A Note on Computation of a Ray Bending Path

Jerry Tessendorf

April 20, 2022

1 starting point

Index of refraction spatial variations within a volume induce bending of the ray marching path that light takes through the participating medium. The driving equations for this bending describes the shape of the path $\vec{\mathrm{x}}(s)$ and direction of propagation $\hat{\mathrm{n}}(s)$ as functions of the arclength s along the path. Both are modified by variations in the index of refraction n via the equations

$$\frac{d\vec{\mathrm{x}}(s)}{ds} = \hat{\mathrm{n}}(s) \tag{1}$$

$$\frac{d\hat{\mathrm{n}}(s)}{ds} = \vec{\mathrm{F}}(s) - \hat{\mathrm{n}}(s)\left(\hat{\mathrm{n}}(s)\cdot\vec{\mathrm{F}}(s)\right) \tag{2}$$

where $\vec{\mathrm{F}}(s)$ is the gradient of the log of the index of refraction:

$$\vec{\mathrm{F}}(s) = \nabla \log n\left(\vec{\mathrm{x}}(s)\right) \tag{3}$$

The direction of propagation is also the tangent of the path. This sets up an initial value problem: Given the values at the initial arclength s_0 of $\vec{\mathrm{x}}(s_0) = \vec{\mathrm{x}}_0$ and $\hat{\mathrm{n}}(s_0) = \hat{\mathrm{n}}_0$, compute from these equations the position and direction for arclengths $s > s_0$ of interest.

Here we consider a situation in which the the gradient vector $\vec{\mathrm{F}}$ is constant along a stretch of arclength of interest. There are two situations in which this occurs, and many more may be relevant. One of them is when the volume is modeled as a stack of layers and the gradient is constant within each layer. Another is the volume rendering ray march process with $\vec{\mathrm{F}}$ varying with each ray march step, but constant along the length of a single step. This latter situation is consistent with other assumptions in ray marching, which assumes that the material density and color are also constant over the distance of a single ray march step.

In the next section, equation 2 in solved exactly under the constant gradient assumption by expressing $\hat{\mathrm{n}}(s)$ as a rotation about an axis. The path is found in section 3 by integrating equation 1 exactly. A process for specifying the desired accuracy to auto-adjust the step size is presented in section 4. This process is analyzed using the example of curved light ray paths near black holes in the weak gravitation limit. Intersections of the curved paths with planes is discussed in section 5.

2 ray bending as a direction rotation

Ray bending alters the direction of the path tangent, while preserving the fact that the tangent vector is a unit vector. This means that the alteration of the path tangent can be represented by a rotation of the tangent vector. A general rotation is described by a unit vector $\hat{\alpha}(s)$ for the axis of rotation, and an angle $\alpha(s)$ for the amount of rotation about the axis. Using these two quantities, the rotation is expressible as

$$\hat{\mathrm{n}}(s) = \hat{\mathrm{n}}_0\ \cos\alpha(s) + \hat{\alpha}(s)\ (\hat{\mathrm{n}}_0\cdot\hat{\alpha}(s))\ (1 - \cos\alpha(s)) + \hat{\mathrm{n}}_0\times\hat{\alpha}(s)\ \sin\alpha(s) \tag{4}$$

along with the initial condition $\alpha(s_0) = 0$. Ray bending with a constant $\vec{\mathrm{F}}$ provides a natural rotation axis: the path is contained in a plane defined by the two vectors $\hat{\mathrm{n}}_0$ and $\hat{\mathrm{F}} = \vec{\mathrm{F}}/|\vec{\mathrm{F}}|$, so the axis of rotation is constant and perpendicular to both of these:

$$\hat{\alpha} = \frac{\hat{\mathrm{n}}_0\times\hat{\mathrm{F}}}{\left|\hat{\mathrm{n}}_0\times\hat{\mathrm{F}}\right|} \tag{5}$$

Because of this choice of rotation axis, the second term on the RHS of equation 4 is zero, and the third term has the vector

$$\hat{\mathrm{n}}_0 \times \hat{\alpha} \;=\; \frac{(\hat{\mathrm{n}}_0 \cdot \hat{\mathrm{F}})\;\hat{\mathrm{n}}_0 \;-\; \hat{\mathrm{F}}}{\left|\hat{\mathrm{n}}_0 \times \hat{\mathrm{F}}\right|} \tag{6}$$

which is a unit vector:

$$\left|\hat{\mathrm{n}}_0 \times \hat{\alpha}\right|^2 \;=\; \frac{1 - (\hat{\mathrm{n}}_0 \cdot \hat{\mathrm{F}})^2}{\left|\hat{\mathrm{n}}_0 \times \hat{\mathrm{F}}\right|^2} \;=\; 1 \tag{7}$$

This brings the expression for the path tangent to

$$\hat{\mathrm{n}}(s) \;=\; \hat{\mathrm{n}}_0\;\cos\alpha(s) \;+\; \hat{\mathrm{n}}_0 \times \hat{\alpha}\;\sin\alpha(s) \tag{8}$$

and the problem of describing the ray bending path tangent has been reduced to solving for the angle $\alpha(s)$.

To solve for $\alpha(s)$, this expression for $\hat{\mathrm{n}}(s)$ can be inserted into equation 2. The derivative on the LHS becomes

$$\frac{d\hat{\mathrm{n}}(s)}{ds} \;=\; \frac{d\alpha(s)}{ds}\;\{-\hat{\mathrm{n}}_0\;\sin\alpha(s) \;+\; \hat{\mathrm{n}}_0 \times \hat{\alpha}\;\cos\alpha(s)\} \tag{9}$$

Since equation 2 has reduced to a differential equation for a single degree of freedom, taking the inner product of both sides with $\hat{\mathrm{n}}_0$ simplifies the expressions. The LHS becomes

$$\hat{\mathrm{n}}_0 \cdot \frac{d\hat{\mathrm{n}}(s)}{ds} \;=\; -\frac{d\alpha(s)}{ds}\;\sin\alpha(s) \tag{10}$$

and the RHS side is

$$\hat{\mathrm{n}}_0 \cdot \left\{\vec{\mathrm{F}}(s) \;-\; \hat{\mathrm{n}}(s)\;\left(\hat{\mathrm{n}}(s) \cdot \vec{\mathrm{F}}(s)\right)\right\} \;=\; \sin\alpha(s)\;\left\{(\hat{\mathrm{n}}_0 \cdot \vec{\mathrm{F}})\;\sin\alpha(s) \;-\; \left((\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \vec{\mathrm{F}}\right)\cos\alpha(s)\right\} \tag{11}$$

producing the differential equation

$$\frac{d\alpha(s)}{ds} \;=\; -(\hat{\mathrm{n}}_0 \cdot \vec{\mathrm{F}})\;\sin\alpha(s) \;+\left((\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \vec{\mathrm{F}}\right)\cos\alpha(s) \tag{12}$$

