Description of the Reachability Set Adaptive Mesh Algorithm

Erik Komendera

Department of Computer Science
University of Colorado
Boulder CO 80309-0430

Technical Report CU-CS-1090-12
February 2012

Description of the Reachability Set Adaptive Mesh Algorithm

Erik Komendera

February 14, 2012

Abstract

This paper will describe the algorithms that comprise the reachability set project. First, the parameters will be described. Then, from the highest level to the lowest level, the algorithms used will be described. Finally, data from initial tests will be presented.

1 Parameters

The following is a list of parameters used in the reachability set algorithms, with a brief description of what each does. The parameters are listed in order of input from a text file. Where there is an indentation, the indented parameters are only used if the head parameter is used.

- System Type S_{type} : Only "ThreeBody".
- S_{type} = ThreeBody:
    - μ : The μ parameter for the three body problem. Body one will be placed on the x-axis at (1.0-μ), body two at (-μ), and neither will move in the frame of reference.
    - R_1 : The radius of body one.
    - R_2 : The radius of body two.
    - Impact DT Δt_i : The interval at which samples are taken to see if a trajectory impacts a planet. If this paramter is too large, a trajectory may pass through a planet without being caught, creating a false negative.
    - Escape Freeze f_e : True causes trajectories to be frozen if they escape as found by the Kepler Energy condition.
- Vertex Count C_V : The initial count of vertices (trajectories).
- Add Vertex Count C_{AV} : At each resampling of points, the number of new vertices to add.

- Delete Old Vertices d_{ov} : If true, instead of adding new vertices to the mesh, the mesh is completely replaced by the C_{AV} new vertices.
- Dimensions D : The number of dimensions for the ΔV sphere. Can range from 0-6, but only 2 and 3 are used.
    - Sphere Indices $\mathbf{I}_S$: The D indices of the ΔV sphere. 0,1,2 stands for X, Y, Z, and 3,4,5 stands for V_X, V_Y, V_Z. For example, a 2-dimensional sphere along the V_X and V_Y axes would have sphere indices of 3 and 4.
- Initial Center $\mathbf{C}$: A 6D vector containing the starting position and velocity to be considered. Each chosen point in the ΔV sphere is added to the initial center for that trajectory.
- Sphere Radius R_S : The radius of the ΔV sphere.
- Start Time T_0 : A float representing the beginning of the test. 0.0 is only used in the tests.
- Max Phases P_{max} : The number of phases. Each phase is an interval of time. At the end of a phase, a check is performed to see if more vertices should be added. The Runge-Kutta algorithm is always run over a single phase. Each additional phase has the same starting time as the previous one.
- Phase Length Δt_{phase} : The time interval for a phase.
- Data Per Phase N_{phase} : The number of states saved for a trajectory for the duration of a phase. By limiting this value, memory resouces can be conserved.
- Subdivision Function $f_{sub}()$: The name of the heuristic function used to intelligently resample the mesh. Can be "EndResult", "Growth", "Growth-NoFreeze", or "Size".
- $f_{sub}()$ = EndResult:
    - Fraction of Subdivided Edges With Different End Results f_{diff} : The expected fraction of edges randomly chosen for subdivision that have different end results. If this value is 1, every edge chosen for subdivision will have differing end results.
    - Bounded Vertices N_{bound} : The number of initial seed vertices that are placed on the boundary of the ΔV sphere. If a seed ΔV sphere has very few vertices, the resultant sphere will not resemble a sphere, and the addition of new points does not favor rounding out the sphere, so forcing a round shape is necessary to get the full ΔV sphere.
- $f_{sub}()$ = Growth: no specific inputs are required.

- $f_{sub}()$ = GrowthNoFreeze or Size:
  - No Growth If All Have Same End Result g_{diff} : If all vertices in a simplex have the same end result, do not allow that simplex's growth to be considered for future remeshing.
  - Check Impact Without Freezing i_{keep} set to True (not input): This prevents trajectories from being terminated (frozen) if they impact.
  - Escape Freeze f_e set to False (not input): Do not terminate trajectories if they escape.
- Check Freeze i_{freeze} : If True, this will terminate trajectories if they impact. Cannot be True if i_{keep} is also True.
- Check All Simplices Σ_{check} : If False, then only simplicies that are new as of the last remeshing are considered for remeshing. All tests set this to True.
- Random Factor F : The size of the first standard deviation in remeshing.

