yt SVR Algorithm Handover Package

Chris Gyurgyik, Ariel Kellison, and Youhan Yuan

GitHub Repository

www.github.com/spherical-volume-rendering/svr-algorithm

-Introduction-

This document provides the details of the algorithm for spherical voxel traversal developed and implemented by Chris Gyurgyik, Ariel Kellison, and Youhan Yuan for the yt open source visualization package. The first part of the document provides informal pseudocode for the algorithm. This is followed by descriptions of the implementation API, the testing methodology with a prototypical example, and benchmarking results.

-Algorithm-

The spherical coordinate traversal for ray tracing algorithm developed for integration into the yt package consists of two main phases: initialization and traversal. An outline of the algorithm and relevant theorems is provided below.

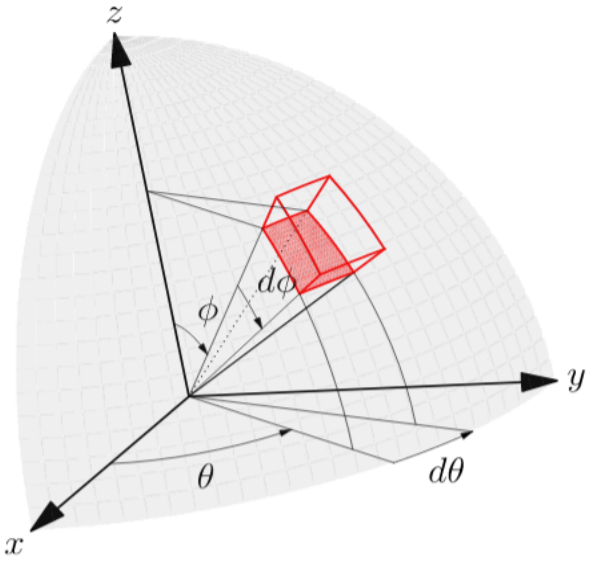


Figure 1. A spherical voxel is outlined in red. The values $d\theta$ and $d\phi$ are the polar and spherical measurements of the voxel; the radial measure of the voxel is the difference between the two spherical shells bounding the voxel (the inner shell is shaded in red)[1].

Spherical Voxel Traversal Algorithm

Givens: the number of spherical shells $N(r)$, the number of polar sections $N(\theta)$, the number of azimuthal sections $N(\theta)$, the ray origin and direction, the initial traversal time t_0, the maximum traversal time t_{end}, the maximum sphere radius r_{max}, and the center of the sphere.

Initialization

1. Determine if the ray intersects the coordinate grid (c.f. [1]).
2. Determine the initial radial voxel by comparing the radial coordinate of the ray at time t_0 to the radius of each spherical shell:
   for each spherical shell r_i for $i = N(r), \ldots, 1$ **do**
   if the radial coordinate of the ray at time t_0 is less than the radius of r_i then

[1]The image was generated following the outline of tex.stackexchange.com/questions/159445/draw-in-cylindrical-and-spherical-coordinates.

set the initial radial voxel to r_i

end

3. For each angular coordinate (Θ, ϕ), use the given total number of angular sections $(N(\theta), N(\phi))$ to find the *totally ordered* set *S* of points marking the angular voxel boundaries from $0$to 2π along the radius of the maximum spherical shell.

4. Determine the initial angular voxels using Theorem 2:
   for each angular coordinate Θ, ϕ **do**

   Step 1. Find the vector $\overline{u}$ between the ray location at t_0 and the grid center

   Step 2. Find the point of intersection $p$between $\overline{u}$ and the radius of the maximum spherical shell, r_{max}.

   Step 3. Determine the angular voxel containing p:

   for each consecutive pair of points $s_1, s_2 \in S$

   Determine if p lies in the angular voxel determined by s_1 and s_2 using Theorem 2.

   end

end

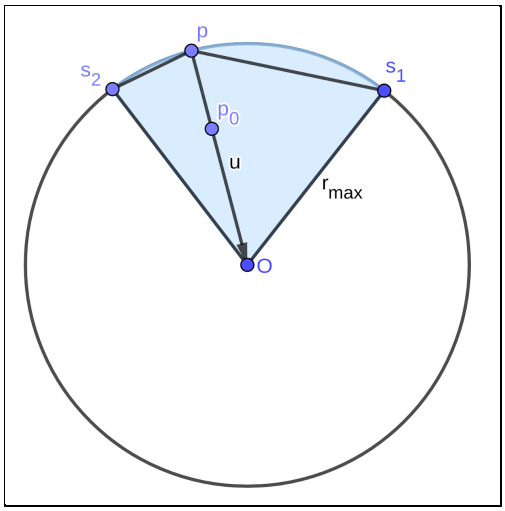


Figure 2 : Step 4 of initialization stage.