This expression can be solved for α by setting up the integral equation

$$\int_0^{\alpha} \frac{dx}{\left((\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \vec{\mathrm{F}}\right)\cos x \;-\; (\hat{\mathrm{n}}_0 \cdot \vec{\mathrm{F}})\;\sin x} \;=\; s - s_0 \tag{13}$$

This integral has a closed form expression[1]

$$\log\tan\left(\frac{x \;+\; \beta}{2}\right)\Bigg|_0^{\alpha(s)} \;=\; (s - s_0)\;\sqrt{(\hat{\mathrm{n}}_0 \cdot \vec{\mathrm{F}})^2 \;+\; \left((\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \vec{\mathrm{F}}\right)^2} \tag{14}$$

where $\hat{\mathrm{n}}_0 \cdot \hat{\mathrm{F}} = \cos\beta$. Using

$$\sqrt{(\hat{\mathrm{n}}_0 \cdot \vec{\mathrm{F}})^2 \;+\; ((\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \vec{\mathrm{F}})^2} \;=\; |\vec{\mathrm{F}}| \;\equiv\; F, \tag{15}$$

[1] Gradshteyn and Ryzhik 2.557.4, pg. 149.

the solution is

$$\tan\left(\frac{\beta \ - \ \alpha(s)}{2}\right) \ = \ \tan(\beta/2) \ e^{F(s-s_0)} \tag{16}$$

Using some identities, this can be manipulated into the form

$$\cos(\alpha(s)) \ = \ \frac{(1-Q^2)\cos\beta \ + \ 2Q\sin\beta}{1+Q^2} \tag{17}$$

$$\sin(\alpha(s)) \ = \ \frac{(1-Q^2)\sin\beta \ - \ 2Q\cos\beta}{1+Q^2} \tag{18}$$

where

$$Q \ = \ \tan(\beta/2) \ \exp\left(F(s-s_0)\right) \tag{19}$$

This is a complete solution for the tangent of the ray bending path in a region of constant gradient.

3 integrating the path

The ray bending path $\vec{\mathrm{x}}(s)$ follows from integrating equation 1 using equations 8, 17-19:

$$\vec{\mathrm{x}}(s) \ = \ \vec{\mathrm{x}}_0 + \hat{\mathrm{n}}_0 \int_{s_0}^{s} ds' \ \cos\alpha(s') \ + \ \hat{\mathrm{n}}_0 \times \hat{\alpha} \int_{s_0}^{s} ds' \ \sin\alpha(s') \tag{20}$$

These integrals have the generic form

$$\int dt \ \frac{A \ + \ B \ e^{Ft} \ + \ C \ e^{2Ft}}{1 \ + \ D \ e^{2Ft}} \tag{21}$$

for appropriate values of A, B, C, D. Using some integral identities[2] this evaluates to

$$\begin{aligned}\int dt \ \frac{A \ + \ B \ e^{Ft} \ + \ C \ e^{2Ft}}{1 \ + \ D \ e^{2Ft}} \ &= \ \frac{A}{2F}\left(2Ft \ - \ \log\left(1 \ + \ D \ e^{2Ft}\right)\right) \\ &+ \ \frac{B}{F\sqrt{D}} \ \tan^{-1}\left(\sqrt{D} \ e^{Ft}\right) \\ &+ \ \frac{C}{2DF} \ \log\left(1 \ + \ D \ e^{2Ft}\right)\end{aligned} \tag{22}$$

Assembling all of the algebra that follows from these expressions,

$$\begin{aligned}\int_{s_0}^{s} ds' \ \cos\alpha(s') \ &= \ \frac{\cos\beta}{F}\left(F(s-s_0) \ + \ \log\left(\frac{1 \ + \ \tan^2(\beta/2)}{1 \ + \ \tan^2(\beta/2) \ e^{2F(s-s_0)}}\right)\right) \\ &+ \ \frac{2\sin\beta}{F}\left(\tan^{-1}\left(\tan(\beta/2) \ e^{F(s-s_0)}\right) - \beta/2\right)\end{aligned} \tag{23}$$

and

$$\begin{aligned}\int_{s_0}^{s} ds' \ \sin\alpha(s') \ &= \ \frac{\sin\beta}{F}\left(F(s-s_0) \ + \ \log\left(\frac{1 \ + \ \tan^2(\beta/2)}{1 \ + \ \tan^2(\beta/2) \ e^{2F(s-s_0)}}\right)\right) \\ &- \ \frac{2\cos\beta}{F}\left(\tan^{-1}\left(\tan(\beta/2) \ e^{F(s-s_0)}\right) - \beta/2\right)\end{aligned} \tag{24}$$

[2] Gradshteyn and Ryzhik 2.313.1, pg. 92; 2.124.1, pg. 60

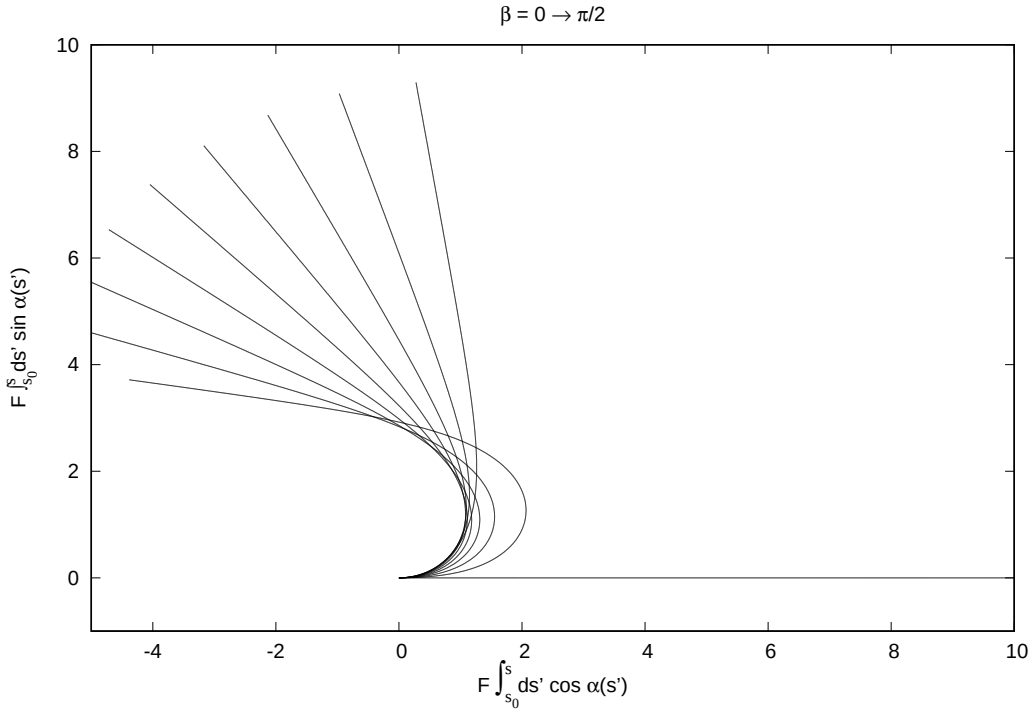


Figure 1: The two components of the ray bending path, plotted parametrically. Each line is a different value of β, which ranges in value from 0 to $\pi/2$.

