's projection parameter (as shown in Figure 6) and then pull N as close to the viewpoint as possible. Formula (5) describes the mapping relationship of our implemented Z-buffer.

$$\lim_{N \to 0} z_n = \frac{N}{-eye_z} \tag{5}$$

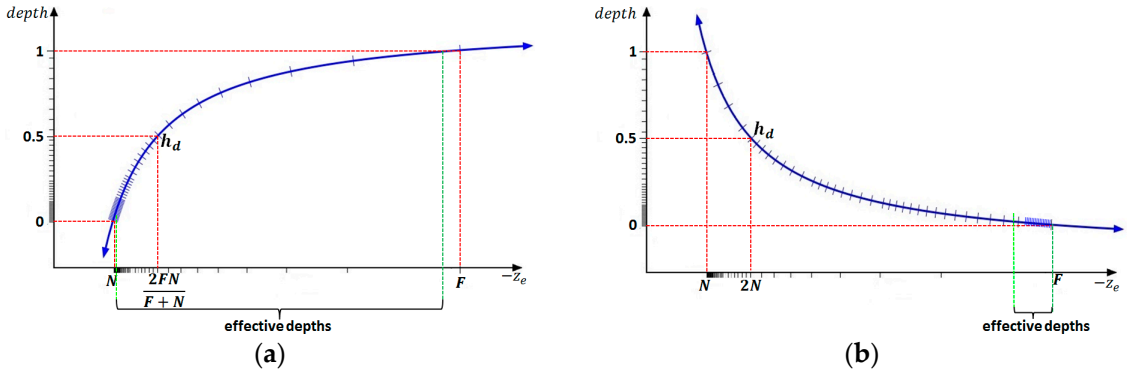


Figure 5. Depth distribution in the Z-buffer. (**a**) Depth distribution in the default Z-buffer; and (**b**) depth distribution in the reversed Z-buffer.

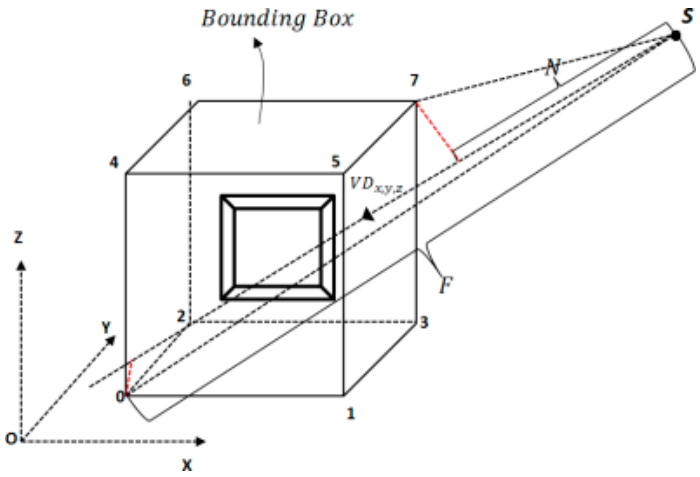


Figure 6. Computing the near and far planes according to a 3D surface's bounding box and a view's projection parameter. S: viewpoint; $VD_{x,y,z}$: normalized view direction. The near and far planes are equal to the projection length along the view direction.

2.1.2. IVM Generation and Spatial Properties

Rasterization in CG converts vectorial geometries to discrete fragments. An efficient and simple algorithm used in rasterization is the linear edge function (LEF) [44]. Given a triangle t whose vertices are in the counterclockwise orientation and a fragment f to be tested, LEF is defined as

$$lef_{(v_i,v_j)}(f) = (f_x - v_{ix}) \times (v_{jy} - v_{iy}) - (f_y - v_{iy}) \times (v_{jx} - v_{ix}) \tag{6}$$

- $lef_{(v_i,v_j)}(f) < 0$ if the center of f is to the left of one edge;
- $lef_{(v_i,v_j)}(f) = 0$ if the center of f is on one edge;
- $lef_{(v_i,v_j)}(f) > 0$ if the center of f is to the right of one edge.

where (v_i, v_j) denotes the edge formed by vertices v_i, v_j, (f_x, f_y) is the center of f. Thus, if $lef_{(v_i,v_j)}(f) < 0$, $(i,j)\,\epsilon\,\{(0,1),(1,2),(2,0)\}$, then f is an inner fragment of t (as shown in Figure 7). The fragment value v_f of f is computed as

$$\begin{cases} \lambda_0 = \frac{\frac{1}{2} lef_{(v_1,v_2)}(f)}{a} \\ \lambda_1 = \frac{\frac{1}{2} lef_{(v_2,v_0)}(f)}{a} \\ \lambda_2 = \frac{\frac{1}{2} lef_{(v_0,v_1)}(f)}{a} \\ \lambda_0 + \lambda_1 + \lambda_2 = 1 \end{cases} \tag{7}$$

$$v_f = \frac{\frac{\lambda_0 f_0}{w_0} + \frac{\lambda_1 f_1}{w_1} + \frac{\lambda_2 f_2}{w_2}}{\frac{\lambda_0}{w_0} + \frac{\lambda_1}{w_1} + \frac{\lambda_2}{w_2}} \tag{8}$$

In Formulas (7) and (8), a represents the signed area of t; $\lambda_0, \lambda_1, \lambda_2$ are the barycentric coordinates of f; f_0, f_1, f_2 are the attribute values associated with the three vertices of t; and w_0, w_1, w_2 are the clip coordinates.

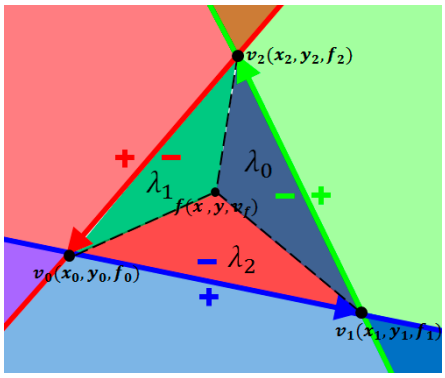


Figure 7. LEF-based triangle rasterization.

After rasterization, visible fragments are written to target buffers if early fragment tests are enabled [45]. In other words, visible parts of 3D surfaces are projected as those visible fragments. We use an item buffer to store the visible primitives' indexes as pixels. In this manner, the continuous visibility issue in 3D space is converted to a discrete visibility issue in 2D space [22,34]. IVMs formed by the visible fragments have three spatial properties, which is the foundation of our visibility classification.

Handiness: Only visible primitives are recorded, and fully occluded primitives are discarded.
Coherence: Visibility property of a face in 3D space remains unchanged in IVMs.
Continuity: Fragments' values within a fully visible primitive are uniform; otherwise, this primitive is partially visible (as shown in Figure 8).

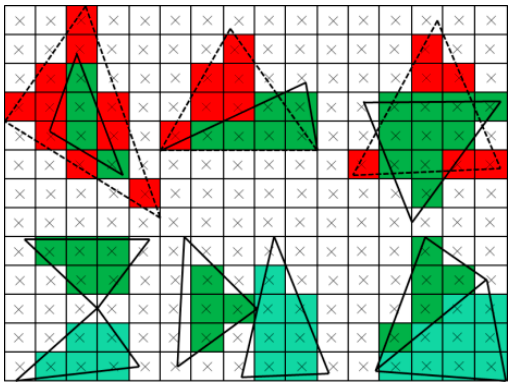


Figure 8. The continuity of IVMs. The fully visible primitives are drawn with solid lines and filled with green pixels; the partially visible ones are drawn with dotted lines and filled with red pixels.

2.2. *Visibility Classification*

Visibility classification based on IVMs marks a primitive with a fully visible or partially visible label, which represents the visibility issue in 2D space. The 2D visibility computation has been treated as hundreds of polygon-polygon intersection tests in [9,10]. Instead, based on the spatial properties of IVMs, lazy-projection coverage correction (LPCC), and hierarchical iterative vertex-edge-region sampling (HIVERS) are proposed to improve visibility computation.

2.2.1. Lazy-Projection Coverage Correction

(1) Lazy Projection

The backface culling and frustum culling are the two classical strategies that are widely used to reduce needless visibility tests. After the processing of these methods, remaining faces of a 3D surface in the viewshed consist of fully visible faces F, partially visible faces P and fully occluded faces

O. The related methods used in texturing need to handle each face's visibility in $F \cup P \cup O$. Instead, lazy projection is proposed to avoid addressing the faces' visibilities in O. Based on the fact that fully occluded faces are automatically discarded by the depth test, we use a shader storage buffer object, which is a continuous array, to record visible faces' indexes at corresponding subscripts (as shown in Figure 9). Given a visible face stored in the array, we project the face onto IVMs to obtain its initial projection coverage (IPC). In this manner, we only process the faces in $F \cup P$.

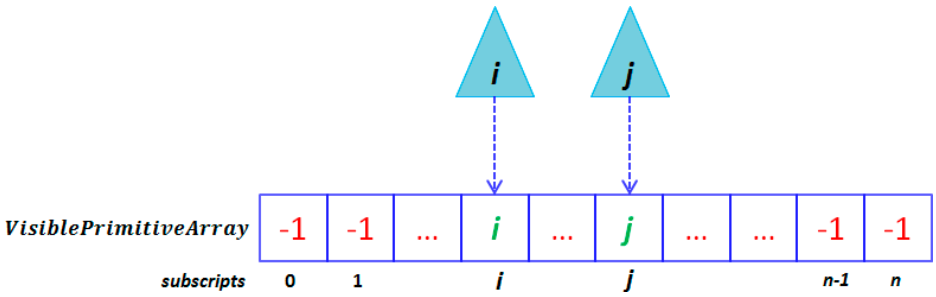


Figure 9. Recording visible faces' indexes in a shader storage buffer object.

(2) Projection Coverage Correction

After lazy projection, visibility judgment for a face only needs to apply the IVMs' continuity. However, due to the following two concerns, we should provide some correction to a face's IPC during visibility classification. First, the fragments whose centers lie on an edge do not always belong to a face (as shown in Figure 10). Second, the rasterization of an edge contains errors (see Figure 11). These two details interfering with visibility computation are overlooked by the fragment-based visibility tests in References [14,16,31–33]. Without considering these details, the continuity of a primitive will be destroyed.

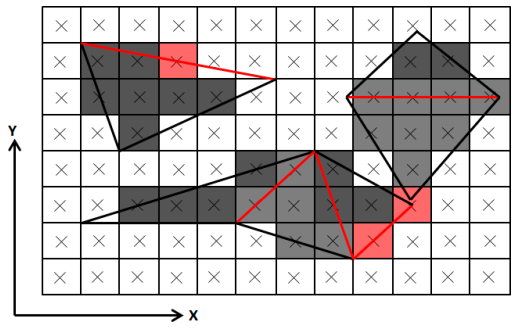


Figure 10. The top-left rule used to rasterize edges that cross fragments' centers. The edges in red are the ones that cross fragments' centers. The gray brightness of a fragment denotes the triangles to which the fragment belongs. The red fragments whose centers lie on the edges do not belong to the corresponding triangles because the edges are not left edges.

In CG, rendering adjacent primitives' shared edges twice is avoided by using rules to maintain consistency. The top-left rule (TLR) [46] is the criterion used in Direct3D to address this situation. Given a triangle whose vertices are in the counterclockwise orientation, if an edge (s, e) (s is start, and e is end) is perfectly horizontal, and the x-coordinate of e is smaller than the x-coordinate of s, then (s, e) is a top edge; if the y-coordinate of e is smaller than the y-coordinate of s, then (s, e) is a left edge. If a fragment's center lies on a top edge, then the fragment belongs to the triangle that is below the top edge; if the center lies on a left edge, then the fragment belongs to the triangle that is to the right of the left edge (see Figure 10).

