9.5

A GPU Panorama Viewer for Generic Camera Models

Frank Nielsen, **Sony Computer Science Laboratories, Inc.**

Introduction

Digital panoramas are nowadays omnipresent in Internet virtual tours (see *http://www.world-heritage-tour.org/*). A *panorama* basically stores light information arriving at a single position from a wide field of view. Panoramas are particular light fields that conveniently sample the plenoptic function in image-based rendering systems at discrete positions. We distinguish between *spherical panoramas,* which cover the full sphere of directions (4π steradians), from *cylindrical panoramas,* which cover only 360 horizontal degrees over a partial vertical field of view. Panoramas are widely used in computer graphics not only as backdrops (also called skyboxes), but also as *environment maps* for dynamic reflections and more recently as *light probes* for the rendering of convincing lighting. In the old days, environment maps were merely captured using a tele-lens camera (approximating the required orthographic projection), capturing the reflections of a spherical mirror ball [Nielsen05a]. Nowadays, stitching allows us to calibrate and precisely register a sequence of pictures acquired from a same *nodal point*: the *center of projection* (COP) [Nielsen05b]. Thus, an environment map is a *complete panorama ray map* obtained from a single COP, while a pinhole image is interpreted as a *local ray map* partially sampling the environment map from the same COP. The most common environment maps used for simulating real-time reflections in computer graphics are the latitude-longitude map (also called the equirectangular map), the cubic map (six quad faces), and the dual paraboloid map (two images) [Nielsen05a]. Wong et al. further introduced the HEALPix map (12 quads) [Wong05], which improves the spherical ray sampling distribution over the cube map for computing real-time reflections on the GPU.

Figure 9.5.1 depicts a 1024×512 latitude-longitude spherical environment map.

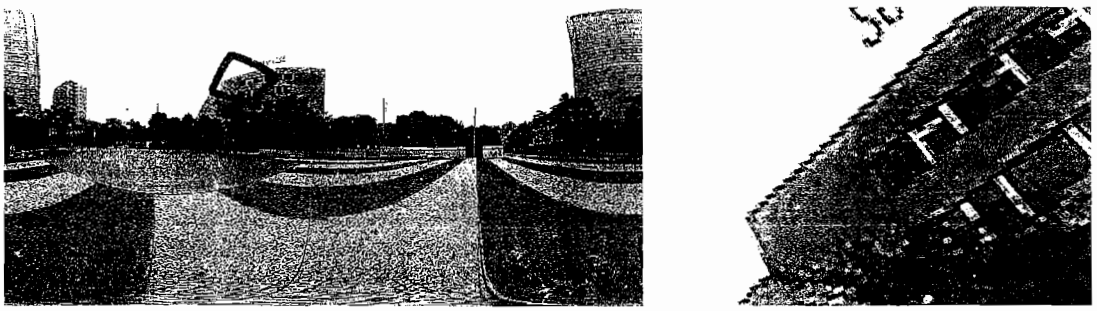

FIGURE 9.5.1 Example of (**A**) a latitude-longitude environment map from which (**B**) a virtual pinhole camera image is synthesized. The corresponding border of the virtual pinhole camera is traced in the environment map.

Synthesizing Pinhole Camera Views

To render a view as if obtained by a virtual pinhole camera anchored at the same COP of the environment map, we need to partially remap the ray map using a pinhole camera model. The pinhole camera is defined by a set of *extrinsic parameters* (roll, pitch, and yaw attributes stored in a rotation matrix and defining the aim and orientation of the camera image plane) and *intrinsic parameters* (image dimension, principal point, and focal length). Remapping, a *warping* operation, can either proceed by mapping forward pixels of the environment map to the virtual pinhole camera image (*forward mapping*), or vice-versa (*backward mapping*). This local ray map conversion can be carried out using intensive per-pixel CPU computations, or per-triangle units using the texture primitives of graphics cards (2D or 3D triangles), but this is time consuming. On one hand, per-pixel warping offers the highest picture quality by explicitly controlling the interpolation scheme. On the other hand, texturing triangles allows us to relieve the CPU from excessive computations by leaving the texturing operation of interpolating intermediate values to the graphics engine. The drawback is that texturing uses the standard (trilinear) interpolation scheme. Let us now quickly review the per-pixel backward mapping and the 2D/3D per-triangle forward mapping method before introducing the GPU panorama shader.

