EXPLORING BACKTRACING AS AN $O(n)$ ALTERNATIVE TO GENERALIZED SDF FOR THE TORUS AND MORE

Abstract. The use of generalized signed distance functions (GSDF) for raytracing provides a constant-time evaluation for a single ray's distance to a subject. For the torus, however, finding a closed-form algebraic solution is not as simple as for other shapes. One exists (REFERENCE TORUS PAPER), but it is rather long and costly to evaluate. Here, I offer "backtracing", an alternative to the GSDF which also runs with $O(n)$ complexity with respect to the number of pixels to be rendered. One parameterizes the torus, computes a parameterization resolution bounded by the image resolution, tabulates focus-to-surface distances at the computed resolution, and accesses them as needed when rendering pixels. Several potential methods are presented, with only one—recursive sampling—being immediately applicable to the torus. Although recursive sampling is shown to run in $O(n)$, there exist further optimizations that may allow it to be even faster.

1. Introduction

1.1. **Motivation.** In raytracing, speed is of high interest. One of the main obstacles to ray tracing being a means of animation is its high time cost compared to other more common methods like rastering. Ray tracing can produce very realistic images, due to their adherence to physics, but real-time animation remains difficult, as ray tracers are often time-costly.

Raytracers are most commonly implemented with signed distance functions of the form $\Phi(\mathbf{p}) : \mathbb{R}^3 \to \mathbb{R}$ which return the shortest distance from a point, $\mathbf{p}$, to an object in the scene. Iterative "ray marching" methods are used to determine the point of contact between a ray originating from the focal point and objects in the scene.

Generalized SDFs (GSDF) have been presented as an alternative to SDFs for many "primitive" shapes, with which many scenes can be constructed, have been determined and can be evaluated smoothly. This is a faster alternative to the conventional ray marching technique which uses (nongeneral) SDFs and recurses at each pixel to detemine the contact point of a ray with the subject within a certain tolerance. The GSDFs tend to be more complicated than the nongeneral SDF counterpart. The sphere, for example, has the following SDF (essentially a distance function):

$$\|(\vec{p} + \phi\hat{v})\| = r$$

, and its GSDF is:

$$\Phi_G(= -\hat{\mathbf{v}} \cdot \mathbf{p} \pm \sqrt{r^2 + (\hat{\mathbf{v}} \cdot \mathbf{p})^2 - \|\mathbf{p}\|^2}$$

. The GSDF is more complicated and involves more computations, but succeeds in circumventing the need for an iterative method to determine points of intersection. GSDFs for many other convex shapes—the box, plane, cone, and cylinder to name a few—have been determined and are proven to work effectively. Such GSDFs are

often determined by solving algebraic constructions of the form:

$$\mathbf{p} + \hat{\mathbf{v}}\Phi \in S$$

, solving a characteristic equation for the distance at which a ray originating from $\mathbf{p}$ is in the shape, S. In convex shapes, these characteristic equations are generally quadratic and always have two roots, as a ray through a convex shape is guatanteed to only enter the shape once and exit it once. For the convex torus however, the maximum number of contact points is four and its characteristic equation is a quartic. (UNIV W. BOHEMIA) details that quartic, but it is visibly quite complex and solving quartics is too costly for our purposes. It would likely be easier and faster to iteratively evaluate the SDF, which is quite simply expressed:

- float sdTorus(vec3 p, vec2 t) {
  - vec2 q = vec2(length(p.xz)-t.x,p.y);
  - return length(q)-t.y;
- }

(CITE QUILEZ). Finding a closed form GSDF is not a viable option, so we search for an alternative.

1.2. **Overview of Back Tracing.** For this reason, we introduce "back tracing". The torus has a fairly simple parameterization so identifying points on its surface is easy. "Tracing" a ray from the focus of the scene, through the desired pixel, and onto the surface of the torus is difficult. The idea of backtraceing is to obtain this information in reverse. If we sample points on the surface of the torus and construct the lines that connect them to the focus, the intersections of those lines and the screen—which can be easily obtained by the plane GSDF—may be enough to render the desired image. When we say that a torus point back traces to a certain point on the screen, we mean that that screen point lies along the line joining the torus point and the focus.