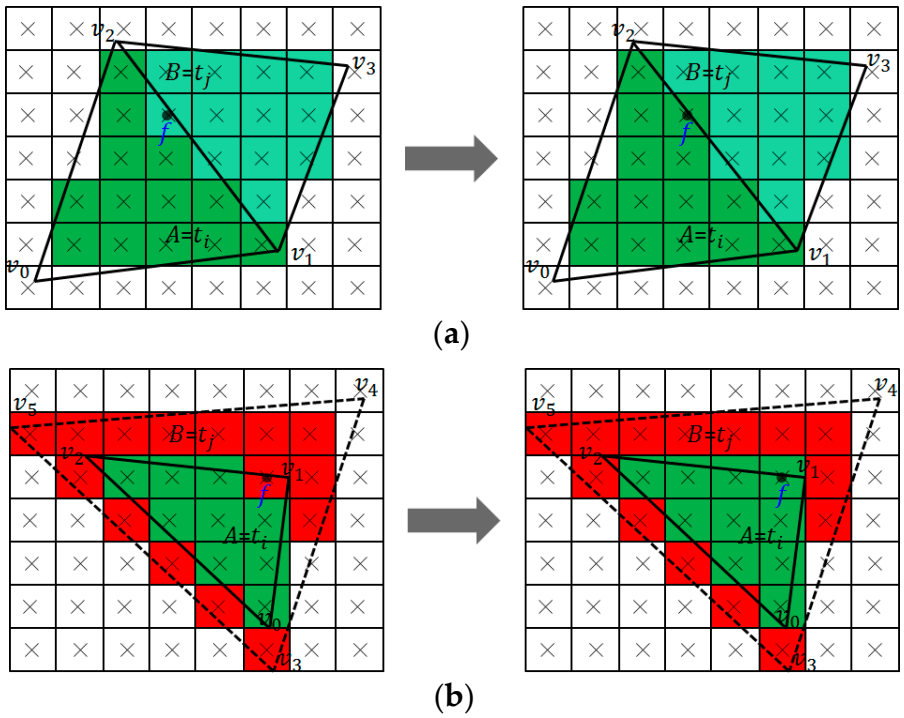


Figure 11. Rasterization error correction. (**a**) Rasterization error correction between two adjacent triangles. (**b**) Rasterization error correction between two overlapped triangles.

The rasterization error of edges often has little impact on the visual effect, but is nonnegligible in fragment-based visibility tests. As depicted in Figure 11, the rasterization error of edges indicates that a fragment f within triangle A is rasterized incorrectly as one of triangle B. In our study, we find that the pixel offset caused by the rasterization error remains within $(0, \frac{\sqrt{2}}{2})$. Thus, we propose the following Algorithm 1. LPCC to recover the correct continuity of a visible primitive.

Algorithm 1. LPCC

Definition: Given a visible triangle t_i (i represents the index value of t_i), IPC_i represents the initial projection coverage of t_i, f is one fragment crossed by each edge of triangle t_i, v_f is the fragment value of fragment f, and d_i is the distance from the viewpoint to f's center in the plane of triangle t_i.

Function LPCC(t_i)**:**

project t_i and get IPC_i
foreach f crossed by each edge of t_i **do**
  if f belongs to t_i **and** $v_f \neq i$ **then**
    $j \leftarrow v_f$
    project t_j and get IPC_j
    if f does not belong to t_j **then**
      modify the value v_f of fragment f as i
    else if $d_i < d_j$ **then**
      modify the value v_f of fragment f as i
    end if
  end if
end foreach

Algorithm 1 consists of the following steps:

1. Use LEF and TLR to judge whether a given fragment f belongs to t_i. If not, judge next fragment; otherwise, go to step 2
2. Determine whether the value v_f of fragment f equals t_i's index value i. If yes, go to step 1; otherwise, go to step 3.
3. Determine whether f is an inner fragment of triangle t_j. If not, f is an exceptional fragment of triangle t_i, then modify the value v_f of fragment f as i (as shown in Figure 11a); otherwise, go to step 4.
4. Determine whether triangle t_i is in front of triangle t_j at f's center. If yes, f is an exceptional fragment of triangle t_i, then modify the value v_f of fragment f as i (as shown in Figure 11b); otherwise, go to step 1.

2.2.2. Hierarchical Iterative Vertex-Edge-Region Sampling

HIVERS aims to identify a primitive's visibility with fewer samplings. Based on the continuity of IVMs, if a primitive p is considered to be fully visible, then p must satisfy the following conditions.

Vertex visibility condition (VVC): the inner fragments corresponding to p's vertices maintain a consistent value (see Figure 12a).
Edge visibility condition (EVC): the inner fragments corresponding to p's edges maintain a consistent value (see Figure 12b).
Region visibility condition (RVC): the inner fragments within p's subregions maintain a consistent value (see Figure 12c).

The three conditions are necessary to define a fully visible primitive. In addition, according to the continuity of IVMs and the spatial structure of a manifold surface, we can make the following inference.

Inference 1: If a given primitive p of a manifold surface satisfies VVC and EVC, then this primitive is fully visible; otherwise, the primitive is partially visible. In other words, we only need to sample the inner fragments corresponding to f's vertices and edges in a manifold surface. We denote this inference as VES.

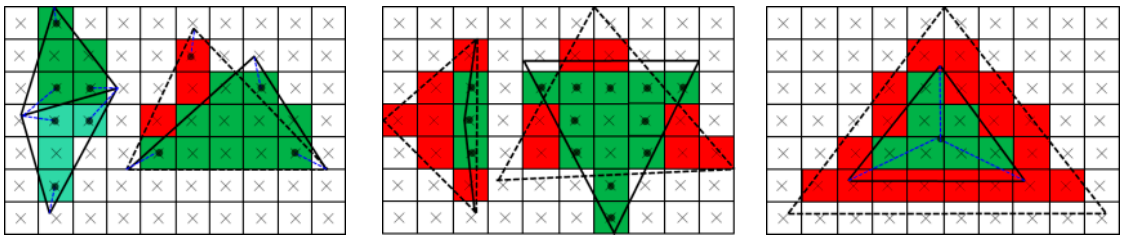


Figure 12. The description of VVC, EVC and RVC. (**a**) VVC: the fully visible primitives satisfy VVC, while the partially visible primitives do not. (**b**) EVC: the fully visible primitives satisfy EVC, while the partially visible primitives do not. (**c**) RVC: the fully visible primitives satisfy RVC, while the partially visible primitives do not.

To be compatible with the occlusion situations in nonmanifold surfaces, we integrate *VVC*, *EVC* and *RVC* to form HIVERS. As depicted in Figure 13, at each level of HIVERS, the three conditions are checked in order; we call this process one iteration at one level. The detailed procedure of Algorithm 2. HIVERS is as follows.

Algorithm 2. HIVERS

Definition: Given a visible triangle t, B is the barycenter of t, fv represents t is fully visible and pv represents t is partially visible.
Function HIVERS(t):
$t \leftarrow fv$
if t does not satisfy VVC **or** EVC **then**
  $t \leftarrow pv$
  return
end if
if all fragments within t have been sampled **then**
  return
else do
  subdivide t into subregions $\{t_0, t_1, t_2\}$ acorrding to B
  use **HIVERS**(t_i) to address $\{t_0, t_1, t_2\}$ recursively
end if

Algorithm 2 consists of the following steps:

1. Determine whether triangle t satisfies VVC. If not, then t is partially visible; otherwise, go to step 2.
2. Determine whether t satisfies EVC. If not, then t is partially visible; otherwise, go to step 3.
3. Check whether all fragments within t have been sampled. If yes, t is fully visible; otherwise, go to step 4.
4. Subdivide t into subregions $\{t_0, t_1, t_2\}$ according to t's barycenter B, and address each subregion from step 1 to step 3. If any of the subregions is partially visible, then t is partially visible; if all of the subregions are fully visible, then t is fully visible.

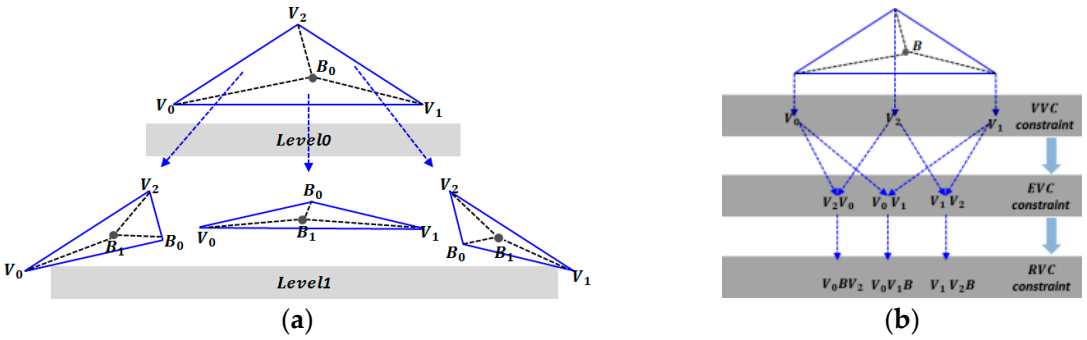


Figure 13. The framework of HIVERS. (**a**) The hierarchical structure of HIVERS. (**b**) One iteration at one level.

3. Implementation in Graphics Pipeline

The proposed framework is implemented in the graphics pipeline based on OpenGL. The implementation consists of IVM generation in the fixed graphics pipeline and visibility classification in the parallel computation stage.

3.1. IVM Generation in Fixed Graphics Pipeline

Zhang et al. [47] simulated digital photography by passing the projection parameter to the default graphics pipeline, which may produce the off-center projection dislocation. Instead, we present a shader-based rendering to generate IVMs. Given a 3D object and an undistorted digital image, the transformation from 3D to 2D in CV can be expressed as

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P_{3\times 4} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{9}$$

Formula (9) is equivalent to

$$\begin{cases} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - S\right) \\ x' = x/z \\ y' = y/z \\ u = f_x \times x' + c_x \\ v = f_y \times y' + c_y \end{cases} \tag{10}$$

In Formula (10), R is the rotation matrix, S is the viewpoint, f_x and f_y are the camera's focal length expressed in the pixel unit, and (c_x, c_y) is the principal point.

The transformation from 3D to 2D in CG can be expressed as

$$\begin{cases} (x_e, y_e, z_e, w_e)^T = M \times (X, Y, Z, 1)^T \\ (x_c, y_c, z_c, w_c)^T = P \times (x_e, y_e, z_e, w_e)^T \\ (x_n, y_n, z_n)^T = \left(\frac{x_c}{w_c}, \frac{f \times y_c}{w_c}, \frac{z_c}{w_c}\right)^T \\ (x_w, y_w, z_w)^T = \begin{pmatrix} (x_n + 1)\frac{ViewportWidht}{2} + O_x \\ (y_n + 1)\frac{ViewportHeight}{2} + O_y \\ s \times z_n + b \end{pmatrix} \end{cases} \tag{11}$$

In Formula (11), M is the matrix that transforms the object coordinate $(X, Y, Z, 1)^T$ into the eye coordinate $(x_e, y_e, z_e, w_e)^T$; P is the matrix that transforms the eye coordinate into the clip coordinate $(x_c, y_c, z_c, w_c)^T$; $(x_n, y_n, z_n)^T$ is the normalized device coordinate, where $f = 1$ when the viewport's origin is lower-left and $f = -1$ when the viewport's origin is upper-left; $(x_w, y_w, z_w)^T$ is the window coordinate; (O_x, O_y) represents the viewport's lower-left corner.

To relate Formula (10) with Formula (11), our shader-based rendering is designed as follows. First, according to Formula (10), we compute one vertex's projection $\begin{bmatrix} u \\ v \end{bmatrix}$ in the vertex shader. Second, we let z and $\begin{bmatrix} u \\ v \end{bmatrix}$ in Formula (10) be w_c and $\begin{bmatrix} x_w \\ y_w \end{bmatrix}$ in Formula (11), respectively; we then substitute z and $\begin{bmatrix} u \\ v \end{bmatrix}$ into Formula (11) to compute $(x_c, y_c, z_c, w_c)^T$ as

$$\begin{cases} x_c = 2 \times \frac{u - O_x}{ImageWidth} - 1 \\ y_c = 2 \times \frac{v - O_y}{ImageHeight} - 1 \\ z_c = \frac{N}{N-F} + \frac{FN}{(F-N)\times z} \\ w_c = 1.0 \end{cases} \tag{12}$$

In Formula (12), w_c is normalized to improve rendering efficiency, z_c is computed according to Formula (4) to improve the Z-buffer's precision, and viewport size is replaced with image size. Finally, in the fragment shader, we use an off-screen texture to store visible fragments and a shader

storage buffer object *VisiblePrimitiveArray* to record visible primitives' indexes at the corresponding subscripts (as shown in Figure 9).

3.2. Visibility Classification in Parallel Computation Stage

The parallel computation stage in OpenGL exists as the compute shader. Compared with OpenCL, the compute shader is a lightweight framework with less complexity and interoperability with shared resources. This stage is completely independent of any predefined graphics-oriented semantics. The compute shader can read and modify OpenGL buffers and images. The tool is, therefore, able to modify data that are the input or output of the fixed graphics pipeline. Due to the independence between the compute shader and the fixed graphics pipeline, we must consider synchronization. We disable the compute shader after completing visibility classification in the current frame and recover it after IVM generation in the next frame.

