Vis Comput (2009) 25: 549–557
DOI 10.1007/s00371-009-0335-3

ORIGINAL ARTICLE

Towards multi-perspective rasterization

Xuan Yu · Jingyi Yu · Leonard McMillan

Published online: 19 March 2009

Abstract We present a novel framework for real-time multi-perspective rendering. While most existing approaches are based on ray-tracing, we present an alternative approach by emulating multi-perspective rasterization on the classical perspective graphics pipeline. To render a general multi-perspective camera, we first decompose the camera into piecewise linear primitive cameras called the general linear cameras or GLCs. We derive the closed-form projection equations for GLCs and show how to rasterize triangles onto GLCs via a two-pass rendering algorithm. In the first pass, we compute the GLC projection coefficients of each scene triangle using a vertex shader. The linear raster on the graphics hardware then interpolates these coefficients at each pixel. Finally, we use these interpolated coefficients to compute the projected pixel coordinates using a fragment shader. In the second pass, we move the pixels to their actual projected positions. To avoid holes, we treat neighboring pixels as triangles and re-render them onto the GLC image plane. We demonstrate our real-time multi-perspective rendering framework in a wide range of applications including synthesizing panoramic and omnidirectional views, rendering reflections on curved mirrors, and creating multi-perspective faux animations. Compared with the GPU-based ray tracing methods, our rasterization approach scales better with scene complexity and it can render scenes with a large number of triangles at interactive frame rates.

Keywords Multi-perspective rendering · Graphics hardware · Real-time rendering · Rasterization

Electronic supplementary material The online version of this article (http://dx.doi.org/10.1007/s00371-009-0335-3) contains supplementary material, which is available to authorized users.

X. Yu · J. Yu (✉)
University of Delaware, Newark, USA
e-mail: jingyiyu@udel.edu

X. Yu
e-mail: xuanyu@udel.edu

L. McMillan
University of North Carolina at Chapel Hill, Chapel Hill, USA

1 Introduction

A perspective camera captures the spatial relationships of objects in a scene as they appear from a single viewpoint. In art, the use of perspective cameras is surprisingly rare: artists, architects, and engineers regularly combine what is seen from several viewpoints into a single image. Despite their incongruity of view, these multi-perspective images are able to preserve spatial coherence. They can depict, within a single context, details of a scene that are simultaneously inaccessible from a single view.

In Computer Graphics, image-based approaches are commonly used to render multi-perspective images [3, 27, 30]. Their rendering quality depends on the sampling of the plenoptic function [1] and aliasing artifacts can be introduced during initial sampling and final reconstruction. Alternatively, ray tracing can be used to render high quality multi-perspective cameras. However, ray tracing is often too slow for interactive rendering. Recently, GPU-based ray tracing techniques [4, 19] have been developed to efficiently determine ray–object intersections. However, a fundamental problem in GPU ray tracing is that it requires random access to a database of scene triangles, which does not fit well with the SIMD feed-forward graphics pipeline [11].

In this paper, we present a novel rasterization framework for rendering multi-perspective images in real-time. To

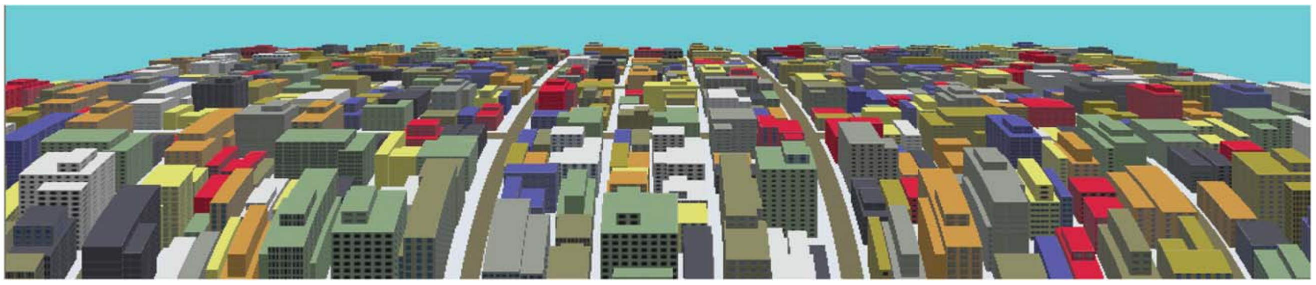

Fig. 1 Our framework interactively renders a complex city scene of 15k triangles at 70 fps. A cross-slit camera is used to render the scene at an 1024×320 resolution

render a general multi-perspective camera, we first decompose the camera into piecewise primitive multi-perspective cameras. We use the recently proposed general linear cameras (GLC) model that describes typical pinhole and orthographic cameras, as well as many commonly studied multi-perspective cameras. We derive the closed-form projection equations for GLCs and show how to rasterize triangles onto the GPU via a two-pass rendering algorithm.

In the first pass, we compute the GLC projection coefficients of each scene triangle using a vertex shader. The linear raster on the graphics hardware then interpolates these coefficients at each pixel. Finally, we use these interpolated coefficients to compute the projected pixel coordinates using a fragment shader. In the second pass, we move these pixels to their actual projected positions. To avoid holes, we treat neighboring pixels as triangles and re-render them onto the GLC image plane. We demonstrate our real-time multi-perspective rendering framework in a wide range of applications including synthesizing panoramic and omnidirectional views, rendering reflections on curved mirrors, and creating multi-perspective faux animations. Compared to GPU-based ray tracing methods, our rasterization approach scales better with scene complexity and can render highly complex scenes with a large number of triangles at interactive frame rates.

2 Previous work

Historically, multi-perspective images have been frequently employed by pre-Renaissance and post-impressionist artists to depict more than can be seen from any specific point [29]. Escher uses highly curved projection models to generate “impossible” perspectives of a scene. Picasso and other Cubism pioneers made effective use of rearranging different parts of the depicted scene while maintaining their local spatial relationships, which results in an incongruous spatial system [8].