A Simple Pinhole Camera Model

The pinhole camera model of image dimension width w and height h and (horizontal) field of view (hfov) maps image pixels to 3D rays anchored at the COP as follows:

$$PinholeXY2TP(x,y) = \left(\arctan\left(\frac{x-c_x}{f}\right), \arctan\left(\frac{y-c_y}{a\sqrt{(x-c_x)^2+f^2}}\right)\right) = (\theta,\phi),$$

where $(c_x, c_y) = \left(\frac{w}{2}, \frac{h}{2}\right)$ is the camera principal point, $a = \frac{h}{w}$ is the *aspect ratio,* and f is the (horizontal) focal length in pixel units computed from the field of view as:

$$f = hfov2fx(w, hfov) = \frac{w}{2\tan\frac{hfov}{2}}$$

Conversions between 3D Cartesian and 2D spherical coordinates as shown in Figure 9.5.2 are processed using the conventional formula:

$$Cartesian2Spherical(X, Y, Z) = \left(\arctan\left(\frac{X}{Z}\right), \arctan\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right)\right) = (\theta, \phi)$$

$$Spherical2Cartesian(\theta, \phi) = \left(\sin\theta\cos\phi, \sin\phi, \cos\theta\cos\phi\right) = (X, Y, Z)$$

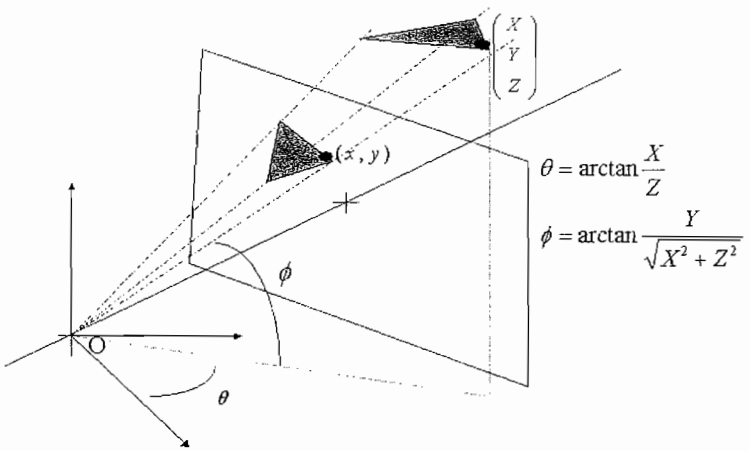

FIGURE 9.5.2 Conversion between spherical and Cartesian coordinates.

Per-Pixel Backward Ray Mapping

We first need to align the principal point of the camera image (image center) with the aim of the camera using a rotation matrix defined as

$$R(r, p, y) = \begin{pmatrix} \cos r\cos y - \sin r\sin p\sin y & -\sin r\cos p & \cos r\sin y + \sin r\sin p\cos y \\ \sin r\cos y + \cos r\sin p\sin y & \cos r\cos p & \sin r\sin y - \cos r\sin p\cos y \\ -\cos p\sin y & \sin p & \cos p\cos y \end{pmatrix},$$

where the roll denotes the angle around the z-axis, the pitch denotes the angular amplitude around the x-axis, and the yaw denotes the inclination around the y-axis. Then we create a *synthetic* pinhole camera view by looking up for each pinhole image pixel xy the corresponding $\theta\phi$ ray, rotating that ray and the original orthonormal frame using the rotation matrix, and finding the corresponding pixel in the environment ray map using the `LatitudeLongitudeTP2XY` function:

$$LatitudeLongitudeTP2XY(\theta,\phi)=\left(w_p\left(\frac{\theta+\pi}{2\pi}\right),h_p\left(\frac{\phi+\pi/2}{\pi}\right)\right),$$

where w_p and h_p denote, respectively, the equirectangular panorama width and height.

Interpolation schemes can be chosen as the nearest neighbor interpolant for speed or bilinear or better interpolation methods (e.g., Lanczos). Listing 9.5.1 shows the source code.