In order to do this, we must carefully determine which points on the torus to sample. If not, we may have blank pixels on the screen which would make the method unusable. If we can guarantee that there is at least one backtraceed ray that intersects each pixel that should be hit by a torus-intersecting ray, we guarantee that we have no blank pixels. In the rendered image, many pixels will represent a point on the torus that does not correspond to the ray passing through middle of the pixel as we have with the GSDF. However, this provides an error which is smaller than than the pixel size. In all instances of imaging—both biological and digital—there is an effective pixel size, below which error is discarded. In the human eye, it is the size of the cones and rods. In a digital camera, it is the size of each individual photodiode in the CCD.

2. Theory

2.1. **Parameterization.** First, we define our parameterization of the torus.

$$\mathbf{x}(\theta,\phi) = R(\mathbf{b}_1\cos\theta + \mathbf{b}_2\sin\theta) + r((\mathbf{b}_1\cos\theta + \mathbf{b}_2\sin\theta)\cos\phi + \mathbf{b}_3\sin\phi)$$

$$\mathbf{x}(\theta,\phi) = (\mathbf{b}_1\cos\theta + \mathbf{b}_2\sin\theta)(R + r\cos\phi) + \mathbf{b}_3 r\sin\phi$$

(INSERT TORUS IMAGE), where $\mathbf{b}_3$ is the central axis of the torus, $\mathbf{b}_1, \mathbf{b}_2$ are an orthonormal basis of the plane orthogonal to $\mathbf{b}_3$, r is the small radius, and R is the larger radius. In other words, we imagine the torus as a hoop of radius R with

thickness $2r$. Rather conveniently, the unit normal presents itself as a term in the first expression of the parameterization:

$$\hat{\mathbf{n}} = (\mathbf{b}_1 \cos\theta + \mathbf{b}_2 \sin\theta)\cos\phi + \mathbf{b}_3 \sin\phi$$

.

The usefulness of this feature is (at least) two-fold. Firstly, we can choose to only deal with parts where $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}}_{\text{screen}} \leq 0$, since the rest of the torus is invisible. Secondly, when deciding how to shade the torus, we need information about the normal vector at each point to be able to determine the angle of incidence of rays from the light source.

2.2. **Pixel Size and Grid Size.** As previously mentioned, it is necessary to ensure that the parametric grid is sufficiently sized for each pixel to intersect at least one backtraced ray. One way to ensure this is to ensure that consecutive points on the torus back trace to consecutive pixels on the screen. A simple geometric argument shows why this is the case:

- Consider the back traced lines of two consecutive torus points that intersect at the focus (by construction).
- Construct the cone whose axial angle is twice the angle between the two rays, i.e. such that the two rays are diametrically opposed.
- The cross section of this cone on the screen is an ellipse.
- Any ray who maps to a pixel between those of these two rays must have an intersection point inside the ellipse on the plane of the screen.
- Construct a third ray that also passes through the focus and a third point in the parametric direction between the first two torus points, but lies outside those points.
- We notice that that third ray has a larger angle with the cone's central axis than the first two rays, and is therefore not in the cone.
- Thus, any elliptical cross section will not contain the third ray's intersection with that plane.

Then, we observe that the two adjacent torus points back trace to adjacent points on the screen, and their separation must be less than the pixel size. When implementing some form of back tracing, we will need to ensure that adjacent torus points back trace to adjacent pixels—a condition we will call the the adjacency condition (AC). The ideal backtracing method finds the minimal number of points such that AC is held. From here, we explore a few methods—some of which have proven to succeed on the torus, and others which have proven not to, but may work on other shapes with sufficient geometric conditions. Sections 2.4 and 2.5 contain descriptions to methods which do not work but have potential to be efficient in other scenarios, and section 2.6 contains a working method for rendering the torus.