In Computer Graphics, multi-perspective images have been widely used in cel-animations [27] and omnidirectional visualizations [15, 20]. A commonly used technique for creating these images is to combine strips from different pinhole images. This approach, often called a strip camera, has appeared quite often in Graphics literature. For example, computer generated multi-perspective panoramas [27] combined elements of multiple pinhole strips into a single image using a semi-automatic image registration process. The concentric mosaics of [24] and [17] are another type of multi-perspective image that is useful for exploring captured environments.

Durand [6] suggests that specifying multi-perspective cameras can also be an interactive process and uses them as an example to distinguish between picture generation and user interaction. Examples of such approaches include the 3D-based interactive rendering systems by Agrawala et al. [3] and Hanson and Wernert [10]. Roman et al. [21, 22] provide a semi-interactive system that uses a linear camera to combine photographs into panoramas of street scenes. Agrawala et al. [2] proposed to composite large regions of ordinary perspective images. They reduce the degree of user interaction by identifying the dominant plane and then use graph cuts to minimize multi-perspective distortions.

Recently, real-time multi-perspective rendering techniques have been developed based on polygonal graphics hardware. These include techniques for supporting multiple centers of projection in VR applications [12, 25], rendering general curved reflections or refractions [16, 28], and synthesizing special panoramic effects [7, 29]. The work by Hou et al. [11] decomposes an arbitrary multi-perspective image into piecewise-linear multi-perspective primitives. They then render each primitive camera by implementing a non-linear beam-tracing using a pair of vertex and fragment programs.

Finally, multi-perspective images have received attention from the computer vision community for analyzing structure revealed via motion [17, 23] and generating panoramic images with a wide field-of-view using mirrors [26]. Several researchers have proposed alternative multi-perspective camera models which capture rays from different points in space. These multi-perspective cameras include pushbroom cameras [9], which collect rays along parallel planes from

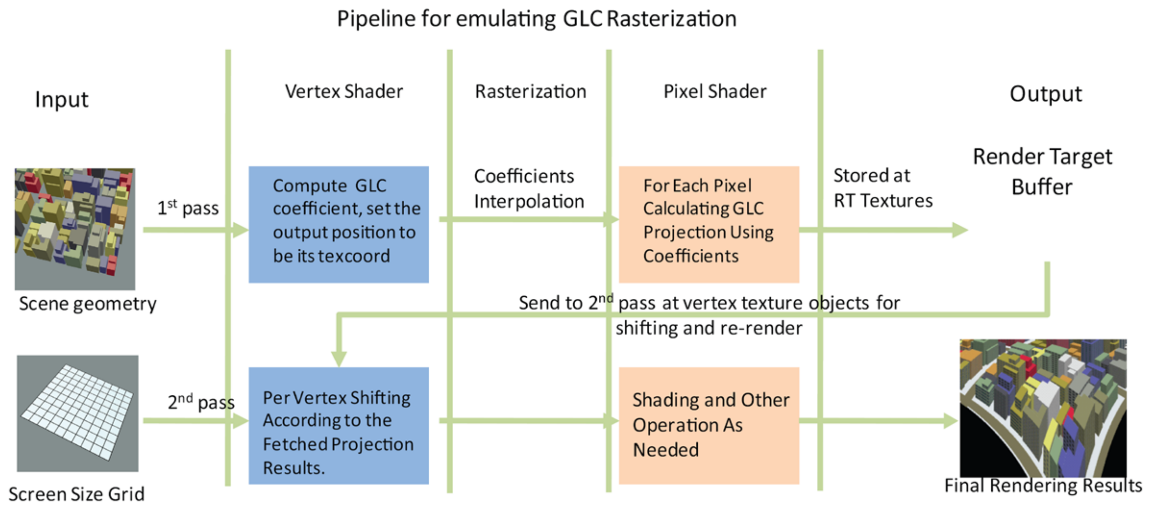
Pipeline for emulating GLC Rasterization
Input
Vertex Shader
Rasterization
Pixel Shader
Output
1st pass
Compute GLC coefficient, set the output position to be its texcoord
Coefficients Interpolation
For Each Pixel Calculating GLC Projection Using Coefficients
Stored at RT Textures
Render Target Buffer
Scene geometry
Send to 2nd pass at vertex texture objects for shifting and re-render
2nd pass
Per Vertex Shifting According to the Fetched Projection Results.
Shading and Other Operation As Needed
Screen Size Grid
Final Rendering Results

Fig. 2 A diagram showing the pipeline of our rendering algorithm

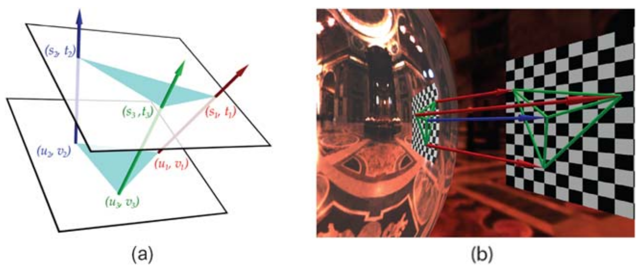

Fig. 3 (**a**) A General Linear Camera is described by three generator rays parametrized under two parallel planes (2PP). (**b**) We can decompose a reflection image of a curved mirror into piecewise GLCs

points swept along a linear trajectory, and two-slit cameras [31], which collect all rays passing through two lines.

3 Multi-perspective decomposition

To render an arbitrary multi-perspective camera, we first decompose the camera into piecewise linear primitive multi-perspective cameras whose projection models can be easily characterized.