Figure 1 shows these two functions, plotted parametrically by varying the value of $F(s - s_0)$, for a range of values of β.

4 automatic adjustment of ray march step

This curved-path formula can be used in an environment in which the index of refraction varies spatially, by ray marching over small steps Δs. This process assumes two dimensionless threshold values, T_{max} and T_{min}, with $T_{max} > T_{min}$. The threshold T_{max} is maximum fractional change in the gradient within a step that is tolerable, and when the fractional change exceeds this threshold, the step size is reduced and the step recomputed with the smaller step size. When the fractional change in the gradient is less than the threshold T_{min}, the step size is increased because we are willing to tolerate some amount of error in return

for bigger steps and faster ray marching. These two thresholds drive the step size to smaller values in regions of substantial change in the gradient, but restores larger steps in regions of relatively small change in the gradient. A third parameter, N_{max} is an integer for the maximum number of steps allowed along a segment before terminating. Procedurally, this ray march algorithm has the following steps:

1. At the start of the ray march, initialize the position $\vec{\mathrm{x}}$, ray direction $\hat{\mathrm{n}}$, and step size Δs.
2. If the position $\vec{\mathrm{x}}$ is outside of the medium, terminate the ray march.
3. Compute the new position, direction, and step size:
   (a) Initize a step counter $N_{steps} = 0$.
   (b) Compute the gradient $\vec{\mathrm{F}}$ at $\vec{\mathrm{x}}$.
   (c) Compute the new position $\vec{\mathrm{x}}(\Delta s)$ according to equation 20.
   (d) Increment the counter $N_{steps} += 1$.
   (e) If $N_{steps} > N_{max}$ march has failed to converge. Terminate the ray march.
   (f) Compute the gradient $\vec{\mathrm{F}}'$ at the new position.
   (g) Compute the gradient fractional difference $\epsilon = \left|\vec{\mathrm{F}}' - \vec{\mathrm{F}}\right| / \left|\vec{\mathrm{F}}\right|$.
   (h) If $\epsilon > T_{max}$, reduce the set size by $\Delta s \to \Delta s \xi$, where ξ is a random number from a uniform random number generator with values between 0 and 1. Return to step 3c.
   (i) If $\epsilon < T_{min}$, increase the step size by $\Delta s \to \Delta s / \xi$, where ξ is a random number from a uniform random number generator with values between 0 and 1.
   (j) Compute the new direction $\hat{\mathrm{n}}(\Delta s)$ using equations 8, 17, and 18.
4. Update the position $\vec{\mathrm{x}} = \vec{\mathrm{x}}(\Delta s)$.
5. Update the direction $\hat{\mathrm{n}} = \hat{\mathrm{n}}(\Delta s)$.
6. Return to step 2.

4.1 Example demonstration of automatic step adjustment

Here we show an example of this curved ray marching using the example of weak gravitational lensing from a Reissner-Nordstom black hole. Gravitational lensing describes light propagating in a geometric optics approximation though a gravitational field. In the full general relativistic description of this, the path taken by the light is a null geodesic in spacetime. In the weak-field limit, the differential equation for the null geodesic reduces to equation 2, where the index of refraction derives from an analysis of the Christoffel symbols in this limit. For a Reissner-Nordstom black hole[3], the index of refraction is

$$n(\vec{\mathrm{x}}) = \frac{r^2}{r^2 - r + q^2} \tag{25}$$

where $r = |\vec{\mathrm{x}} - \vec{\mathrm{x}}_C|/r_S$ is the distance from the black hole center scaled by the Schwarzschild radius r_S. The dimensionless constant q is the charge.

The raymarching for this demonstration was implemented as a shader in Gilligan, and is shown here:

[3]Saswati Roy and Asoke Kumar Sen, (2019), "Study of gravitational deflection of light ray", J. Phys.: Conf. Ser. 1330 012002 https://iopscience.iop.org/article/10.1088/1742-6596/1330/1/012002

```
namespace ash{
   namespace Shaders{
      class VolumeCurvedRayMarch : public Shader
      {
        public:
          VolumeCurvedRayMarch( const lux::ScalarField& IOR, const float step, ash::Vector& llc, ash::Vector urc  ) :
            ior (IOR),
            step_ds (step),
            aabb(lux::makeAABB( lux::Vector(llc.X(), llc.Y(), llc.Z()), lux::Vector(urc.X(), urc.Y(), urc.Z()) )),
            max_steps (1000000),
            step_tolerance_max (0.1),
            step_tolerance_min (0.0001)
            {}

          ~VolumeCurvedRayMarch(){};

          void eval(ShaderData* shaderdata);

          double max_tolerance() const { return step_tolerance_max; }
          double min_tolerance() const { return step_tolerance_min; }
          void set_tolerances( const double ma, const double mi )
          {
             step_tolerance_max = ma;
             step_tolerance_min = mi;
          }

        private:
          const lux::ScalarField ior;
          double step_ds;
          lux::AABB aabb;
          int max_steps;
          double step_tolerance_max;
          double step_tolerance_min;
          lux::UniformPRN prn;

          double integrated_curved_raymarch(double& ds, lux::Vector& D, lux::Vector& P);
      };



      void VolumeCurvedRayMarch::eval(ShaderData* data)
      {
          ash::Vector endPoint = data->hitPosition;
          ash::Vector startPoint = data->incomingRay->P;

          lux::Vector eP( endPoint.X(), endPoint.Y(), endPoint.Z() );
          lux::Vector sP( startPoint.X(), startPoint.Y(), startPoint.Z() );

          lux::Vector direction = (eP-sP).unitvector();
          double far_max = (eP-sP).magnitude();

          double near;
          double far;
          if( !aabb->intersection( sP, direction, near, far ) ){ return; }
          if( near < 0.0 && far < 0.0 ){ return; }

          if(near > far_max){ return; }

          if( far > 0.0 )
          {
              if( far > far_max){ far = far_max; }
              eP = sP + far * direction;
          }

          if( near < far )
          {
              sP += near*direction; // starting point of curved ray march
              bool done = false;
              double total_path = near;
              int step_count = 0;
              double step = step_ds;
              while(!done)
              {
                  total_path += integrated_curved_raymarch(step, direction, sP);  // variable step size
                  done = !(aabb->isInside(sP));
                  step_count++;
                  if(step_count > max_steps){ done = true; }
              }
              if(step_count <= max_steps)
              {
                 ash::Vector newD( direction.X(), direction.Y(), direction.Z() );
                 ash::Vector newP( sP.X(), sP.Y(), sP.Z() );
                 Ray outputRay;
                 outputRay.initRay(newD, newP);
                 outputRay.pixel_id = data->incomingRay->pixel_id;
                 outputRay.color = data->incomingRay->color;
                 outputRay.spectralOpacity = data->incomingRay->spectralOpacity;
                 outputRay.terminated = false;
                 outputRay.zDepth = data->incomingRay->zDepth + total_path;
                 data->outputStack->push(outputRay);
              }
```

```
            data->spectralOpacity = ash::Color(1,1,1,1);  // this ray should not produce anything
            data->incomingRay->spectralOpacity = ash::Color(1,1,1,1);  // this ray should not produce anything
        }
    }

    double VolumeCurvedRayMarch::integrated_curved_raymarch( double& ds, lux::Vector& D, lux::Vector& P )
    {
        lux::Vector F = ior->grad(P)/ior->eval(P);
        double Fmag = F.magnitude();
        lux::Vector F_hat = F/Fmag;
        lux::Vector alpha_hat = (D^F_hat).unitvector();
        lux::Vector D_alpha_hat = D^alpha_hat;
        double cos_beta = D*F_hat;
        double beta = std::acos(cos_beta);
        double sin_beta2 = std::sin(beta*0.5);
        double tan_beta2 = std::tan(beta*0.5);
        double sin_beta2sq = sin_beta2*sin_beta2;
        double tan_beta2sq = tan_beta2*tan_beta2;
        double sin_beta = std::sqrt( 1.0 - cos_beta*cos_beta );
        double T = step_tolerance_max;
        double ef = 0.0;
        double actual_step = ds;
        while(T >= step_tolerance_max)
        {
            actual_step = ds;
            ef = std::exp(Fmag*ds);
            double denom = 1.0 + tan_beta2sq*ef*ef;
            double Q = Fmag*ds + std::log( (1.0 + tan_beta2sq)/denom );
            double R = std::atan( tan_beta2*ef ) - beta*0.5;
            double integrated_cos_alpha = (cos_beta/Fmag)*Q + (2.0*sin_beta/Fmag)*R;
            double integrated_sin_alpha = (sin_beta/Fmag)*Q - (sin_beta/Fmag)*(1.0-tan_beta2sq)*R/tan_beta2;

            lux::Vector Pnew = P + D * integrated_cos_alpha + D_alpha_hat * integrated_sin_alpha;

            lux::Vector Fp = ior->grad(Pnew) / ior->eval(Pnew);
            T = (Fp-F).magnitude()/Fmag;
            if(T>=step_tolerance_max)
            {
                float random_number = 1.0;
#pragma omp critical
  {
    random_number = prn.eval();
        }
                ds *= random_number;
            }
            else
            {
                P = Pnew;
                double cos_alpha = ( cos_beta + 4.0*sin_beta2sq*ef + sin_beta2sq*( tan_beta2sq-1.0 )*ef*ef )/denom;
                double sin_alpha = sin_beta*(1.0-ef)*(1.0 + tan_beta2sq*ef)/denom;
                D = D * cos_alpha +  D_alpha_hat * sin_alpha;
                if(T<step_tolerance_min)
                {
                    float random_number = 1.0;
#pragma omp critical
      {
        random_number = prn.eval();
             }
                    ds /= random_number;
                }
            }
        }
        return actual_step;
    }
  };
};
```