2.3. **Constant Grid.** The idea of the constant grid method is to find the maximum gap we can expect between consecutive screen points. We do this by finding the point of least convergence (PLC) on the torus; the point that, given a constant grid size, backtraces to the largest separation between its neighbors on the screen. We set the grid size such that this separation is equal to the pixel size in both parametric directions, and apply this grid size to the entire torus. This guarantees that AC is satisfied. Analytically determining that point on the torus is a geometric problem that remains yet unsolved, making this method yet unusable, but we can

detail the steps to using it in the case where the torus is centered in the field of view. More formally, we can describe this as the case where the following are true:

- $\hat{\mathbf{n}}_{\text{torus}} \parallel \hat{\mathbf{n}}_{\text{screen}}$ at the PLC
- the PLC is the closest point on the torus to the focus
- the PLC lies on the exterior of the torus (and not the hole-side), i.e. $\cos\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$

The first step in the computation is to determine the point where

$$(\mathbf{b}_1 \cos\theta + \mathbf{b}_2 \sin\theta)\cos\phi + \mathbf{b}_3 \sin\phi = \hat{\mathbf{n}}_{\text{screen}}$$

, which can be done by solving a 3-by-3 system for $\cos\theta\cos\phi, \sin\theta\cos\phi, \sin\phi$ then a 2-by-2 system for ϕ, θ. We then seek an expression relating the on-screen gaps of rays backtraceed from consecutive grid points. In particular, we see how the length of a short walk, from the PLC, along the circular parametric contours can impact the screen projection distance between the destination of that walk and the PLC. The angle, α, between the backtraceed rays from consecutive grid points, where $(\mathbf{b}_1 \cos\theta + \mathbf{b}_2 \sin\theta)\cos\phi + \mathbf{b}_3 \sin\phi = \hat{\mathbf{n}}_{\text{screen}}$, with respect to the focus can be expressed as

$$\frac{(\mathbf{b}_1 \cos(\theta+\Delta\theta) + \mathbf{b}_2 \sin(\theta+\Delta\theta))(R + r\cos\phi + \mathbf{b}_3 r \sin\phi - \mathbf{f}}{\|(\mathbf{b}_1 \cos(\theta+\Delta\theta) + \mathbf{b}_2 \sin(\theta+\Delta\theta))(R + r\cos\phi) + \mathbf{b}_3 r\sin\phi - \mathbf{f}\|} \cdot \hat{\mathbf{n}}_{\text{screen}} = \cos\alpha$$

or

$$\frac{(\mathbf{b}_1 \cos\theta + \mathbf{b}_2 \sin\theta)(R + r\cos(\phi+\Delta\phi)) + \mathbf{b}_3 r \sin(\phi+\Delta\phi) - \mathbf{f}}{\|(\mathbf{b}_1 \cos\theta + \mathbf{b}_2 \sin\theta)(R + r\cos(\phi+\Delta\phi)) + \mathbf{b}_3 r\sin(\phi+\Delta\phi) - \mathbf{f}\|} \cdot \hat{\mathbf{n}}_{\text{screen}} = \cos\alpha$$

depending on the direction of the step, where the grid size is defined by the step sizes, $\Delta\theta, \Delta\phi$. Then,

$$l_f \tan\alpha = \frac{\cos\alpha}{\sqrt{1-\cos^2\alpha}} l_f = h$$

gives the gap size, where l_f is the length from the focus to the screen. We hope to solve for $\Delta\theta, \Delta\phi$ which is difficult to tackle analytically. We instead consider using a first-order taylor approximation instead to linearize the system, solving for distances along the tangent plane at the PLC.

To find a desired grid size for our Taylor approximation, we think about the walk along the torus we described earlier. We break this walk up coordinate-wise along the θ-direction and then along the ϕ-direction. These two walks are both along circles, so we can easily determine their lengths: $(R + r\cos\phi)\Delta\theta$ and $r\Delta\phi$. This gets us from $\mathbf{x}(\theta, \phi)$ to $\mathbf{x}(\theta + \Delta\theta, \phi + \Delta\phi)$. For our approximation, we start at $\mathbf{x}(\theta, \phi)$ and walk $(R + r\cos\phi)\Delta\theta$ and $r\Delta\phi$ along the $\hat{\theta}$ and $\hat{\phi}$ directions on the tangent plane. This is an under approximation of the grid size, as this walk is entirely in the direction of the screen plane, meaning walking that same distance along the torus would result in a smaller projection difference on the screen.