The following data are passed into our compute shader. (1) Multiview's IVMs. One view's IVM is passed as an "image2D" object. (2) *VisiblePrimitiveArray* and projection parameter. The projection parameter is stored as a uniform matrix. (3) 3D model data. Vertices and faces are stored in two shader storage buffer objects. (4) Face and vertex number. To classify visibility between multiple models, the face and vertex numbers of each model are stored in a uniform buffer object, which act as search indexes (as shown in Figure 14).

In the compute shader, our visibility classification proceeds as follows. Each visible primitive stored in *VisiblePrimitiveArray* is visited by an independently parallel thread. In each thread, our visibility classification is executed.

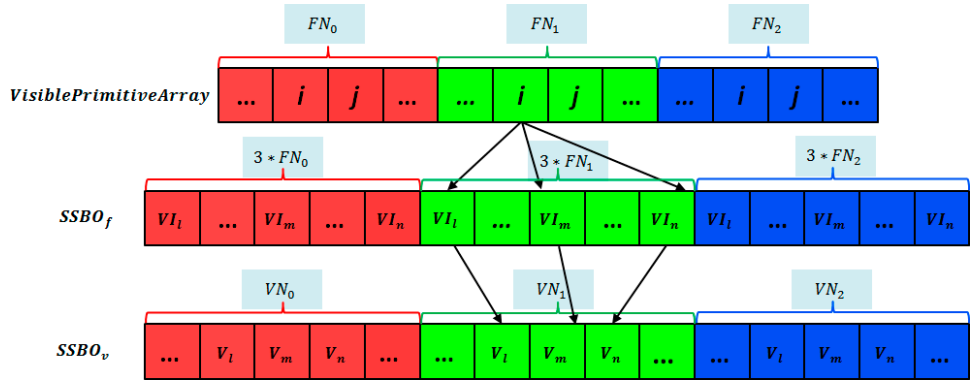


Figure 14. Visiting a specific face in multiple models. FN_* represents one model's face number, VN_* represents one model's vertex number, VI_* represents one face's vertices' indexes, V_* represents one vertex's coordinate, $SSBO_f$ is the shader storage buffer object storing faces and $SSBO_v$ is the shader storage buffer object storing vertices.

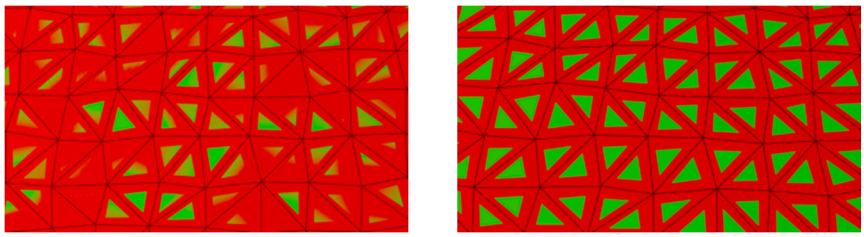
4. Experiments

The following tests are conducted in our experiments. With respect to accuracy, we use related methods to detect visibility for three synthetic models. The visibility of the three synthetic models is known from a predefined viewpoint. With respect to efficiency, we compare the related methods' number of processed faces, sampling count, and time consumption. With respect to texture reconstruction, comparisons between GPVC and other available libraries and software are made. The key steps of GPVC are also presented in our experiments. The experimental platform is Win8.1 x64 OS, with 16 GB computer memory, I7 CPU (2.5 GHz, 4 cores, and 8 threads) and a GTX 860 M video card.

4.1. IVM Generation

4.1.1. Z-Buffer Precision Improvement

Figure 15 shows the visual effect when rendering very close faces based on different Z-buffer configurations. The default Z-buffer with the symmetric projection results in the Z-fighting problem due to the infinite precision. The reversed Z-buffer with the asymmetric projection used in GPVC exhibits good performance to avoid the problem, which ensures the continuity of IVMs. The format of the Z-buffer used in this paper is set to a 32-bit floating point.



(**a**) (**b**)

Figure 15. Comparison of the visual effect between the default Z-buffer and the reversed Z-buffer. (**a**) Rendering effect of the default Z-buffer. (**b**) Rendering effect of the reversed Z-buffer used in GPVC.

4.1.2. Shader-Based Rendering

In this section, we demonstrate the significance of our shader-based rendering in the simulation of digital photography. To present the feature boundaries clearly, we fuse the five views, rendering images with the corresponding digital images, as shown in Figure 16. To prove the validity of our approach, we compare our shader-based rendering with the default rendering in detail. As shown in Figure 17, the default rendering cannot accurately simulate the off-center perspective projection, resulting in distinct object dislocations relative to the digital images. Compared with the default approach, the shader-based rendering can maintain the corresponding objects spatially aligned.

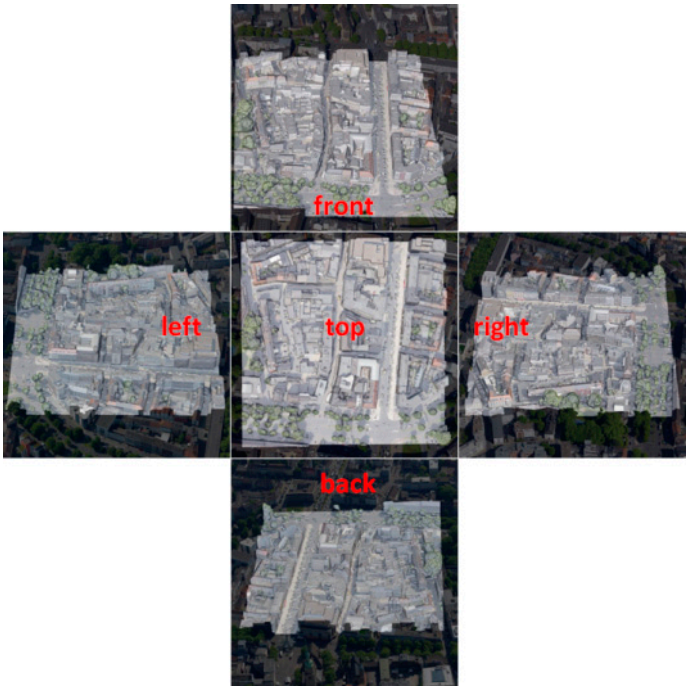
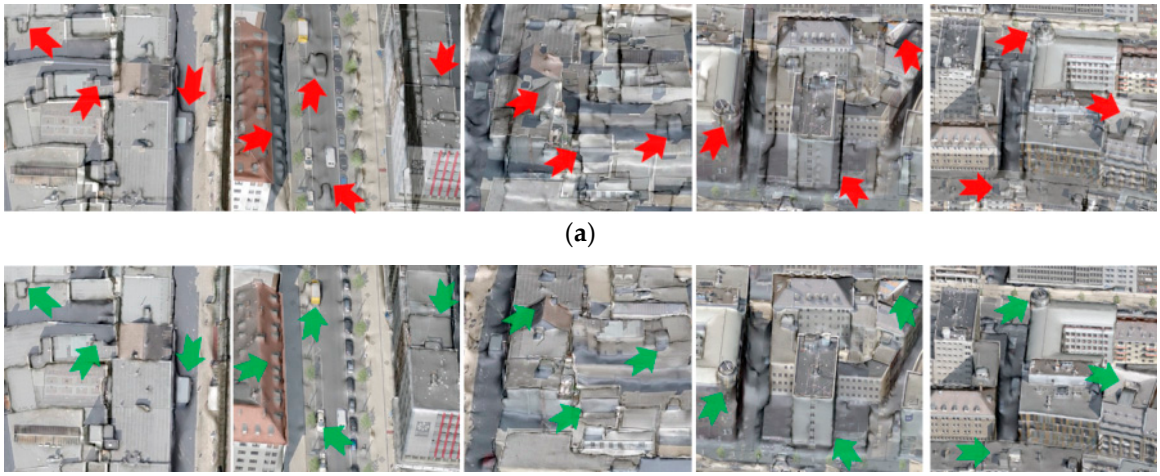


Figure 16. The fusion of rendering images and oblique digital images from five views.



(**a**)

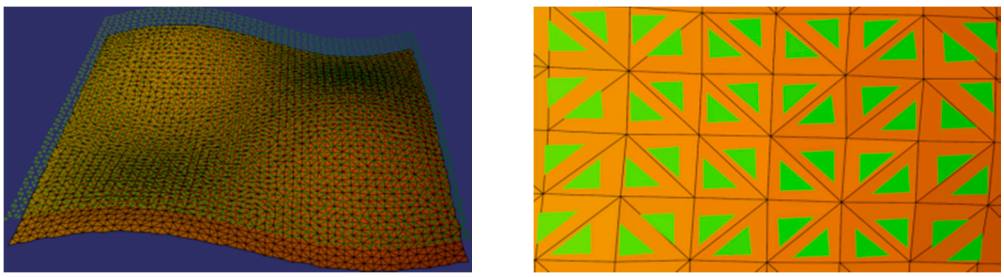
(**b**)

Figure 17. Detailed contrasts between the shader-based rendering and the default rendering. (**a**) Detailed features in the default rendering images. (**b**) Detailed features in the shader-based rendering images.

4.2. Visibility Classification

4.2.1. Visibility Accuracy Statistics

We design three synthetic models A, B, C to examine the accuracy of related methods. As shown in Figure 18a, each model consists of two uneven terrain layers (upper layer U and lower layer L); each layer is a triangulation network containing 4082 faces. For A, the vertical distance between U and L is 0.01 m; for B, the vertical distance is 1 m; and for C, the vertical distance is 100 m. Given a predefined top view (the viewpoint is 6100 m from the center of a synthetic model), the triangles of U are fully visible, and the triangles of L are partially occluded (as shown in Figure 18b). Based on these features, we present related methods' performances.



(**a**) (**b**)

Figure 18. Synthetic models. (**a**) Global view. (**b**) Top view in detail.

(1) Shadow-Mapping Algorithm

The nonlinear depth distribution in the Z-buffer is highly sensitive to the projection matrix, particularly to the near and far planes. Without the loss of generality, we use two projection configurations to present the performance of the shadow-mapping algorithm. Symmetric projection configuration (SPC): the symmetric projection matrix with a pair of adaptive planes. The adaptive planes are computed according to Figure 6. Asymmetric projection configuration (APC): the asymmetric projection matrix with a "zero near plane" and an adaptive far plane; this mechanism is recommended by our GPVC. In this design, given the predefined top viewpoint, the two projection configurations are both sufficient to avoid Z-fighting when rendering the synthetic models. Tables 1 and 2 show the accuracy statistics. The epsilon is used to determine whether a pixel within a face is

visible: if $|computedDepth - storedDepth| \leq epsilon$, then the pixel is visible; otherwise, the face is occluded. We attempt to enumerate the *epsilon* from 0.1 to 1×10^{-13} to prove the difficulty of choosing an appropriate bias. In each enumeration, the new epsilon is reduced to half of the original value. For simplicity, in the following tables, V represents the visible face number, O represents the occluded face number, and Ac represents the detection accuracy.