We set up the render using the environment map shown in figure 2. The camera is placed 5 m from the center of the black hole, which has a Schwarzschild radius of 1 cm. The horizontal field of view of the camera is 40 degrees. Figure 3(a) shows the scene from the camera POV without the black hole present. The black hole is present in figure 3(b), but the tolerance values have been set to effectively prevent automatic step size adjustment. The same black hole is rendered in figure 3(c), with tolerances set to induce automatic step size adjustment. The automatic step size adjustment is able to resolve the black hole center, unlike the fixed step size case.

For a path that skirts the limb of the black hole, figure 4 shows the components of the ray path perpendicular to the axis between the camera and the center of the black hole. The path nears the Schwarzchild radius r_s, then deflects away without coming in contact. Figure 5 shows the step size as a function of path length for the path in figure 4. As the path approaches the black hole, the step size drops by several orders of magnitude, then grows larger once the path is well away from the Schwarzchild radius.

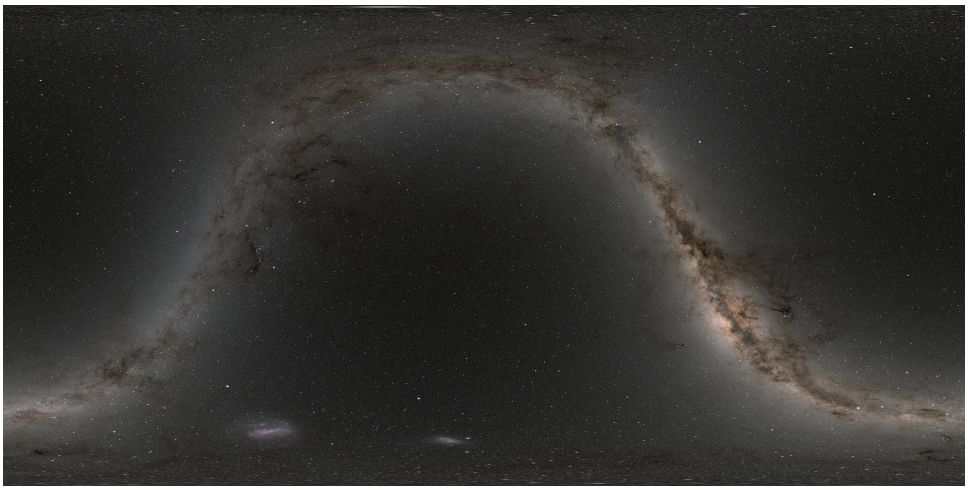


Figure 2: Environment map.

Choosing a ray that stops at the Schwarzchild radius, figure 6 shows the path and its termination. The autmatically adjusted step size is in figure 7. In this figure, the step size is plotted as a function of step number, rather that path length. The step size plunges in value as the path approaches the Schwarzchild radius, reaching a value between 10^{-15} and 10^{-21}, fluctuates in this small range of values, effectively never advancing further, and the path is frozen at the Schwarzchild radius. This plot shows this behavior for 10^5 steps, but the full data set shows this behavior for 10^6 steps before termination by a threshold test on the number of steps.