3.1 General linear cameras

Our decomposition is based on the recently proposed general linear camera or GLCs [30]. GLCs unify traditional perspective, orthographic, and multi-perspective cameras models such as the pushbroom and the cross-slit cameras. In the GLC framework, every ray is parameterized by its intersections with the two parallel planes, where $[s,t]$ is the intersection with the first and $[u,v]$ the second, as shown in Fig. 3(a). This parametrization is often called a two-plane parametrization (2PP) [13]. In this paper, we further simplify the analysis of [30] by substituting $\sigma = s - u$ and $\tau = t - v$ and use $[\sigma,\tau,u,v]$ to represent rays. We also assume the default uv plane is the default image plane and is at $z=0$ while st plane is at $z=1$.

A GLC is defined as the affine combination of three rays:

$$\mathrm{GLC} = \bigl\{r : r = \alpha\cdot[\sigma_1,\tau_1,u_1,v_1] + \beta\cdot[\sigma_2,\tau_2,u_2,v_2] + (1-\alpha-\beta)\cdot[\sigma_3,\tau_3,u_3,v_3], \forall\alpha,\beta\bigr\}. \quad (1)$$

We treat the problem of multi-perspective rendering as one of specifying the set of rays collected by the camera. Yu and McMillan [29] have shown that these GLC cameras, when constrained by an appropriate set of rules, can be laid out on the image plane, thereby generating arbitrary multi-perspective renderings. In fact, to a first order, the GLC-based framework can describe any multi-perspective image that is consistent with a smoothly varying set of rays. In Sect. 5.1, we demonstrate rendering 180 or 360 degree panoramas by stitching piecewise cross-slit GLCs. In Sect. 5.3, we show, by dynamically composing a multi-perspective camera from smoothly varying specific GLCs, that it is also possible to create faux animations from static models.

GLCs can also be used to render reflections on arbitrarily curved mirrors (Sect. 5.2). Given a mirror surface represented as a triangle mesh, we can associate the reflection ray with each vertex so that each triangle corresponds to a ray triplet, as shown in Fig. 3(b). We then specify the uv parametrization plane for each ray triplet as the plane of the triangle itself and render each GLC individually. Finally, we can compose multiple GLC images into a single reflection image.

Next, we derive the GLC projection equation. To further simplify our analysis, we assume the uv plane is the image plane of the GLC. We also assume the three generators have the form $[\sigma_1,\tau_1,0,0]$, $[\sigma_2,\tau_2,1,0]$, and $[\sigma_3,\tau_3,0,1]$,

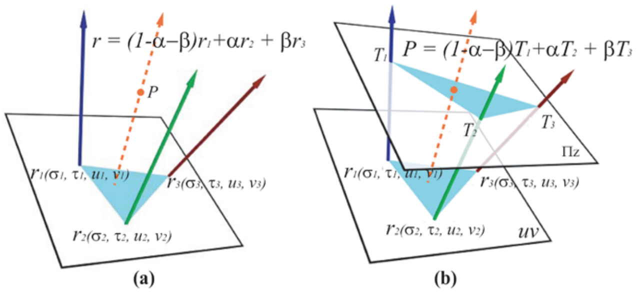

Fig. 4 (**a**) Projecting a point P to a ray in the GLC. (**b**) The projection of P can be computed using the affine coordinate on the sweeping plane Π_z

i.e., these three generator rays originate from pixels $[0,0]$, $[1,0]$, and $[0,1]$, respectively. Under these three generator rays, every ray r in the GLC can be written as the following affine combination:

$$r[\sigma,\tau,u,v] = (1-\alpha-\beta)\cdot[\sigma_1,\tau_1,0,0] + \alpha\cdot[\sigma_2,\tau_2,1,0] + \beta\cdot[\sigma_3,\tau_3,0,1]. \tag{2}$$

It is easy to see that $\alpha = u$ and $\beta = v$ under this simplification. Therefore, computing the affine coefficients of r is equivalent to projecting r onto the image plane.

To compute the projection of a 3D point $P(x,y,z)$ in the GLC, we sweep a plane Π_z parallel to the uv plane and passing through $\dot{P}$. The three generator rays will intersect Π_z at $\dot{T}_1$, $\dot{T}_2$, $\dot{T}_3$, where

$$\begin{aligned}
\dot{T}_1 &= (0,0,0) + z\cdot(\sigma_1,\tau_1,1) = (\sigma_1 z, \tau_1 z, z),\\
\dot{T}_2 &= (1,0,0) + z\cdot(\sigma_2,\tau_2,1) = (\sigma_2 z + 1, \tau_2 z, z),\\
\dot{T}_3 &= (0,1,0) + z\cdot(\sigma_3,\tau_3,1) = (\sigma_3 z, \tau_3 z + 1, z).
\end{aligned} \tag{3}$$

Our goal is to compute the affine combination $[\alpha,\beta]$ of the three generator rays that pass through P. Notice, the same affine coefficients should apply to point P with respect to triangle $\Delta\dot{T}_1\dot{T}_2\dot{T}_3$, i.e.,

$$\dot{P} = (1-\alpha-\beta)\cdot\dot{T}_1 + \alpha\cdot\dot{T}_2 + \beta\cdot\dot{T}_3. \tag{4}$$

Recall that $[\alpha,\beta]$ can be computed using the ratio of the signed areas formed by triangle $\Delta\dot{T}_1\dot{P}\dot{T}_3$, $\Delta\dot{T}_1\dot{T}_2\dot{P}$ over $\Delta\dot{T}_1\dot{T}_2\dot{T}_3$ as shown in Fig. 4(b); we have

$$u = \frac{\Delta\dot{T}_1\dot{P}\dot{T}_3}{\Delta\dot{T}_1\dot{T}_2\dot{T}_3} = \frac{\begin{vmatrix} z\sigma_1 & z\tau_1 & 1\\ x & y & 1\\ z\sigma_3 & 1+z\tau_3 & 1\end{vmatrix}}{\begin{vmatrix} z\sigma_1 & z\tau_1 & 1\\ 1+z\sigma_2 & 0+z\tau_2 & 1\\ z\sigma_3 & 1+z\tau_3 & 1\end{vmatrix}},$$

$$v = \frac{\Delta\dot{T}_1\dot{T}_2\dot{P}}{\Delta\dot{T}_1\dot{T}_2\dot{T}_3} = \frac{\begin{vmatrix} z\sigma_1 & z\tau_1 & 1\\ 1+z\sigma_2 & z\tau_2 & 1\\ x & y & 1\end{vmatrix}}{\begin{vmatrix} z\sigma_1 & z\tau_1 & 1\\ 1+z\sigma_2 & 0+z\tau_2 & 1\\ z\sigma_3 & 1+z\tau_3 & 1\end{vmatrix}}. \tag{5}$$