Table 1. Accuracy statistics of the shadow-mapping algorithm with SPC.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Epsilon	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**
0.00156~0.1	9603	1	50.01	0.003~0.1	9603	1	50.01	7.81×10^{-4}~0.1	4735	4869	99.30
7.81×10^{-4}	9592	12	50.00	0.00156	4718	4886	99.13	3.91×10^{-4}	4324	5280	95.02
3.91×10^{-4}	8331	1273	50.26	7.81×10^{-4}	4711	4893	99.05	1.95×10^{-4}	3585	6019	87.33
1.95×10^{-4}	7051	2553	49.86	3.91×10^{-4}	4140	5464	93.11	9.77×10^{-5}	3175	6429	83.06
9.77×10^{-5}	6272	3332	49.98	1.95×10^{-4}	3512	6092	86.57	4.88×10^{-5}	2934	6670	80.55
4.88×10^{-5}	5915	3689	50.11	9.77×10^{-5}	3136	6468	82.65	2.44×10^{-5}	2812	6792	79.28
2.44×10^{-5}	5750	3854	50.10	4.88×10^{-5}	2916	6688	80.36	1.22×10^{-5}	2747	6857	78.60
1.22×10^{-5}	2789	6815	79.04	2.44×10^{-5}	2813	6791	79.29	6.10×10^{-6}	2713	6891	78.25
6.10×10^{-6}	2765	6839	78.79	1.22×10^{-5}	2768	6836	78.82	3.05×10^{-6}	2682	6922	77.93
3.05×10^{-6}	2712	6892	78.24	6.10×10^{-6}	2746	6858	78.59	1.53×10^{-6}	1382	8222	64.39
1.53×10^{-6}	1374	8230	64.31	3.05×10^{-6}	2695	6909	78.06	7.63×10^{-7}	278	9326	52.89
7.63×10^{-7}	304	9300	53.17	1.53×10^{-6}	1285	8319	63.38	3.81×10^{-7}	37	9567	50.39
3.81×10^{-7}	45	9559	50.47	7.63×10^{-7}	287	9317	52.99	1.91×10^{-7}	7	9597	50.07
1.91×10^{-7}	10	9594	50.10	3.81×10^{-7}	49	9555	50.51	9.54×10^{-8}	2	9602	50.02
1×10^{-13}~9.54×10^{-8}	0	9604	50.00	1.91×10^{-7}	8	9596	50.08	1×10^{-13}~4.77×10^{-8}	0	9604	50.00
				1×10^{-13}~9.54×10^{-8}	0	9604	50.00				

Table 2. Accuracy statistics of the shadow-mapping algorithm with APC.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Epsilon	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**
1.16×10^{-11}~0.1	9604	0	50.00	1.16×10^{-11}~0.1	9604	0	50.00	1.16×10^{-11}~0.1	9604	0	50.00
1.82×10^{-13}~5.82×10^{-12}	9603	1	50.01	1.82×10^{-13}~5.82×10^{-12}	9603	1	50.01	5.82×10^{-12}	9603	1	50.01
								2.91×10^{-12}	9602	2	50.02
								1.82×10^{-13}~1.46×10^{-12}	4735	4869	99.30

(2) Ray-Tracing Algorithm

The ray-tracing algorithm used here is an open-source library [36], which is employed by Waechter et al. [15] with a sparse sampling implementation. Under the sparse sampling mode, only the three vertices of a triangle are checked; if one vertex is occluded, then the triangle is invisible. The sparse sampling does not work well for our synthetic models; thus, based on the library, we design a dynamic dense sampling (as shown in Figure 19). The dynamic dense sampling indicates that the sampling points within a triangle remain at the centers of the inner fragments of a triangle. Because all calculations involving ray tracing will introduce a bias, we need an *epsilon* to determine a checking point's visibility: If the distance between the checking point and the first intersection point is smaller than the *epsilon*, then the checking point is visible. We also enumerate the *epsilon* from 0.1 to 1×10^{-13} to present the detection accuracy with different biases. Tables 3 and 4 show the accuracy statistics.

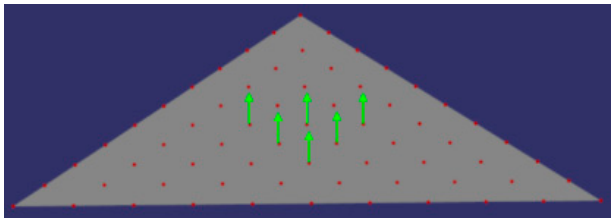


Figure 19. Ray tracing with the dynamic dense sampling.

Table 3. Accuracy statistics of ray tracing with the sparse sampling.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Epsilon	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**
9.77×10^{-5}~0.1	9604	0	50.00	1.95×10^{-4}~0.1	9604	0	50.00	1.95×10^{-4}~0.1	7514	2090	71.76
4.88×10^{-5}	9352	252	51.90	9.77×10^{-5}	9603	1	49.99	9.77×10^{-5}	7512	2092	71.74
2.44×10^{-5}	8261	1343	59.34	4.88×10^{-5}	9343	261	51.80	4.88×10^{-5}	7393	2211	72.14
1.22×10^{-5}	6813	2791	68.04	2.44×10^{-5}	8264	1340	59.37	2.44×10^{-5}	6742	2862	75.57
6.10×10^{-6}	5142	4462	76.36	1.22×10^{-5}	6816	2788	68.08	1.22×10^{-5}	5724	3880	79.97
3.05×10^{-6}	3971	5633	79.08	6.10×10^{-6}	5122	4482	76.16	6.10×10^{-6}	4379	5225	82.26
1.53×10^{-6}	3583	6021	79.79	3.05×10^{-6}	3981	5623	79.19	3.05×10^{-6}	3636	5968	82.03
7.63×10^{-7}	3439	6165	79.81	1.53×10^{-6}	3578	6026	79.74	1.53×10^{-6}	3413	6191	82.04
3.81×10^{-7}	3381	6223	79.56	7.63×10^{-7}	3428	6176	79.70	7.63×10^{-7}	3344	6260	82.15
1.91×10^{-7}	3363	6241	79.46	3.81×10^{-7}	3379	6225	79.54	3.81×10^{-7}	3328	6276	82.13
9.54×10^{-8}	3355	6249	79.46	1.91×10^{-7}	3358	6246	79.40	2.38×10^{-8}~1.91×10^{-7}	3326	6278	82.13
4.77×10^{-8}	3353	6251	79.48	9.54×10^{-8}	3349	6255	79.39				
				4.77×10^{-8}	3346	6258	79.40	1.82×10^{-13}~1.19×10^{-8}	3325	6279	82.12
1.82×10^{-13}~2.38×10^{-8}	3352	6252	79.49	2.38×10^{-8}	3345	6259	79.41				
				1.82×10^{-13}~1.19×10^{-8}	3344	6260	79.40				

Table 4. Accuracy statistics of ray tracing with the dynamic dense sampling.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Epsilon	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**	**Epsilon**	**V**	**O**	**Ac (%)**
0.0125~0.1	9604	0	50.00	1.95×10^{-4}~0.1	4802	4802	100.00	1.95×10^{-4}~0.1	4802	4802	100.00
1.95×10^{-4}~0.006	4802	4802	100.00	9.77×10^{-5}	4763	4841	99.59	9.77×10^{-5}	4753	4851	99.49
9.77×10^{-5}	4774	4830	99.70	4.88×10^{-5}	4233	5371	94.08	4.88×10^{-5}	4168	5436	93.40
4.88×10^{-5}	4342	5262	95.21	2.44×10^{-5}	3077	6527	82.04	2.44×10^{-5}	3190	6414	83.22
2.44×10^{-5}	3327	6277	84.64	1.22×10^{-5}	2080	7524	71.66	1.22×10^{-5}	2147	7457	72.36
1.22×10^{-5}	2346	7258	74.43	6.10×10^{-6}	1275	8329	63.28	6.10×10^{-6}	1151	8453	61.98
6.10×10^{-6}	1532	8072	65.95	3.05×10^{-6}	821	8783	58.55	3.05×10^{-6}	759	8845	57.90
3.05×10^{-6}	1052	8552	60.95	1.53×10^{-6}	655	8949	56.82	1.53×10^{-6}	669	8935	56.97
1.53×10^{-6}	873	8731	59.09	7.63×10^{-7}	592	9012	56.16	7.63×10^{-7}	653	8951	56.80
7.63×10^{-7}	807	8797	58.40	3.81×10^{-7}	562	9042	55.85	3.81×10^{-7}	649	8955	56.76
3.81×10^{-7}	772	8832	58.04	1.91×10^{-7}	549	9055	55.72				
1.91×10^{-7}	759	8845	57.90					1.82×10^{-13}~1.91×10^{-7}	648	8956	56.75
9.54×10^{-8}	753	8851	57.84	1.82×10^{-13}~9.54×10^{-8}	547	9057	55.70				
1.82×10^{-13}~4.77×10^{-8}	752	8852	57.83								

(3) GPVC

Visibility computation based on GPVC can be achieved without choosing an improper bias. The accuracy of GPVC reaches 100% after projection coverage correction. Tables 5 and 6 show the accuracy statistics.

Table 5. Accuracy statistics of GPVC before projection coverage correction.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Projection Configuration	**V**	**O**	**Ac (%)**	**Projection Configuration**	**V**	**O**	**Ac (%)**	**Projection Configuration**	**V**	**O**	**Ac (%)**
SPC	4736	4868	99.31	SPC	4717	4887	99.11	SPC	4734	4870	99.29
APC	4736	4868	99.31	APC	4717	4887	99.11	APC	4734	4870	99.29

Table 6. Accuracy statistics of GPVC after projection coverage correction.

Synthetic Model *A*				Synthetic Model *B*				Synthetic Model *C*			
Projection Configuration	**V**	**O**	**Ac (%)**	**Projection Configuration**	**V**	**O**	**Ac (%)**	**Projection Configuration**	**V**	**O**	**Ac (%)**
SPC	4802	4802	100.00	SPC	4802	4802	100.00	SPC	4802	4802	100.00
APC	4802	4802	100.00	APC	4802	4802	100.00	APC	4802	4802	100.00

4.2.2. Visibility Efficiency Statistics

We use a manifold mesh (611,055 faces) reconstructed from 43 views to present the efficiency of related methods. Tables 7–9 are the statistics pertaining to the number of processed faces, sampling count and time consumption. The number of processed faces represents the total number of faces after backface culling and frustum culling in 43 views. The sampling count represents the total sampling number during computing visibility for the processed faces.

Table 7. Efficiency statistics of the shadow-mapping algorithm.

Projection Configuration	Epsilon	Processed Face Number	Sampling Count	Time Consumption (s)
SPC	0.00001	14514510	186659322	20.54
SPC	0.0001	14514510	269039318	26.90
SPC	0.001	14514510	295725638	29.28
SPC	0.01	14514510	303479553	29.52
APC	0.00001	14514510	425847450	39.29
APC	0.0001	14514510	425847450	39.25
APC	0.001	14514510	425847450	39.96
APC	0.01	14514510	425847450	40.79

Table 8. Efficiency statistics of the ray-tracing algorithm.

Sampling Type	Epsilon	Processed Face Number	Sampling Count	Time Consumption (s)
Sparse Sampling	0.00001	14514510	34716374	261.43
Sparse Sampling	0.0001	14514510	34722053	255.46
Sparse Sampling	0.001	14514510	34928523	202.51
Sparse Sampling	0.01	14514510	37680002	110.45
Dense Sampling	0.00001	14514510	263249353	1666.48
Dense Sampling	0.0001	14514510	263421883	1622.45
Dense Sampling	0.001	14514510	265066824	1248.05
Dense Sampling	0.01	14514510	295846279	706.68

Table 9. Efficiency statistics of GPVC.

Sampling Type	Processed Face Number	Sampling Count	Time Consumption (s)	
			CPU	GPU
Dense Sampling	10230279	272005618	30.14	0.16
HIVERS	10230279	246995434	28.47	0.14
VES	10230279	194294149	24.73	0.11

4.3. Texturing Contrast

In this section, we apply related methods to multiview-based texture reconstruction and compare the textured models generated by available libraries and software. We use two datasets to prove our method's superiority. The first dataset pertains to the campus of Wuhan University (as shown in Figure 20a), and the second is provided by ISPRS and EuroSDR (as shown in Figure 20b). First, we use the famous business software ContextCapture to reconstruct the triangular meshes of the two datasets. Next, we use related methods to compute the visibility of each view. Finally, we use graph cuts to select the optimal fully visible textures.

(**a**) (**b**)

Figure 20. The textured models based on GPVC. (**a**) The textured model of Wuhan University. (**b**) The textured model reconstructed from the ISPRS and EuroSDR datasets.

5. Discussion

In this study, a graphics pipeline-based visibility classification (GPVC) method is proposed to improve the visibility handling involving multiview-based texture reconstruction. Experimental results demonstrate that our method outperforms related methods and available tools.

During IVM generation, we use the reversed Z-buffer and the shader-based rendering to ensure the accuracy of IVMs. The reversed Z-buffer designed in this paper can avoid Z-fighting when rendering very close faces (as shown in Figure 15), which guarantees the continuity of IVMs. Our shader-based rendering is proposed to achieve the locational coherence of corresponding objects between rendering images and digital images (as shown in Figure 17). The default rendering used in [47] fails to maintain the corresponding objects spatially aligned (as shown in Figure 17a). In our study, given an 8176×6132 UAV image whose principal point is at $(4035.1894, 2975.0360)$, the off-center projection dislocation can reach more than 50 pixels in the x-axis direction and 90 pixels in the y-axis direction. The dislocation causes some visible primitives to exit the viewshed or that some invisible ones enter the viewshed.