  For edges, this is the size of the semi-major axis of the one-standard-deviation covariance ellipsoid as a multiple of half of the length of the edge squared. For example, if the length of the edge is L, the semi-major axis will be $F\ L^2$. The semi-minor axes are always half the length of the semi-major axis. A randomly resampled vertex on this edge has a 68 percent chance of being within the covariance ellipsoid (within one standard deviation). The smaller the factor, the closer an expected new vertex will be to the center of the edge.

  For other simplices, this represents how close to a simplex a random point is expected to be. If $F = 1$, then the chance of a point being within the simplex is one standard deviation. If $F = 2$, there is a one-standard-deviation chance of the point being within a doubly-scaled simplex concentric with the simplex in question. In general, the lower that F is, the more likely that a new point will be near the center of the simplex.
- Weight Exponent e_w : For weighted selection of simplices, the exponent applied to the previously calculated weight.
- Bounded Factor B : If true, this limits how large and small weights can become. This is in place to counteract wildly varying weights and to prevent, for example, one extremely highly weighted simplex from being subdivided for every single addition to the mesh.
  - Low Factor B_{low} : In the sorted list of weights, the lowest fraction of weights are set to the weight at the Low Factor entry in the list.
  - High Factor B_{high} : In the sorted list of weights, the highest fraction of weights are set to the weight at the High Factor entry in the list.

- No Subdivide Below Length L_{min} : If the D th root of the volume of a D -dimensional simplex is smaller than this, it will not be considered for future subdivision.
- Mesh Indices $\mathbf{I}_M$: A 3D set of indices indicating which of the six dimensions are to be plotted for visualization.
- Max Checks Per Phase C_{phase} : How many times per phase that a remeshing can be performed. Setting a limit prevents an infinite amount of remeshing, and allows for the later phases to be simulated.
- Times To Run N_{runs} : The number of times this particular test is to be performed.

2 Algorithms

2.1 Reachability Set

This is the top level algorithm. It controls the creation of the reachability set and its simplices, the execution of the Three Body system of equations, the adaptive remeshing of the simplices, the removal of erroneous trajectories, and the addition of new vertices.

1. Make and fill a unit ΔV sphere with an initial set of C_V vertices, and create a Delaunay triangulation of those vertices using the Bowyer-Watson algorithm. Call the set of vertices V_R, and the sets of edges, triangles, and tetrahedrons S_2, S_3, S_4.
2. Scale the unit ΔV sphere to a radius of R_S , and add it to $\mathbf{C}$ to find the initial trajectory starting points. Set their current phase to 0. A trajectory is considered finished when its current phase is equal to P_{max} .
3. While there are still trajectories that have not been fully explored:
   (a) Find the minimum phase p_{min} not yet calculated for some or all of the trajectories. Because there are new vertices added every loop, the current phase for any vertex may differ from that of another vertex.
   (b) If all trajectories are frozen, quit the loop. A frozen trajectory is one that has impacted or escaped, and if all are frozen, no further integration is possible.
   (c) If p_{min} is greater than P_{max} , quit the loop. This means that every phase has been completed for every vertex. This happens when the algorithm discontinues the addition of vertices.
   (d) Find all vertices that need to be updated for the p_{min} phase. Call this set V_M.
   (e) Call Mathematica to run NDSolve on V_M over Δt_{phase} , and collect the N_{phase} samples of state data for that phase.

i. Send state and system data to Mathematica to be run on its native Runge-Kutta solver.

ii. If i_{freeze} or i_{keep} are true, do a search to find the precise time that a trajectory impacts a body. Keeping track of impacts allows for intelligent decisions to be made based on impact data.

iii. If i_{freeze} is true, remove all states that occur after the impact time and declare the trajectory "frozen". This prevents a failed trajectory from demanding unnecessary computational time.

(f) Update the phase count for the updated vertices. This prevents duplication of data.

(g) Check to see if any of V_M escape, using the Keplerian Energy requirement: if the positional distance from the origin is greater than 10 units, and the energy of the trajectory is above 0, then the trajectory has escaped. If f_e , remove all states that occur after the impact and declare the trajectory "frozen".