LISTING 9.5.1 Source Code

```
CPUPanoramaViewer.cpp
indexpi=0;
for(y=0;y<hpi;y++)
    for(x=0;x<wpi;x++)
    {
     xy[0]=x;xy[1]=y;

    PinholeXY2TP(xy,tp);
    Spherical2Cartesian(tp,xyz);
    Rotation(xyz,R,xyzrot);
    Cartesian2Spherical(xyzrot,tp);
    LatitudeLongitudeTP2XY(tp,xy);
    // Nearest interpolation scheme for compactness
    xx=(int)xy[0];yy=(int)xy[1];
    indexll=3*(yy*widthpan+xx);

    pinholeimage[indexpi++]=environmentmap[indexll++];// R
    pinholeimage[indexpi++]=environmentmap[indexll++];// G
    pinholeimage[indexpi++]=environmentmap[indexll++];// B
    }
```

ON THE CD Please refer to subfolder CPU Panorama viewer in this article's folder on the CD-ROM.

Per-Triangle 3D Forward Ray Mapping

To speed up the rendering process, we may compute the ray conversion at sparse positions, and let the graphics engine render the textured primitives. A typical example is rendering a 3D unit sphere using texture coordinates of the environment map. We represent the environment map (θ,ϕ) as a grid on a regular mesh and compute the 3D vertex position on the sphere for each 2D grid vertex using the `Spherical2Cartesian` procedure. Deciding whether a triangle is to be rendered or not, and clipping partially visible triangles, is handled by the graphics engine.

Per-Triangle 2D Forward Ray Mapping

Another strategy consists in rendering 2D triangles and determining for ourselves the out-of-view triangles. That is, once we get the 3D triangle vertex positions, we find the angular parameters (θ',ϕ') and project back to the screen space using the `PinholeTP2XY` primitive:

$$PinholeTP2XY(\theta,\phi) = (f\tan\theta + c_x, a\sqrt{(x-c_x)^2}\tan\phi + c_y) = (x,y)$$

Although the vertex positions are precisely computed, the barycentric 2D triangle interpolation does not produce a perfectly correct result. However, this is quite unnoticeable, and the interpolation approximation error decreases as we refine the triangulation. Note that for textured primitives, backward mapping as shown in Figure 9.5.3 would be challenging, as some 2D *xy* triangles of the pinhole camera image may be cut into several parts in the environment mapping (for example, a triangle containing the latitude-longitude north pole in its interior).

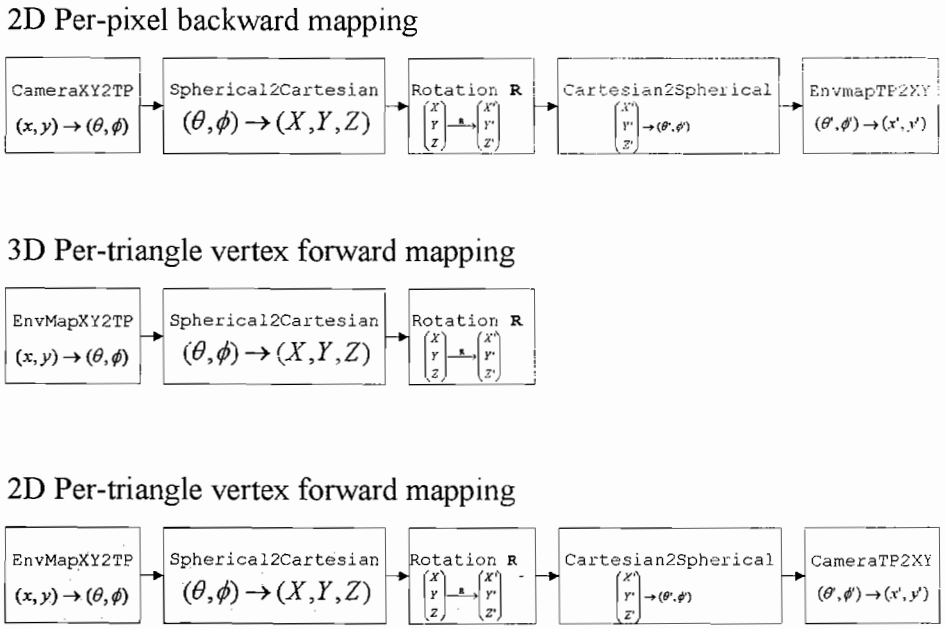

FIGURE 9.5.3 Backward per-pixel and forward per-triangle coordinate pipelines.