For GLC-based decompositions, the smoothness is guaranteed if all cameras are parameterized under the same 2PP. However, when GLCs are used to model more complicated phenomenons such as reflections, it is common practice to use different parametrization planes (e.g., local tangent planes), for minimizing the approximation errors. Therefore, the projection continuity for two adjacent GLCs is not guaranteed in a general configuration [18]. In [11], Hou et al. proposed a similar multi-perspective camera decomposition scheme. Their approach used the normalized vector $[n_x, n_y, n_z]$ to represent the ray direction instead of the 2PP. Their parametrization naturally maintains smoothness across the camera boundaries. However, their projection equation is more complicated and it may have multiple (up to 4) solutions, and hence, additional steps are needed to determine the actual solution. Popescu et al. [18] extended GLCs to a continuous form, but at the cost of a cubic projection equation. For multi-perspective rasterization, the linear projection not only has the advantage of simpler computation, but also the more important advantage of single projection.

4 Multi-perspective rasterization

Our goal is to emulate GLC-based triangle rasterization on the perspective camera graphics pipeline. In this section, we first briefly review the perspective-camera based linear rasterization and then identify the difficulty in implementing GLC rasterization. Finally, we present a simple and efficient technique for emulating GLC rasterization on the GPU.

4.1 Linear rasterization

In a perspective camera, 3D lines project to 2D lines via perspective projection. Therefore, to render a triangle, we can simply first project the vertices of the triangle onto the image plane of the camera and then linearly interpolate the pixels between the projected vertices. Therefore, the raster on the graphics hardware serves simply as a linear interpolation engine. In contrast, a triangle maps to a curved triangle in a multi-perspective camera. Thus, the linear raster cannot be directly used to rasterize the triangles.

Modern graphics hardware has been focused on adding programmable shading to their capabilities, to allow each vertex or pixel be processed by a short program [5, 14]. For example, it is easy to write simple vertex and fragment programs to support Phong shading using per-pixel-based lighting, surface normal and positions. However, very little effort has been made to change the linear raster. In fact, rasterization of curved triangle still remains an open problem in computer graphics.

4.2 Emulating GLC rasterization

We present a simple and efficient GLC rasterization scheme on the GPU. Our goal is to use the raster to interpolate the linear components in the GLC projection equation and rely on the fragment shader to conduct the nonlinear operations.

Recall that both the numerator and the denominator in the GLC projection equation (5) can be written as the determinant of matrices. Furthermore, all entries in the two matrices are linear in the x, y, and z coordinates of 3D points. Therefore, we can linearly interpolate these projection coordinates within the triangle. Specifically, to render each scene triangle, we first compute the per-vertex-based projection coefficients (i.e., each entry in the numerator and denominator matrices) using a vertex shader. The raster then interpolates these coefficients. On the fragment shader, we then compute the projected pixel coordinates using the GLC projection equation and store them in a source texture S. Next, we need to move these pixels from S to their actual projected positions in the GLC image T. A simple solution is to map each pixel in S to a pixel in T. However, this simple mapping will introduce holes in the target GLC image, i.e., some pixels in T may not correspond to any pixels in S due to discretization. To resolve this problem, we treat every three neighboring pixels in S as a triangle and re-render these triangles onto T. Similar to frustum culling for perspective cameras, we also discard triangles that lie completely outside the image plane in the second pass. Fig. 2 shows the complete pipeline of our approach.

To maintain correct depth ordering, we also compute per-pixel ray depth. Specifically, once we compute the projected pixel coordinate $[u, v]$ of a 3D point $P(x, y, z)$ in the GLC, we calculate the ray depth d_P as $\|(x-u)^2 + (y-v)^2 + z^2\|^{1/2}$. This step can be efficiently combined with the GLC projection at the end of the first pass in the same fragment shader. When we re-rasterize the triangles from S to T in the second pass, we simply use z-buffer to resolve the visibility problem.

4.3 GLC projection equation

To render the complete 3D scene, it is important that we rasterize each scene triangle without overlapping in the first pass. We benefit from the large texture memory size on commodity graphics card. In our experiment, we use up to 1K $\times$ 1K textures and each scene triangle is set to have 10×10 image resolution. This allows us to interactively render complex scenes with over 10K triangles.

4.4 Results

We have implemented our algorithm using DirectX 9.0c with Shader model 3.0. All experiments are recorded on a PC with 2.13 GHz intel Core2 Duo CPU, 2 GB memory, and an NVidia 8800 GTX graphics card. Figure 10 summarizes the performance data. The computational overhead in our GLC rasterization pipeline is comparable to the classical perspective rendering. This is because to render a pinhole camera, all scene vertices need to be transformed via perspective transformation and tested for view frusturm culling whereas our GLC rasterization scheme separates the projection and culling in two separate passes. Our approach, however, will incur larger overhead due to context switching between the two passes. When rendering cameras using a single or a small number of GLCs, the overhead caused by context switching is negligible and our algorithm scales linearly with the resolution used to render each scene triangle. However, when rendering a large number of GLCs, this overhead can severely affect the performance of our algorithm (Sect. 6).