Traversal

while the current time is less than t_{end} **do**

1. Find the time of intersection between the ray and the boundaries of the current radial voxel; determine whether to increment or decrement the radial voxel ID:

Step 1. Determine the time of intersection with each of the radial voxel boundaries, r_1 and r_2(the algorithm described in [1] is sufficient). Only one of these intersection times will be less than the current time; set $tMax_r$ to this value.

Step 2. If $r_1 < r_2$, the radial voxel ID should be decremented: set $tStep_r = -1$. If $r_2 < r_1$, the radial voxel ID should be incremented: set $tStep_r = 1$. If $r_2 = r_1$, the ray is in the innermost radial voxel and the voxel ID should remain unchanged.

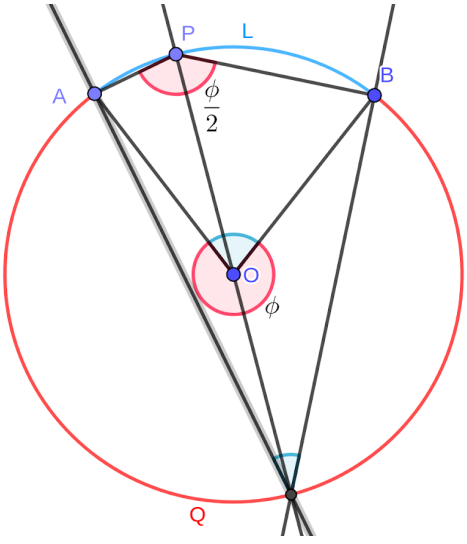
2. Find the time of intersection between the ray and the boundaries of the current angular (azimuthal and polar) voxel; determine whether to increment or decrement the angular voxel ID:

for each angular coordinate Θ, ϕ **do**

Create the vectors $\overline{Os_{min}}$ and $\overline{Os_{max}}$between the origin of the grid, O , and the boundary points $s_{min}, s_{max} \in S$ of the current angular voxel. If the ray intersects $\overline{Os_{min}}$ then set $tStep_{ang} = -1$ (indicating that that the current angular voxel ID should be decremented), if the ray intersects $\overline{Os_{max}}$ set $tStep_{ang} = 1$; set $tMax_{ang}$($tMax_{ang} = tMax_\theta \; or \; tMax_\phi$) to the corresponding intersection time. If no intersection occurs, the ray never leaves the current angular voxel during the grid traversal; the voxel ID remains unchanged and $tMax_{ang} = t_{end}$.

end

3. Compare $tMax_\theta$, $tMax_\phi$, $tMax_r$. Set the current time to the minimum of these values. Update the voxel ID of the coordinate with minimum intersection time using the relevant $tStep$.



Theorem 2. (Points on minor arc) *The point P lies on the minor arc between two points A and B on circle C with center O iff the angle ∡APB is obtuse.*

Proof (Theorem 2): The proof follows from the inscribed angle theorem: an angle θ inscribed in a circle is half of the central angle that subtends the same arc on the circle. For, if $L$is the arc on

C containing *P*, and $Q$is the arc on *C* that does not contain *P*, then the central angle ϕ that subtends $Q$is twice the inscribed obtuse angle$\measuredangle$APB and is therefore a reflex angle; the arc L on the circle *C* with center *O* is therefore minor. See Figure 3.

Figure 3: Theorem 2.

-API-

Requirements

Listed below are the build requirements to run Cython_SVR.walk_spherical_volume. All of these are already used in the yt library, and therefore will not cause issues during the integration phase.

- Python3
- Cython
- Numpy
- distutils

Currently, before use of the algorithm, the code must be compiled and linked on your machine:

```
> python3 cython_SVR_setup.py build_ext --inplace
```

The cythonized version of the algorithm is contained in a function named walk_spherical_volume with the inputs and return type listed in the tables below. Here, *np* is an alias for the Python package Numpy.

