Vol. 7, No. 4: 33–42

Hardware-Assisted Rendering of Cylindrical Panoramas

Dongho Kim and James K. Hahn
The George Washinigton University

Abstract. Cylindrical panorama is being used in many applications. Cylindrical panoramic viewers render the panorama from the center of the cylinder by projecting the cylindrical map onto a planar screen. This process involves nonlinear image warping, so many panoramic viewers are implemented in software. Hardware acceleration may be used if the panorama is resampled onto the polygonal models.

This paper presents an algorithm to render cylindrical panorama with hardware acceleration while using the input panorama as is. The rendering equation of cylindrical panorama is decomposed into linear approximation and nonlinear residuals. Nonlinear parts are encoded as bump maps to perturb the texture coordinates on a per-pixel basis. This process can be performed by environment-mapped bump mapping (EMBM) hardware.

1. Introduction

Cylindrical panorama viewers, as popularized by QuicktimeVR, are very popular. They typically use software rendering to perform the nonlinear cylindrical image warping, and as such are limited by CPU speed and occupy the CPU, preventing simultaneous application processing. [Chen 95], [McMillan, Bishop 95], [Shum, He 99], [Szeliski, Shum 97], [panoguide xx].

In this paper, we describe a new method of hardware-assisted rendering of cylindrical panorama. It implements nonlinear image warping using bump mapping hardware. Therefore, high-resolution cylindrical panorama can be rendered efficiently without intensive use of the CPU.

© A K Peters, Ltd.

1086-7651/02 $0.50 per page

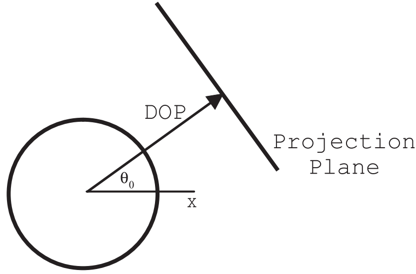

Figure 1. Projection plane and direction of projection (DOP).

2. Main Idea

Rendering with cylindrical panorama is a process of warping a cylindrical texture map onto a plane. First, we define the *projection plane*, onto which a cylindrical map is warped locally on-the-fly. The projection plane lies parallel with the axis of the cylinder. Second, we define the *direction of projection* (DOP), which becomes the normal vector of the projection plane. As users specify the viewing direction, DOP is set as the horizontal component of viewing direction. Because DOP is perpendicular to the cylinder, it can be defined simply by θ_0, which is the angle of DOP from the x-axis. Then, the projection plane is placed at unit distance from the origin with the normal vector along DOP. Figure 1 shows a two-dimensional view of this configuration.

From now on, (θ, v) denotes the cylindrical texture coordinates in [0, 1] for the cylindrical panorama, and (u', v') is the planar texture coordinates on the projection plane. The three-vector (x, y, z) is a location in the projection coordinate system, where the origin is at the center of the cylinder. The z-axis is aligned with the axis of the cylinder, and the x-axis is aligned with $\theta = 0$.

2.1. Decomposition of Rendering Equation

Let H be the height of the unit cylinder given as $\mathrm{H} = 2\tan(0.5 \times \mathrm{FOV})$, where FOV is the vertical field of view of the cylindrical panorama. Then, (θ, v) is related to (x, y, z) as given in Equation 1. This is the rendering equation of cylindrical panorama. Here, atan2() is the arctangent function in a standard C library, which gives the angle in [$-\pi..\pi$] for output.

$$\begin{aligned} \theta &= \frac{\mathrm{atan2}(y, x)}{2\pi} + 0.5 \\ v &= \frac{1}{\mathrm{H}} \frac{z}{\sqrt{x^2 + y^2}} + 0.5 \end{aligned} \qquad (1)$$

Suppose that (x', y') is a projection of (x, y, z) onto the local coordinates of the projection plane. Because the projection plane is placed at unit distance from the center of the cylinder, Equation 1 becomes the following equation which determines $(\theta, v)$for a given (x', y').

$$\begin{aligned} \theta &= \theta_0 + \frac{\text{atan}(x')}{2\pi} \\ v &= \frac{1}{\text{H}} \frac{y'}{\sqrt{1 + x'^2}} = 0.5 \end{aligned} \tag{2}$$