In Fig. 5, we render a cross-slit camera and compare the rendering result with ray tracing. We have modified the PovRay ray-tracer to support all types of GLCs. Our test scene consists of geometric primitives like spheres, cylinders, and boxes. We discretize the sphere by 260 triangles, the cylinder by 40 triangles, and the box by 12 triangles. In Fig. 5(b), we set the scene triangle resolution to be 5×5 pixels and we observe slight polygonization artifacts on the ground plane. However, since we re-render the projected pixels as triangles in the second pass, the final rendering does not contain holes. Furthermore, as we increase the resolution for rasterizing the scene triangles, the polygonization artifacts disappear.

5 Applications

5.1 Multi-perspective panoramas

Multi-perspective panoramas attempt to combine a series of views into a single image that is viewed in pieces. One way to construct multi-perspective panoramas is to stitch GLCs together. In Fig. 7(a), we approximate a circular cross-slit panorama [31] by stitching piecewise linear cross-slit GLCs. Specifically, all these GLCs share a common slit that passes through the center of the circle. We then discretize the circle using a sequence of slit segments. In Fig. 6(b), we render a 360 degree view of a cow model of 5800 triangles. We are able to fuse both the head and the tail of the cow into a single multi-perspective image. Our system renders at 47 fps and we can interactively rotate the cow as shown in the supplementary video.

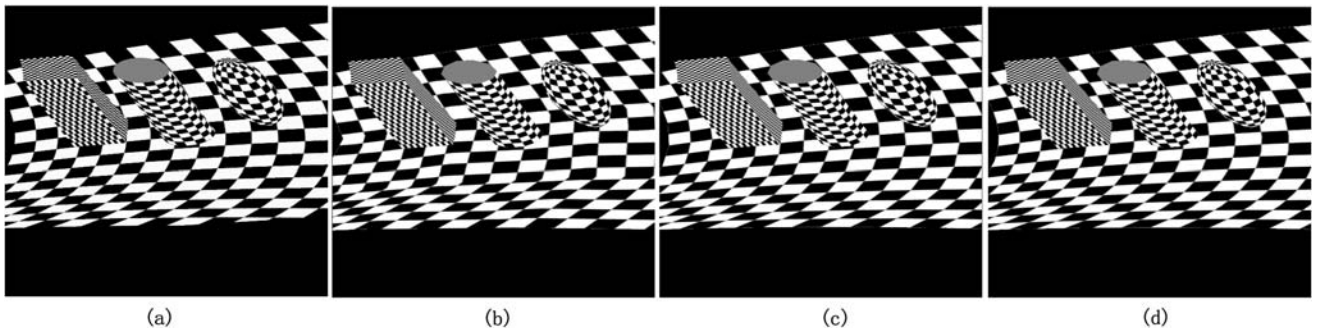

Fig. 5 (**a**) Rendering a cross-slit camera using ray tracing. (**b**)–(**d**) show the rendering result of our approach using 5 × 5 (**b**), 10 × 10 (**c**), and 20 × 20 (**d**) rasterization resolution for scene triangles. Notice at a low rasterization resolution, polygonization artifacts can occur in the final rendering (e.g., the ground plane in (**b**))

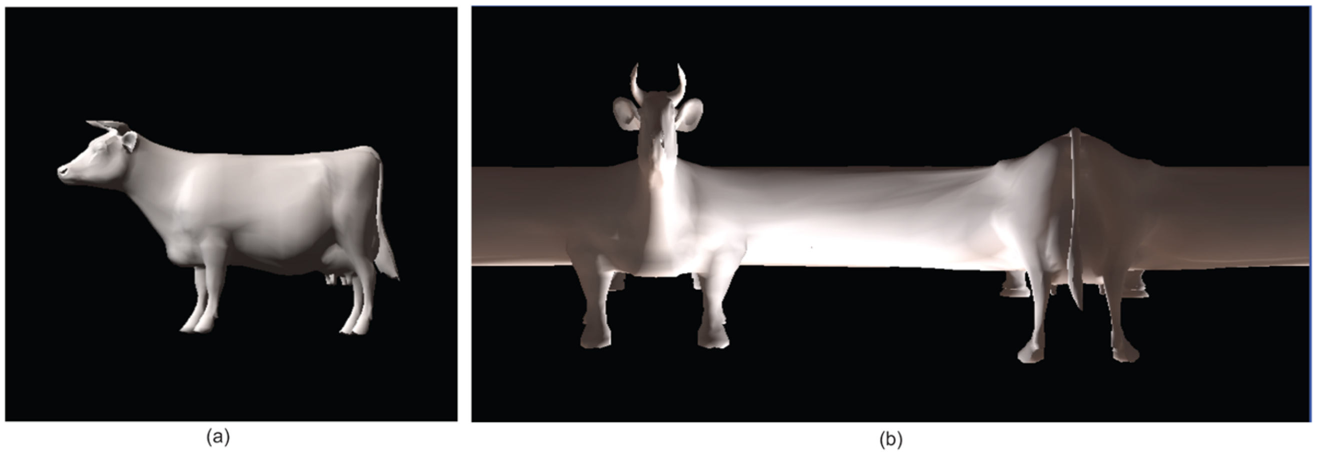

Fig. 6 Real-time rendering of 360 degree panoramas using GLC rasterization. (**a**) A perspective image of a cow model that consists of 5800 triangles. (**b**) Our panoramic rendering of the cow model at 47 fps at an 800 × 600 resolution

In Fig. 7, we render an omnidirectional view of a teapot model that consists of 14k triangles at 32 fps at a 800 × 600 resolution, reminiscent of an MCOP image [20]. The classical pinhole camera 7(b) cannot simultaneously illustrate the nose and the handle of the teapot. Using multi-perspective rendering, we can fuse both the nose and the handle in a single image 7(c) while maintaining low image distortions. Figure 7(d) shows the top view of the teapot.

