Stereographic Projection Visualization Mathematics

Lin Yang

October 30, 2012

Part I
Stereographic Projection

1 Particle flux and profile

For each particle, it has six basic parameters: x, y, z, $mass$, $density$ and $hsmooth$. The flux of the an particle is determined by

$$f = \frac{unitfactor \times mass \times density}{4\pi d^2} \tag{1}$$

, where d is the distance from the particle to the observer point$(x_o,\ y_o,\ z_o)$,

$$d = \sqrt{(x - x_o)^2 + (y - y_o)^2 + (z - z_o)^2} \tag{2}$$

$hsmooth$ specifies the flux profile. On the sphere, the flux profile[1] is

$$f(\theta,\phi)d\Omega = \begin{cases} N\exp\left(-1.5\cos^{-1}(\vec{r}\cdot\vec{r_p})^2/\delta\theta^2\right) & \cos^{-1}(\vec{r}\cdot\vec{r_v}) \le \delta\theta \\ 0 & \cos^{-1}(\vec{r}\cdot\vec{r_p}) > \delta\theta \end{cases} \cdot d\Omega \tag{3}$$

, where N is a normailation factor, $\delta\theta$ is defined as

$$\delta\theta = \frac{hsmooth}{d} \tag{4}$$

$\vec{r_p}$ is the normalized particle vector,

$$\vec{r_p} = (x - x_o,\ y - y_o,\ z - z_o)/d \tag{5}$$

and

$$\vec{r} = (\sin\theta\cos\phi,\ \sin\theta\sin\phi,\ \cos\theta) \tag{6}$$

[1] Mike: could you check this formula?

2 Stereographic Projection Mathematics

In order to use the GPU primtives (here I used OpenGL point sprite), we need to project the flux profile on the sphere to the plane. The plane is the so-called seterographic projection plane. The setting up is the same with Wikipedia[2]. To avoid the siglarity of this projection, I divided the sphere into 2 parts: the upper hemisphere and lower hemisphere. With the same notation of Wikipedia, I choose N (projection point) point be the north pole for the lower hemisphere and south pole for the upper hemisphere. The stereographic projection matches our requirement since it projects a circle on the sphere to the a circle on the projection plane (here the circle is the particle profile).

2.1 Project particle circle to the plane

To describe a particle profile on the sphere, we have 3 basic quantities: θ, ϕ and $\delta\theta$. When project to the projection plane, we have the two points $P(\theta+\delta\theta,\phi)$ and $P(\theta-\delta\theta,\phi)$ are on the same diameter (here P stands for the projection). So we have the parameter of the circle on the projection plane is

$$\begin{aligned} r(\theta,\phi,\delta\theta) &= \frac{1}{2}\left(\frac{\sin(\theta-\delta\theta)}{1-\cos(\theta-\delta\theta)} - \frac{\sin(\theta+\delta\theta)}{1-\cos(\theta+\delta\theta)}\right) \\ x_c(\theta,\phi,\delta\theta) &= \frac{1}{2}\left(\frac{\sin(\theta-\delta\theta)}{1-\cos(\theta-\delta\theta)} + \frac{\sin(\theta+\delta\theta)}{1-\cos(\theta+\delta\theta)}\right)\cos\phi \\ y_c(\theta,\phi,\delta\theta) &= \frac{1}{2}\left(\frac{\sin(\theta-\delta\theta)}{1-\cos(\theta-\delta\theta)} + \frac{\sin(\theta+\delta\theta)}{1-\cos(\theta+\delta\theta)}\right)\sin\phi \end{aligned} \tag{7}$$

The three parameters in terms with the setting up of the OpenGL is using to calculate the point sprite.