A GPU Fragment Shader

In this section, we describe the ray map conversion by a short fragment shader. The parameters of the shader are the rotation matrix and the texture image dimension. Using the shader, we can render at full-screen resolution with maximal frame rate (usually 60 fps, but this may vary according to your monitor's refresh rate).

Zooming in or zooming out is achieved by decreasing or increasing the field of view, which impacts the focal length. This can be implemented using another shader parameter that we omitted here for simplicity. Moreover, rotational motion blur effect can be added purposely using the OpenGL accumulation buffer [Nielsen05b]. Listing 9.5.2 is an excerpt of the file pinhole.cg.

LISTING 9.5.2 Excerpt from pinhole.cg

```
samplerRECT PanoramaImage;
float widthpan,heightpan;
float3x3 R;
float PI=3.14159265;

float2 Cartesian2Spherical(float3 p)
{
float2 tp;

tp[1]=atan(p[1]/sqrt(p[0]*p[0]+p[2]*p[2]));
tp[0]=atan2(p[0],p[2]);

return tp;
}

float3 Spherical2Cartesian(float2 tp)
{
float3 xyz;

xyz[0]=cos(tp[1])*sin(tp[0]);
xyz[1]=sin(tp[1]);
xyz[2]=cos(tp[1])*cos(tp[0]);

return xyz;
}

// Pinhole X-Y -> Theta-Phi -> Panorama X-Y
float2 PinholeXY2TP(float x,float y)
{
// focal length in pixel unit
float f=1000;
// principal point
float cx=widthpan/2.0;
float cy=heightpan/2.0;
// aspect ratio
float aspect=heightpan/widthpan;
float t,p;

t=atan2(x-cx,f);
p=atan2((y-cy)/aspect,sqrt((x-cx)*(x-cx)+f*f));

return float2(t,p);
}
```

```
// Environment map
float2 LatitudeLongitudeTP2XY(float t, float p)
{
float x,y;

x=widthpan*((t+PI)/(2.0*PI));
y=heightpan*((p+PI/2.0)/PI);

return float2(x,y);
}

// Receives RPY in matrix R and warp accordingly
// Backward mapping: Pinhole->Environment mapping
float3  WarpPanorama(float2 texcoord : TEXCOORD0) : COLOR0
{
float3 pp,xyz;
float2 tp,xy;

tp=PinholeXY2TP(texcoord[0],texcoord[1]);
xyz=Spherical2Cartesian(tp);
pp=mul(R,xyz);
tp=Cartesian2Spherical(pp);
xy=LatitudeLongitudeTP2XY(tp[0],tp[1]);

return f3texRECT(PanoramaImage, xy);
}
```

Using the same conversion framework, we can also remove the radial lens distortion effects using the GPU. Let us consider Tsai's radial distortion model [Nielsen05b]. We simply need to define the primitive `TsaiXY2TP`, which we do by first remapping the (distorted) source image into an undistorted ideal pinhole image and then applying the regular `PinholeXY2TP` transformation. Please refer to subfolder GPUPanoramaViewer1 on the CD-ROM.

Generic Camera Models

A generic camera model (yielding either a partial or complete environment map) is defined concisely using two change-of-coordinate functions: `genericcameraXY2TP` and `genericcameraTP2XY`. These functions are potentially partially defined. For example, the fisheye camera only (re)projects the environment mapping onto an image disk (undefined elsewhere in the rectangular image). Also noteworthy, the origin and axis of the environment map can be readjusted using the GPU by specifying the new origin and frame axes using the rotation matrix. To illustrate the generic camera functions, let us consider the conversion of the latitude-longitude equirectangular map to the front face of the dual paraboloid using the function `ParaboloidUpXY2TP`. A normalized pixel (x, y) (with $x \in [-1,1]$ and $y \in [-1,1]$) in the front paraboloid maps to a corresponding downward ray direction defined by the following 3D vector:

$$\left(\frac{2x}{x^2+y^2+1}, \frac{2x}{x^2+y^2+1}, \frac{x^2+y^2-1}{x^2+y^2+1}\right)$$