5.2 Real-time reflection

A special class of multi-perspective images are reflections on curved mirrors. To render reflections, we reuse the default triangulation of the reflector and compute per-vertex reflection rays. We then treat each individual triangle as a GLC and use the triangle plane as the parametrization plane of the GLC. Finally, we render each GLC separately and stitch them to generate the final reflection image.

In Fig. 8, we render reflections from a mirror sphere. The butterfly scene contains 1100 triangles. We tessellate the sphere with 960 GLC triangles and render each GLC at a 256 × 256 resolution. We set each scene triangle to be rasterized at a 6 × 6 resolution. Our approach is able to render at 10 fps and the reflections appear highly smooth.

Since our algorithm uses two passes to render each GLC, context switching between each pass can introduce additional overhead. In particular, with very fine triangulation of the reflector (∼1K triangles), our algorithm is slower than the GPU ray-tracing approach [11] where scene triangle projection and ray tracing are combined in a single pass. The main advantage of our approach over ray tracing is that it scales well with scene complexity whereas the performance of GPU ray-tracing can significantly decrease when rendering thousands of triangles.

5.3 Multi-perspective faux-animation

Finally, it is possible to use multi-perspective rendering to create fake or faux-animations from static models by dynamically constructing the multi-perspective images. In Fig. 9, we show three frames from a synthesized animation.

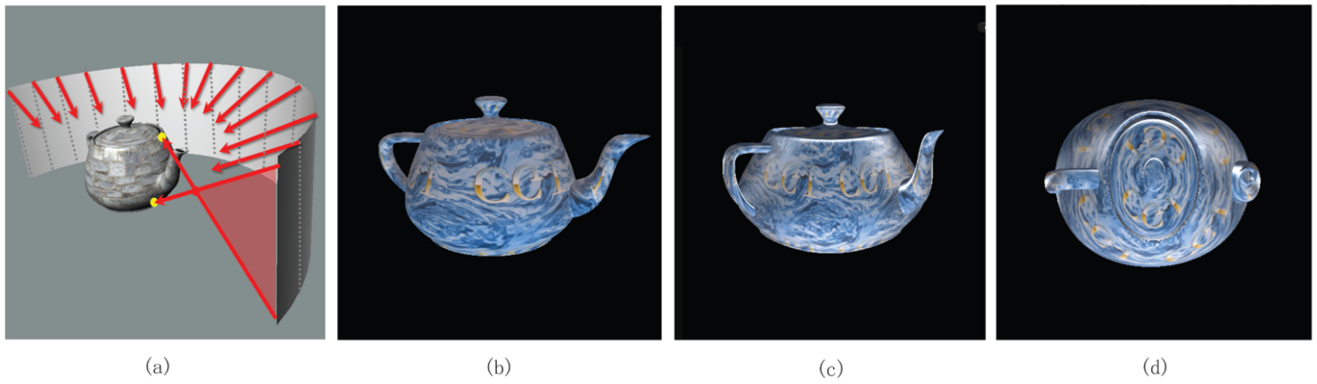

Fig. 7 (**a**) We construct the multi-perspective panorama by stitching a sequence of cross-slit cameras. (**b**) shows a perspective view of the teapot. (**b**) shows a 270 degree view of the teapot. Notice the CGI textures near the handle and side can be simultaneously seen in the panorama. (**d**) shows a top-down view of the teapot. Our system renders at 32 fps at an 800 × 600 resolution

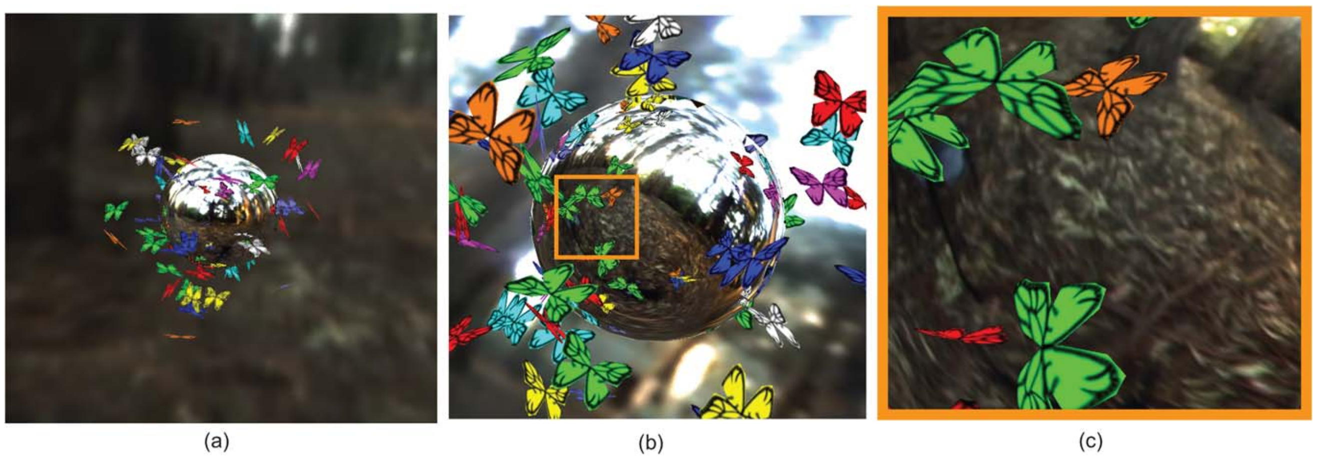

Fig. 8 Rendering reflections using the GLC rasterization framework. We decompose the mirror sphere into 960 triangles, each corresponding to a GLC. (**a**) We render the sphere in a butterfly scene at 10 fps. The rendering results (**b**) are highly smooth. (**c**) shows a close-up view of (**b**)