(h) Check to see if any trajectories in V_M blew up. A trajectory is considered blown up if the positional distance from the origin is higher than 10000 units. This is a system instability that occurs if a trajectory skims too close to a center of gravity. Call the set of blown up trajectories V_{blown}

(i) Delete the V_{blown} blown up trajectories and recalculate the S_2, S_3, S_4 using the Bowyer-Watson Delaunay triangulation algorithm. Doing so maintains a ΔV sphere in which every point belongs to a tetrahedron.

(j) Using $f_{sub}()$ (Growth, End Result, or Flatness), define a new set of vertices equal to C_{AV} $+V_{blown}$, call this set V_N. These vertices will be given a starting position in the unit ΔV sphere of D dimensions. Do not add any new vertices if C_{phase} has been exceeded for this particular phase; using an upper limit of C_{phase} allows future phases to be executed.

(k) If there are new vertices:

i. If d_{ov} is true, $V_R \leftarrow V_N$.

ii. Using the Bowyer-Watson algorithm, recalculate the Delaunay triangulation to find S_2, S_3, S_4.

iii. Scale the unit ΔV sphere to R_S , and add it to the initial center to find the initial trajectory starting points for the new vertices. Set their current phase to 0. This indicates that the new vertices have to start at the first phase.

4. Return V_R, S_2, S_3, S_4.

5. Save both the returned data and the parameter set for future examination.

2.2 End Result Heuristic

The intention of the End Result heuristic is to focus subdivision on the boundaries between end result conditions. By repeatedly adding new vertices in the vicinity of boundaries, the end result regions can be identified to a higher precision. This, in turn, allows a planner to predict with higher accuracy whether a trajectory will impact, escape, or stay in the system for a time span $\Delta 4$. This heuristic only looks considers edges (S_2) for subdivision.

1. Define two empty lists of item-weights for simplices that will be considered for subdivision, called Different Weights W_d and Same Weights W_s. Two lists are required because the algorithm requires a fraction f_{diff} of new vertices to be selected from the set of edges with different end results. Allowing subdivision for some edges that have the same end result provides a non-zero chance that a hidden region may be revealed in a region that has no obvious boundaries.

2. If Σ_{check} is false, consider only those edges with at least one vertex just updated. No tests set Σ_{check} to false, but the benefit of doing so is that it forces further examination of regions that were previously updated. The major problem with Σ_{check} = false is that regions that were skipped over will continue to be skipped, resulting in a lesser-quality mesh. Thus, all tests set Σ_{check} to true.

3. Define a set of edges, Different Ends E_d, whose end results (impact either body, escape, or stay in system) are different, and a set of edges, Same Ends E_s, whose end results are the same.

4. For each edge in S_2:

   (a) If the edge is smaller than L_{min} in the ΔV sphere, skip.

   (b) Otherwise, place it into either E_d or E_s based on whether its vertices' end results differ.

5. For each of the sets E_d and E_s:

   (a) Calculate weight: L^{e_w}, and place the edge-weight pair in either W_d or W_s. The weight exponent e_w allows for variation of the importance of long or short edges. For example, if $e_w > 1$, the weight scheme favors long edges.

6. Sort both W_d and W_s by weight.

7. For each of the sets W_d and W_s:

   (a) If B is true, change the range of allowable weights to be a subset in the middle of either list. This helps eliminate cases with extremely high or low weights:

   i. The lower bound of weights is set to $W_d[B_{low} \ /Length(W_D)]$ or $W_s[B_{low} \ /Length(W_D)]$.
   ii. The higher bound of weights is set to $W_d[B_{high} \ /Length(W_D)]$ or $W_s[B_{high} \ /Length(W_D)]$.
   iii. For all weights outside the bounds, change the factor to the nearest bound.

8. Use the Edge Subdivision Random Algorithm with W_d and W_s to get a set V_N of vertices with a count $C_{AV} \ +V_{blown}$.

9. Return V_N.

2.3 Growth and Size Heuristics

The intention of the growth and size heuristics is to focus the adaptive meshing toward regions that grow fast or have a large size. The intuition here is that large simplices will necessarily have large errors internally due to their scope. By focusing subdivision on such regions, tests have shown that the overall error of the mesh is reduced in most cases.

1. Define an empty list of item-weights W for simplices that will be considered for subdivision.

2. For each simplex in one of S_2, S_3, S_4:

   (a) If Σ_{check} is false, and this simplex has no vertices that were just updated, skip. Like with the end result heuristic, Σ_{check} is always true in the tests.