The main idea of our work is to decompose these equations into a linear approximation and nonlinear residuals, and fill the bump map with the nonlinear part. The bump map is used to perturb texture coordinates determined by the linear part. For this process, Equation 2 is decomposed as follows.

$$\begin{aligned} \theta &= \theta_0 + ax' + R_\theta(x') \\ v &= 0.5 = by' + R_v(x', y') \end{aligned} \tag{3}$$

Therefore, the residual parts are given as follows.

$$\begin{aligned} R_\theta(x') &= \frac{\text{atan}(x')}{2\pi} - ax' \\ R_v(x', y') &= \frac{1}{\text{H}} \frac{y'}{\sqrt{1 + x'^2}} - by' \end{aligned} \tag{4}$$

Here, a and b are the coefficients of the linear approximations of (θ, v) according to (x', y'). They are computed before rendering so that the maximum magnitudes of the residual parts are minimized.

2.2. Computation of Linear Coefficients

Suppose x' is in $[-X_m, X_m]$ and y' is in $[-Y_m, Y_m]$. These ranges are determined by the possible values of horizontal and vertical fields of view during real-time rendering. For instance, if we always use the fields of view less than $90°$, the ranges are $[-1, 1]$ and $[-1, 1]$ for x' and Ty', respectively, since $\tan(0.5 \times 0.5\pi)$ is 1.

Figure 2(a) shows the first and second terms of $R_\theta(x')$ in Equation 4. While we change the slope a, the difference of the two terms is minimal, when $R_\theta(X_c)$and $R_\theta(X_m)$ have the same magnitude and opposite signs. This is the first condition. X_c is obtained by setting the derivative of $R_\theta(x')$ to zero, which becomes the second condition. Therefore, we have two unknowns, X_c and a, and two equations. The equations can be solved by substitution,

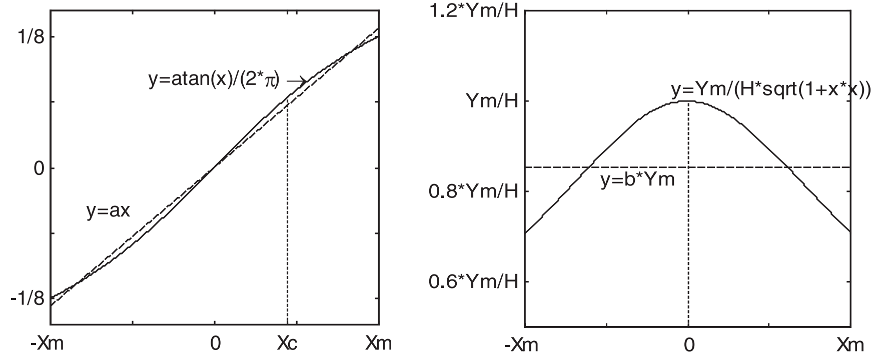

Figure 2. Determination of linear coefficients.

followed by a few iterations of a simple bisection root-finding algorithm [Press et al. 88].

For $R_v(x', y')$ in Equation 4, it is apparent that the magnitude is at a maximum when y' is equal to Y_m. Figure 2(b) is the plot of the two terms of $R_v(x', Y_m)$. Similar to $R_\theta(x')$, the maximum magnitude of $R_v(x', Y_m)$ is minimal, when $R_v(0, Y_m)$and $R_v(X_m, Y_m)$ have the same absolute values and opposite signs. This gives the equation to solve for b.

3. Implementation

Recent improvements in texture mapping hardware introduced multitexture mapping. With multitexture mapping, two or more textures can be applied to a polygon in a single rendering pass., while a texture is applied at each stage. The result of the previous stage and the texture for the current stage can be combined by various operations, such as add, subtract, multiply, etc.

One of the operational modes of multitexturing is environment mapped bump mapping (EMBM), where texture coordinates in the second stage are perturbed by the texel values sampled in the first stage. This functionality is designed originally for bumped reflection, where the environment is reflected using environment mapping. In this work, however, this functionality is used to perturb texture coordinates and perform nonlinear warping with hardware support. Because residual parts shown in Equation 4 are approximated as a bump perturbation map, the resolution of the bump map may affect the quality of projective texturing. A 256×256 bump map is used in this work, and there are no noticeable artifacts due to this approximation.