The walk has a length of $\sqrt{[(R + r\cos\phi)\Delta\theta]^2 + [r\Delta\phi]^2}$. Knowing that the walk was along a plane parallel to the screen, we can use simple trigonometry to solve

$$\frac{\sqrt{[(R + r\cos\phi)\Delta\theta]^2 + [r\Delta\phi]^2}}{\Phi(\mathbf{p}_{\text{PLC}})} = \frac{h}{l_f}$$

, where $\Phi(\mathbf{p}_{\text{PLC}})$ is the value of the SDF of the torus evaluated at the focus.

GOOD PLACE FOR A FIGURE

$$[(R + r\cos\phi)]^2[\Delta\theta]^2 + r^2[\Delta\phi]^2 = (\frac{\Phi(\mathbf{p}_{\text{PLC}})h}{l_f})^2$$

for solutions to $\Delta\phi$ and $\Delta\theta$. Since we can break the walk into components, and we only need adjacent pixels along each direction to (separately) backtrace to distances less than h, we can just solve

$$r\Delta\phi = \frac{\Phi(\mathbf{p})h}{l_f} \implies \Delta\phi = \frac{\Phi(\mathbf{p}_{\text{PLC}})h}{rl_f}$$

and

$$(R + r\cos\phi)\Delta\theta = \frac{\Phi(\mathbf{p})h}{l_f} \implies \Delta\theta = \frac{\Phi(\mathbf{p}_{\text{PLC}})h}{(R + r\cos\phi)l_f}$$

This works well and is quite low-cost. We need only do multiplication and evaluate some trigonometric functions We get a solution for our parametric grid size which upper bounds the screen projection of our backtraceed vectors to h. However, we must consider the adverse effects of an over approximation. We know that the backtraceed screen projection distance at the point where $\hat{\mathbf{n}}_{\text{torus}} \parallel \hat{\mathbf{n}}_{\text{screen}}$ is maximum. However, at points on horizon of the torus, consecutive gridpoints are separated along directions nearly orthogonal to the screen plane, and thus project to pixels much closer than those we analyzed to calculate h. Due to this "clustering"effect, we may end up with much more data than we need, and therefore wasted calculations. The Taylor approximation may further this problem, as it provides an underestimate on the necessarry grid size. We would therefore have two considerations for sources of potentially-costly over-estimation, with clustering being the more profound.

2.4. **Back tracing with Iterative Sampling.** One way we may combat the clustering effect at the boundary is to continuously adjust the resolution, so that we do not use undersized gridlines and save some calculations. This method of iterative sampling is incompatible with the geometry of the torus, but could work on other shapes as we will see. Additionally, we must remember that this suffers the same pitfall as the previous method since the determination of the PLC remains unsolved. The steps are detailed here in the context of the torus, as this method could be used to render a part of the torus, but could be adapted to other shapes, so long as we can parameterize them and express walk lengths along their parametric contours. The idea is to start with the closest point on the torus and find a reasonable angular step in one direction, either $\hat{\phi}$ or $\hat{\theta}$, as we did above. We then walk along the torus according to the resolution we determined. Then, we use the first degree Taylor approximation, constructing the tangent plane to the torus as we did before, and look for the distance along that tangent plane we need to walk to get point that backtraces to a screen point h away from the previous point. If we continue this until we are in the range where $\hat{\mathbf{n}}_{\text{torus}} \cdot \hat{\mathbf{n}}_{\text{screen}} \geq 0$, and we are out of the visible sector of the torus. This gives us backtraces of an entire row or column of gridpoints (which are not evenly spaced, but spaced such that we do not have erroneous blank pixels). Once we get this row or column, we can use the each computed point as a starting point to calculate its entire row or column.