2.2 Project particle flux profile to the plane

To project the profile to the plane we are going to get profie function $g(x,y)$ such that

$$g(x,y)\mathrm{d}x\mathrm{d}y = f(\theta,\phi)\mathrm{d}\Omega \tag{8}$$

Since for stereographic projection,

$$4/(1+r^2)^2\mathrm{d}x\mathrm{d}y = \sin\theta\mathrm{d}\theta\mathrm{d}\phi = \mathrm{d}\Omega \tag{9}$$

We have

$$4/(1+r^2)^2\mathrm{d}x\mathrm{d}y = \sin\theta\mathrm{d}\theta\mathrm{d}\phi = \mathrm{d}\Omega \tag{10}$$

We have

$$g(x,y) = 4/(1+r^2)^2 \times f[\theta(x,y),\phi(x,y)] = 4/(1+r^2)^2 \times f(\vec{r_p}) \tag{11}$$

, where $\vec{r_p}(x,y) = [2x/(1+x^2+y^2),\ 2y/(1+x^2+y^2),\ (x^2+y^2-1.0)/(x^2+y^2+1.0)]$ Finally we have the flux for each pixel of the projection plane is

$$f_i = C_i g(x_i,y_i) \tag{12}$$

[2]http://en.wikipedia.org/wiki/Stereographic_projection

, where C_i is the normaliztion factor such that the total flux projected to the projection plane is not changing and (x_i, y_i) is the coordinates of the i-th pixels on the projection plane. Hence,

$$f_0/C_i = \sum_S f_i \tag{13}$$

, where S is the disk on the projection plane. C_i is actually a function of r and r_0 (r the radius of the disk and $r_0 = \sqrt{(x^2+y^2)}$). So C_i could be pre-calculated and import to the GPU to accelerate the calculation speed.

3 Edge Condition

Since I divided the sphere into two parts, I must deal with particles locates on both spheres. So if a particle on the plane edge will be caculated twice. The CPU code is

Listing 1: OpenGL draw flux code

```
//lower sphere
fbufferL->bindBuf();
fshaderL->begin();
{
    glDrawArrays(GL_POINTS, 0, reader->getMemparts());
    glFlush();
}
fshaderL->end();
fbufferL->unbindBuf();

//upper sphere
fbufferU->bindBuf();
fshaderU->begin();
{
    glDrawArrays(GL_POINTS, 0, reader->getMemparts());
    glFlush();
}
fshaderU->end();
fbufferU->unbindBuf();
```

The "glDrawArrays" is actually doing the drawing and called for both upper sphere and lower sphere. It calls the shader code like this

Listing 2: GLSL code of drawing particles

```
float distance = length(pvec);

//angular radius
dtheta = parameter.b / distance;
```

```
    //rotation and normalize
    vec3 npvec = normalize(rotmatrix * pvec);
    float costheta = npvec.z;//dot(npvec, nzaxis);
    float theta = acos(costheta);

    if((theta > PI / 2.0 || theta + dtheta >= PI / 2.0)
        && dtheta < PI / 2.0){
        float sintheta = sin(theta);
        float sinphi;
        float cosphi;
...
```

Part II
To HEALPIX Re-projection

3.1 Basics

For each HEALPIX pixel, I calculate its corresponding (θ, ϕ). Based on θ, the code decide whether get the pixel value from the upper sphere or the lower sphere. After that, use bi-linear interpolation to get the interpolated value from the neighboring 4 pixels. The code also considering the condition that the 4 pixels are from different sphere.

On each pixel of the projection plane, the flux data is

$$f_i = g(x_i, y_i)\Delta x \Delta y \tag{14}$$

, where $\Delta x = \Delta y = 1/(\text{window size on pixels})$. When convert to HEALPIX map, the flux data should be

$$f'_i = f(\vec{r_i})\Delta\Omega \tag{15}$$

, where $\Delta\Omega = 4\pi/(12 \times \text{NSIDE}^2)$. Therefore, when interpolate a value of f_i from the projection plane, there is an additional factor c to convert it from seterographic projection to HEALPIX projection f'_i, such that $f'_i = c_i f_i/(4/(1+r_i^2)^2)$. Hence

$$f'_i = \frac{f_i}{4/(1+r_i^2)^2} \times \frac{\Delta\Omega}{\Delta x \Delta y} \tag{16}$$

and

$$c_i = \frac{\Delta\Omega}{\Delta x \Delta y} = \frac{4\pi \times \text{WSIZE}^2}{12 \times \text{NSIDE}^2} \tag{17}$$

In the following code, the "_getpixflux" function get $f_i/(4\pi/(1+r_i^2)^2)$ from corresponding pixel. After 4 pixels read, a interpolation is made to guess the value of f'_i. The factor c_i is multiplyed after the interplation.