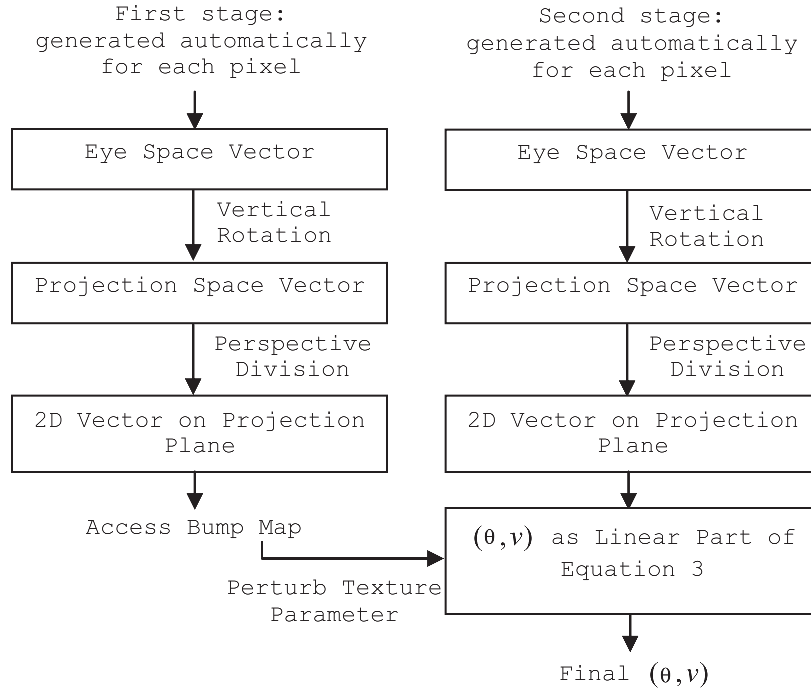

Figure 3. Determination of texture coordiantes using projective texture mapping.

Our algorithm can be implemented in one of two ways as explained in the next two subsections. Although DirectX 8.1 is used for the implementation, OpenGL could also be used.

3.1. Implementation Using Projective Texture Mapping

Rendering of a cylindrical panorama can be accomplished by rendering a rectangle for the screen with a multitexture configuration as shown in Figure 3. This screen rectangle is always attached to the viewing frustum and rotated while the user changes the viewing direction.

For the first texture stage, the texture operation is set as EMBM and a bump perturbation map is used as a texture. The bump map contains the amounts of perturbation needed for all texels, which are computed from the residual parts of cylindrical parameterization given in Equation 4. For the second texture stage, the cylindrical panorama is used as texture. In Figure 3, a projection space means a coordinate system, where the x and y axes are in the projection plane and the z axis is in the direction of DOP. For

both stages, texture coordinates are generated as eye space coordinates by automatic texture generation, and transformed appropriately by the texture transformation matrix. Most of the procedure is similar to standard projective texture mapping [Blythe et al. 00], [Weinhaus, Devich 99]. With projective texture mapping, eye space coordinates are transformed to texture space coordinates and the x and y components are divided by the z component, in order to obtain 2D coordinates. In our case, perspective division in projection space computes (x', y'), the coordinates on the projection plane. Note that the entire process is performed on a per-pixel basis by graphics hardware.

The texture transformation matrix for the first stage (M_0) and the second stage (M_1) are set as given in Equation 5. These matrices are multiplied by the eye coordinate vector for each pixel, so that following perspective division can give (x', y'). In Equation 5, M_{rot} is the vertical rotation matrix, which transforms from the eye coordinates to the projection space coordinates. Note that a vertex is represented as a row vector and the transformation matrix is post-multiplied in DirectX. Minus signs appear at the (2,2) elements in both matrices, because textures are addressed from the top in DirectX.

$$M_0 = M_{rot} \cdot \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & -b & 0 & 0 \\ \theta_0 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_1 = M_{rot} \cdot \begin{bmatrix} \frac{1}{2X_m} & 0 & 0 & 0 \\ 0 & -\frac{1}{2Y_m} & 0 & 0 \\ 0.5 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Figure 4 shows a part of the DirectX source code for this configuration. This implementation works well with very recent graphics hardware, such as nVIDIA GeForce4 Ti 4600 and ATI Radeon 8500, as well as DirectX reference rasterizer. However, we found that some 3D graphics hardware have problems with this implementation. For instances, nVIDIA GeForce3 does not work correctly when the texture coordinates obtained by projective texture mapping are perturbed in the next stage. ATI Radeon cannot perform projective texture mapping correctly when both stages use projective texture mapping. It seems that some hardware did not implement the entire functionality because this usage is beyond the scope of its original objective, which is bumped environment mapping.