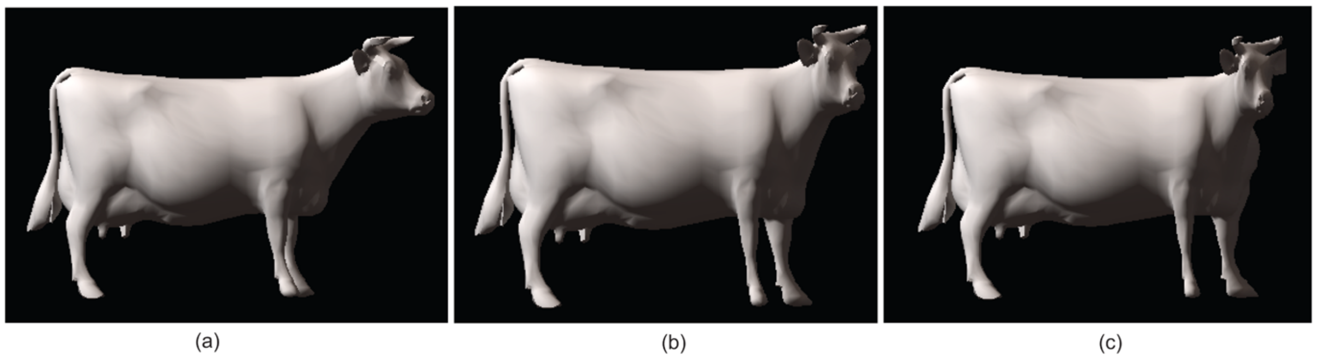

Fig. 9 (**a**) shows a perspective view of the cow model. (**b**), (**c**) Multi-perspective renderings were used to turn the head quarters of the cow in a fake animation

We first construct a multi-perspective camera composed of two GLCs. The first GLC, a cross-slit camera, captures the torso and the tail of the cow model. The second cross-slit GLC hinges on the first one and can rotate towards or away from the head of the cow. As we rotate the second camera, we are able to synthesize realistic head rotations of the cow model while maintaining continuity across the two GLC cameras. Zomet [31] used similar approach by using single cross-slit camera to achieve rotation effects from a sequence of images.

Fix Scene Complexity			Fix GLCs & Image resolution		
Scene Triangle: 1100		Resterization resolution: 6x6	# of GLCs: 480		GLC Image resolution: 256x256
# of GLCs	GLC Image resolution	fps	Scene triangle	Resterization resolution	fps
126	128x128	87.1	550	6x6	28.4
126	512x512	43.1	550	10x10	14.4
360	128x128	27.8	1100	6x6	22.4
360	512x512	16.7	1100	10x10	10.8
640	128x128	16.6	2200	6x6	9.5
640	512x512	10.4	2200	10x10	6.9

Fig. 10 The performance of our algorithm in reflection rendering. (**a**) We fix the number of the scene triangles and each triangle's rasterization resolution while changing the number of GLCs used to approximate the mirror sphere and each GLC's image resolution. (**b**) We fix the number of GLCs and the GLC image resolution while changing the scene complexity and each scene triangle's rasterization resolution

6 Discussions and future work

Our GLC-rasterization framework has several advantages. First, our method is very easy to implement: the mapping from a 3D point to its image in the GLC is unique and can be directly computed on the GPU whereas other multi-perspective cameras such as the ones used in [11] require solving higher-order polynomial equations and selecting the proper solution. Second, since we use rasterization in place of ray tracing, the performance is expected to ride in proportion with advances in commodity graphics hardware. Finally, our framework will naturally support fully dynamic scenes since it does not rely on acceleration data structures commonly used for ray tracing.

The main limitation of our framework is that it does not scale well with the number of the GLCs. In our experiments, we find that this is caused by context switching between the two passes. When rendering multi-perspective cameras composed of a large number of GLCs, the overhead due to context switching can significantly decrease the overall performance of our algorithm. Although the ray-tracing approach [11] also used multi-perspective decomposition and stitching, it is not an issue for their method as the authors combined the bounding triangle computation and ray tracing in a single pass. In the future, we plan to investigate how to combine the ray-tracing and rasterization methods into a single framework to handle both complex scenes and complex camera compositions. It may also be possible to combine our technique with existing GPU-based algorithms for rendering more sophisticated visual phenomenons such as glossy reflections, refractions and caustics under subscattering. Finally, we will explore efficient methods to extend the GLCs to support both linear projection and projection continuity.

We envision our proposed multi-perspective rendering framework will serve as conceptual inspiration for designing new graphics hardware. Specifically, we anticipate that our framework would motivate the hardware community to develop nonlinear rasterization units to support general multi-perspective rendering. Notice that any multi-perspective camera can be locally approximated by the GLCs. Therefore, if the rasterization unit can support the GLC projection model (i.e., quadratic rational functions), it may be used to directly render arbitrary multi-perspective effects.

Acknowledgements This work has been supported by the National Science Foundation under grant MSPA-MCS-0625931.

References

1. Adelson, E.H., Bergen, J.R.: The plenoptic function and the elements of early vision. In: Computational Models of Visual Processing, pp. 3–20 (1991)
2. Agrawala, M., Zorin, D., Munzner, T.: Artistic multiprojection rendering. In: Proceedings of the Eurographics Workshop on Rendering Techniques 2000, pp. 125–136 (2000)
3. Aseem, A., Agrawala, M., Cohen, M., Salesin, D., Szeliski, R.: Photographing long scenes with multi-viewpoint panoramas. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, pp. 853–861 (2006)
4. Carr, N.A., Hall, J.D., Hart, J.C.: The ray engine. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (2002)
5. Diefenbach, P.J., Badler, N.I.: Multi-pass pipeline rendering: realism for dynamic environments. In: Proceedings of the Symposium on Interactive 3D Graphics, pp. 59–ff. (1997)
6. Durand, F.: An invitation to discuss computer depiction. In: NPAR '02: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering, pp. 111–124 (2002)
7. Gascuel, J.D., Holzschuch, N., Fournier, G., Péroche, B.: Fast non-linear projections using graphics hardware. In: SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, pp. 107–114 (2008)
8. Glassner, A.S.: Cubism and cameras: Free-form optics for computer graphics. Tech. Rep. MSR-TR-2000-05, January (2000)
9. Gupta, R., Hartley, R.I.: Linear pushbroom cameras. IEEE Trans. Pattern Anal. Mach. Intell. **19**(9), 963–975 (1997)
10. Hanson, A.J., Wernert, E.A.: Image-based rendering with occlusions via cubist images. In: VIS '98: Proceedings of the Conference on Visualization '98, pp. 327–334 (1998)
11. Hou, X., Wei, L.Y., Shum, H.Y., Guo, B.: Real-time multi-perspective rendering on graphics hardware. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches, p. 79 (2006)
12. Kitamura, Y., Konishi, T., Yamamoto, S., Kishino, F.: Interactive stereoscopic display for three or more users. In: SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 231–240 (2001)