4.2 Path deflection: behavior related to the parameters T_{max}, T_{min}, and N_{max}

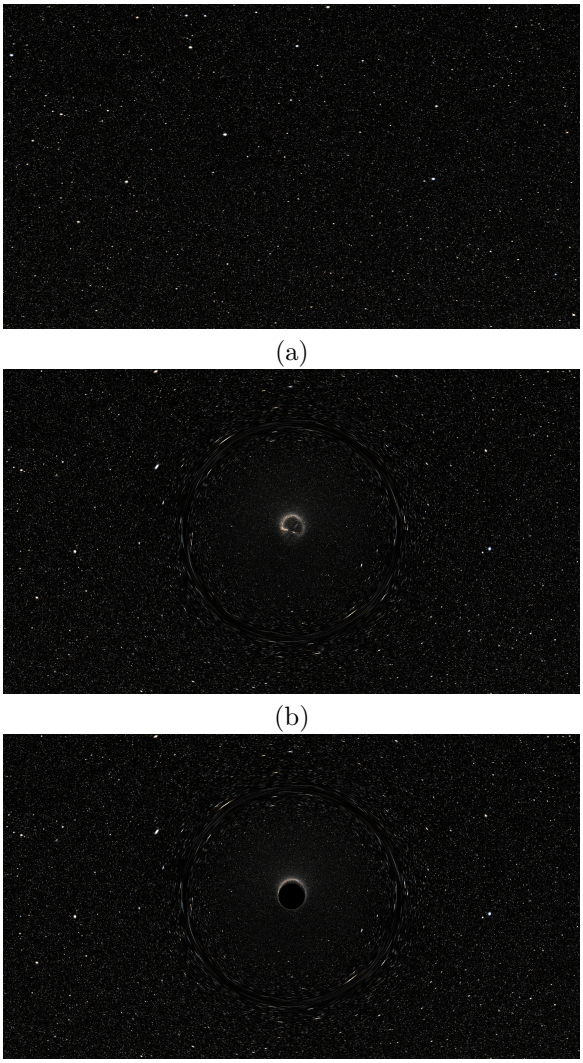
The algorithm described in section 4 is driven by three user-selected parameters: A threshold T_{max} for the maximum allowed percentage change in the vector $\vec{\mathrm{F}}$ at the start and end of a path segment and which induces reduction of the step size Δs, a threshold T_{min} of allowable amount of percentage change in $\vec{\mathrm{F}}$ which induces increases in the size of Δs, and the maximum number N_{max} of path segments in a path before bailing out of the path as incompletable. In this section we use a well-understood case for testing the accuracy of the curved ray march, and its behavior as these parameters are varied.

The weak gravitation limit of general relativity provides a well-understood test case for ray bending, in the form of the deflection angle of the ray path. In GR, this deflection angle is commonly represented as a function of the impact parameter, which is the ratio $2r_s/b$, where r_s is the Schwarzschild radius and b is the closest-approach distance of the ray from the center of the gravitational object. The deflection angle is the angle between the incoming and outgoing directions of the ray:

$$\cos\phi \;=\; \hat{\mathrm{n}}(0)\cdot\hat{\mathrm{n}}\,(s_{max}) \tag{26}$$

For Schwarzschild metric, this relationship is plotted in figure 8, based on the pertubation expansion[4]

[4]Carlos Rodriguez and Carlos Marín, "Higher-order corrections for the deflection of light around a massive object," https://doi.org/10.48550/arXiv.1701.04434



(a)

(b)

(c)

Figure 3: Camera POV with black hole. $r_S = 0.01$, $q = 0.1$. Default step size is 0.1. The images are 1920×1080. Render times reported based on 12 threads on a linux machine. (a) No black hole present. Render time is approximately 5 seconds. (b) $T_{max} = 10000$, $T_{min} = 0$, to suppress automatic updates of step size. Render time is approximately 9:03 minutes. (c) $T_{max} = 0.1$, $T_{min} = 0.0001$. Render time is approximately 3.25 hours.

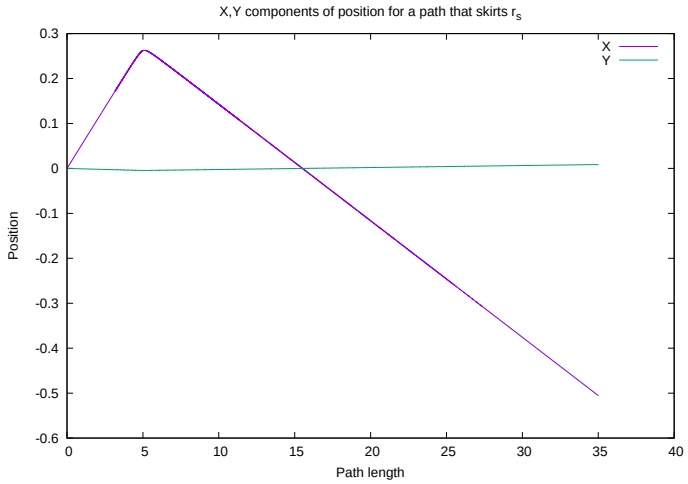


Figure 4: Path components for a ray that skirts the Schwarzchild radius. $T_{max} = 0.01$, $T_{min} = 0.0001$.

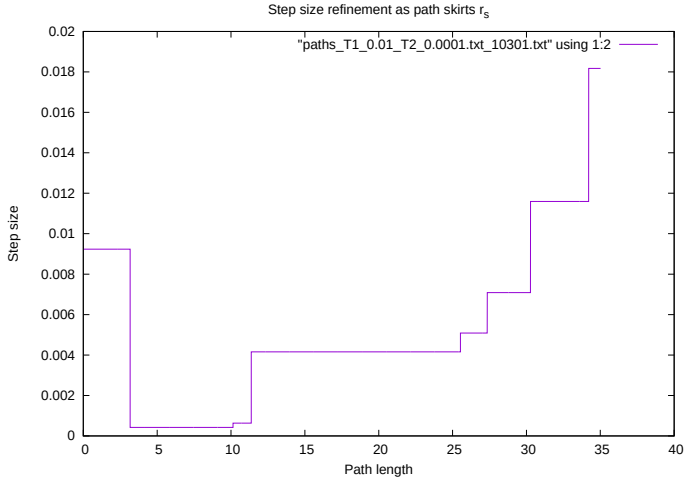


Figure 5: Automatically adjusted step size for a ray that skirts the Schwarzchild radius. $T_{max} = 0.01$, $T_{min} = 0.0001$.

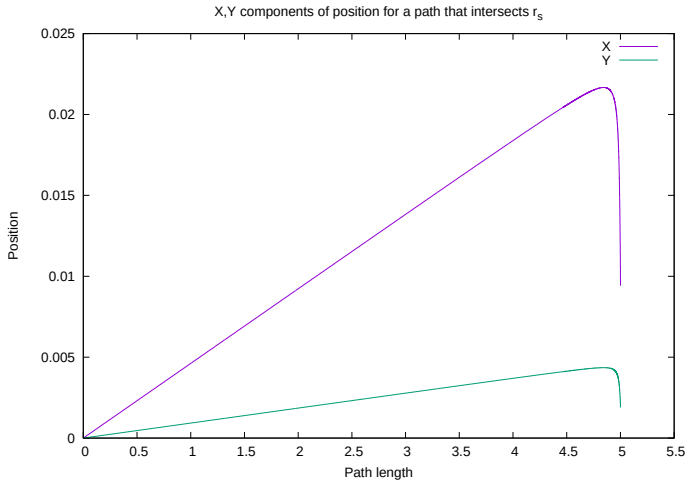


Figure 6: Path components for a ray that stops at the Schwarzchild radius. $T_{max} = 0.01$, $T_{min} = 0.0001$.