   (b) If the D -dimensional volume of this simplex is less than L^D_{min}, skip.

   (c) If g_{diff} is true and this simplex's vertices all have the same end result, skip. This condition incorporates some of the end result heuristic in that it forces examination at regions with boundaries, but none of the tests set g_{diff} to true.

   (d) If $f_{sub}()$ = Growth: Find the volume of the simplex in state space at the beginning V_b and the end V_e of the phase time interval Δt_{phase} for the minimum phase p_{min}. This estimates the rate of growth for a simplex over Δt_{phase} . If $f_{sub}()$ = Size: Find the volume of the simplex in state space at the end of the phase time interval Δt_{phase} for the minimum phase p_{min}.

   (e) If the volume at the end of Δt_{phase} is less than or equal to 0, skip. This is an indicator that the simplex is degenerate, and should not be considered.

   (f) If $f_{sub}()$ = Growth: The weight is $(V_e/V_b)^{e_w}$. If Size: The weight is $V_e^{e_w}$. Like the End Result heuristic, the weight exponent e_w allows for the modification of the importance of large volume ratios or large sizes.

    (g) Add the simplex-weight pair to W.

3. Sort W by weight.

4. For each item in W:

    (a) If B is true, change the range of allowable weights to be a subset in the middle of either list. This helps eliminate cases with extremely high or low weights:

        i. The lower bound of weights is set to $W[B_{low} / Length(W_D)]$.

        ii. The higher bound of weights is set to $W[B_{high} / Length(W_D)]$.

        iii. For all weights outside the bounds, change the factor to the nearest bound.

5. Use the Simplex Subdivision Random Algorithm with W to get a set V_N of vertices with a count $C_{AV} + V_{blown}$.

6. Return V_N.

2.4 Edge Subdivision Random Algorithm

This algorithm randomly chooses an edge from either W_s or W_d repeatedly until $C_{AV} + V_{blown}$ new vertices have been added. Since edges can be chosen multiple times, the Multivariate Vector Random Position algorithm chooses the locations of new vertices to minimize the issue of degeneracy. The parameter f_{diff} forces the selection of edges from W_s or W_d to adhere to a user-specified ratio over the long run.

1. Define an empty set New Vertices V_N.

2. While the length of V_N is less than $C_{AV} + V_{blown}$:

    (a) Choose a random number ρ between 0 and 1. ρ is used to determine whether to choose an edge from W_S or W_D.

    (b) Define E as the identity of the edge chosen for subdivision.

    (c) If the number of edges in W_S is zero, call Weighted Random Choice on W_D to choose the subdividing edge. Because W_S has no elements, W_D is forced to be chosen regardless of the value of ρ.

    (d) Else if the number of edges in W_D is zero, call Weighted Random Choice on W_S to choose the subdividing edge.

    (e) Else if ρ is less than or equal to f_{diff} , call Weighted Random Choice on W_D to choose the subdividing edge. The parameter f_{diff} guarantees over the long run that the number of edges with different weights chosen for subdivision goes to f_{diff} .

    (f) Else if ρ is greater than f_{diff} , call Weighted Random Choice on W_S to choose the subdividing edge.

   (g) Use the Multivariate Vector Random Position function on E to randomly place a new vertex in the vicinity of E.

   (h) Add the new vertex to V_R and to V_N.

3. Return V_N.

2.5 Simplex Subdivision Random Algorithm

This algorithm randomly chooses an edge from W repeatedly until C_{AV} $+V_{blown}$ new vertices have been added. As in the Edge Subdivision Random Algorithm, simplices may be chosen several times for the addition of new vertices; therefore simply placing new vertices at the center of a simplex may result in degeneracy. Thus, this algorithm calls the Simplex Shell Method to randomly place a new vertex in the vicinity of a simplex.

1. Define an empty set New Vertices V_N.
2. While the length of V_N is less than C_{AV} $+V_{blown}$:
   (a) Choose a simplex S from the set of subdividing simplices using Weighted Random Choice.
   (b) Use the Simplex Shell Method function to randomly place a new vertex in the vicinity of S.
   (c) Add the new vertex to V_R and to V_N.
3. Return V_N.

2.6 Weighted Random Choice

This algorithm allows for the random choice from a list in which each element has a weight associated with it.