Compared with the shadow-mapping [14,16,31–33] and ray-tracing [15,35] methods used in texturing, the main contribution of GPVC is that visibility computation is performed without a bias. To demonstrate the shortcomings of the bias-based methods, we design three models with known visibility from a predefined viewpoint. As shown in Table 1, the shadow-mapping algorithm with the symmetric projection configuration only works well when the epsilon remains in the ranges specified by enumeration (3.05×10^{-6}~1.22×10^{-5} for synthetic model A, 3.91×10^{-4}~0.00156 for synthetic model B, 3.91×10^{-4}~0.1 for synthetic model C). As the two layers of the synthetic models move closer, the optimal epsilon becomes increasingly difficult to specify. The reason is that the computed depths and the depths stored in the Z-buffer tend to be approximate, requiring a smaller epsilon to distinguish the correct visibility. If the selected epsilon increases, certain occluded faces are identified as visible; if the selected epsilon decreases, certain visible faces are identified as occluded. This problem becomes more serious when using the shadow-mapping algorithm with the asymmetric projection configuration, as shown in Table 2, only when the epsilon is in the range (1×10^{-13}~4.77×10^{-8}) is the performance satisfactory for synthetic model C; using other epsilons misses many occlusions. The reason is that the computed and stored depths are distributed over the range with more precision (very close to 0.0). Only a very small epsilon can work, even if the two layers of synthetic model C are distant. In this respect, the shadow-mapping algorithm with the asymmetric projection configuration is unfit to compute visibility in texture reconstruction. As shown in Tables 3 and 4, the dense ray tracing outperforms the sparse ray tracing for our synthetic models. The reason is that our three synthetic models are not manifold, and the sparse ray tracing fails to identify certain occluded faces. Within certain biases (1.95×10^{-4}~0.006 for synthetic model A, 1.95×10^{-4}~0.1 for synthetic models B and C), the accuracy of the dense ray tracing can reach 100%. However, in the dense ray-tracing mode, increasingly more faces are labeled as occluded if the epsilon decreases outside of the abovementioned ranges. This effect results from the ability of an increased number of tracings to generate more self-intersections. Compared with that of the two methods mentioned above, our GPVC's performance is robust for the three models due to the use of the continuity of IVMs. After projection coverage correction, the accuracy of GPVC can reach 100% (as shown in Tables 5 and 6). We also apply our

projection coverage correction to the shadow-mapping algorithm and obtain the following results. With the aid of our projection coverage correction, the accuracy of the shadow-mapping algorithm with the symmetric projection configuration (epsilon = 0.00156 for synthetic model *B*, epsilon remains in 7.81×10^{-4}~0.1 for synthetic model *C*) can reach 100%; the accuracy of the shadow-mapping algorithm with the asymmetric projection configuration (epsilon remains in 1.82×10^{-13}~1.46×10^{-12} for synthetic model *C*) can also reach 100%.

To demonstrate the improvement of GPVC in texturing, we present many comparisons pertaining to textured models generated by related methods, open-source libraries (MVS-Texturing) and business software (ContextCapture). As shown in Figures 21–25, the detailed areas are the three parts of one textured model reconstructed from the ISPRS and EuroSDR datasets. Unlike our synthetic models (in one synthetic model, the distance between two layers is fixed), the urban models reconstructed from multiview images have dense faces and complex geometry structures. In the case of the shadow-mapping algorithm and ray-tracing methods, some areas' visibility can be computed correctly, and textures are mapped well by using a given bias. However, this bias is not robust for every part of the complex model. Thus, many areas are mapped with inaccurate textures (as depicted in the red boxes in Figures 21–24). The textured results based on the shadow-mapping algorithm with the symmetric projection configuration are highly sensitive to the selected bias because the depths are distributed in [0, 1] dispersedly; thus, a small change in the bias can generate different visibility, producing different textures. As shown in Figure 21, the texturing gets better as the used bias increases because more visible textures are correctly detected. While the textured results based on the shadow-mapping algorithm with the asymmetric projection configuration appear to be insensitive to the bias because the depths are distributed at approximately 0.0 densely; thus, a large change in the epsilon has little effect on the visibility results, producing nearly unchanged textures. As shown in Figure 22, because the four used biases are not sufficiently small to identify the occluded faces, the occluded textures are mapped to the corresponding areas. The left screenshot in Figure 22 is textured with the correct color because the corresponding area is a nearly flat structure without complex occluders, thus the identified visible textures are mapped to the correct locations. When comparing the texturing in Figure 22 with the one in Figure 25a, it can be seen that the texturing based on the shadow-mapping algorithm with the asymmetric projection configuration is nearly as same as the one without visibility tests. The reason is that almost all of the faces are detected as visible by the shadow-mapping algorithm with the asymmetric projection configuration, which is nearly equivalent to skipping visibility tests. Compared with that afforded by the sparse ray tracing, the improvement of the dense ray tracing is not as obvious. The reason is that for a manifold surface, most faces' correct visibility can be determined by checking three vertices with an appropriate bias. In some cases, the dense ray tracing is worse than the sparse ray tracing (as shown in Figures 23c and 24c). The reason is that a face's visibility tends to be incorrect as sampling time grows without an appropriate bias. If the bias is small, then self-occlusions occur, and visible faces become invisible, leaving holes in the textured models (as shown in Figures 23a–c and 24e–g); if the bias is large, then occluded faces become visible, leaving incorrect textures on the textured models. Figure 25 shows the performance of our method. Without visibility tests, there are serious deviations contained in the right two screenshots (see Figure 25a) due to the error mapping of occluded textures. Before projection coverage correction, the textured results contain some artifacts because the optimal textures are not identified as visible, and some other inadequate visible textures have a chance to be mapped (as shown in Figure 25b). One concern is that given a manifold mesh, without projection coverage correction, the textured areas generated by HIVERS and VES are the same. The reason is that the rasterization error only occurs near the edges of polygons; if a visible triangle face contains exceptional fragments, it will be detected as invisible by HIVERS and VES. After projection coverage correction, the artifacts disappear. The only difference of the texture areas within the green boxes in Figure 25c,d results from the corresponding area of the reconstructed mesh containing nonmanifold structures (as shown in Figure 26, the nonmanifold parts are trees that are not reconstructed completely). Thus, after projection coverage correction, the target faces' visibility

computed by HIVERS and VES is not the same. While the other two textured areas in Figure 25 are strictly manifold, the textured results of HIVERS and VES are the same. This finding proves the validity of Inference 1.

Figure 21. Textured areas based on the shadow-mapping algorithm with SPC. (**a**) Textured areas when epsilon = 0.00001. (**b**) Textured areas when epsilon = 0.0001. (**c**) Textured areas when epsilon = 0.001. (**d**) Textured areas when epsilon = 0.01.

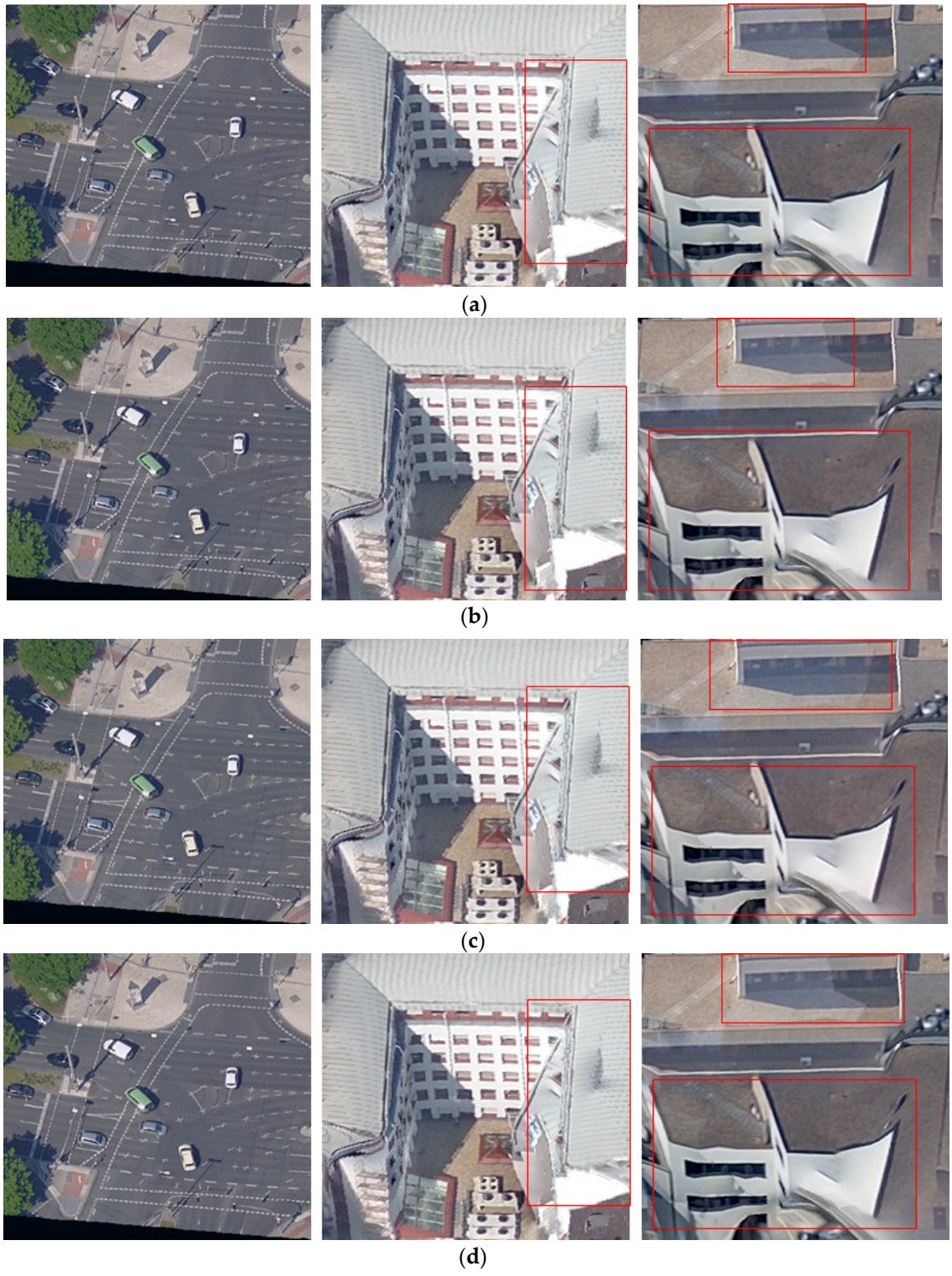


Figure 22. Textured areas based on the shadow-mapping algorithm with APC. (**a**) Textured areas when epsilon = 0.00001. (**b**) Textured areas when epsilon = 0.0001. (**c**) Textured areas when epsilon = 0.001. (**d**) Textured areas when epsilon = 0.01.