Finding the walk distance is not as simple as it was in the case where the tangent plane was parallel to the screen, but still manageable with high school-level techniques. We will end up applying this technique recursively, so let us assign

indexed notation. The point computed from the previous torus walk is $\mathbf{p}_{n-1}$, $\Phi_{n-1} \equiv \Phi(\mathbf{p}_{n-1})$, and $\hat{\mathbf{v}}_{n-1}$ will refer to the unit normal in the direction from $\mathbf{p}_{n-1}$ to the focus, $\mathbf{f}$, all of which were computed at the previous step. We consider the triangle between $\mathbf{p}_{n-1}$, $\mathbf{f}$, and the yet undetermined new point along the tangent plane. We know the angle at the vertex of $\mathbf{f}$ is $\alpha \equiv \arctan(\frac{h}{l_{n-1}})$ where l_{n-1} is the distance from the screen projection of the previous point to the focus (easily calcualted from the information stored about last iteration). We also know the angle at the vertex of $\mathbf{p}_{n-1}$ is $\beta \equiv \arccos(\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1})$. This is enough to use the law of sines to determine the distance, D, we need to walk along the normal plane.

$$\frac{\Phi_{n-1}}{\sin(\beta)} = \frac{D}{\sin(\frac{\pi}{2} - \alpha - \beta)}$$

$$\sin(\frac{\pi}{2} - \alpha - \beta)\frac{\Phi_{n-1}}{\sin(\beta)} = \cos(\alpha + \beta)\frac{\Phi_{n-1}}{\sin(\beta)} = D$$

. We can bypass the need to compute inverse trigonometric functions by using the pythagorean identity and angle addition formulas.

$$\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1} = \cos\beta \implies \sin\beta = \sqrt{1 - [\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}]^2}$$

, and

$$\frac{h}{l_{n-1}} = \tan\alpha \implies \cos\alpha = \frac{1}{\sqrt{[\frac{h}{l_{n-1}}]^2 + 1}}, \sin\alpha = \sqrt{\frac{[\frac{h}{l_{n-1}}]^2}{[\frac{h}{l_{n-1}}]^2 + 1}}$$

. This may look messier, but computing trigonometric functions is costlier (especially inverses), and speed is king in raytracing. The angle addition formula reads

$$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$

$$D = \cos(\alpha+\beta)\frac{\Phi_{n-1}}{\sin(\beta)} = \left[\frac{\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}}{\sqrt{[\frac{h}{l_{n-1}}]^2 + 1}} - \sqrt{\frac{[\frac{h}{l_{n-1}}]^2}{[\frac{h}{l_{n-1}}]^2 + 1}}\sqrt{1 - [\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}]^2}\right] \frac{\Phi_{n-1}}{\sqrt{1 - [\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}]^2}}$$

$$D = \left[\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1} - \frac{h}{l_{n-1}}\sqrt{1 - [\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}]^2}\right] \frac{\Phi_{n-1}}{\sqrt{(1 - [\hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}]^2)([\frac{h}{l_{n-1}}]^2 + 1)}}$$

. If we store $A = \hat{\phi} \cdot \hat{\mathbf{v}}_{n-1}$ and $B = \frac{h}{l_{n-1}}$, we can rewrite this as

$$D = \left[A - B\sqrt{1 - A^2}\right] \frac{\Phi_{n-1}}{\sqrt{(1 - A^2)(1 + B^2)}}$$

, which minimizes the square roots and recomputing repeating expressions. Then, depending on which direction we are stepping in, $D = (R + r\cos\phi)\Delta\theta$ or $D = r\Delta\phi$. We then compute the corresponding value of $\Delta\theta$ or $\Delta\phi$, take the next step along the torus and repeat.

The method requires that the region we step towards is less parallel to the screen than the tangent at the previous point. Without that condition, the reasoning we used to guarantee that the taylor method under approximates the grid size is no longer valid. This method is not applicable to the torus since it has parametric contours along which this condition does not hold, for example, when transitioning from the front side to the back side of the torus.