1. Starting from 0 for the first choice, and $\sum_{i=1}^{X-1} W_i$ for choice X, calculate the valid range of values for the choice. Each valid range for a choice X is $[\sum_{i=1}^{X-1} W_i, \sum_{i=1}^{X-1} W_i + W_X]$.
2. Pick a uniform random number R between 0 and $\sum_{i=1}^{N} W_i$.
3. Find the choice C_h whose range's lower bound is less than R and whose range's upper bound is more than R.
4. Return C_h.

2.7 Multivariate Vector Random Position

Adding new points in the area of an edge poses a few problems that this algorithm addresses. Simply putting a new vertex at the midpoint of an edge will guarantee a degenerate Delaunay triangulation. Likewise, placing multiple new vertices may also lead to degeneracy if it is not done carefully. Since the choice of edges is randomized, one edge may be chosen several times in a round. Thus, this algorithm places new vertex by selecting from a distribution shaped like an ellipsoid in which the semi-major axis is half the length of the edge, and the other axes are half the length of the semi-major axis. If F is one, then there is a one-standard-deviation chance that a new vertex will be inside the ellipsoid that spans from vertex to vertex.

Conveniently, ellipsoidal distributions can be made with covariance matrices whose eigenvectors are all normal to each other. Since one eigenvector will be parallel to the edge, the other eigenvectors can be arbitrarily defined so long as all eigenvectors are orthogonal. Once the covariance matrix is found, the Multivariate Normal Distribution can be used to choose a random position for a new vertex.

1. Create a covariance matrix centered at the midpoint of the chosen edge, whose distribution resembles an ellipsoid with a one-standard-deviation semi-major axis of length equal to one half the random factor times the square of the length of the edge, and two semi-minor axes whose lengths are one quarter the random factor times the square of the length of the edge:
   (a) Define vector A to be $V_2 - V_1$, where V_i refers to an endpoint of the edge.
   (b) Define Magnitude M to be $1/2FL^2$, where F is the Random Factor, and L is the length of the edge. If F is one, then the one-standard-deviation ellipsoid semi-major axis will be exactly half the length of A. M is the major eigenvalue whose vector is parallel to the edge, and $1/2M$ is the value of the other eigenvalues.
   (c) If D is 2:
      i. Define vector B to be a vector normal to A. This will be the semi-minor axis.
      ii. Define vectors n_A, n_B to be the normalized versions of A and B.
      iii. Define an eigenvector square matrix:
      $$Q = \begin{bmatrix} n_{Ax} & n_{Bx} \\ n_{Ay} & n_{By} \end{bmatrix}$$
      .
      iv. Define an eigenvalue diagonal matrix:
      $$E = \begin{bmatrix} M & 0 \\ 0 & 1/2M \end{bmatrix}$$
      .

(d) If D is 3:

   i. Define vectors B_1, B_2 to be two vectors normal to both each other and to A. These will be the semi-minor axes.

   ii. Define vectors n_A, n_{B_1}, n_{B_2} to be the normalized versions of A, B_1, B_2.

   iii. Define an eigenvector square matrix:

$$Q = \begin{bmatrix} n_{Ax} & n_{B_1x} & n_{B_2x} \\ n_{Ay} & n_{B_1y} & n_{B_2y} \\ n_{Az} & n_{B_1z} & n_{B_2z} \end{bmatrix}$$

.

   iv. Define an eigenvalue diagonal matrix:

$$E = \begin{bmatrix} M & 0 & 0 \\ 0 & 1/2M & 0 \\ 0 & 0 & 1/2M \end{bmatrix}$$

.

(e) Define $Qi = Q^{-1}$.

(f) Calculate the covariance matrix $C = Q * E * Qi$.

(g) Calculate $Ci = C^{-1}$, $D_C = Determinant(C)$. These values are used for making a new vertex.

2. Calculate P_m, the maximum probability density, which occurs at the mean of the multivariate Gaussian distribution. This provides an upper bound for rejection sampling — the probability density will not be higher at any location. This, in turn, reduces the average number of rejections before a good point is selected. Use the following equation for calculating the probability density of the multivariate Gaussian distribution for a vector $\mathbf{x}$ in ΔV space:

$$F(x_1, \ldots, x_k) = \frac{1}{(2\pi)^{k/2} D_C^{1/2}} e^{-1/2(\mathbf{x}-\mu)^T Qi(\mathbf{x}-\mu)}$$

   where μ is the mean vector in ΔV space.