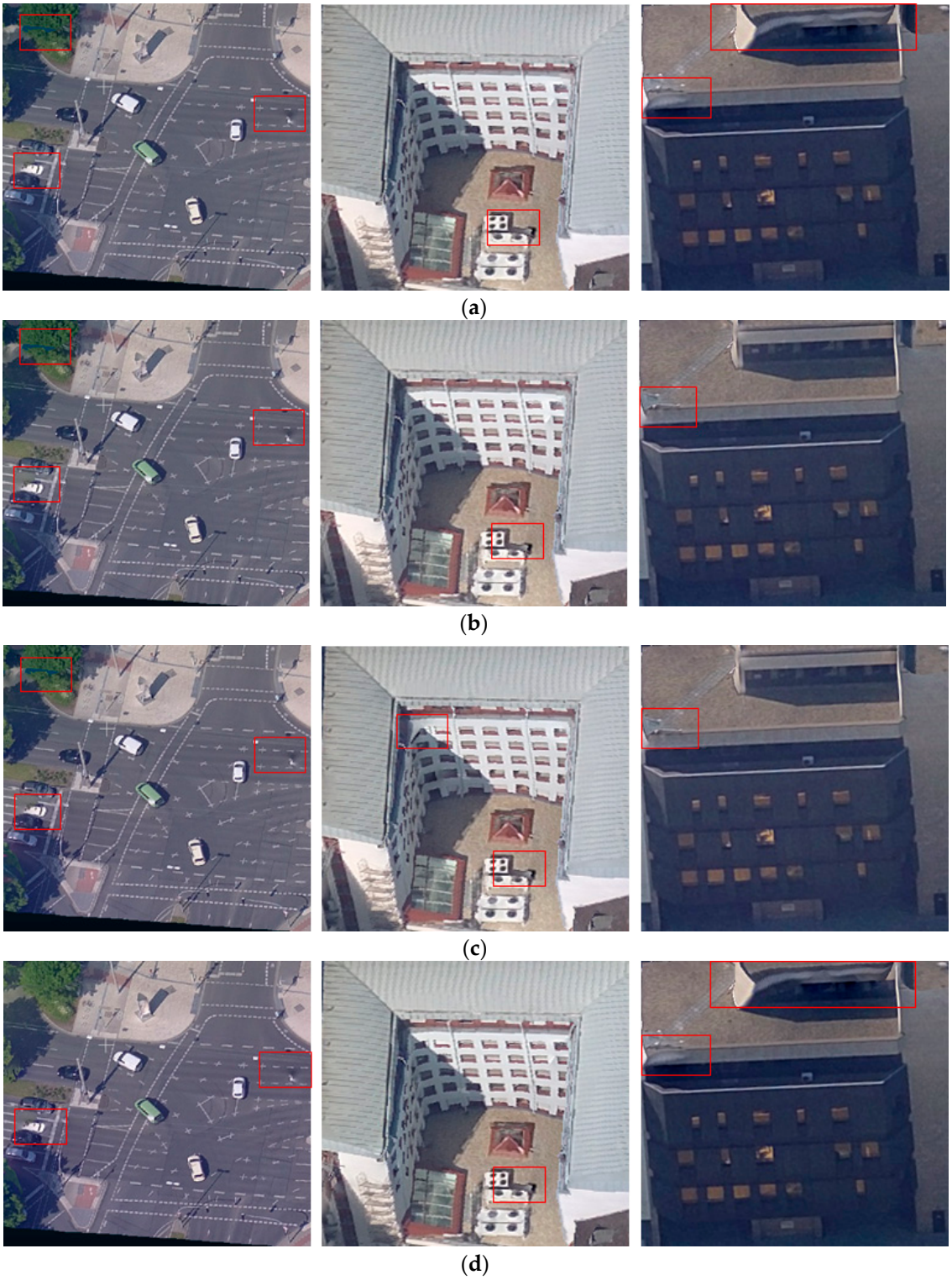


Figure 23. Textured areas based on the sparse ray-tracing. (**a**) Textured areas when epsilon = 0.00001. (**b**) Textured areas when epsilon = 0.0001. (**c**) Textured areas when epsilon = 0.001. (**d**) Textured areas when epsilon = 0.01.

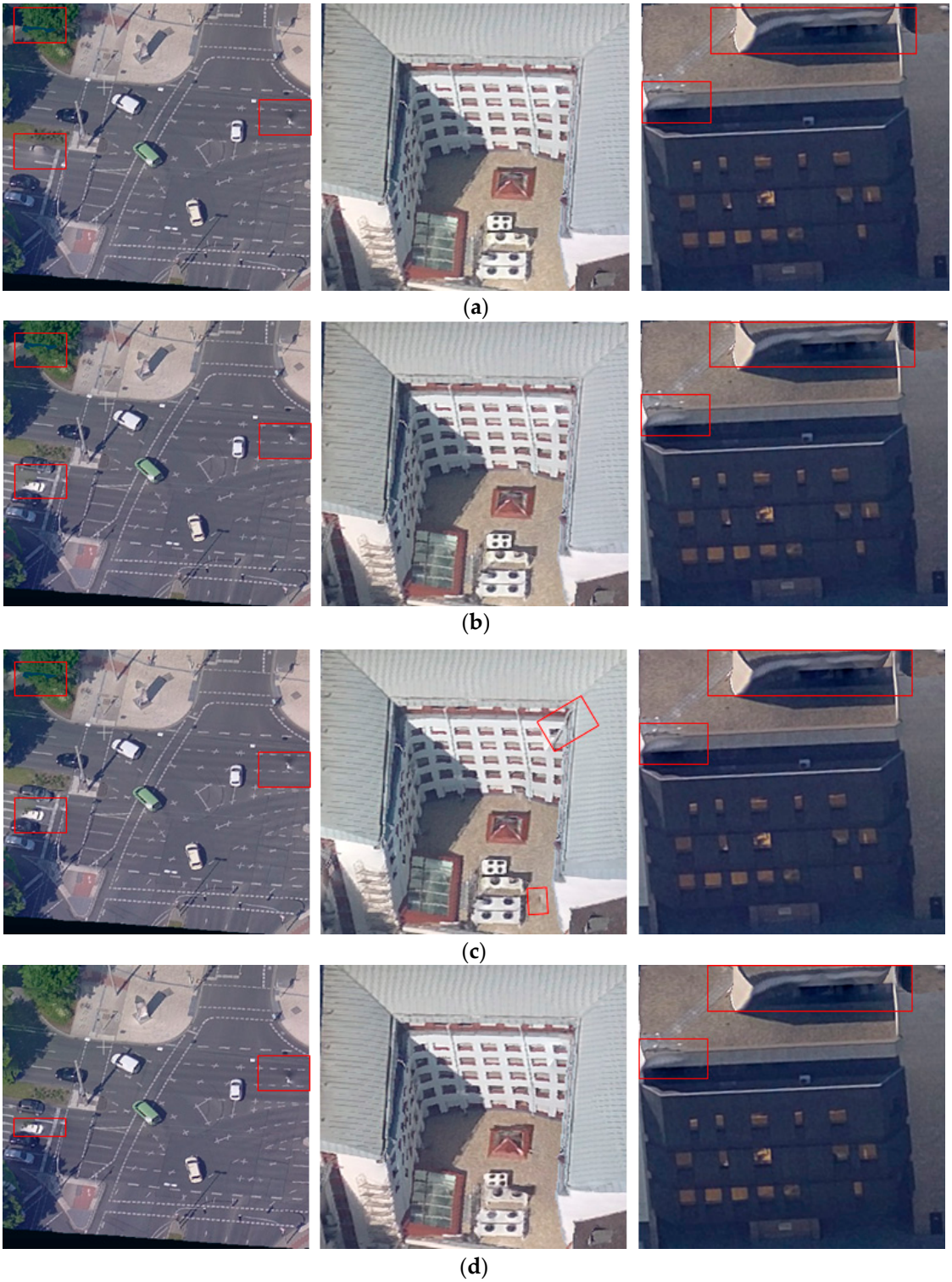


Figure 24. Textured areas based on the dense ray-tracing. (**a**) Textured areas when epsilon = 0.00001. (**b**) Textured areas when epsilon = 0.0001. (**c**) Textured areas when epsilon = 0.001. (**d**) Textured areas when epsilon = 0.01.

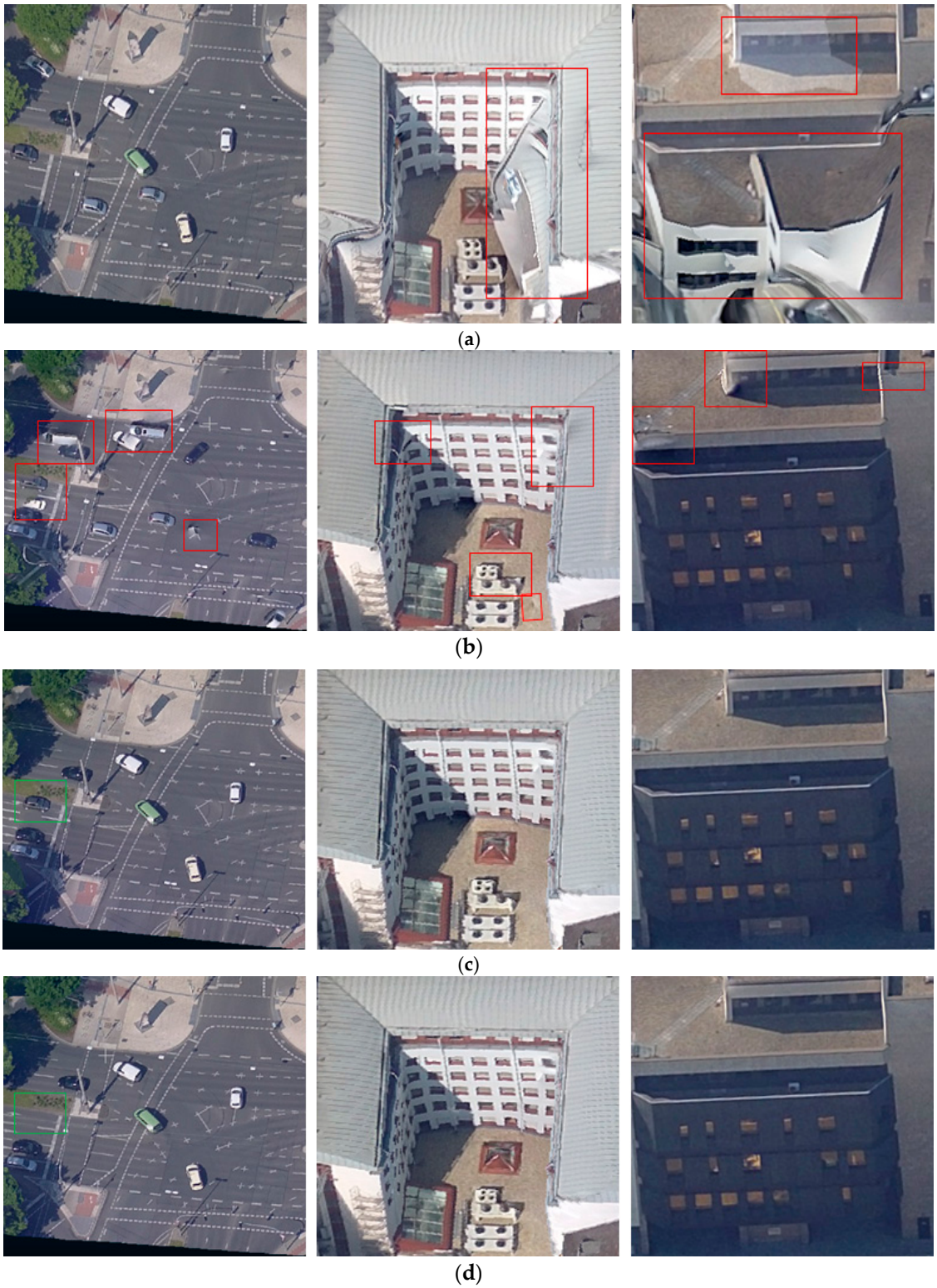


Figure 25. Textured areas of our method. (**a**) Textured areas without visibility tests. (**b**) Textured areas based on GPVC but without projection coverage correction. (**c**) Textured areas based on GPVC with projection coverage correction (HIVERS). (**d**) Textured areas based on GPVC with projection coverage correction (VES).

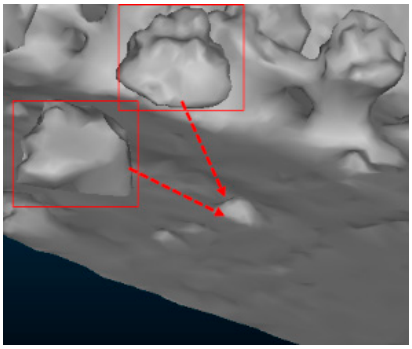


Figure 26. The nonmanifold structures of the reconstructed mesh.

Figures 27–29 presents texturing comparisons between GPVC and other software. It can be seen that the distortions, dislocations, and holes contained in the textured models generated by ContextCapture and MVS-Texturing result from the inaccurate selection of textures. In the textured models based on GPVC, these noises disappear, which proves the superiority of GPVC. GPVC attempts to compute the true visibility of each face on a surface. In other words, the accuracy of GPVC will not be affected by the artifacts of a reconstructed geometry. As shown in Figure 30a (to show the artifacts obviously, we enable each face's normal during rendering), the reconstructed geometry contains many artifacts such as the distortions, uneven areas and fragments. And these artifacts still occur in the textured model. Some of the serious artifacts have an effect on their neighbors' visibilities, resulting in the optimal textures being excluded, but the selected textures are still reasonable. Compared with the reconstructed geometry, no artifacts occur in the relatively accurate geometry in Figure 30b (the accurate geometry is extracted from the reconstructed geometry by using the method proposed by Verdie et al. [48]). Thus, fewer artifacts occur in the textured model. But there are still dislocations in the red boxes of Figure 30b. The reason is that the extracted geometry misses some occluders such as trees and roofs, and these occluders' textures are mapped to the corresponding areas. From the above analysis, visibility tests based on GPVC and texturing are sensitive to the structures of reconstructed geometries.