2.5. **Recursive Sampling.** The previous methods of iterative sampling and constant grid sampling have unresolved problems and are not usable. This method, however, works and has been proven to exhibit the desired properties with respect to running time and generalizability. The method of "recursive sampling" is presented now, as a divide-and-conquer style alternative that uses a breadth-first search. It too deals with the idea that different parts of the torus will need to be sampled at different rates due to the geometry, but uses little actual geometric insight. It goes as follows:

- Choose a grid in θ and ϕ, step sizes dt, dp
- Initialize an empty queue
- Initialize an empty hash table (as described in the hashing section)
- For each parametric grid square:
  - Add it to the queue, with one point in each square (with the same relative position in each square) defined as the "representative".
  - Hash the representative into the table
- While the queue is not empty:
  - Pop the first element in the queue and consider its grid square
  - If the representative backtraces to a pixel adjacent to those of the three other grid square points, remove it from the queue
  - If at least one of the other points is does not backtrack to a pixel adjacent to the representative's,
    - Define five new points and construct the four new grid sqares that subdivide the current grid square and add them to the back of the queue
    - Hash all of the five new points into the table

We can prove this algorithm is correct in satisfying the condition:

We are given some points on the torus. At the first pass, we check whether each grid square (a neighborhood of points on the torus) admits any blank pixels. If it does not, then that region is "safe". If it does, then we subdivide the region completely (recursively, as deeply as necessary), and verify that those subregions are safe. Since the union of the subregions is the original region, we know that all of the subregions being safe is equivalent to the original region being safe. We generalize this argument to the whole torus. Once all of the original grid squares (which fully cover the torus) are safe, we are guaranteed that the entire torus is safe.

3. Numerics and Implementation of Desired Method

3.1. **Tabulation and Pixel Determination.** We must be sure that a back tracing method will provide us with the information we need to seamlessly interface with a ray tracer that may use GSDFs for the remainder of its shapes. The idea is that the back tracing algorithm will determine a distance to the torus associated with each pixel on the screen, that distance being the distance from the focus along the ray direction that passes through that pixel and the focus. In any instance of a composite scene of several shapes, the GSDFs of the other shapes will be evaluated on a pixel-by-pixel basis, with one ray direction being evaluated for each pixel. If we tabulate one value for each pixel which intersects a ray that contacts the torus, then we can compare the value with other GSDFs as usual to determine which

object is actually seen. This allows us to preserve the properties of GSDFs that allow us to make unions and intersections of shapes.

For each point of interest on the torus, we tabulate the screen intersection of the back tracing ray at each gridpoint. This is done using the GSDF of the plane. We compute

$$-\frac{(\mathbf{p}-\mathbf{c})\cdot\hat{\mathbf{n}}}{\hat{\mathbf{v}}\cdot\hat{\mathbf{n}}}\Phi_{\mathbf{G},\mathbf{p}}$$

, where $\hat{\mathbf{n}}$ is the unit normal of the screen, $\mathbf{p}$ is the gridpoint, $\hat{\mathbf{v}}$ is the unit vector along the direction from $\mathbf{p}$ to the focus, and $\mathbf{c}$ is any point on the screen. The screen point is

$$\mathbf{p}+\Phi_{\mathbf{G},\mathbf{p}}\hat{\mathbf{v}}$$

. We save these screen points in a hash table. The hash keys for each ray are the cooordinates of the pixel through which that ray passes. This can be computed by calculating its coordinates in the basis of the screen plane and dividing them by the pixel size to determine the indices where they lie. These are the same indices over which we iterate when computing the color data for each pixel. There may be several rays passing through a pixel, and hashing them in the same key compromises the desired $O(1)$ lookup time. In order to bypass this, we must decide what to do with hashing collisions. To have higher accuracy to GSDF behaviour, whenever we have a collision when hashing a new ray's screen point, we compare it to the value which is already there and store only the point which is closer to the center of the pixel. On the other hand, we could prioritize speed by only taking the first point to collide with each pixel and discarding any rays that might map to that pixel in the future. Either method has the benefit that we only need to store, at most, as many points as pixels.

Once we have hashed the backtraced rays for each grid point, we need only iterate over each pixel in the image and see whether it collided with some ray. If it did, we continue on with checking the incident rays from the light source and evaluating some shading function to get color data.

3.2. **Degree of Overestimation in Constant Grid Sampling.** In order to estimate the extend of the clustering effect, we use what we derived in section 2.5 to come up with a Taylor approximation for how the screen projection distances change at various points along circular contours.