3.2. Implementation with Per-Vertex Projection

While the previous implementation is limited to recent graphics hardware, the second rendering method presented here can be used with any graphics

```
pd3dDev->SetTexture(0, m_pBumpMap);
pd3dDev->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_BUMPENVMAP);
pd3dDev->SetTextureStageState(0, D3DTSS_TEXCOORDINDEX,
D3DTSS_TCI_CAMERASPACEPOSITION);
pd3dDev->SetTextureStageState(0, D3DTSS_TEXTURETRANSFORMFLAGS,
D3DTTFF_COUNT3 | D3DTTFF_PROJECTED);

pd3dDev->SetTexture(1, m_pPanorama );
pd3dDev->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_SELECTARG1);
pd3dDev->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_TEXTURE);
pd3dDev->SetTextureStageState(1, D3DTSS_TEXCOORDINDEX
 D3DTSS_TCI_CAMERASPACEPOSITION);
pd3dDev->SetTextureStageState(1, D3DTSS_TEXTURETRANSFORMFLAGS,
D3DTTFF_COUNT3 | D3DTTFF_PROJECTED);

D3DXMATRIX mat;
... /* matTexture is set as M0 in Eq 5 */
pd3dDev->SetTransform(D3DTS_TEXTURE0, &matTexture);
... /* matTexture is set as M1 in Eq 5 */
pd3dDev->SetTransform(D3DTS_TEXTURE1, &matTexture);
```

Figure 4. Source code for multitexture configuration.

hardware with EMBM capability. Moreover, the performance is better than the previous method, since it does not require per-pixel projective texture mapping.

In this implementation, projective texture mapping is applied explicitly for each vertex of the screen rectangle before rendering takes place. Then, the resultant 2D texture coordinates are used for texture mapping. Therefore, there is no need to use per-pixel projective texture mapping. For the four vertices of the rectangle, transformation matrices in Equation 5 are multiplied and perspective division is performed. This gives 2D texture coordinates for the two stages defined on the projection plane.

But correct results cannot be obtained if the screen rectangle is rendered as is. This is because 2D texture coordinates are distorted perspectively. In other words, the mapping is not linear inside the rectangle. In order to get correct texture mapping inside the rectangle, the rectangle should also be projected onto the projection plane. If the viewing direction is not horizontal, the rectangle is projected to a trapezoid. Rendering of this trapezoid gives the same results as the method in Section 3.1, since the 2D geometry and texture coordinates are defined on the same two-dimensional domain, which is the projection plane. For this implementation, texture transforms should be disabled from the source code in Figure 4, and the locations and texture coordinates are pretransformed per vertex.

3.3. Zoom In/Out

Zooming in can be implemented by reducing the field of view for real-time rendering. Similarly, the enlargement of the field of view gives zooming out. The screen rectangle should be scaled appropriately.

4. Dynamic Range of Bump Map

For the texture format of bump maps, DirectX provides 8-bit format (D3DFMT_V8U8) and 16-bit format (D3DFMT_V16U16). If the 16-bit format is supported by the hardware, the perturbation values in [-1, 1] are represented by 16 bits for u and v, respectively. Therefore, perturbation can be represented in much detail.

Currently, ATI Radeon 8500 is the only hardware with the support of D3DFMT_V16U16. An 8-bit bump map can represent the values of only 256 levels in the [-1, 1] range. In order to utilize this dynamic range effectively, DirectX provides the scaling of bump map texels. This is done by setting D3DTSS_BUMPENVMAT00 and D3DTSS_BUMPENVMAT11 parameters with the SetTextureStageState() API. In other words, the perturbation values are scaled up when they are encoded in the bump map, and they are scaled down to the original values when they are used during rendering. Therefore, 8-bit bump maps can be used effectively if the magnitudes of perturbation values are small. This is the reason why optimal linear coefficients should be found in Section 2.2.