3.2 Code

Listing 3: OpenGL re-projection code

```
for(int i = 0; i < npix; i++){
    double x, y,r, factor;
    int j;
    pix2ang_ring(nside, i, &theta, &phi);
    //now we have theta and phi and dtheta

    //convert (theta, phi, dtheta) to the projection plane, we have
    phi = 2*PI - phi + PI;
    bool isupshere = false;
    if(theta < PI/2){
        theta = PI - theta;
        isupshere = true;
    }

    int d = round(windowSize / 2.0);

    //bilinear interpolation
    double pr = sin(theta)/(1-cos(theta));
    double pxc = pr * cos(phi);
    double pyc = pr * sin(phi);

    double xc = (pxc) * (double)d;
    double yc = (pyc) * (double)d;
    double x1 = floor(xc);
    double x2 = x1 + 1;
    double y1 = floor(yc);
    double y2 = y1 + 1;

    float f11 = _getpixflux(round(x1), round(y1), isupshere);
    float f12 = _getpixflux(round(x1), round(y2), isupshere);
    float f21 = _getpixflux(round(x2), round(y1), isupshere);
    float f22 = _getpixflux(round(x2), round(y2), isupshere);

    double flux = 0;
    double fr1 = (x2 - xc) / (x2 - x1) * f11 + (xc - x1) / (x2 - x1) * f2
    double fr2 = (x2 - xc) / (x2 - x1) * f12 + (xc - x1) / (x2 - x1) * f22
    flux = (y2 - yc) / (y2 - y1) * fr1 + (yc - y1) / (y2 - y1) * fr2;
    healmap[i] = flux * params->FLUXFACTOR / (4 * PI / npix) *
    4 * PI * (windowSize * windowSize) / npix;
```

For "_getpixflux", here

Listing 4: OpenGL get pixel flux code

```
double render::_getpixflux(int x1, int y1, bool isupshere){
    //inside the circle
    double f11 = 0;
    int d = round(windowSize / 2.0);
    double _r = sqrt((double)(x1 * x1 + y1 * y1)/(double)(d * d));
    if(x1 * x1 + y1 * y1 <= d * d){
        if(isupshere){
            //up sphere
            f11 = fluxmapU[(d - y1) * windowSize + x1 + d];
        }else{
            f11 = fluxmapL[(d - y1) * windowSize + x1 + d];
        }
        f11 =  f11 / (4.0 / (1 + _r*_r)/(1 + _r*_r));
    }else{
        //converted it to theta phi and then to another sphere
        double _y = (double) y1 / (double)d;
        double _x = (double) x1 / (double)d;
        double _sinphi = _y / _r;
        double _cosphi = _x / _r;
        double _theta;
        _theta = (_r == 0.0) ? PI : 2.0 * atan(1.0 / _r);
        _theta = PI - _theta;
        double r1 = sin(_theta)/(1-cos(_theta));
        double _px = r1 * _cosphi;
        double _py = r1 * _sinphi;
        int kx = floor((_px + 1.0) * windowSize / 2);
        int ky = floor((_py + 1.0) * windowSize / 2);

        double _flux = 0;
        if(!isupshere){
            //lower sphere
            _flux = fluxmapU[(windowSize - ky) * windowSize + kx];
        }
        else{
            //upper sphere
            _flux = fluxmapL[(windowSize - ky) * windowSize + kx];
        }
        f11 = _flux / (4.0 / (1 + r1*r1)/(1 + r1*r1));
    }
    return f11;

}
```