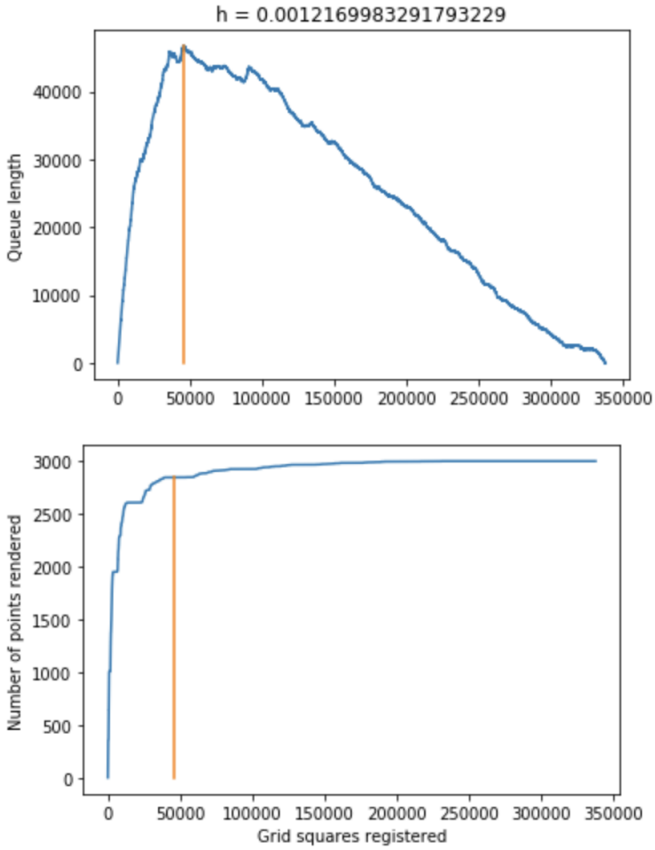
3.3. **Running Time for Recursive Sampling.** To analyze the running time of this algorithm, we consider the length of the queue at each pass over the remaining grid. Suppose we start with k grid points and k grid squares. At the next pass over the torus, in the worst case, we needed to refine every single grid. There are then $4k$ grid squares. This continues until we reach the number of points we actually need to render the torus, n. We can sum up these evaluations:

$$k+4k+...\frac{n}{4}+n$$

. Since $\frac{k}{n}$ is not necessarily a clean power of four, we add an extra term to account for the worst-case the event that $k = 3.99999...0$ and we need to quadruple again to a value close to $4n$:

$$k+4k+...\frac{n}{4}+n+4n = n(4+1+\frac{1}{4}+\frac{1}{16}+...\frac{k}{n}) \leq 5\frac{1}{3}n$$

FIGURE 4.1. Queue size (top) and table size (bottom) over time, pixel width: $h = 0.0012$.



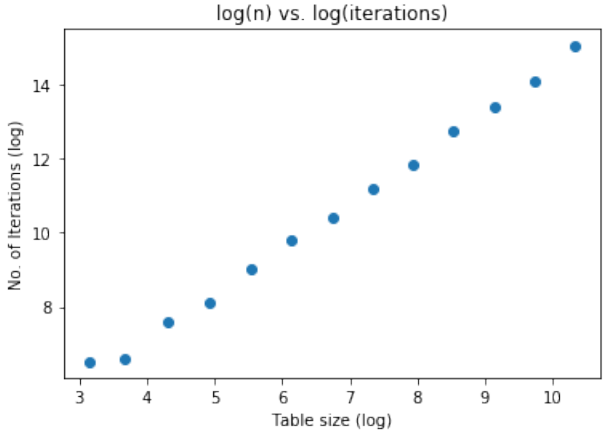
. We see that our algorithm's running time is bounded from above by a first degree polynomial, and is therefore $O(n)$.

4. RESULTS

Here we analyze the behavior of the recursive sampling method and data on its running time. (4.1) shows the rate at which the algorithm amasses data points.