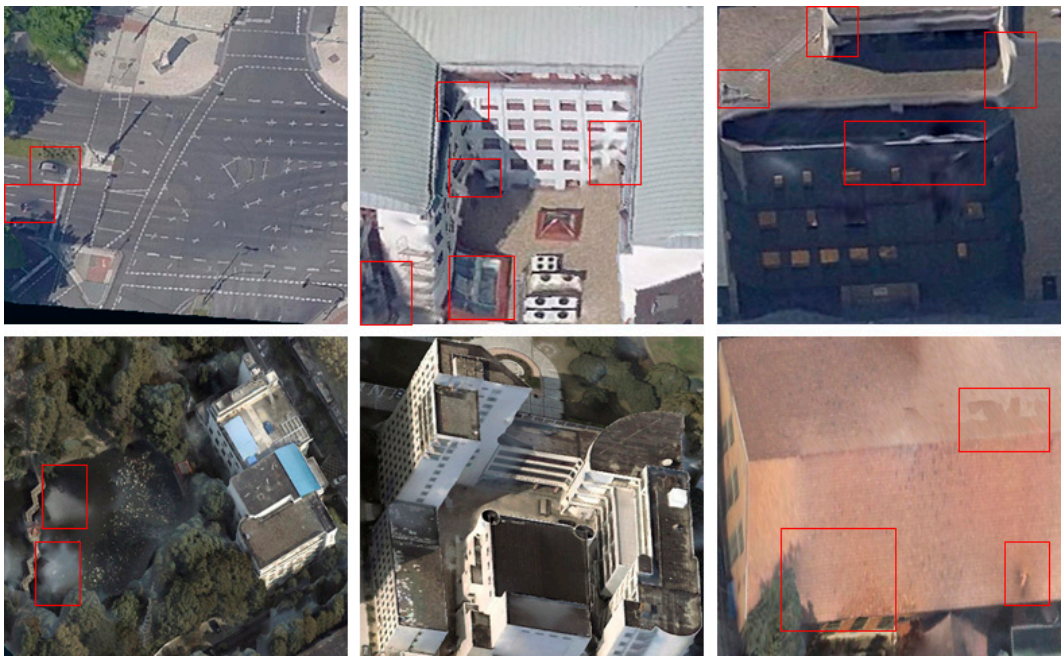


Figure 27. Textured areas generated by ContextCapture.

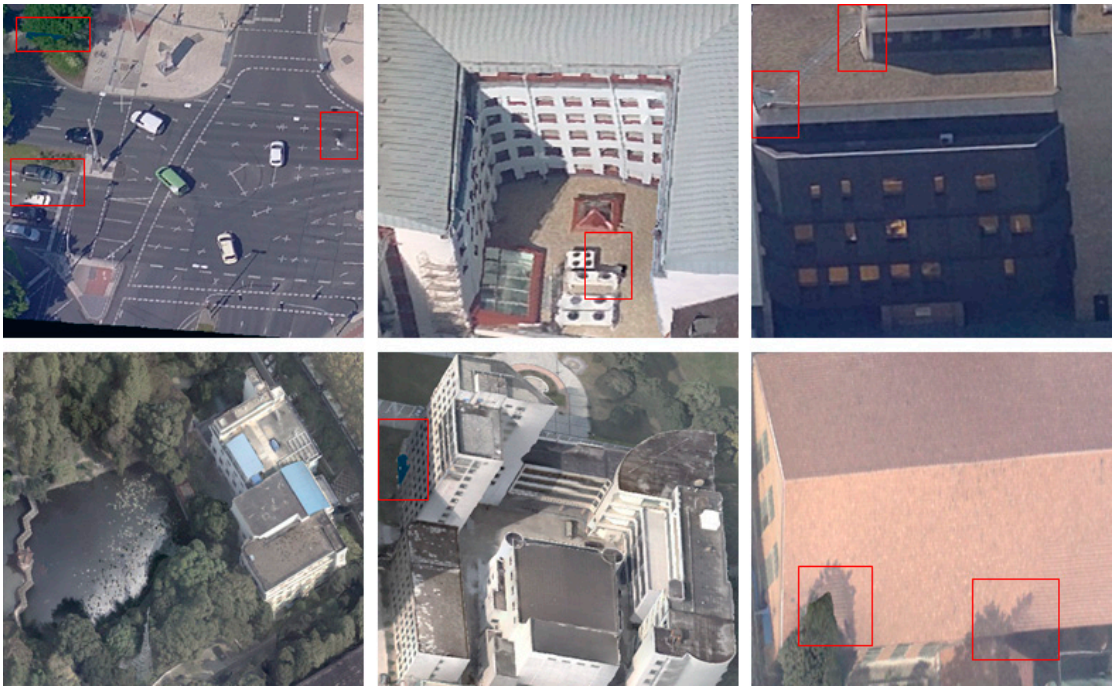


Figure 28. Textured areas generated by MVS-Texturing [15].

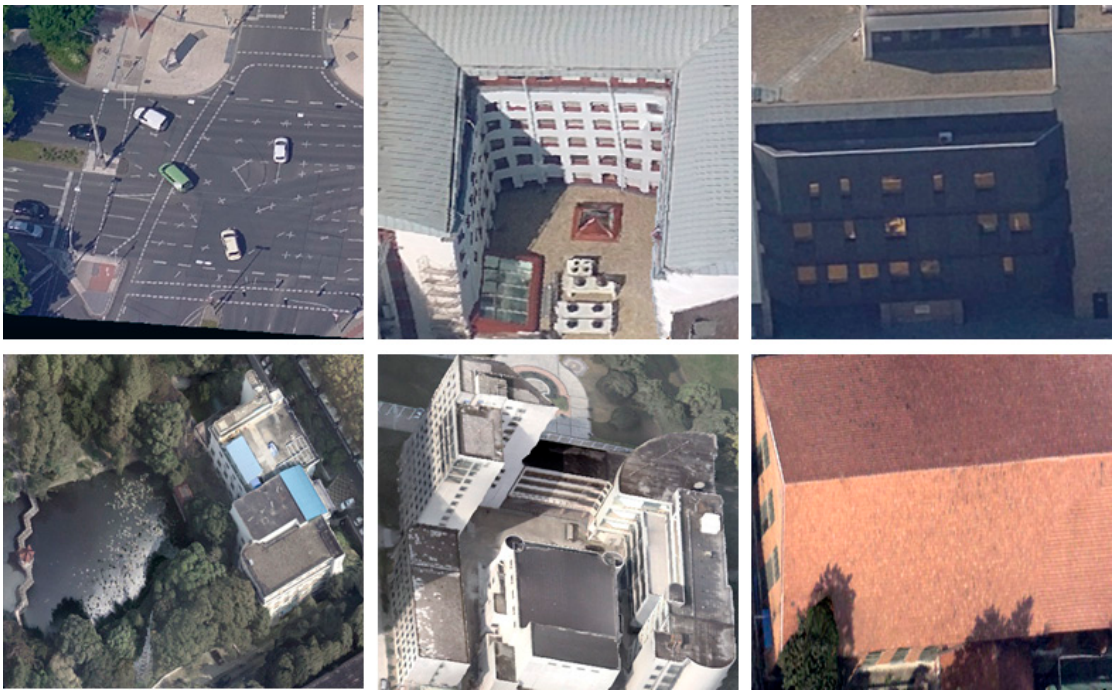


Figure 29. Textured areas based on GPVC.

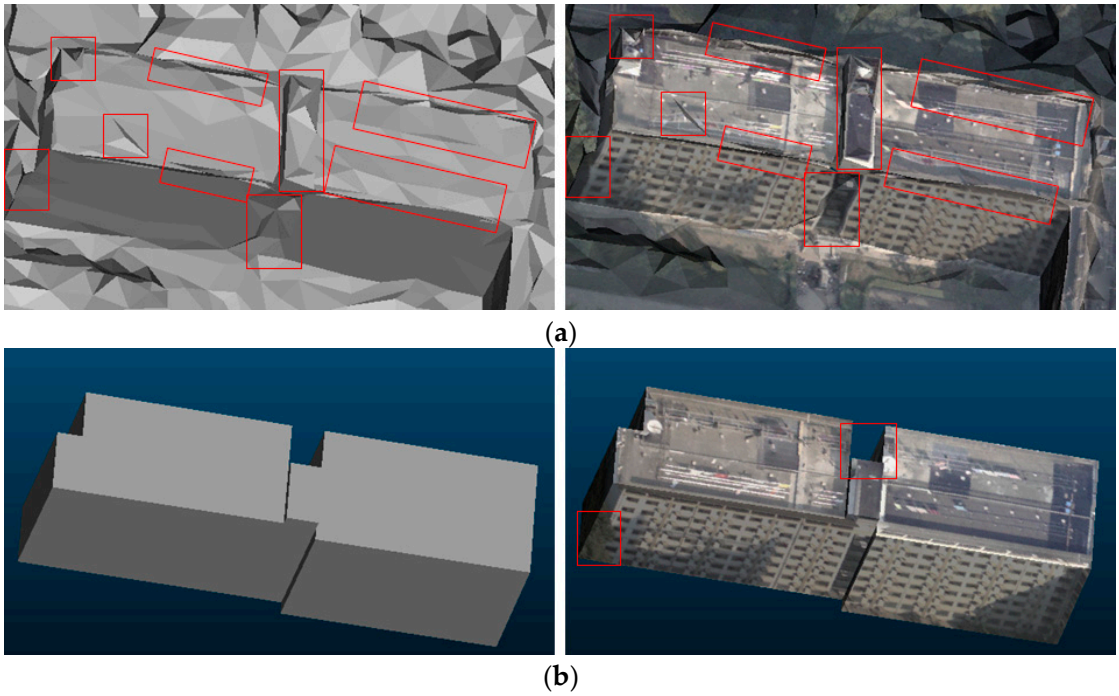


Figure 30. Textured comparisons between inaccurate geometries and accurate geometries. (**a**) Left: An inaccurate reconstructed geometry; right: the reconstructed geometry's texturing. (**b**) Left: The relatively accurate geometry extracted from the reconstructed geometry; right: the extracted geometry's texturing.

In terms of efficiency, we present the statistics regarding the number of processed faces, sampling count and time consumption of related methods. As shown in Tables 7–9, the number of processed faces in GPVC decreases to nearly 1/3 of those in the shadow-mapping and ray-tracing methods. The reason is that our GPVC only handles the visible faces that are stored in the shader storage buffer object (as shown in Figure 9), while the other two methods must handle the visible and invisible faces. With respect to the sampling count, the sparse ray tracing has the lowest value because it only processes the three vertices of a triangle. The shadow-mapping algorithm with the symmetric projection configuration (epsilon = 0.00001) is second. The reason is that the epsilon in use is too small for the experimental mesh, and many faces are labeled as occluded after sampling a few inner fragments. As discussed above, the performance of the shadow-mapping algorithm with the asymmetric projection configuration is not insensitive to the selected epsilon; thus, this approach's efficiency remains nearly unchanged. Compared with other methods, the improvement of GPVC with dense sampling is not as obvious. The reason is that many visible faces are identified as occluded by other bias-based methods without sampling all inner fragments. GPVC with dense sampling must sample all inner fragments of visible faces to ensure correct visibility. This situation can be improved by using HIVERS. Compared with dense sampling, the main function of HIVERS is that HIVERS can identify occluded faces with fewer samplings (as shown in Figure 31). Unlike the hierarchical structures used in References [24–26,49], in which hierarchy trees are built for a global scene as look-up indexes, the hierarchical structure of HIVERS decomposes a single triangle face into vertices, edges and subregions iteratively (see Figure 13). HIVERS samples the inner fragments corresponding to vertices and edges in each subregion to identify each face's visibility. According to our Inference 1, HIVERS can be replaced by VES when addressing a manifold scene. VES not only performs well for a partially visible primitive but also avoids sampling all of the inner fragments within a fully visible primitive; thus, the improvement on the sampling count increases by nearly 1/3 (as shown in Table 9). In terms of time consumption, the shadow-mapping algorithm with the symmetric projection configuration requires more time as epsilon increases. The reason is that a larger epsilon produces

more visible fragments, requiring more samplings to inspect a visible primitive. In contrast, although the sampling count increases as epsilon increases, the time cost of the ray tracing decreases. The reason is that a larger epsilon used in Reference [36] can reduce the traversal of hierarchy tree nodes and reach leaves with less time. However, the ray-tracing still exhibits the worst performance due to its limitation in finding intersections on a complex surface. Compared with the bias-based methods, GPVC exhibits good performance with respect to efficiency as well as accuracy. When deploying our visibility classification in the compute shader, the task can be performed within 0.2 s.