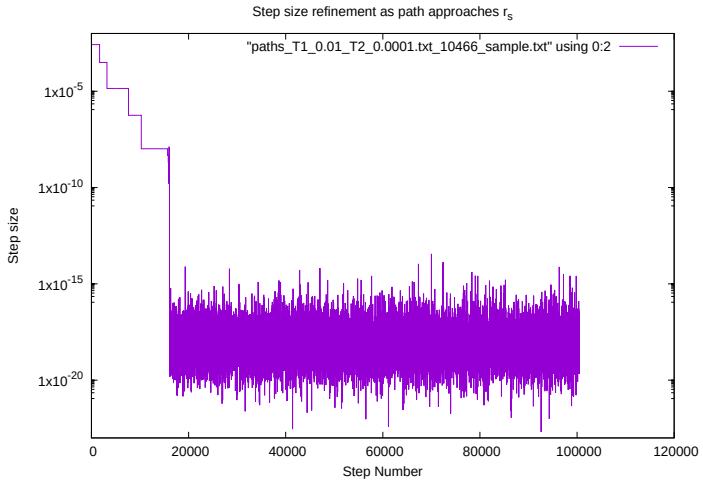


Figure 7: Automatically adjusted step size for a ray that stops at the Schwarzchild radius. $T_{max} = 0.01$, $T_{min} = 0.0001$.

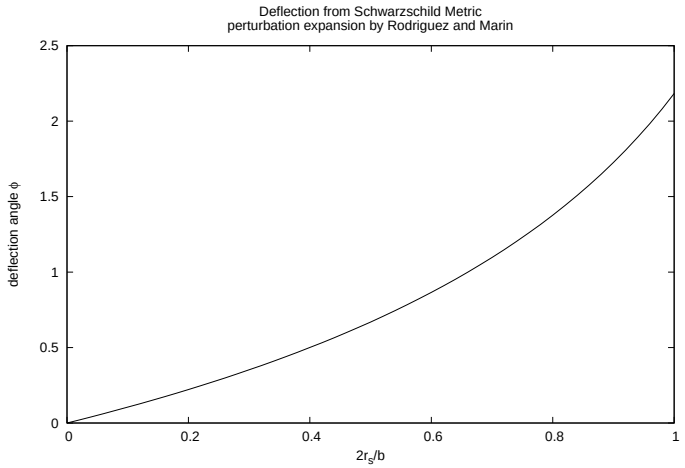


Figure 8: Deflection from a Schwarzschild metric according to the pertubation expansion from Rodriquez and Marin.

$$\phi \;=\; \sum_{i=1}^{20} \kappa_i \left(\frac{3x}{4}\right)^i \tag{27}$$

with the impact parameter $x \;=\; 2r_s/b$, and the coefficients κ_i shown in the results from Rodriguez and Marín. This relationship is for the full GR deflection, whereas our ray bending physically applies for the weak gravitation limit only, and should be physically valid only in some range of small values of the impact parameter. Nevertheless, we calculate ray bending for larger values of the impact parameter, beyond the range of the physical validity of weak gravitation, as a way of testing the numerical stability and self-consistency of the algorithm, but should expect disagreement with equation 27 for larger values of x.

Figure 9 shows both the rendered image and deflection data for a "baseline" case with relatively modest thresholds and maximum number of steps. The scatter data for the deflection angle clusters with near-neglible variance around a curve. The deflection angle exceeds the theory for small impact parameter b, but agrees well with the theory for large impact parameter.

In the top image of figure 10, the maximum error threshold T_{max} was reduce by a factor of 10 from the case in figure 9. The parameters T_{min} and N_{max} are unchanged from figure 9. This lower maximum error continues to produce an image with some reasonable visual accuracy. The images in the middle row of figure 10 reduce the T_{min} value by a factor of 100, and artifacts appear near the Schwarzschild radius. These artifacts are a consequence of the ray step size Δs remaining very small because of the reduced T_{min}, so that some paths exceed N_{max} steps and terminate too soon. In the right image the number of steps N_{max} is increased by a factor of 10, but the artifact remains, although improved slightly. The bottom image increases N_{max} by another factor of 10 and the artifact is no longer visible. There appears to be a close relationship between T_{min} and N_{max}: decreasing the minimum error requires increasing the number of steps to support

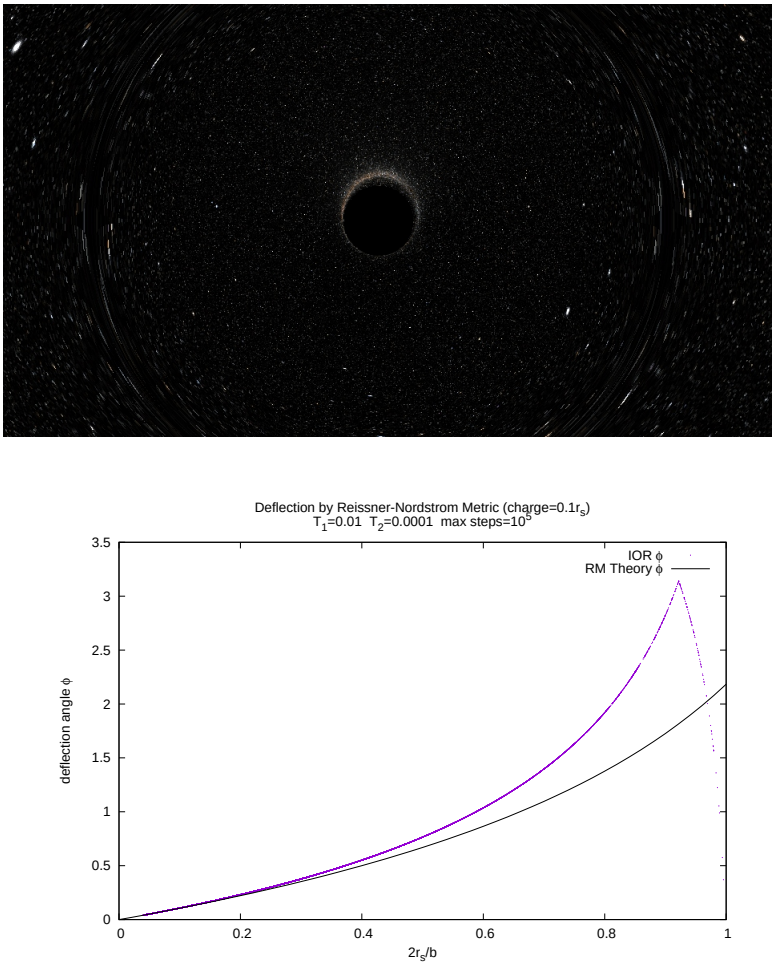


Figure 9: Deflection of rays through a Reissner-Nordstrom metric, with tolerance settings $T_{max} = 10^{-2}$, $T_{min} = 10^{-4}$, $N_{max} = 10^5$. Top: rendered image of a background star field deflected through a Reissner-Nordstrom index of refraction. Bottom: Scatter plot of deflection angles for the computed paths, compared to the pertubative expression for deflection for a Schwarzschild metric in equation 27. Note the agreement in the weak gravitation regime $2r_s/b \lesssim 0.3$.

that minimum error.