5. Results and Discussion

With the implemented panoramic viewer, the user can look around with horizontal and vertical rotations. Zooming in and out is also supported. Figure 5(b) shows the rendering results from the panorama given in Figure 5(a). Rendering performance is shown in Figure 6. In this figure, #1 indicates the implementation in Section 3.1 and #2 indicates the method in Section 3.2. R1 through R5 represent the resolutions of $640 \times 480, 800 \times 600, 1024 \times 768, 1280 \times 960$, and 1600×1200, respectively. GeForce3, GeForce4 Ti 4600, and Radeon 8500 were used for the test.

Errors are introduced by using a look-up table. We define the error as the Euclidean distance between the accurate texture coordinates and the approximated values in 2D texture space with range [0, 1] . This can be measured by rendering the texture with color-encoded texture coordinates. Then, the measured average per-pixel error is about 8.66×10^{-6}. Because the bump map is interpolated linearly, increasing the bump map resolution does not reduce the error significantly.

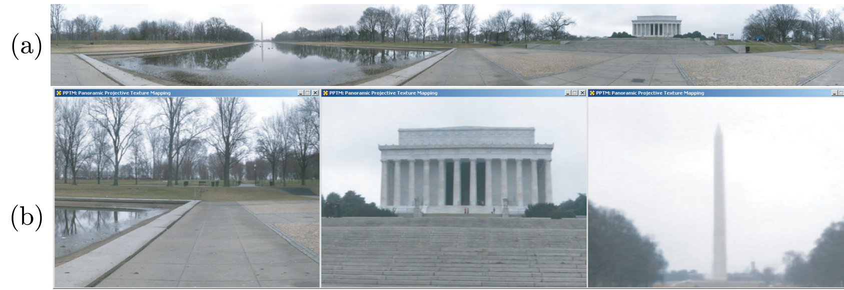

Figure 5. Result of cylindrical panoramic rendered.

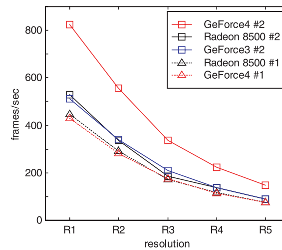

Figure 6. Rendering performance.

We believe that hardware-based nonlinear warping could overcome the limitation of texture mapping hardware that the texture coordinates are interpolated only linearly. Therefore, it could be used in many real-time applications.

Acknowledgements. This work was supported in part by GW Presidential Fellowship and ONR Grant N000140110582.

References

[Blythe et al. 00] D. Blythe et al. “Advanced Graphics Programming Techniques using OpenGL.” *Siggraph 2000 Course Notes*, New York: ACM SIGGRAPH, 2000.

[Chen 95] S.E. Chen. “Quicktime VR—An Image-Based Approach to Virtual Environment Navigation.” In *Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series*, edited by Robert Cook, pp. 29–38, Reading, MA: Addison-Wesley, 1995.

[McMillan, Bishop 95] L. McMillan and G. Bishop. “Plenoptic Modeling: An Image-Based Rendering System.” In *Proceedings of SIGGRAPH 95, Computer*

Graphics Proceedings, Annual Conference Series, edited by Robert Cook, pp. 39–46, Reading, MA: Addison-Wesley, 1995.

[Shum, He 99] H. Y. Shum and L. W. He. "Rendering with Concentric Mosaics." In *Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series*, edited by Alyn Rockwood, pp. 299–306, Reading, MA: Addison-Wesley, 1999.

[Szeliski, Shum 97] R. Szeliski and H.Y. Shum, "Creating Full Panoramic Mosaics and Environment Maps." In *Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series*, edited by Turner Whitted, pp. 251–258, 1997.

[Weinhaus, Devich 99] F. M. Weinhaus and R. N. Devich. "Photogrammetric Texture Mapping onto Planar Polygons." *Graphical Models and Image Processing* 61: 2 (1999), 63–83.

[Press et al. 88] W.H. Press et al. *Numerical Recipes in C.* Cambridge, UK: Cambridge University Press, 1988.

[Microsoft xx] Microsoft. *DirectX 8.1 Programmer's Manual*, xxxx.

[panoguide xx] Available from World Wide Web (http://www.panoguide.com/). xxxx

Web Information:

http://www.acm.org/jgt/papers/KimHahn02

Dongho Kim, Department of Computer Science, The George Washington Unviersity, Washington, DC (dkim@gwu.edu)

James K. Hahn, Department of Computer Science, The George Washington Unviersity, Washington, DC (jhahn@gwu.edu)

Received April 2002; accepted September 2002.