Input Arguments

Input Value Name	Type	Functionality	Notes
ray_origin	np.ndarray [np.float64_t, ndim=1, mode="c", size = 3]	The origin of the ray.	The origin of the ray may be inside or outside the sphere.
ray_direction	np.ndarray [np.float64_t, ndim=1, mode="c", size = 3]	The direction of the ray.	The ray is not assumed to be normalized. Therefore, during one unit of time, the ray will travel one iteration of the vector ray_direction.

min_bound	np.ndarray [np.float64_t, ndim=1, mode="c", size = 3]	The minimum bound in the form (radial, polar, azimuthal).	It should hold that each value of the minimum bound is strictly less than each value of the maximum bound.
max_bound	np.ndarray [np.float64_t, ndim=1, mode="c", size = 3]	The maximum bound in the form (radial, polar, azimuthal). The max_{radial} is used to determine the sphere's maximum radius. These bounds are also used to traverse over a sector of the sphere.	Example: If one wants to travel over only the upper hemisphere, then: $min_{bound} = [0,\ 0,\ 0]$and $max_{bound} = [radius_{max},\ 2\ *\ \pi,\ 2$
num_radial_sections	int	The number of radial sections used to determine the voxel division of the sphere.	To determine the delta radius, one can use the following formula: $\frac{(max_{radius} - min_{radius})}{N_{radial\ sections}}$
num_polar_sections	int	The number of polar sections used to determine the voxel division of the sphere.	To determine the delta polar value, one can use the following formula: $\frac{(max_{polar} - min_{polar})}{N_{polar\ sections}}$
num_azimuthal_sections	int	The number of azimuthal sections used to determine the voxel division of the sphere.	To determine the delta azimuthal value, one can use the following formula: $\frac{(max_{azimuthal} - min_{azimuthal})}{N_{azimuthal\ sections}}$
sphere_center	np.ndarray [np.float64_t, ndim=1, mode="c", size = 3]	The center of the sphere.	N/A
t_begin	np.float64_t	The beginning time of the ray traversal.	The time is not normalized; this is different than how yt currently implements *walk_volume*.
t_end	np.float64_t	The end time of the ray traversal.	The traversal will continue until either t_end is reached or it has exited the spherical volume.

Return Value

Return Value Type	Functionality	Notes
np.ndarray [dtype=int, ndim=3, size = N]	A Numpy array of the spherical voxel coordinates. The voxel coordinates are as follows: For coordinate i in numpy array v: v[i, 0] = radial_voxel v[i, 1] = polar_voxel v[i, 2] = azimuthal_voxel	The size N is dependent on the number of voxels traversed. Transitions can be made for one, two, or all three voxel types. In other words, the following are all possible: $\{1,\ 1,\ 1\} \Rightarrow \{2,\ 1,\ 1\}$ $\{1,\ 1,\ 1\} \Rightarrow \{2,\ 2,\ 1\}$ $\{1,\ 1\ ,1\} \Rightarrow \{2,\ 2,\ 2\}$

Example

```
# Example of the spherical voxel traversal algorithm.
# Compile code before use:
#   python3 cython_SVR_setup.py build_ext --inplace

import cython_SVR
import numpy as np

ray_origin    = np.array([-13.0, -13.0, -13.0])
ray_direction = np.array([1.0, 1.0, 1.0])
sphere_center = np.array([0.0, 0.0, 0.0])
sphere_max_radius      = 10.0
num_radial_sections    =  4
num_polar_sections     =  4
num_azimuthal_sections =  4
min_bound = np.array([0.0, 0.0, 0.0])
max_bound = np.array([sphere_max_radius, 2 * np.pi, 2 * np.pi])
t_begin   = 0.0
t_end     = 30.0
voxels = cython_SVR.walk_spherical_volume(ray_origin, ray_direction, min_bound,
                                          max_bound, num_radial_sections,
                                          num_polar_sections,
                                          num_azimuthal_sections, sphere_center,
```

```
                                                sphere_max_radius, t_begin, t_end)
# Expected voxels: [ [1, 2, 2], [2, 2, 2], [3, 2, 2], [4, 2, 2],
#                    [4, 0, 0], [3, 0, 0], [2, 0, 0], [1, 0, 0] ]
```

-Testing-

We developed a series of test cases for verification of the spherical voxel traversal algorithm described above. Two testing methodologies were included: an enumeration of the voxels traversed and a test on the chord length traversed by the ray. Enumeration test cases enabled verification of the behavior of the algorithm on edge cases; these tests also ensured the functionality of the algorithm on a relatively small grid (i.e. a small number of radial, azimuthal, and polar sections relative to the maximum spherical shell). However, for larger grids (i.e. a large number of radial, azimuthal, and polar sections relative to the maximum spherical shell), it is impossible to enumerate by hand all expected voxels traversed in the grid.

Due to the large number of floating point operations used in the spherical traversal, verification on large grids with small voxels is an important task. To verify algorithm behavior on larger grids, tests on the chord length traversed by the ray were used. Chord length tests compare the expected time the ray would spend in the grid given the maximum spherical radius with the actual time it takes the ray to traverse from voxel to voxel. For all test cases developed, even on large grids, the relative difference between the expected and actual time for ray traversal was on the order of machine epsilon.

Below, we provide a sample test case that was used in our test suite with step-by-step traversal to show the expected behavior of the algorithm.