3. Until a random vector is calculated, use Rejection Sampling:

   (a) Create a randomized position vector R_C subject to the constraints that the distance is within $3L$ from the midpoint of the edge, and also inside the unit ΔV sphere. The $3L$ constraint is necessary because the probability density for an edge is low enough throughout the ΔV sphere that an enormous number of rejections are possible before a point is chosen. Culling the choices to be within $3L$ reduces the number of rejections while modifying the probability density function only for extreme cases.

   (b) Calculate a random uniform variable $V \in [0, P_m]$. V will be subject to rejection sampling. If V is lower than the probability density for R_C, it will be accepted.

   (c) Find the probability density P at R_C using the multivariate Gaussian distribution.

   (d) If $V \leq P$, return R_C. Else, V has been rejected: restart loop.

2.8 Simplex Shell Method

The Simplex Shell Method allows for new vertices to be added in the vicinity of a simplex following a distribution that is shaped like the simplex itself. Like the Multivariate Vector Random Position method, a Gaussian distribution is incorporated. In this case, each simplex "shell" (surface) concentric to the chosen simplex has a specific probability density. The smaller the shell, the higher the density, with the simplex centroid having the highest density. The routine chooses a shell of some size, and randomly places a new point somewhere on the surface of that shell.

Once the shell's size has been chosen (on a scale in which 1 means the shell is the same size as the simplex), the next step is to determine which face will have the point. Each face has the same chance of being chosen, regardless of its size, and the point will be uniformly distributed on that face.

The facilitate the calculation of positions, barycentric coordinates are used. For a tetrahedron, a point $\mathbf{X}$ can be converted to a set of barycentric coordinates L_1, L_2, L_3, L_4 using the following relation: $\mathbf{X} = L_1\mathbf{X_1} + L_2\mathbf{X_2} + L_3\mathbf{X_3} + L_4\mathbf{X_4}$ where $\mathbf{X_i}$ is the ith vertex of the tetrahedron. For triangles, just remove the fourth term. Barycentric coordinates have the additional constraint that $L_1 + L_2 + L_3 + L_4 = 1$. For a point $\mathbf{X}$ to be on the surface of the tetrahedron, at least one of L_1, L_2, L_3, L_4 must be exactly 0, and the others must fall in the closed interval $[0, 1]$. Therefore, generating random points on the surface of the simplex must create a tuple of L_i values such that all these conditions are met. One index will be chosen to be 0, one index will be set aside for summation, and the others will be randomly selected from $[0, 1]$. Once those values are defined, the summation index will be found as $L_j = 1 - \sum_{i=1, i\neq j}^{N_v} L_i$.

1. Calculate the center of the simplex and call it C.

2. Define a scale factor S to be a randomly chosen point on the normal distribution with mean 0 and standard deviation F, where F is the Random Factor.

3. Define N_v =D +1 to be the number of vertices in the simplex.

4. Until a random vector is calculated:

   (a) Randomly choose an index $I_e \in [1, N_v]$ to represent the surface simplex on which to place this point. This is the surface simplex index.

(b) Randomly choose an index $I_s \in [1, N_v], I_s \neq I_e$, to represent the index reserved for fulfilling the summation $L_j = 1 - \sum_{i=1, i \neq j}^{N_v} L_i$ for barycentric coordinates.

(c) Define L_C to be a randomized barycentric position where $L_{C_i} \in [0, 1], L_{C_{I_e}} = 0$.

(d) Calculate $L_{C_{I_s}} = 1 - \sum_{i=1, i \neq I_s}^{N_v} L_i$.

(e) If $0 \leq L_{C_{I_s}} \leq 1$, then L_C is a barycentric vector on the surface of the simplex. Else, restart the loop.

(f) Calculate R to be the ΔV space transformation of L_C into a position on the surface of the simplex using $\mathbf{R} = L_1\mathbf{X_1} + L_2\mathbf{X_2} + L_3\mathbf{X_3} + L_4\mathbf{X_4}$.

(g) Transform R onto the surface of the shell by calculating $R_S = C + S(R - C)$.

(h) Finally, if R_S is inside the unit ΔV sphere, return R_S. Else, restart the loop.