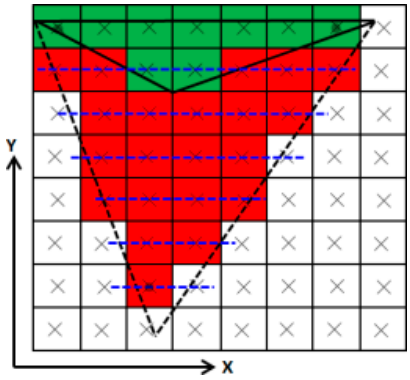


Figure 31. The improvement of HIVERS. In the worst case, the dense sampling must traverse 20 fragments to identify the occluded primitive, while HIVERS only traverses 2 fragments.

6. Conclusions

Visibility computation is highly important in multiview-based texture reconstruction. However, the bias-based methods used in this field are unable to achieve a good performance for a complex surface. Thus, in this paper, we propose GPVC which does not depend on an uncontrollable bias, to make an improvement. According to our experimental results, GPVC outperforms the bias-based methods and other related libraries and software. Based on GPVC's framework, certain applications, such as shadow generation, visibility picking, and visibility query, can also be derived.

Author Contributions: X.H. proposed the idea and wrote the manuscript; Q.Z. provided the experimental data; W.J. conceived and designed the experiments; and X.H. performed the experiments and analyzed the data.

Funding: This work was supported by the Key Technology program of China South Power Grid under Grant GDKJQQ20161187.

Acknowledgments: The authors would like to acknowledge the provision of the datasets by ISPRS and EuroSDR, released in conjunction with the ISPRS scientific initiative 2014 and 2015, led by ISPRS ICWG I/Vb.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jimenezdelgado, J.J.; Segura, R.J.; Feito, F.R. Efficient collision detection between 2d polygons. *J. WSCG* **2004**, *12*, 191–198.
2. Feng, Y.T.; Owen, D.R.J. A 2d polygon/polygon contact model: Algorithmic aspects. *Eng. Comput.* **2004**, *21*, 265–277. [CrossRef]
3. Greminger, M.A.; Nelson, B.J. A deformable object tracking algorithm based on the boundary element method that is robust to occlusions and spurious edges. *Int. J. Comput. Vision* **2007**, *78*, 29–45. [CrossRef]
4. Huang, Y.; Essa, I.A. Tracking multiple objects through occlusions. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 20–25 June 2005; pp. 1051–1058.
5. Han, B.; Paulson, C.; Lu, T.; Wu, D.; Li, J. Tracking of multiple objects under partial occlusion. In Proceedings of the SPIE Defense, Security, and Sensing, Orlando, FL, USA, 5 May 2009.

6. Yang, Q.; Wang, L.; Yang, R.; Stewenius, H.; Nister, D. Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. *IEEE Trans. Pattern Anal. Mach. Intell.* **2009**, *31*, 492–504. [CrossRef] [PubMed]
7. Eisert, P.; Steinbach, E.G.; Girod, B. Multi-hypothesis, volumetric reconstruction of 3-d objects from multiple calibrated camera views. In Proceedings of the IEEE International Conference on Acoustics Apeech and Signal Processing, Phoenix, AZ, USA, 15–19 March 1999; pp. 3509–3512.
8. Lee, Y.J.; Lee, S.J.; Park, K.R.; Jo, J.; Kim, J. Single view-based 3d face reconstruction robust to self-occlusion. *EURASIP J. Adv. Signal Process.* **2012**, *2012*, 176. [CrossRef]
9. Kuffner, J.J.; Nishiwaki, K.; Kagami, S.; Inaba, M.; Inoue, H. Footstep planning among obstacles for biped robots. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, HI, USA, 29 October–3 November 2001; pp. 500–505.
10. Ahuja, N.; Chien, R.T.; Yen, R.; Bridwell, N. Interference detection and collision avoidance among three dimensional objects. In Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford, CA, USA, 18–21 August 1980; pp. 44–48.
11. Nakamura, T.; Asada, M. Stereo sketch: Stereo vision-based target reaching behavior acquisition with occlusion detection and avoidance. In Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN, USA, 22–28 April 1996; pp. 1314–1319.
12. Jiang, S.; Jiang, W. Efficient sfm for oblique uav images: From match pair selection to geometrical verification. *Remote Sens.* **2018**, *10*, 1246. [CrossRef]
13. Pages, R.; Berjon, D.; Moran, F.; Garcia, N.N. Seamless, static multi-texturing of 3d meshes. *Comput. Graph. Forum* **2015**, *34*, 228–238. [CrossRef]
14. Zhang, W.; Li, M.; Guo, B.; Li, D.; Guo, G. Rapid texture optimization of three-dimensional urban model based on oblique images. *Sensors* **2017**, *17*, 911. [CrossRef] [PubMed]
15. Waechter, M.; Moehrle, N.; Goesele, M. Let there be color! Large-scale texturing of 3d reconstructions. In Proceedings of the 13th European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 836–850.
16. Pintus, R.; Gobbetti, E.; Callieri, M.; Dellepiane, M. Techniques for seamless color registration and mapping on dense 3d models. In *Sensing the Past*; Springer: Berlin, Germany, 2017; pp. 355–376.
17. Symeonidis, A.; Koutsoudis, A.; Ioannakis, G.; Chamzas, C. Inheriting texture maps between different complexity 3d meshes. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2014**, *II-5*, 355–361. [CrossRef]
18. Frueh, C.; Sammon, R.; Zakhor, A. Automated texture mapping of 3d city models with oblique aerial imagery. In Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission, Thessaloniki, Greece, 6–9 September 2004; pp. 396–403.
19. Xu, L.; Li, E.; Li, J.; Chen, Y.; Zhang, Y. A general texture mapping framework for image-based 3d modeling. In Proceedings of the 17th IEEE International Conference on Image Processing, Hong Kong, China, 26–29 September 2010; pp. 2713–2716.
20. Katz, S.; Tal, A.; Basri, R. Direct visibility of point sets. *ACM Trans. Graph.* **2007**, *26*, 343–352. [CrossRef]
21. Cohenor, D.; Chrysanthou, Y.; Silva, C.T.; Durand, F. A survey of visibility for walkthrough applications. *IEEE Trans. Vis. Comput. Graph.* **2003**, *9*, 412–431. [CrossRef]
22. Bittner, J.; Wonka, P. Visibility in computer graphics. *Environ. Plan. B Plan. Des.* **2003**, *30*, 729–755. [CrossRef]
23. Franklin, W.R.; Chandrasekhar, N.; Kankanhalli, M.; Seshan, M.; Akman, V. Efficiency of uniform grids for intersection detection on serial and parallel machines. In *New Trends in Computer Graphics-CGI'88*; Springer: Geneva, Switzerland, 1988; pp. 288–297.
24. Yu, B.T.; Yu, W.W. Image space subdivision for fast ray tracing. In Proceedings of the SPIE's International Symposium on Optical Science, Engineering, and Instrumentation, Denver, CO, USA, 23 September 1999; pp. 149–156.
25. Weghorst, H.; Hooper, G.; Greenberg, D.P. Improved computational methods for ray tracing. *ACM Trans. Graph.* **1984**, *3*, 52–69. [CrossRef]
26. Glassner, A.S. Space subdivision for fast ray tracing. *IEEE Comput. Graph. Appl.* **1984**, *4*, 15–24. [CrossRef]
27. Grammatikopoulos, L.; Kalisperakis, I.; Karras, G.; Petsa, E. Data fusion from multiple sources for the production of orthographic and perspective views with automatic visibility checking. In Proceedings of the CIPA 2005 XX International Symposium, Torino, Italy, 26 September–1 October 2005.

28. Grammatikopoulos, L.; Kalisperakis, I.; Karras, G.; Petsa, E. Automatic multi-view texture mapping of 3d surface projections. In Proceedings of the 2nd ISPRS International Workshop 3D-ARCH, ETH Zurich, Switzerland, 12–13 July 2007; pp. 1–6.
29. Karras, G.; Grammatikopoulos, L.; Kalisperakis, I.; Petsa, E. Generation of orthoimages and perspective views with automatic visibility checking and texture blending. *Photogramm. Eng. Remote Sens.* **2007**, *73*, 403–411. [CrossRef]
30. Chen, Z.; Zhou, J.; Chen, Y.; Wang, G. 3d texture mapping in multi-view reconstruction. In Proceedings of the International Symposium on Visual Computing, Rethymnon, Crete, Greece, 16–18 July 2012; pp. 359–371.
31. Baumberg, A. Blending images for texturing 3d models. In Proceedings of the British Machine Vision Conference, Cardiff, UK, 2–5 September 2002; pp. 404–413.
32. Bernardini, F.; Martin, I.M.; Rushmeier, H.E. High-quality texture reconstruction from multiple scans. *IEEE Trans. Vis. Comput. Graph.* **2001**, *7*, 318–332. [CrossRef]
33. Callieri, M.; Cignoni, P.; Corsini, M.; Scopigno, R. Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3d models. *Comput. Graph.* **2008**, *32*, 464–473. [CrossRef]
34. Floriani, L.D.; Magillo, P. Algorithms for visibility computation on terrains: A survey. *Environ. Plan. B Plan. Des.* **2003**, *30*, 709–728. [CrossRef]
35. Rocchini, C.; Cignoni, P.; Montani, C.; Scopigno, R. Multiple textures stitching and blending on 3d objects. In Proceedings of the Eurographics Symposium on Rendering techniques, Granada, Spain, 21–23 June 1999; pp. 119–130.
36. Geva, A. Coldet 3d Collision Detection. Available online: sourceforge.net/projects/coldet/ (accessed on 19 April 2018).
37. Waechter, C.; Keller, A. Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing. Patents US7952583B2, 31 April 2011.
38. Williams, L. Casting curved shadows on curved surfaces. *ACM Siggraph Comput. Graph.* **1978**, *12*, 270–274. [CrossRef]
39. Annen, T.; Mertens, T.; Seidel, H.P.; Flerackers, E.; Kautz, J. Exponential shadow maps. In Proceedings of the Graphics Interface 2008, Windsor, ON, Canada, 28–30 May 2008; pp. 155–161.
40. Dou, H.; Kerzner, E.; Kerzner, E.; Wyman, C.; Wyman, C. Adaptive depth bias for shadow maps. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3d Graphics and Games, San Francisco, CA, USA, 14–16 March 2014; pp. 97–102.
41. Persson, E.; Studios, A. Creating vast game worlds: Experiences from avalanche studios. In Proceedings of the ACM SIGGRAPH 2012 Talks, Los Angeles, CA, USA, 5–9 August 2012; p. 32.
42. Vasilakis, A.; Fudos, I. Depth-fighting aware methods for multi-fragment rendering. *IEEE Trans. Vis. Comput. Graph.* **2013**, *19*, 967–977. [CrossRef] [PubMed]
43. Vasilakis, A.A.; Fudos, I. Z-fighting aware depth peeling. In Proceedings of the ACM SIGGRAPH, Vancouver, BC, Canada, 7–11 August 2011.
44. Pineda, J. A parallel algorithm for polygon rasterization. In Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, Atlanta, GA, USA, 1–5 August 1988; pp. 17–20.
45. Segal, M.; Akeley, K. *The Opengl Graphics System: A Specication (Version 4.5)*; Technical Report; Khronos Group Inc.: Beaverton, OR, USA, 2016.
46. Davidovic, T.; Engelhardt, T.; Georgiev, I.; Slusallek, P.; Dachsbacher, C. 3d rasterization: A bridge between rasterization and ray casting, graphics interface. In Proceedings of the Graphics Interface 2012, Toronto, ON, Canada, 28–30 May 2012; pp. 201–208.
47. Zhang, Z.; Su, G.; Zhen, S.; Zhang, J. Relation opengl imaging process with exterior and interior parameters of photogrammetry. *Geomat. Inf. Sci. Wuhan Univ.* **2004**, *29*, 570–574.
48. Verdie, Y.; Lafarge, F.; Alliez, P. Lod generation for urban scenes. *ACM Trans. Graph.* **2015**, *34*, 1–14. [CrossRef]
49. Greene, N.; Kass, M.; Miller, G.S.P. Hierarchical z-buffer visibility. In Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, Anaheim, CA, USA, 2–6 August 1993; pp. 231–238.