5 curved path intersection with a plane

When using curved paths in a computer graphics scene, the paths may intersection objects in the scene. When/where this occurs, the fact of the intersection and its location must be established, and the treatment of the consequences of intersection (reflection, refraction, scattering, etc.) must be generated. In this note we do not deal with the intersection consequences, only with the detection of the intersection and determining its location.

As with straight-line rendering, the geometry of primary interest for intersection is (1) axis-aligned bounding boxes (aabb), and (2) triangles. The aabb serve as containers and placeholders for triangles in acceleration structures like kd-trees and Bounding Volume Hierarchy. For both types of geometry, the fundamental first step is to detect the intersection of the path with one or more planes. In the case of the aabb, multiple plane intersections are computed, and for a triangle only one intersection is calculated followed by evaluation of barycentric coordinates for the intersection point. Once the path-plane intersection detection and point of intersection is obtained, the remainder of the intersection algorithms is the same for curved paths as for straight paths. So in this note we focus on (1) detecting the existence of an intersection with a plane, and (2) determining the point of intersection. These are the only changes to the intersection problem.

Following the approach in the algorithm in section 4, we assume a path has been segmented into a sequence of curved paths. The potential of intersection with geometry means that each segment must be tested for intersection with the geometry in the scene. If no intersection is found, then the next path segment is generated. We consider a single path segment with starts at point $\vec{\mathrm{x}}_0$, in direction $\hat{\mathrm{n}}_0$ and forcing $\vec{\mathrm{F}}$ and extends for segment path length Δs.

We can evaluate the task of computing the point of intersection of a curved ray with a plane. The plane is characterized by the implicit equation

$$(\vec{\mathrm{x}} - \vec{\mathrm{x}}_P) \cdot \hat{\mathrm{n}}_P = 0 \tag{28}$$

where $\vec{\mathrm{x}}_P$ is a reference point on the plane, $\hat{\mathrm{n}}_P$ is the normal to the plane, and $\vec{\mathrm{x}}$ is any point in space lying on the plane. Abbreviating the curved ray expression 20 as

$$\vec{\mathrm{x}}(s) = \vec{\mathrm{x}}_0 + \hat{\mathrm{n}}_0 \, f(s) + \hat{\mathrm{n}}_0 \times \hat{\alpha} \, g(s), \tag{29}$$

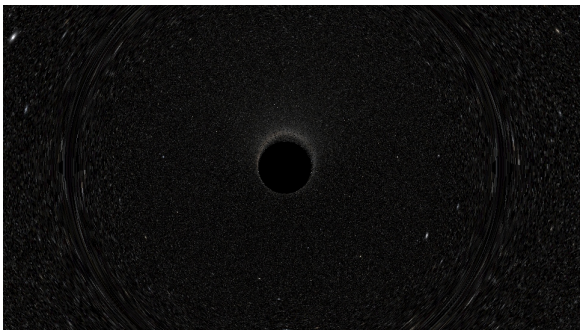
where $f(s)$ is equation 23 and $g(s)$ is equation 24, for $0 \leq s \leq \Delta s$. Figure 11 shows these functions. Note that the function $g(s)$ is monotonic, whereas $f(s)$ may be monotonic or have a single peak. We assume that the segment length is sufficiently small that the value of f remains in the monotonic regime on the left side of the plot.

The equation for intersection of the curved ray with the plane is

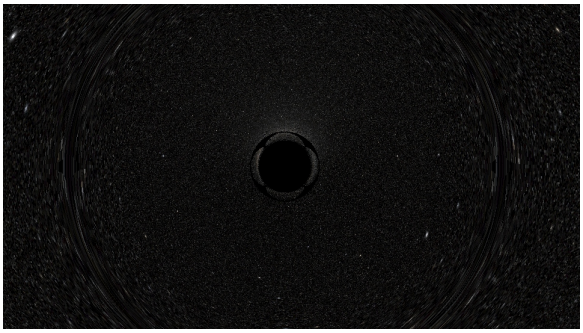
$$L(s) = (\vec{\mathrm{x}}_0 - \vec{\mathrm{x}}_P) \cdot \hat{\mathrm{n}}_P + \hat{\mathrm{n}}_0 \cdot \hat{\mathrm{n}}_P \, f(s) + (\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \hat{\mathrm{n}}_P \, g(s) = 0 \tag{30}$$

This is an implicit function. As a function of s, it is nonlinear. But if we consider this implicit function as function of f, g, it is a straight line.

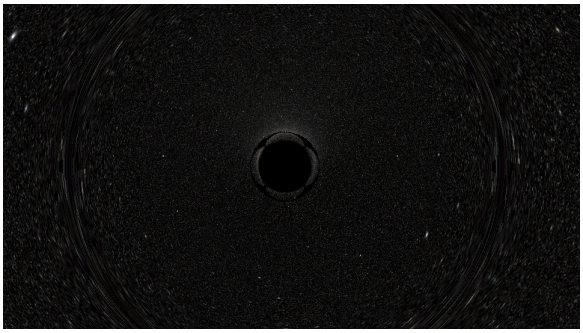
To make this intersection problem manageable, we make the assumption that, within a single segment $0 \leq s \leq \Delta s$, the function $f(s)$ remains monotonic.



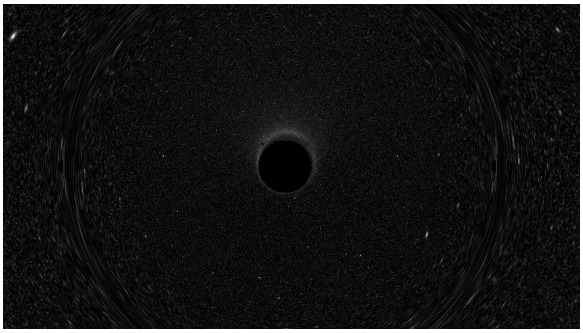
$T_{max} = 10^{-3}$ $T_{min} = 10^{-4}$ $N_{max} = 10^5$



$T_{max} = 10^{-3}$ $T_{min} = 10^{-6}$ $N_{max} = 10^5$



$T_{max} = 10^{-4}$ $T_{min} = 10^{-6}$ $N_{max} = 10^6$



$T_{max} = 10^{-4}$ $T_{min} = 10^{-6}$ $N_{max} = 10^7$

Figure 10: Runs showing errors near the Schwarzschild radius, with some cases having artifacts that are fixed by increasing the maximum number of steps.