The top graph shows the length of the queue over time. What we see fits our qualitative explanation of the algorithm. At first, we see the the size of the queue grow exponentially and then shave off in a slow linear fashion. The first phase corresponds to the rapid quadrupling of the grid size to refine the resolution. Then, once enough grid squares are adequately sized, we would expect the growth to turn into decay, with the algorithm mainly hashing the points which are already on the queue. The bottom graph shows the hash table size over time. The first phase shows the table quickly amassing data, and the second phase shows it tapering off significantly and slowly reaching its complete size.

FIGURE 4.2. log-log plot of Iterations vs. Pixel Count



Across nine different values of h, we measured that the maximum queue size is attained after 17% of the iterations with a standard deviation of 3%. At the time of maximum queue length, 95% of the data that was to be stored had been stored already, with a standard deviation of 2%. Although the rigor of these statistics is questionable on a set of nine data points, the result is telling. Despite the theoretical and empirical (as we will soon see) evidence that this algorithm is $O(n)$, there is an apparent inefficiency when 95% of the data is found in the first 17% of the running time. The remaining 83% of the running time is dedicated to the last 5% of the data, meaning there is a high rate of hashing collision. It may be worthwhile to investigate using a different method to obtain the final 5%. For example, ray marching may be sufficiently fast on the remaining 5% of the data to be effective, allowing us to make use of the front-heaviness of recursive sampling without losing time to the stagnant second phase.

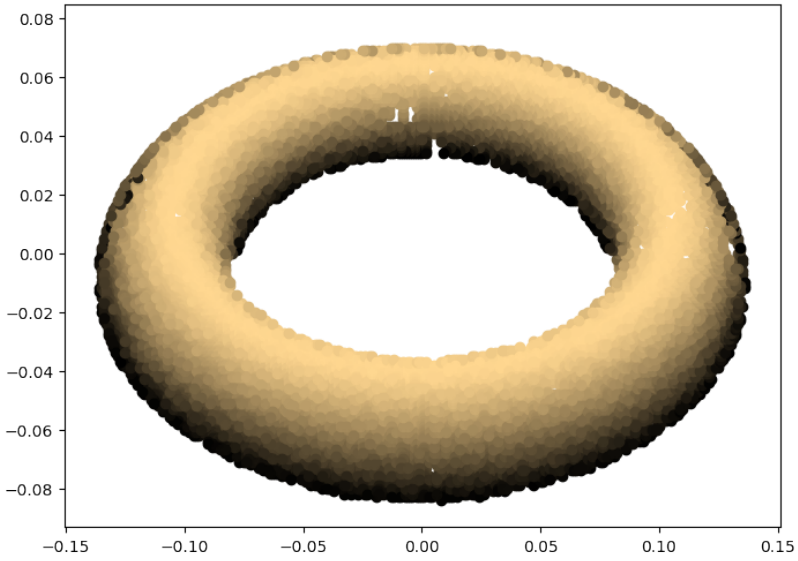
Even still, the model shows evidence of running in linear time. The log-log plot in (FIGURE) yields a 1.12-power relationship, which is indicative of $O(n)$ behavior.

(shows a scatter plot of the locations of the coordinates of the registered pixels with respect to the center of the screen, in the basis of the screen.

5. Conclusion

Back tracing with recursive sampling has proven to be a viable linear-time replacement for a GSDF. In addition, it has the capacity to interface effectively with an otherwise GSDF-based ray tracer which is further support of its viability. The other proposed methods of back tracing, although incomplete and unapplicable to the torus, may be applicable to other shapes with more ideal geometric characteristics. Recursive sampling, although fitting our criteria for a viable alternative to GSDF, showed room for improvement with its capacity to collect data being substantially more concentrated in the first $\sim$ 17% of its running time. Further research into back tracing methods may benefit from combining several methods to bypass the inefficient second phase of the recursive sampling method. It is also worthwhile to investigate the determination of PLC in both the torus and other

FIGURE 4.3. The color was set to $(246, 217, 150)$. A simple shading function was used, where the intensity of the pixel was the angle of incidence of light from the light source. The torus is centered at the origin with $R = 3$, $r = 0.5$ and the light source is at $(0, 0, 10)$



shapes, as it is the only obstacle in constant grid back tracing. Once PLC determination is established, implementations of iterative back tracing can be tested and adapted.