We then need to simply apply the `Cartesian2Spherical` function to retrieve the corresponding (θ,ϕ) angles. Because we use both back and front paraboloid maps to define a complete environment map, it is enough to consider normalized pixels falling within the unit disk:

```
float3 ParaboloidUpXY2TP(float2 xy)
{
float s,t,X,Y,Z;
float3 tpz;

s=(xy[0]-(widthpan/2.0))/(widthpan/2.0);
t=(xy[1]-(heightpan/2.0))/(heightpan/2.0);

if (s*s+t*t<=1.0)
    {
    X=2.0*s/(s*s+t*t+1.0);
    Y=2.0*t/(s*s+t*t+1.0);
    Z=(-1.0+s*s+t*t)/(s*s+t*t+1.0);
    // Cartesian to spherical conversion
    tpz[0]=atan2(X,Z);
    tpz[1]=atan2(Y,sqrt(X*X+Z*Z));
    tpz[2]=1.0;
    }
    else tpz[2]=0.0;

return tpz;
}

float3  WarpPanorama(float2 texcoord : TEXCOORD0) : COLOR0
{
float3 pp,xyz,tpz;
float2 tp,xy;

xy[0]=texcoord[0];
xy[1]=texcoord[1];

tpz=ParaboloidUpXY2TP(xy);
tp[0]=tpz[0];
tp[1]=tpz[1];

if (tpz[2]==1.0){
xyz=Spherical2Cartesian(tp);
pp=mul(R,xyz);
tp=Cartesian2Spherical(pp);
xy=LatitudeLongitudeTP2XY(tp[0],tp[1]);

return f3texRECT(PanoramaImage, xy);}
else
{
return float3(0,0,1);
}
}
```

The basic difference in the previous `pinhole.cg` shader is that the remapping is only effective inside the image disk. Thus, we need to slightly modify the former `WarpPanorama` shader to take into account the domain of definition of the mapping functions. The ray remapping shaders can also be combined altogether in a number of scenarios. For example, we can display on each face of a 3D cube a different camera model viewer obtained from a common environment map (see Figure 9.5.4 and shader file `genericcamera.cg`). Please refer to subfolder GPUPanoramaViewer2 on the CD-ROM.

ON THE CD

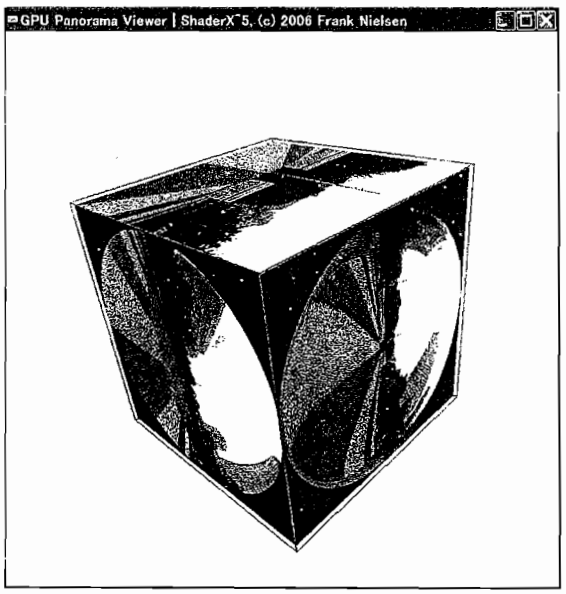

FIGURE 9.5.4 Rendering several generic camera models using the abstract framework.

Conclusion

We have presented an efficient GPU panorama fragment shader for relieving the CPU of the per-pixel and warping procedures. The panorama viewer allows us to render several generic camera models in a same view at maximum frame rate, as well as to convert or remap on-the-fly complete environment maps. The abstraction (θ,ϕ) ray-(x,y) image framework relies on the fact that all rays share a common center of projection. We leave to future work the extension of this abstract camera model and reprojection technique to caustic surfaces particularly observed in catadioptric acquisition systems [Nielsen05b].

References

[Nielsen05a] Nielsen, Frank. "Surround Video: A Multihead Camera Approach." *The Visual Computer*, *21*(1-2), (2005): 92–103.

[Nielsen05b] Nielsen, Frank. *Visual Computing: Geometry, Graphics and Vision*. Charles River Media, 2005.

[Wong05] Wong, Tien-Tsin, Liang Wan, Chi-Sing Leung, and Ping-Man Lam. "Real-time Environment Mapping with *equal solid-angle spherical quad-map*." *ShaderX*^{4}*: Advanced Rendering Techniques*, edited by Wolfgang Engel. Charles River Media, 2005: 221–233.