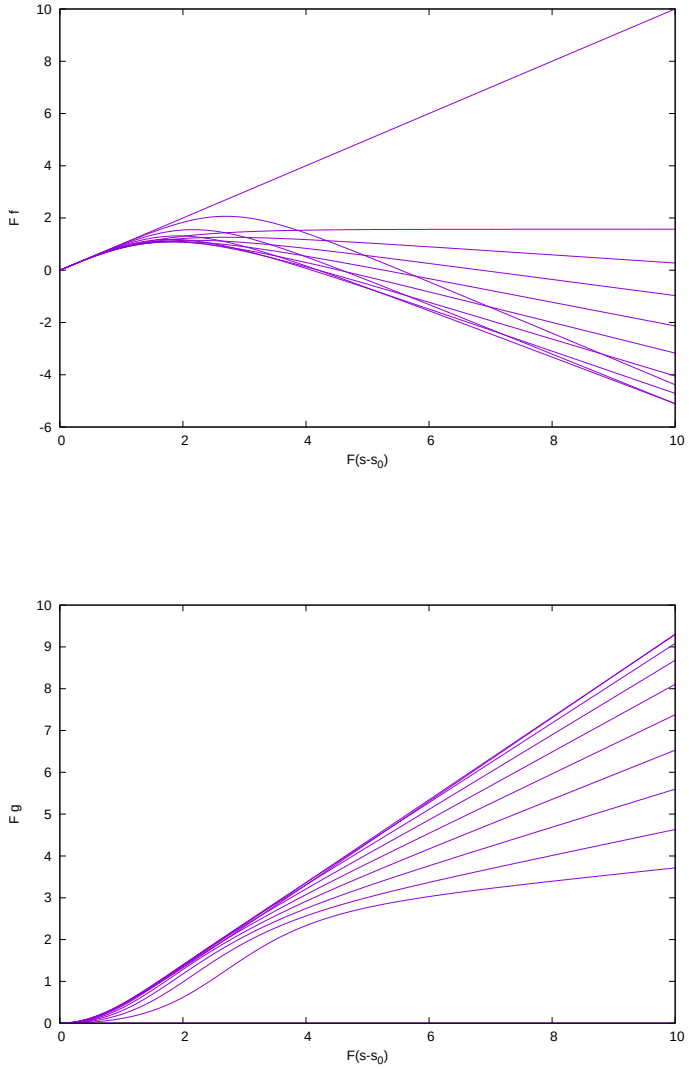


Figure 11: Functions $f(s)$ and $g(s)$ for some choices of β, versus $F(s - s0)$. Top: $F\ f(s)$. Bottom: $F\ g(s)$. Notice that $f(s)$ is quasi-quadratic, but $g(s)$ is monotonic.

5.1 rejecting intersections

Most ray path segments do not intersect geometry, so it is very important to have the simplest possible test of intersection that quickly determines whether there is an intersection or not. Here we lay out such a simple and quick test.

The path segment involves a range of values of the functions f and g that is known in the course of generating the segment. At the beginning of the segment, $s = 0$ and the values of f, g are $0, 0$. At the end of the path, the values of f, g are $f_e \equiv f(\Delta s),\ g_e \equiv g(\Delta s)$. For the purpose of determining whether there is an intersection, the question to address is whether the implicit function $L(s)$ changes sign when evaluated over the range $0 \le s \le \Delta s$. This requires that we evaluate the function at two points:

$$L_0 = L(0) = (\ \vec{\mathrm{x}}_0 - \vec{\mathrm{x}}_P\ ) \cdot \hat{\mathrm{n}}_P \tag{31}$$

$$L_e = L(\Delta s) = (\ \vec{\mathrm{x}}_0 - \vec{\mathrm{x}}_P\ ) \cdot \hat{\mathrm{n}}_P + \hat{\mathrm{n}}_0 \cdot \hat{\mathrm{n}}_P\ f_e + (\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \hat{\mathrm{n}}_P\ g_e \tag{32}$$

If there is no intersection, then the signs of L_0 and L_e will be the same. So the rejection algorithm is:

$$\text{If } L_0 L_e > 0 \text{ then no intersection.} \tag{33}$$

This is a fast test because it is constructed from quantities $f_e, g_e, \hat{\mathrm{n}}_0, \hat{\mathrm{n}}_0 \times \hat{\alpha}$, that are already available from the construction of the path, and linear algebra quantities with $\vec{\mathrm{x}}_P, \hat{\mathrm{n}}_P$. Thus rejection of an intersection with a given plane is a fast operation.

5.2 refined search for intersection point

If $L_0 L_e \le 0$ then there is an intersection and we must determine the location of the intersection, i.e. the value of s where there is an intersection. There are three possible situations that must be evaluated:

1. $L_0 = 0$ and the intersection is at $s = 0$, which is the point $\vec{\mathrm{x}}_0$.
2. $L_e = 0$ and the intersection is at $s = \Delta s$, which is the point $\vec{\mathrm{x}}_0 + \hat{\mathrm{n}}_0 f_e + (\hat{\mathrm{n}}_0 \times \hat{\alpha}) g_e$.
3. Both L_0 and L_e are non zero but of different sign, and the intersection must be found at a point along the path segment.

In situation 3, the simplest solution approach is the bisection method[5], which in this situation looks like:

1. Initialization: set

$$\begin{aligned} s_a &= 0 \\ s_b &= \Delta s \\ f_a &= 0 \\ g_a &= 0 \\ f_b &= f_e \\ g_b &= g_e \\ L_a &= L_0 \\ L_b &= L_e \\ N &= 0 \end{aligned}$$

[5] Bisection Method: https://en.wikipedia.org/wiki/Bisection_method

2. Compute an intermediate point:

$$\begin{array}{rcl} s_i & = & (s_0 + s_1)/2 \\ f_i & = & f(s_i) \\ g_i & = & g(s_i) \\ L_i & = & (\ \vec{\mathrm{x}}_0\ -\ \vec{\mathrm{x}}_P\ ) \cdot \hat{\mathrm{n}}_P\ +\ \hat{\mathrm{n}}_0 \cdot \hat{\mathrm{n}}_P\ f_i\ +\ (\hat{\mathrm{n}}_0 \times \hat{\alpha}) \cdot \hat{\mathrm{n}}_P\ g_i \\ N & += & 1 \end{array}$$

3. If $L_i = 0$, or N exceeds a threshold, then the intersection is at s_i, $\vec{\mathrm{x}}_i\ =\ \vec{\mathrm{x}}_0 + \hat{\mathrm{n}}_0\ f_i\ +\ \ \hat{\mathrm{n}}_0 \times \hat{\alpha}\ g_i$

4. If $L_i L_a < 0$ then the intersection is between s_a and s_i. Set $s_b = s_i$, $f_b = f_i$, $g_b = g_i$, $L_b = L_i$. Return to step 2 and iterate.

5. If $L_i L_b < 0$ then the intersection is between s_b and s_i. Set $s_a = s_i$, $f_a = f_i$, $g_a = g_i$, $L_a = L_i$. Return to step 2 and iterate.

This bisection algorithm can be modified in two ways: (1) the termination test in step 3 could also test for $|L_i|$ to be below a threshold, and consider that threshold to be sufficiently close to 0; (2) instead of evaluating $f(s_i), g(s_i)$ in each iteration, it may be faster to build look-up tables of values of f, g within the interval prior to the start of the algorithm, and interpolate those values (this would be faster, but could be less accurate). This intersection problem has properties (monotonicity) that are what make the bisection algorithm work well.