13. Levoy, M., Hanrahan, P.: Light field rendering. In: SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 31–42 (1996)
14. Lindholm, E., Kligard, M.J., Moreton, H.: A user-programmable vertex engine. In: SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 149–158 (2001)
15. Mei, C., Popescu, V., Sacks, E.: The occlusion camera, In: Proc. of Eurographics 2005. Comput. Graph. Forum. **24**(3), pp. 335–342 (Sept. 2005)
16. Ofek, E., Rappoport, A.: Interactive reflections on curved objects. In: SIGGRAPH '98: the 25th Annual Conference on Computer Graphics and Interactive Techniques, pp. 333–342 (1998)
17. Peleg, S., Ben-ezra, M., Pritch, Y.: Omnistereo: Panoramic stereo imaging. IEEE Trans. Pattern Anal. Mach. Intell. **23**, 279–290 (2001)
18. Popescu, V., Dauble, J., Mei, C., Sacks, E.: An efficient error-bounded general camera model. In: 3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), pp. 121–128. IEEE Computer Society, Washington (2006). DOI: 10.1109/3DPVT.2006.26
19. Purcell, T.J., Donner, C., Cammarano, M., Jensen, H.W., Hanrahan, P.: Photon mapping on programmable graphics hardware. In: ACM SIGGRAPH 2005 Courses, p. 258 (2005)
20. Rademacher, P., Bishop, G.: Multiple-center-of-projection images. Comput. Graph. **32**, 199–206 (1998) (Annual Conference Series)
21. Roman, A., Lensch, H.: Automatic multiperspective images. IEEE Trans. Pattern Anal. Mach. Intell. **25**, 741–754 (2003)
22. Roman, A., Garg, G., Levoy, M.: Interactive design of multi-perspective images for visualizing urban landscapes. In: VIS '04: Proceedings of the Conference on Visualization '04, pp. 537–544. IEEE Computer Society, Washington (2004)
23. Seitz, S.M., Kim, J.: Multiperspective imaging. IEEE Comput. Graph. Appl. **23**(6), 16–19 (2003)
24. Shum, H.Y., He, L.W.: Rendering with concentric mosaics. In: SIGGRAPH '99: the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 299–306 (1999)
25. Simon, A., Smith, R.C., Pawlicki, R.R.: Omnistereo for panoramic virtual environment display systems. In: VR '04: Proceedings of the IEEE Virtual Reality 2004, p. 67 (2004)
26. Swaminathan, R., Grossberg, M.D., Nayar, S.K.: Non-single viewpoint catadioptric cameras: Geometry and analysis. Int. J. Comput. Vis. **66**(3), 211–229 (2006)
27. Wood, D.N., Finkelstein, A., Hughes, J.F., Thayer, C.E., Salesin, D.H.: Multiperspective panoramas for cel animation. In: SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 243–250 (1997)
28. Wyman, C., Davis, S.: Interactive image-space techniques for approximating caustics. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, pp. 153–160 (2006)
29. Yu, J., McMillan, L.: A framework for multiperspective rendering. In: Rendering Techniques (2004)
30. Yu, J., McMillan, L.: General linear cameras. In: the 8th European Conference on Computer Vision (ECCV) (2004)
31. Zomet, A., Feldman, D., Peleg, S., Weinshall, D., Society, I.C.: Mosaicing new views: The crossed-slits projection. IEEE Trans. Pattern Anal. Mach. Intell. **25**, 741–754 (2003)

Xuan Yu is a graduate student in Computer and Information Sciences Department at the University of Delaware. He received his B.Sc. from Shanghai Jiaotong University and is currently pursuing his Ph.D. with Dr. Jingyi Yu at the University of Delaware.

Jingyi Yu is Assistant Professor in Computer and Information Sciences Department at the University of Delaware. He received his B.Sc. from Caltech in 2000, and M.Sc. and Ph.D. degrees in EECS from MIT in 2005. His research interests span a range of topics in computer graphics, computer vision, and image processing, including computational photography, medical imaging, non-conventional optics and camera design, tracking and surveillance, and graphics hardware.

Leonard McMillan is Associate Professor of Computer Science at the University of North Carolina in Chapel Hill. Leonard received his Bachelor's (1983) and Master's (1984) degrees in Electrical Engineering from Georgia Institute of Technology. Leonard received his Ph.D. in Computer Science from the University of North Carolina at Chapel Hill (1997). Leonard has been a Member of Technical Staff at AT&T Bell Laboratories where he worked in the Digital Signal Processing Architecture Group and was a coarchitect of the AT&T Pixel Machine. Leonard has also worked as a Senior Staff Engineer at Sun Microsystems where he helped developing several visualization and multimedia products. Leonard is a pioneer in the field of image-based rendering. Image-based rendering is a new approach to computer graphics where scenes are rendered directly from a collection of reference images rather than a geometric model. Leonard is also interested in a wide range of related topics, including computer graphics rendering, imaging methods and technologies, three-dimensional display technologies, computer graphics hardware, and the fusion of image processing, multimedia, and computer graphics.