Test: The ray begins outside of the maximum radial shell and travels through the sphere center. The test parameters are as follows:

```
ray_origin = [-15.0, 15.0, 15.0];
ray_direction = [1.0, -1.0, -1.0];
sphere_center = [0.0, 0.0, 0.0];
sphere_max_radius = 10.0;
num_radial_sections = 4;
num_angular_sections = 4;
```

```
num_azimuthal_sections = 4;
t_begin = 0.01;
t_end = 30.0;
```

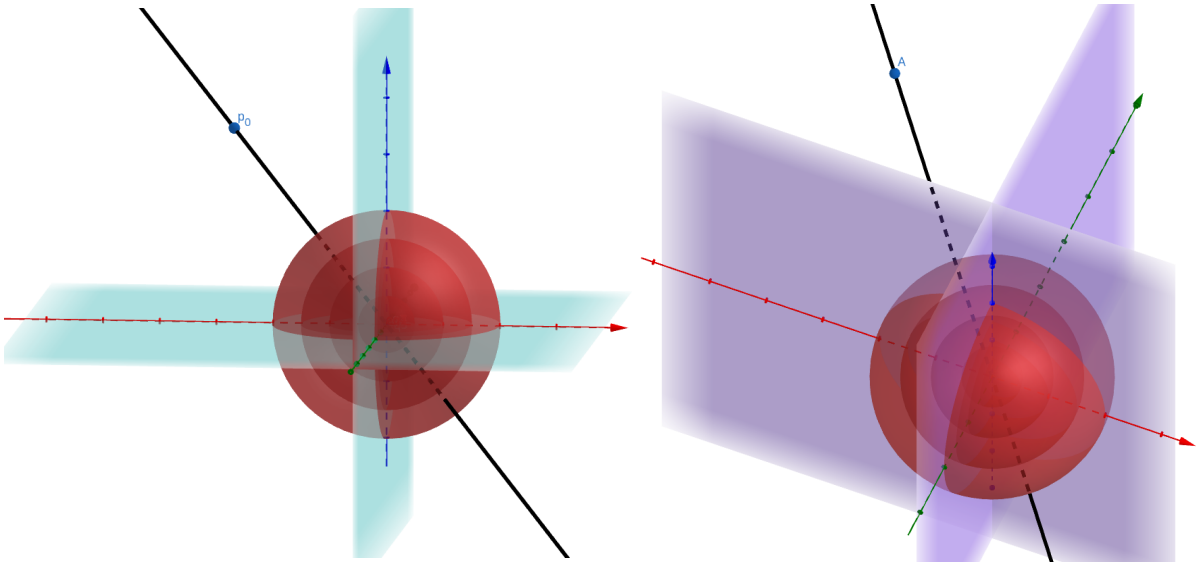


Figure 4: (Left) The azimuthal and radial voxel boundaries for the test sample. (Right) The polar and radial voxel boundaries for the test sample.

Initialization phase: The algorithm determines that the ray intersects with the spherical volume. Since the ray at the beginning time is outside the grid, the algorithm continues.

Traversal phase: The ray will traverse in radial voxels 1, 2, 3, 4 while angular and theta voxels remain the same (voxel 1 for both). When the ray crosses the origin, however, one can see in Figure 4 that both theta and phi step into voxel 3 while the radial voxel remains the same. Lastly, the ray will exit through radial voxels 3, 2, 1 while theta and phi voxels remain the same.

-Benchmarks-

The benchmark results were run on a 2013 MacBook Air (CPU and cache information listed below) using Google Benchmark. Each benchmark sends rays orthographically through the sphere, following two criteria: (1) the ray intersects the sphere, and (2) the ray travels through the entire sphere. Neither parallelization nor concurrency is used. A

scratch paper goal set by the client was 128^2 rays through a 64^3 voxel domain in less than one second. Currently, the algorithm achieves this in less than 0.3 seconds with appropriate optimization flags.

# Rays	# Voxels	CPU Mean (ms)	CPU Median (ms)	CPU Std Dev (ms)
128^2	64^3	233	233	1.0
256^2	64^3	928	926	4.5
512^2	64^3	3717	3713	24.1
128^2	128^3	463	462	2.2
256^2	128^3	1838	1833	12.0
512^2	128^3	7346	7324	16.9

*Run on (4 X 1600 MHz CPUs). CPU Caches: L1 Data 32 KiB (x2), L1 Instruction 32 KiB (x2), L2 Unified 256 KiB (x2), L3 Unified 3072 KiB (x1)

-References-

1. Paul S. Heckbert, editor. Graphics Gems IV. Academic Press Professional, Inc., USA, 1994.